软件工程

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/872402a1-9d49-407b-8d84-c77d1c4ad1cc/%E8%8B%B1%E6%96%87%E7%89%88%E8%BD%AF%E4%BB%B6%E5%B7%A5%E7%A8%8B%E5%A4%8D%E4%B9%A0%E9%A2%98202211%E6%96%B0%E7%89%88.docx

双斜杠：不是重点；

了解：出现在小题；

熟悉/掌握：重要，出现在大题；

其他的罗列一下，别忘了

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/259701bd-8608-4981-b162-b69cbb47432c/%E8%8B%B1%E6%96%87%E7%89%88%E8%BD%AF%E4%BB%B6%E5%B7%A5%E7%A8%8B%E8%AF%95%E9%A2%98%E6%A8%A1%E6%8B%9F%E8%AF%95%E9%A2%982022%E4%B8%8B%E5%8D%8A%E5%B9%B4.docx

复习以课件为主

10个名词解释每个2分

10个判断题每个1分

单项选择题每个2~3分共20分

5个简述题每个题6分（教学回顾内容）

2个综合应用题共20分，第一个类似于例题1/2，第二个类似于第3，

一、对《英文版软件工程复习题202211新版》的见解

第一章 为什么学习软件工程

1.SE的定义、目的、方法及作用：

定义：软件工程由用来明确问题的性质和通过开发工具实现的解决方案组成。

目的：提供高质量软件的解决方案，同时考虑有助于提高软件质量的特性。

方法：

1. 分析：分析问题，从正反两方面测试检查软件。
2. 设计：给出解决方案。
3. 开发团队：对参与到开发的人员描述其角色和责任。
4. 开发：实现软件的解决方案，如棉线对象、活动、封装、测试等。
5. 项目管理：将系统分为多个部分，按步骤定义流程，控制并处理每一种变化。

2.说明错误、缺陷、失败的含义与联系。（请举例说明）

含义：

> 错误error：由于对需求的误解或者错误的代码而导致程序人员在软件生成中产生错误。

缺陷fault：在函数实现中出现的问题。**是客观存在的，一个错误可能导致一个或多个缺陷。**

失败failure：由于缺陷而导致软件运行失败，**是动态产生的**。

联系：

一个错误可能导致一个或多个缺陷，缺陷在测试或者使用过程中会导致软件运行与预期不符，也就是导致了失败。

例子：

例如，软件希望统计一年中国人均收入，而由于编程人员的错误，将软件设计成了统计一年中国所有人的收入总和，这就导致了一个缺陷存在。当测试人员测试软件时候发现软件给出的数据不是中国人均收入，这也导致了软件的运行失败。如果测试人员没有测试这个功能，虽然缺陷仍然静态存在，但是失败动态地消失了。

3.软件质量应从哪几个方面来衡量？论述之。

1. 最终产品的质量，对用户来说拥有满足需求的功能，简单易学的操作；对开发者来说有软件内部的特点。
2. 软件开发及维护过程中的质量，因为众多过程影响了最终产品的质量，因此开发及维护过程拥有同等的重要性。
3. 商业应用背景下的软件商业质量，应该在软件中将技术价值和商业价值统一起来。

4.现代软件工程大致包含的几个阶段及各个阶段文档。

1. 需求分析—《SRS》软件需求规格说明书
2. 系统设计—《SAD》系统结构图
3. 程序设计—模块功能算法与数据描述
4. 程序实现—源代码和注解
5. 单元测试—测试报告
6. 集成测试—测试报告
7. 系统测试—发布前的最后测试

8. 系统提交
9. 维护—维护报告

5.什么是软件过程？软件过程的重要性是什么？包含几个阶段？

软件过程：

C1：软件开发活动中的各种组织及规范方法

C2：软件开发活动中产生某种期望结果的一系列有序任务，设计活动、约束和资源

重要性：

1. 通用性，也叫一致性/结构性，可以使我们知道是否已经做好了工作，同时还能使别人以同样的方式做工作。
2. 自我指导性，可以总结讲演，完善规范，指导新版本的产生

几个阶段：

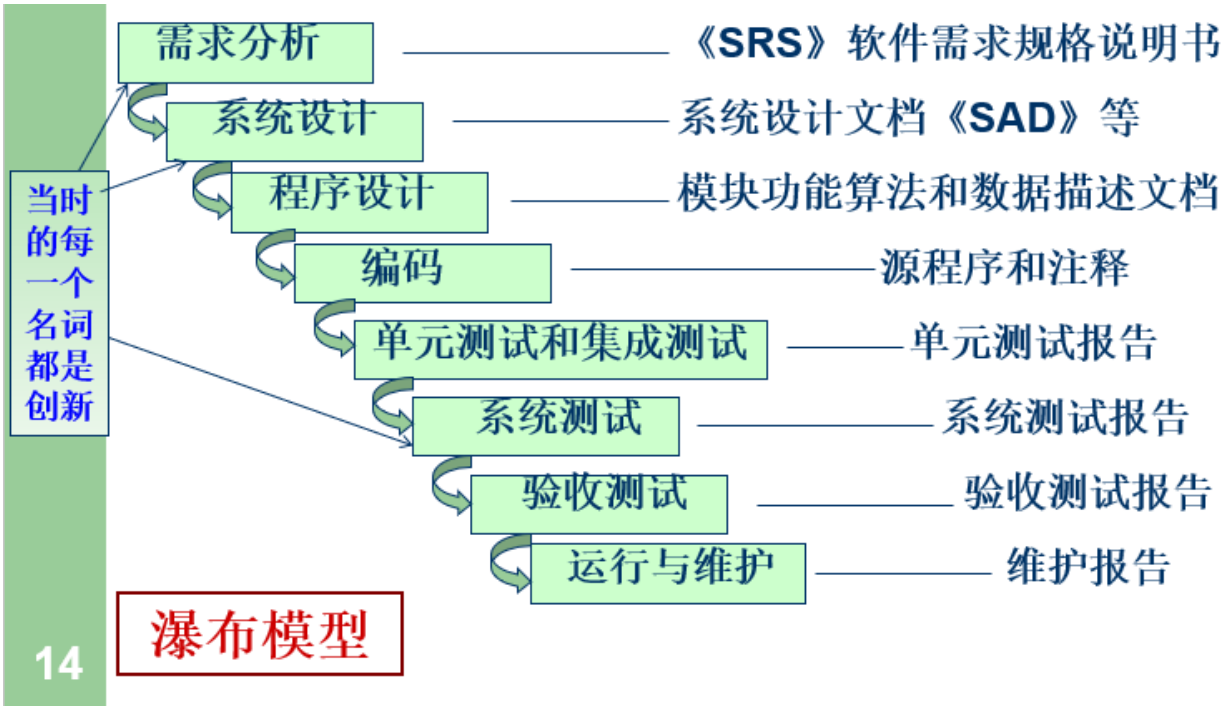
上述九个阶段

6.重用、抽象等现代软件工程Wasserman所述的主要概念

1. 抽象，基于某种层次归纳水平的问题描述，它使我们将注意力集中在问题的关键方面而非细节。
2. 分析、设计方法和符号描述系统，即建模原语/表示法，采用标准的符号表示系统，利于交流，利于建模并检查其完整性和一致性，易于对需求和设计部件进行重用。
3. 用户界面原型化，建立系统的小型版，通常具有有限但关键的功能，以利于用户评价和选择。
4. 软件体系结构，定义了一组体系结构单元及其相互关系集来描述软件系统。
5. 软件过程，即软件开发活动中的各种组织及规范方法。
6. 重用，重复采用以前开发的软件系统中具有共性的不见，用到新的开发项目中去。
7. 度量、测度。一种通用的评价方法和体系评价进程、资源和方法，增加质量。用类数学的语言来描述和评价行动与成果。
8. 开发框架或工具的选择。

第二章 过程模型和生命周期

1.瀑布模型及各阶段文档，优缺点？



优点：

1. 朴素易懂，能够和用户简单地说清楚。
2. 是其他复杂模型的基础，复杂模型添加反馈环和额外的活动。

缺点：

1. 面临软件变动时，无法处理实际过程中的重复开发问题。
2. 文档转换存在困难，难以从一种文档转换成另一种。

2.原型的概念与用途

概念:

> 是一种部分开发的产品，用来让用户和开发者共同研究、提出意见，为最终产品定性。

用途（优点）

1. 用来原型化需求和设计，支持对系统进行多次改进。
2. 可以创建多种解决方案供用户选择。

3.论述分阶段开发模型的含义, 其基本分类及特点是什么？

分阶段开发模型：

系统被设计成部分提交，每次用户只能得到部分功能，而其它部分处于开发的过程中。

基本分类：

分为两个系统：

1. 产品系统，即用户正在使用的版本。
2. 开发系统，即准备代替现有产品系统的下一个版本。

特点：

增量式和迭代式，是原型化模型的改进。

增量式：系统需求按照功能分为若干个子系统，开始建造的版本是规模小的、部分功能的系统，后续版本添加包含新功能的子系统，最后版本是包含全部功能的子系统集。

迭代式：系统开始提供整体功能框架，后续版本陆续增强各个子系统，最后版本使各个子系统功能达到最强性能。

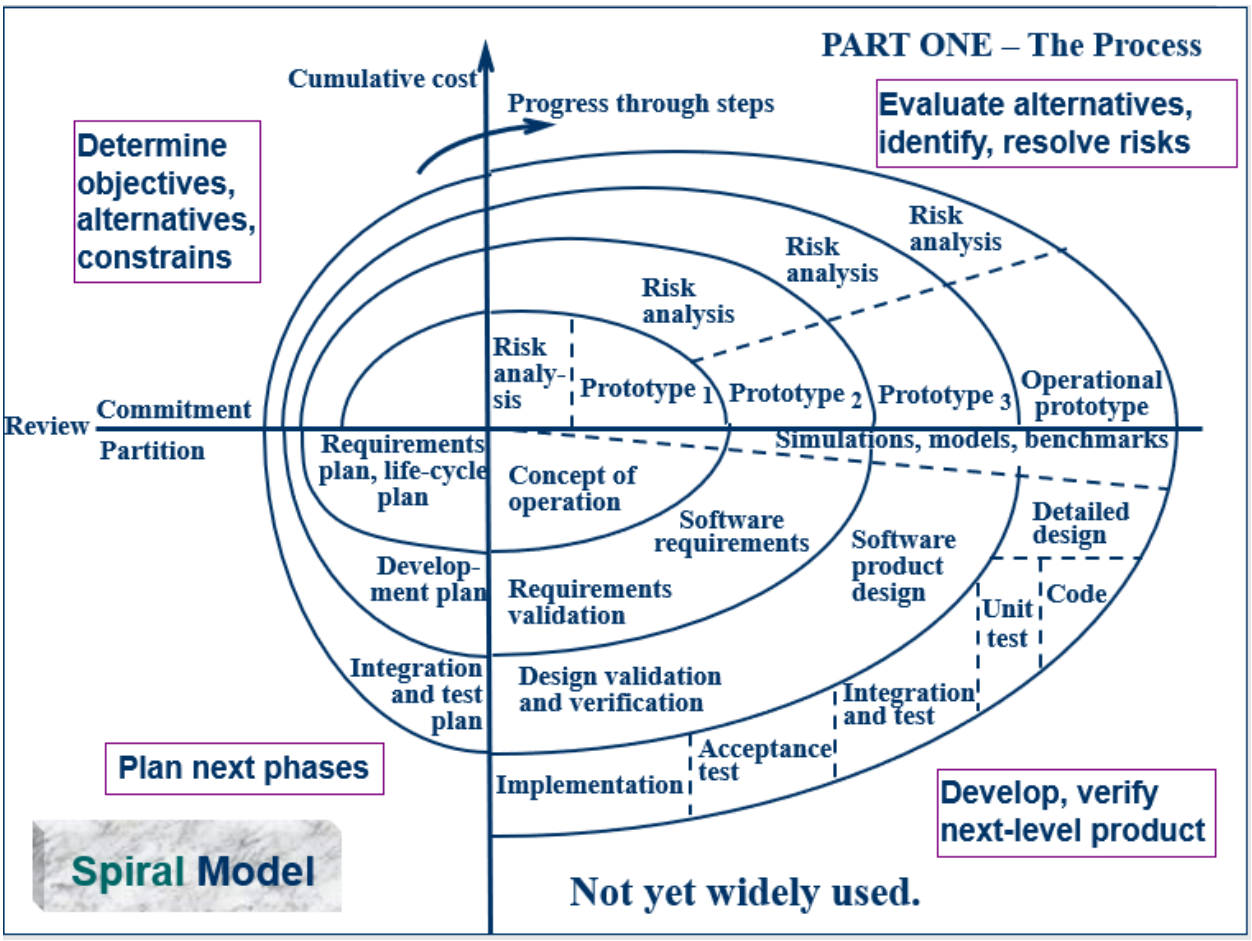
4.螺旋模型四个象限的任务及四重循环的含义？

螺旋模型：将开发活动与风险管理结合起来，降低和控制风险。

四个象限的任务：计划、目标/可选方案、风险评估、开发与测试。（3象限开始）

四重循环：操作概念、软件需求、软件设计、系统实现与部署运行。

每一次循环都根据需求和约束进行风险分析。



5.什么是敏捷方法？以及其代表性方法？

敏捷方法：

> 一种较新型的软件开发方法，不要求遵循传统的软件开发流程，强调快速开发和有效适应需求变化。

代表性方法：

> 极限编程、测试驱动开发。

6.什么是UP， RUP，进化式迭代等市场流行的过程模型?

UP：统一过程，它是用例驱动的、以基本架构为中心的、迭代式和增量性的软件开发过程框架。它将重复一系列生命期，这些生命期构成了一个系统的开发期寿命，每个生命期都以向客户推出一个产品版本而结束。每个周期包括：开始阶段、确立阶段、构建阶段、移交阶段。

RUP：统一开发过程，是IBM提供支持和包装的UP系统。

进化式迭代开发：迭代开发是RUP的关键实践，开发被组织成一系列固定的短期小项目，每次迭代都产生经过测试、集成并可执行的局部系统。每次迭代都具有各自的需求分析、设计、实现和测试，随着时间和一次次迭代，系统增量式完善。

第三章 软件项目管理

1.什么是项目进度？活动？里程碑？项目成本？

项目进度：项目进度是对特定项目的软件开发周期的刻画，包括对项目阶段、步骤、活动的分解，对各个离散活动的交互关系的描述，以及对各个活动完成时间及整个项目完成时间的初步估算。

活动（activity）：项目的一部分，一般占用项目进度计划中的一段时间。

里程碑（milestone)：指特定的时间点，标志着活动的结束，通常伴随着提交物。

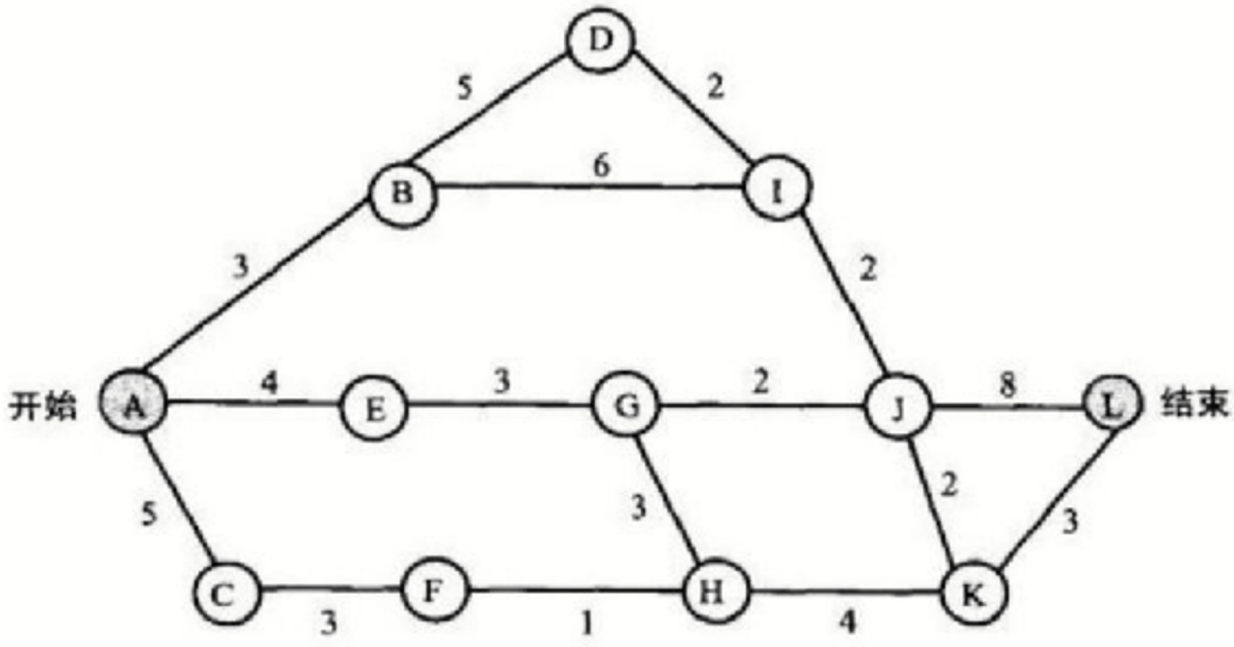
项目成本：为了支持软件开发而购买软件和工具的开支，用于支持需求分析、设计、编码、测试、处理需求变更等开支，再加上工作量开支。

2.如何计算软件项目活动图的关键路径？（习题2，3）冗余时间？最早和最迟开始时间（课堂习题讲解）

项目活动图基本原理：

> 边代表活动，节点代表里程碑。前驱代表本活动完成前需要完成的活动。关键路径（CP）指的是能够标明或计算出完成项目所需最少时间的路径。冗余时间指的是最晚开始时间-最早开始时间

计算关键路径：



关键路径即为从起点到终点花费时间最长的路径。

计算每个活动的最早最晚开始时间：

1. **关键路径**的最早最晚时间是相等的，都是前一个任务开始时间加上前一个任务的持续时间。
2. **从后向前**计算最晚开始时间，从关键路径结束里程碑开始向前，每个任务的最晚开始时间为**与之相连的下一个任务的最晚开始时间减去此任务的持续时间**。

   如果下一个任务有多个，那么选择**（最晚开始时间-此任务持续时间）最早**的那个。（如果选择最晚开始时间晚的减去此任务结束时间作为此任务最晚开始时间，那么此任务完成后的时间已经超过了与之相连的其他任务的最晚开始时间，因此要选最晚开始时间最早的）

   如果下一个任务没有，也就是此任务结束节点是整个活动结束节点，那么此任务最晚开始时间为整个活动结束时间-此任务持续时间

3. **从前向后**计算最早开始时间，从开始里程碑向后，每个任务最早开始时间是**前一个任务的最早开始时间加上前一个任务的持续时间**。

   如果前一个任务有多个，就选**（最早开始时间+任务持续时间）最晚**的那个任务。（如果选了最早的前一个任务加上任务持续时间作为后一个任务的最早开始时

间，那么当此任务以最早开始时间开始时，这个时间小于其他任务的最早结束时间，这个任务将在前一个任务没完成的情况下完成，也不可以）

如果此任务的前一个任务没有，也就是此任务的开始节点是整个任务开始节点，那么此任务最早开始时间是1。

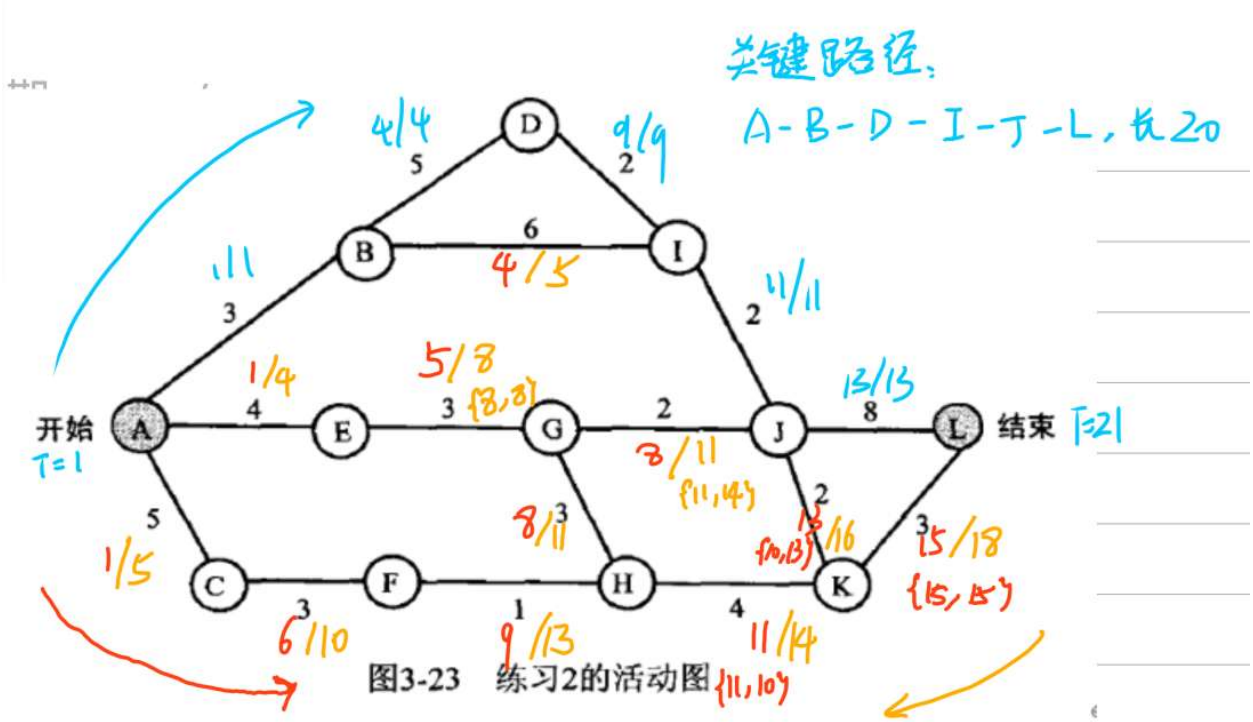


图3-23　练习2的活动图

3.软件项目团队组织的基本结构？

分为两种：

1. 主程序员负责制。由一个主程序员负责系统设计和开发，其他的成员向其汇报，主程序员对每一个决定有绝对决策权。

   优势：交流最小化、有利于迅速做出决定。

   不足：对主程序员要求较高，个人主观性强。

2. 忘我方法。每个成员平等地承担责任，而且过程与个人是分开的。批评是针对产品和结果的，不针对个人。

对于项目组织结构化与创造性的关系：

结构化较强的团队，能按时完成任务，但工作比较循规蹈矩，完成的项目普通但功能完备。当项目人员较多、具有较高稳定性或一致性、项目规模较大时使用较正规的结构。

结构化较弱的团队经常不能按时完成任务，比较任性，难以管理。但未经组织的小组总是具有难以置信的创造性。项目涉及大量的不确定性因素时采用较为民主的方法和相关团队结构。

4.试述COCOMO模型的三个阶段基本工作原理或含义。

模型关键在于针对项目开发的不同阶段来设置工作量的衡量标准，逐步细化，逐渐准确。

表示其工作量的方程：$E = bS^c m(X)$，其中E代表工作量，b，c为常数，S为估算系统规模，m是调整因子。

三个阶段：

阶段一，计划阶段。项目通常构建原型以解决包含用户界面、软件和系统交互、性能和技术成熟性等方面在内的高风险问题。此时人们对正在创建的最终产品的可能规模知之甚少，因此COCOMO II用应用点来估算规模。

阶段二，早期设计阶段。已经决定将项目开发向前推进，但是设计人员必须研究集中可选的体系结构和操作的概念。同样，任然没有足够信息支持准确的工作量和工期估算，但比第一阶段知道的信息多。因此使用功能点对规模进行测量。

阶段三，后体系结构阶段。开发已经开始，而且知道了更多信息。在这个阶段可以根据功能点或代码行来进行规模估算，且可以较为轻松地估算很多成本因素。

5.什么是软件风险？了解主要风险管理活动？有几种降低风险的策略？

软件风险：

在软件生产过程中不希望看到的、有负面结果的时间。

主要的风险管理活动：

~~风险估算：包括风险识别、风险分析、风险优先级分配~~

~~风险控制：包括风险降低、风险管理计划指定、风险化解。~~

产品过大。从一个小的产品内核开始，在以后的开发循环中再添加各种功能。

过难或是复杂的功能。在工程开始时化简这些功能，再考虑它们的代替品。

系统支持问题。建立一个早期原型或者小产品版本，以确定你了解支持系统是如何工作的。（**通过对核心功能的测试，可以确定其他系统对本软件的系统支持程度**）

测试时间。按照TSPi进行工作，使用规范的PSP方法。

产品控制。这就是在工程开始时进行配置管理的原因。

协同工作问题。工作人员合理搭配问题。

三种降低风险的策略：

避免风险：改变功能和性能需求，使风险没机会发生。

转移风险：把风险分配到其他系统中或者购买保险，以便在风险成为事实时弥补经济上的损失。

假设风险：用项目资源接受并控制风险，如在开发时主动有意识地进行测试。

6.找出关键路径：

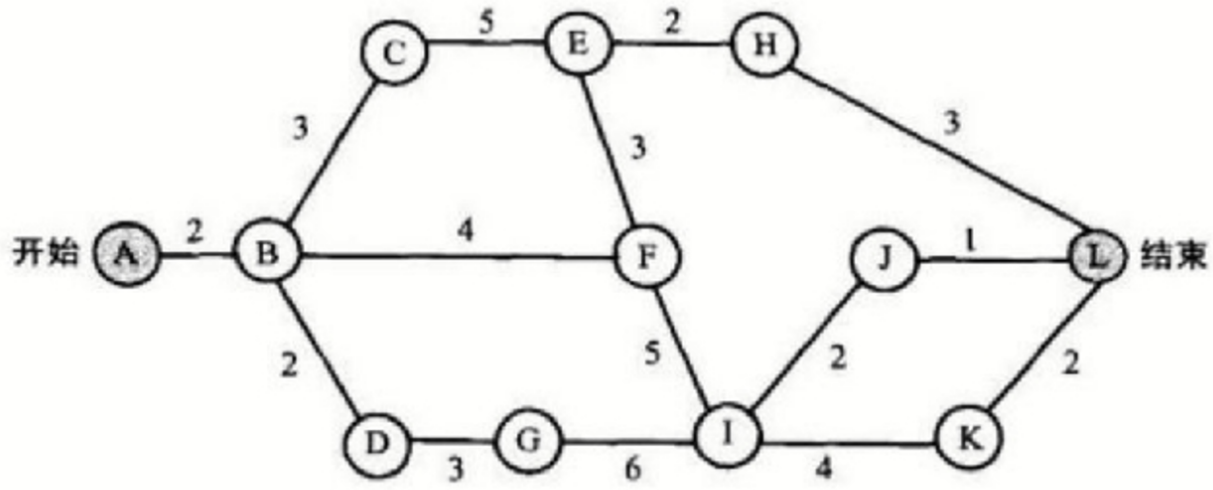


图3-24　练习3的活动图

A-B-C-E-F-I-K-L

第四章 捕获需求

1.需求的含义是什么?

需求是对来自用户的关于软件系统的期望行为的综合描述，设计系统的对象、状态、约束、功能等。

2.需求阶段作为一个工程，其确定需求的过程是什么？

1. 原始需求获取，清楚系统应该做什么，确定客户的需求。
2. 问题分析，明白需求并通过建模或模型化方法进行描述。
3. 规格说明草稿书写，利用符号描述系统将定义规范化表示。
4. 需求核准，开发人员和客户进行核准，有时会测试原型
5. 软件规格说明书书写，行成正式的SRS

3.举例说明获取需求时，若有冲突发生时，如何考虑根据优先级进行需求分类。

需求的优先级划分：

1. 必须要满足的需求
2. 非常值得做但不是必须的需求
3. 有可能做但是可以被消除的（可做可不做的）需求

4.需求文档分为哪两类？

1. 需求定义文档，完整地罗列了用户对系统的期望功能。
2. SRS需求规格说明，将需求重述为关于要构建的系统如何运转的规格说明。

5.什么是功能性需求和非功能性需求（质量需求）？设计约束？过程约束？如何区分？

功能性需求：

描述系统内部功能或系统与外部环境的交互作用，设计系统输入应对、实体状态变化、输出结果、设计约束与过程约束等。

非功能性需求（质量需求）：

描述软件方案必须具备的某些质量特征。

设计约束：

物理环境：对环境或设备的限制，如安装及环境要求等。

接口：设计输入输出的限制或约束条件，如输入格式预定等。

用户：使用者的基本情况，限制几种类型的用户。

过程约束：

资源：材料、人员技能或其他。

文档：类型、数量或其他。

标准：如阅读文档时用户的指派标准。

其他：如什么原因会导致从工资单列表中删除某雇员。

如何区分设计约束和过程约束：

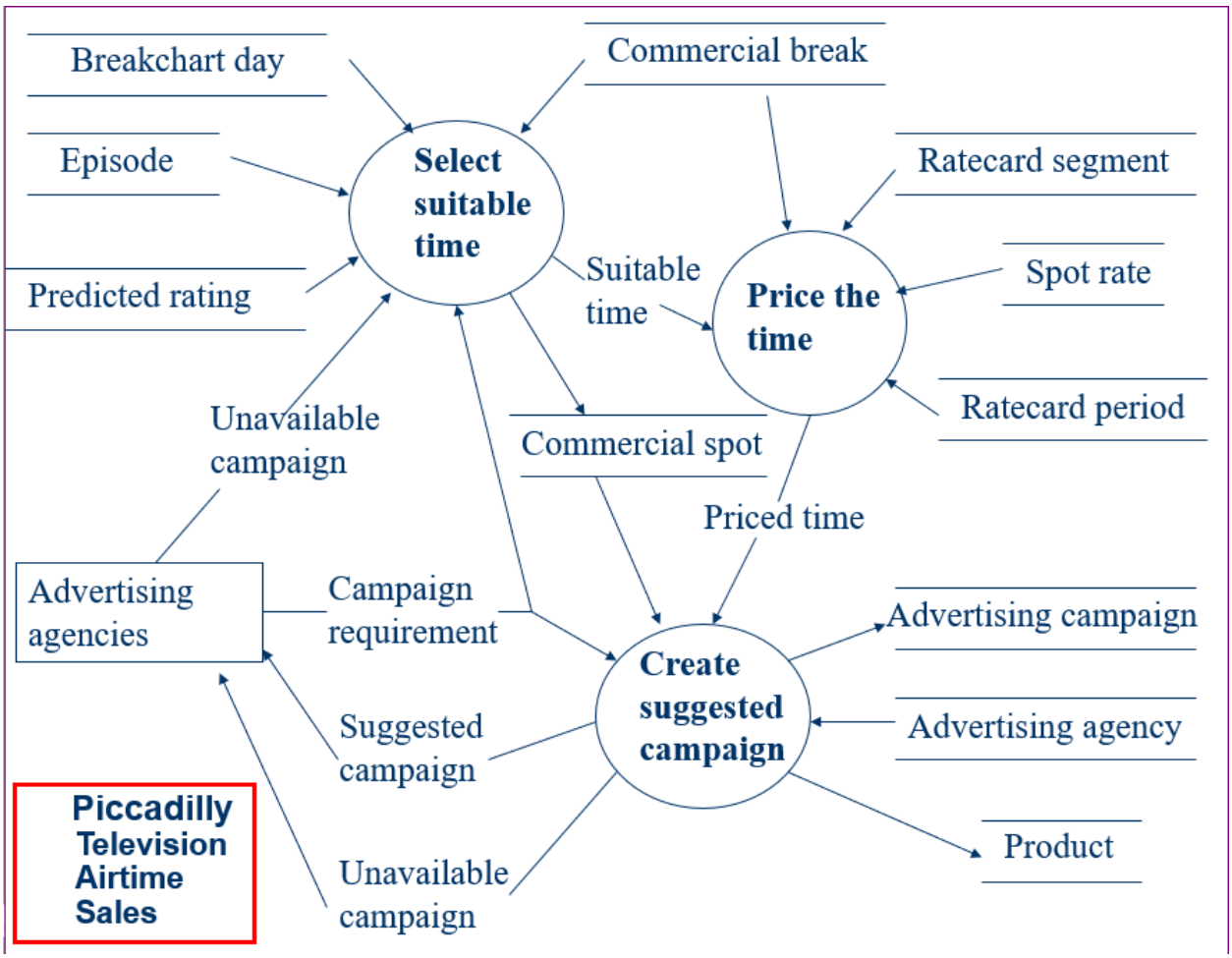
设计约束是已经做出的设计决策或限制问题解决方案集的**设计决策**。涵盖物理环境、接口、用户等方面。

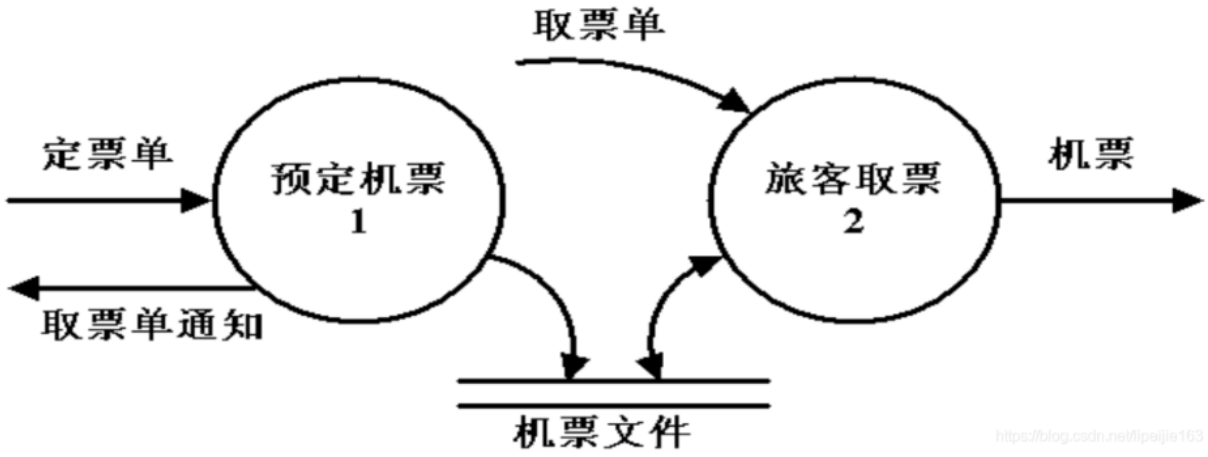
过程约束对用于**构建系统的技术和资源的限制**，涵盖资源、文档等方面

6.了解DFD图的构成及画法。

DFD数据流图，描述数据如何流进系统，如何转换，如何离开系统。

外部项，即数据的源点或终点。


Breakchart day
Commercial break
Episode
Select suitable time
Ratecard segment
Predicted rating
Suitable time
Price the time
Spot rate
Unavailable campaign
Commercial spot
Ratecard period
Priced time
Advertising agencies
Campaign requirement
Advertising campaign
Create suggested campaign
Suggested campaign
Advertising agency
Piccadilly Television Airtime Sales
Unavailable campaign
Product


取票单
定票单
预定机票 1
旅客取票 2
机票
取票单通知
机票文件

第五章 系统设计

1.什么是软件体系结构？设计模式？设计公约？设计？

软件体系结构：

一种软件解决方案，用于解释如何将系统分解为单元，以及单元如何相互关联，还包括这些单元的所有外部特性。

设计模式：

一种针对单个软件模块或少量模块而给出的一般性解决方案，它提供较低层次的设计决策。(此设计决策低于体系结构) (注: 此处为说明,不是定义)

设计公约：

一系列设计决策和建议的集合，用于提高系统某方面的设计质量。。

当一种设计公约发展成熟时，将会被封装成设计模式或体系结构风格，最后可能被内嵌封入程序语言结构。例如：对象、模板等都是编程语言支持的原先设计和编程公约）

设计:

将需求中的问题描述转变成软件解决方案的创造性过程。

2.软件设计过程模型的几个阶段？

1. 初始建模，尝试可能的分解，根据需求描述的系统关键特性等确定软件体系结构风格，进行系统级别的决策。
2. 分析：关注软件系统的功能及质量属性（性能、安全性、可靠性）、关注各种约束。
3. 文档化：确定各个不同的模型视图。
4. 复审：检查文档是否满足所有功能及质量需求。
5. 最终输出：SAD，软件体系结构文档。

3.论述设计用户界面应考虑的问题。

1. 注意解决的关键要素：

   寓意、比喻方面，解决可识别学习的基本术语、图像和概念等。

   思维模型上，解决数据、功能、任务的组织与表示。

   模型的导航规则上，如何在数据、功能、活动和角色中移动及切换。

   外观以及感觉上的问题。

2. 文化差异问题的解决，处理信仰、价值观、到达规范、传统、风俗、传说等问题。两种解决方法：①使用国际设计/无偏见设计，排除特定的文化参考或偏见。②采用定制界面，使不同用户看到不同界面。
3. 用户爱好问题，为具有不同偏好的人选择不同界面。

4. 5.5节----模块独立性----耦合与内聚的概念及各个层次划分？

耦合：两个软件部件之间的相互关联程度。

耦合的六个层次：

非直接耦合：模块相互之间没有信息传递。

数据耦合：模块间传递的是数据。

特征耦合：模块间传递的是数据结构。

控制耦合：模块间传递的是控制量。

公共耦合：不同模块访问公共数据。

内容耦合：一个模块直接修改另一个模块。

内聚：软件部件内部各组成成分的关联程度。

内聚的七个层次：

偶然性内聚：不相关的功能、过程、数据等出现在同一个部件中。

逻辑性内聚：逻辑上相关或相似的功能或数据放置在同一个不见内。

时间性内聚：部件各部分要求在同一时间完成。

过程性内聚：模块各部分有特定次序。

通讯性内聚：各个部分访问共享数据。

顺序性内聚：各个部分有输入输出关系。

功能性内聚：各部分组成单一功能。

5.软件过程中复审的概念，设计复审的重要性。

复审：

检查文档是否满足所有功能及质量需求。

重要性：

(1) 复审中批评和讨论是“忘我”的，能将开发人员更好地团结在一起，提倡并增强了成员之间的交流。
(2) 在评审过程中故障的改正还比较容易，成本还不高，在这时候发现故障和问题会使每一个人受益。

第六章 面向对象

1.什么是设计模式？

一种针对单个软件模块或少量模块而给出的一般性解决方案，它提供较低层次的设计决策。

2.了解OO设计的基本原则？

1. 单一职责原则，一个类只负责一个功能领域中的相应职责。
2. 重用原则，不要重复写一些相似的方法。
3. 开闭原则，软件中的类、对象、函数等对扩展是开放的，对修改是封闭的。
4. 里氏替换原则，如果S是T的子类型，对于S类型的任意对象，如果将他们看作是T类型的对象，则对象的行为也理应与期望的行为一致。
5. 依赖倒置原则，程序要依赖于抽象接口，不要依赖于具体实现。
6. 接口隔离原则。
7. 迪米特法则，一个软件实体应当尽可能少的与其他实体发生相互作用。

3.了解OO开发有何优势？

1. 语言的一致性：采用相同的语义结构（类、对象、接口、属性、行为）描述问题和解决
   方案。
2. 全开发过程的一致性：从需求分析和定义、高层设计、底层设计到编码和测试等，所有
   的过程都采用相同的语义结构。

4.OO开发过程有几个步骤？

1. OO需求设计。

2. OO高层次设计。
3. OO底层设计。
4. OOP。
5. OO测试。

5.掌握用例图的组成和画法，用例的几个要素的含义。

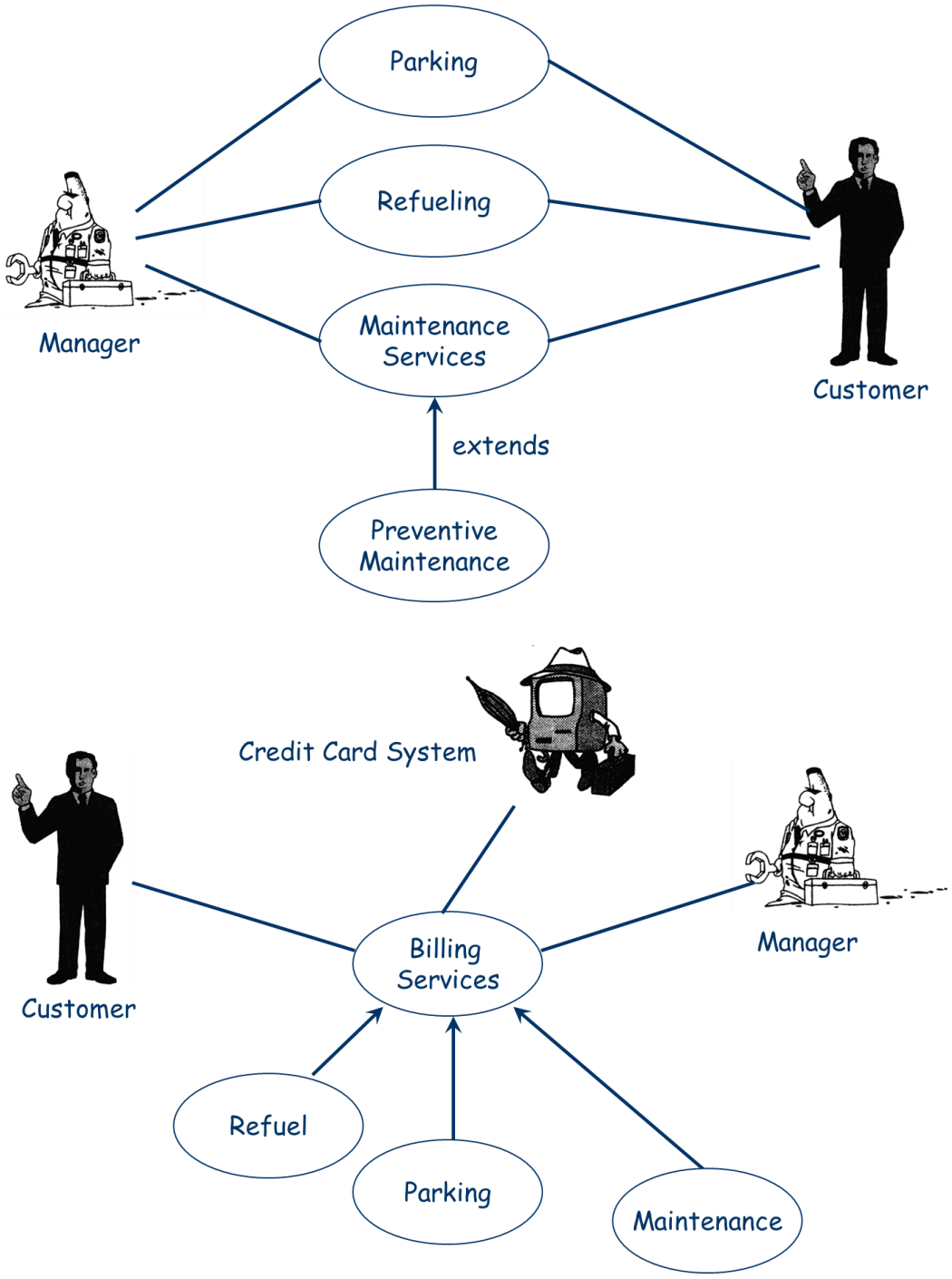
1. 用例图组成：
    a. 用例：描述一个系统应该执行或显示的特定功能。

    b. 执行者：与系统交互的实体，可以是用户、设备等。

    c. 包含：对已经定义的用例的复用

    d. 扩展：扩展用例以说明不同的或更深的视角

2. 例子：



6.掌握用例图的实例解析方法，如何辨识和确定一个用例？

通过对用户、外部系统或其他实体与要开发的系统之间的对话进行建模。

找出参与者，从主要业务开始入手，根据参与者确定与其相关的用例，再细化用例规约。（from web）

7.用例模型相关建模步骤是什么？

- 确定系统边界
- 确定参与者
- 找出所有的用例
- 确定每个用例的级别
- 撰写用例的文字描述
- 画出以整个系统为对象的顺序图

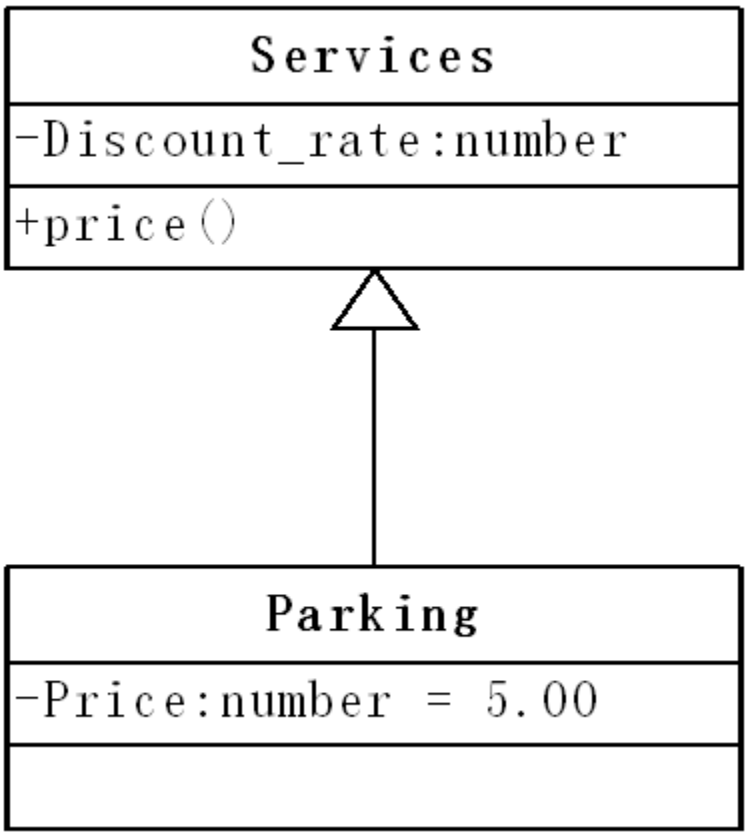
（from web）

8.用例图、类图等针对面向对象的项目开发的意义是什么？

1. 阐明需求
2. 有助于发现需求上的问题。有时问题隐藏在自然语言描写的需求上。
3. 需求本身很复杂而且难以描述，因此需要借助用例图等工具。

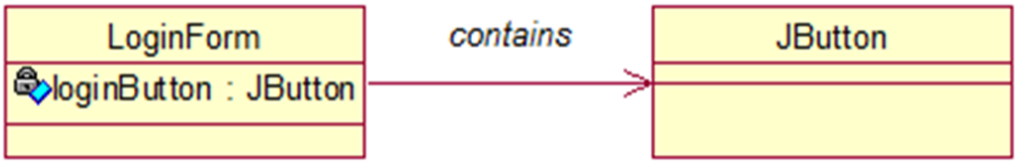
9.熟悉类图中各个类之间的基本关系分类及其含义。

1. 结构：
    a. 类名：定义一个类。
    b. 属性：用名称、类型和初始值描述。
    c. 操作：用名称、参数列表以及返回值描述。
2. 关系：
    a. 继承/泛化：超类概括了自类。



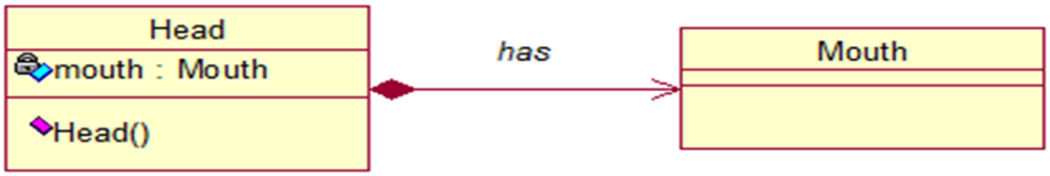
b. 关联：两个类之间有语义关系，如A包含了B。

分为双向关联、单向关联、自关联和多重性关联。

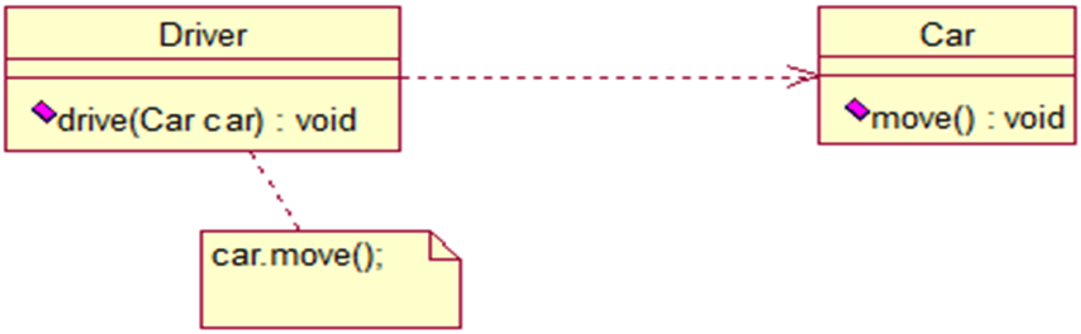


c. 聚合：表示部分与整体之间的关系，但整体消失不代表其对应部分的消失。

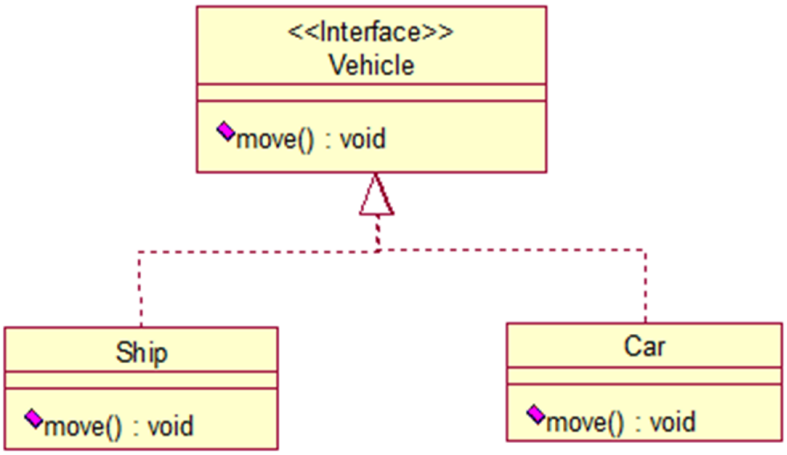
d. 组合：部分与整体之间的关系，但整体可以控制部分的生命周期，如果整体不存在，那么对应的部分也不存在。



e. 依赖：使用关系，某个类使用了另一个类的方法、参数，或者返回值包含这个类参数等。



f. 实现：接口与其实现之间的关系。



10.绘制类图最常用的方法及步骤是什么？识别一个类的基本思路？

方法和步骤：

（1）研究分析问题领域确定系统需求。
（2）确定类，明确类的含义和职责、确定属性和操作。
（3）确定类之间的关系。

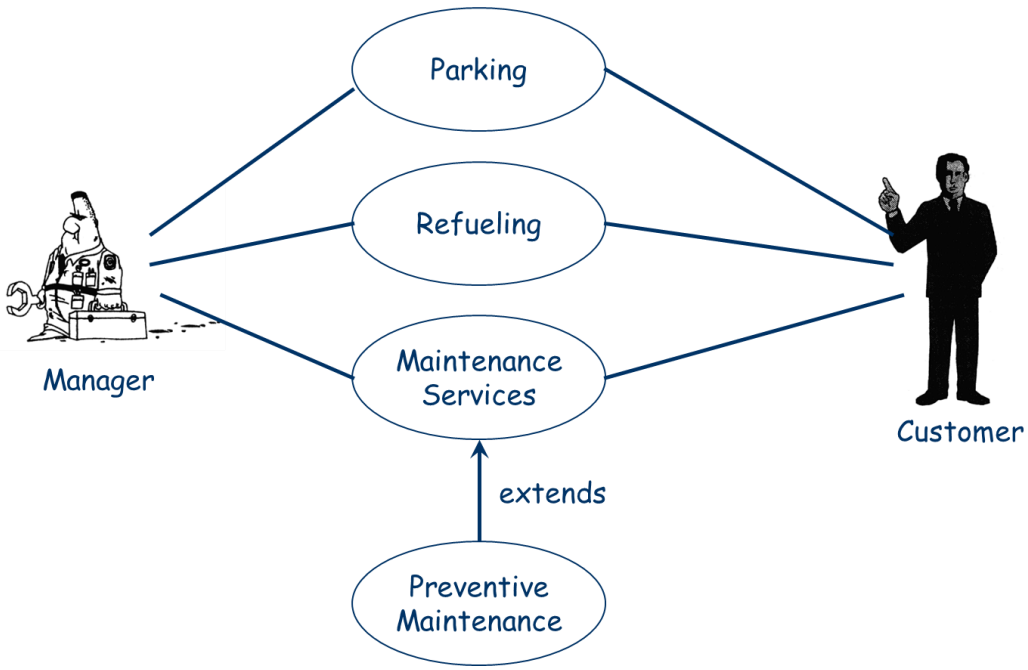
（from web）

基本思路：

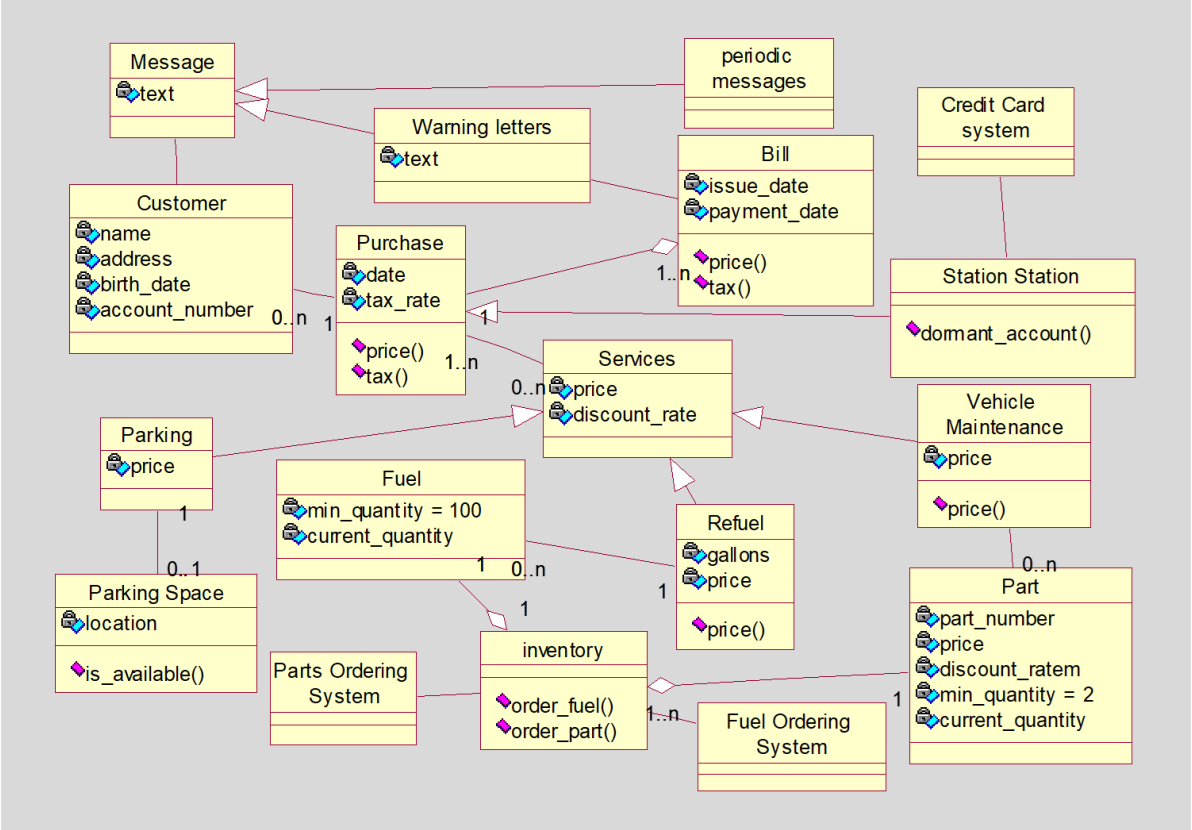
研究问题域和用户需求
策略和启发：组织、物品、需要长期记忆的事件等。
审查和筛选
识别主动对象
对象分类：分析具有主动行为的对象。
建立类图中的类

11.熟悉用例图、类图、状态图的组成和画法。

用例图：



类图：



状态图：

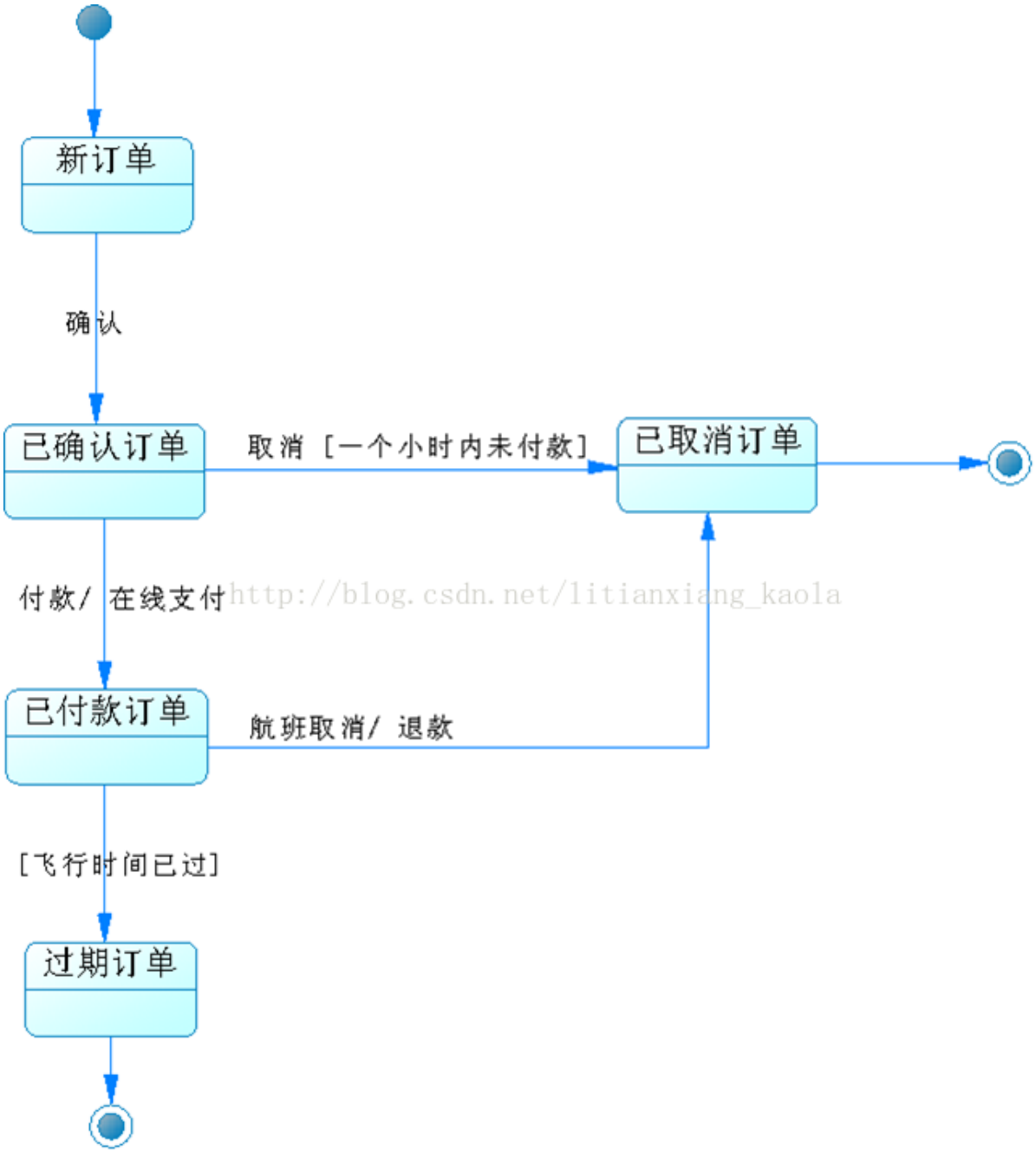
状态图(State Diagram)用来描述一个特定对象的所有可能状态及其引起状态转移的事件。

元素有初始状态、终止状态、状态、转移、守护条件、事件、动作

1，状态图用初始状态（Initial State）表示对象创建时的状态，每一个状态图一般只有一个初始状态，用实心的圆点表示。
2，每一个状态图可能有多个终止状态（Final State），用一个实心圆外加一个圆圈表示。

3，状态图中可有多个状态框，每个状态框中有两格：上格放置状态名称，下格说明处于该状态时，系统或对象要进行的活动（Action）

4，从一个状态到另一个状态之间的连线称为转移（Transition）。状态之间的过渡事件（Event）对应对象的动作或活动（Action）。事件有可能在特定的条件下发生，在UML中这样的条件称为守护条件（Guard Condition），发生的事件可通过对象的动作（Action）进行处理。状态之间的转移可带有标注，由三部分组成（每一部分都可省略），其语法为：事件名 [条件] / 动作名。



12.了解UML其他图示结构的基本用途。

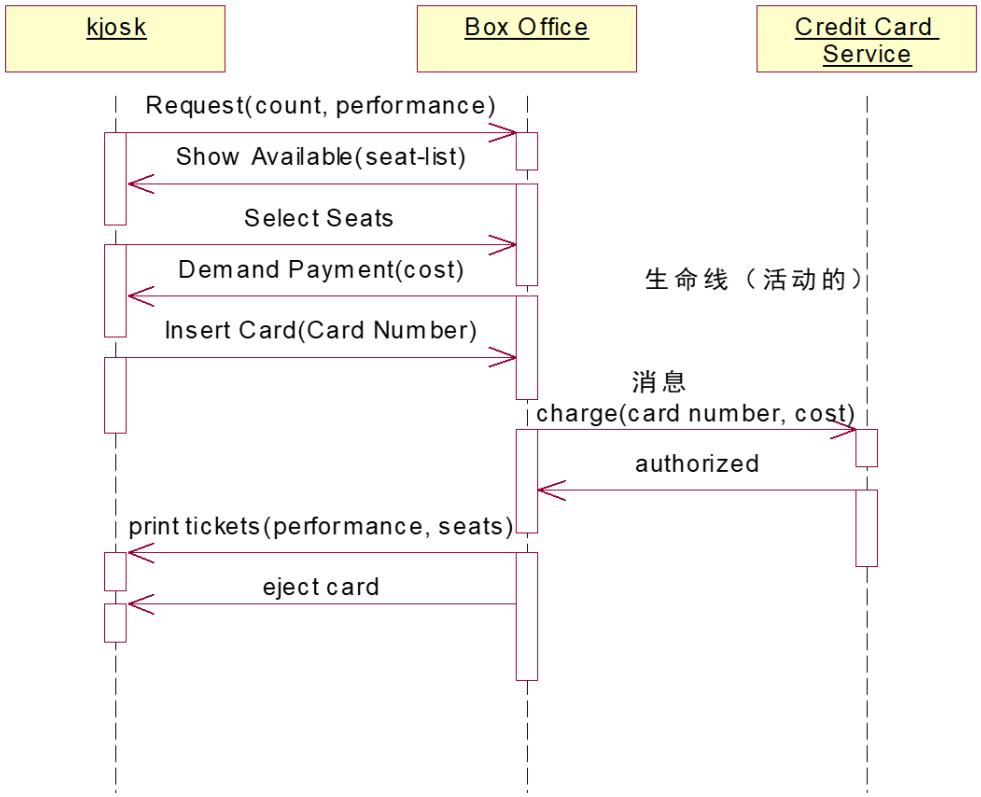
设计过程

step1：类图（改进概念类图）、对象图（解释每个对象）

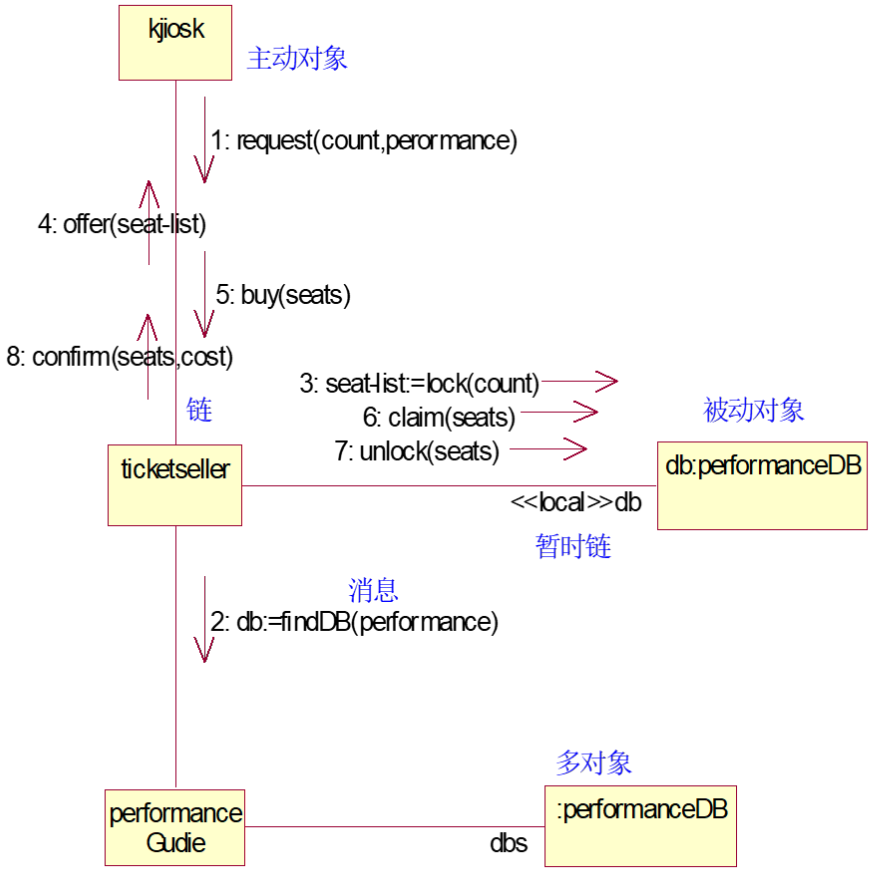
step2：活动图、状态图

step3：顺序图、协作图

顺序图：顺序图用消息的几何排列关系来表达消息的时间顺序，各角色(最上方实体序列可以使用各种角色)之间的相关关系是隐含的。



协作图：用各个角色的几何排列图形来表示角色之间的关系，并用消息来说明这些关系



第七章 编程

1.一般性的编程原则应该从哪三个方面考虑？

1. 控制结构：编程的最高指导原则，要做到轻松控制组件，程序员应该注意于完成什么，而不是控制流是什么。但不管什么样的设计，都应该使程序结构放映设计的控制结构。
2. 算法：在将算法转换成代码时候有极大的灵活性，一般情况下清晰度第一，效率第二。
3. 数据结构：通过改变模块数据结构来简化程序，用数据结构决定程序结构。

2.在编写程序内部文档时，除了HCB外，还应添加什么注释信息？注意什么？

HCB：头部注释块，用来总结信息，如定义程序、描述数据结构、算法以及控制流。

此外添加其他程序注解：帮助读者清楚关于源代码的所有意图。分为：

1. 分段分部分注释，可以对程序正在做什么提供逐行的解释。
2. 代码更改伴随注释更新。
3. 随着时间进行修改的记录。

注意事项：

1. 要写有意义的变量名称和状态标签，表达其用处。
2. 要文档格式化，增强可读性。
3. 数据文档化，记录数据。内部文档应当包含数据结构及其用处的描述。

3.敏捷方法的大致思想？什么是极限编程(XP)? 以及派对编程？

敏捷方法：

> 较新型的软件开发方法，不要求遵循传统的软件开发流程，强调快速开发和有效适应需求变化。

极限编程：

> **一种轻量级的软件开发方法论，属于敏捷开发方法**，从实践中来，是对实践的总结，也经过了实践的检验，主要特征是**要适应环境变化和需求变化，充分发挥开发人员的主动精神。**XP承诺降低软件项目风险，改善业务变化的反应能力，提高开发期间的生产力，为软件开发过程增加乐趣。

派对编程：

> 属于主要的敏捷开发方法，其开发方式是两个程序员共同开发程序，且角色分工明确。一个负责编写程序，另一个负责复审与测试。两人定期交换角色。

第八章 程序测试

1.了解产生软件缺陷的原因？

1. 软件本身要处理大量的系统状态、处理复杂的公式、活动、数据及算法等。
2. 由于客户和设计者对于不确定的、遗忘的、不可能实现的需求，导致设计时的缺陷。
3. 其他因素，如项目的场景及规模因素，诸多参与者等。

补充：

错误error：由于对需求的误解或者错误的代码而导致程序人员在软件生成中产生错误。

缺陷fault：在函数实现中出现的问题。**是客观存在的，一个错误可能导致一个或多个缺陷。**

失败failure：由于缺陷而导致软件运行失败（软件动作与需求描述不相符），**是动态产生的**。

2.有几种主要的缺陷类型？

1. 算法缺陷：算法的某些处理步骤或逻辑有问题，以至于软件部件对给定的输入数据无法产生正确的输出。
2. 计算和精度缺陷：算法或公式在编程实现时出现错误或最终达不到精度要求。
3. 文档缺陷：文档和程序实际做的事情不匹配。
4. 过载缺陷：当运行程序时，数据结构被填充到超过其规定容量。
5. 能力缺陷：当系统活动达到其规定极限时，系统性能变得不可接受。
6. 时序性缺陷：几个同时执行或仔细定义顺序执行的进程之间协调不适当。
7. 性能缺陷：正常条件下，由于组装后测试不充分，性能需求不满足。
8. 恢复性缺陷：当系统发生失败时，系统不能表现出和预期一样的执行效果。
9. 硬件和系统软件缺陷：当支持程序的硬件或软件不正确按照文档给出的状态和过程工作时。
10. 代码的标准和规程缺陷：代码不按照组织好的标准和规程执行。

3.什么是正交缺陷分类？

一种缺陷应该属于一种类别。

被分类的任何一项故障都只属于一个类别，则分类方案是正交的。如果一个故障属于不止一个类，则失去了度量的意义。

4.测试的各个阶段及其任务？涉及的文档？（图8.3）

1. 几个阶段及任务
    a. 单元测试：根据程序设计验证组件功能。
    b. 协同测试：根据程序设计和系统设计验证系统组件是否协同工作。

c. 函数测试：通过SRS软件需求规格说明书检查函数。

d. 性能测试：通过SRS检查性能。

e. 验收测试：检查客户的需求定义。

f. 安装测试：检查在实际环境中的系统。

2. 涉及文档：

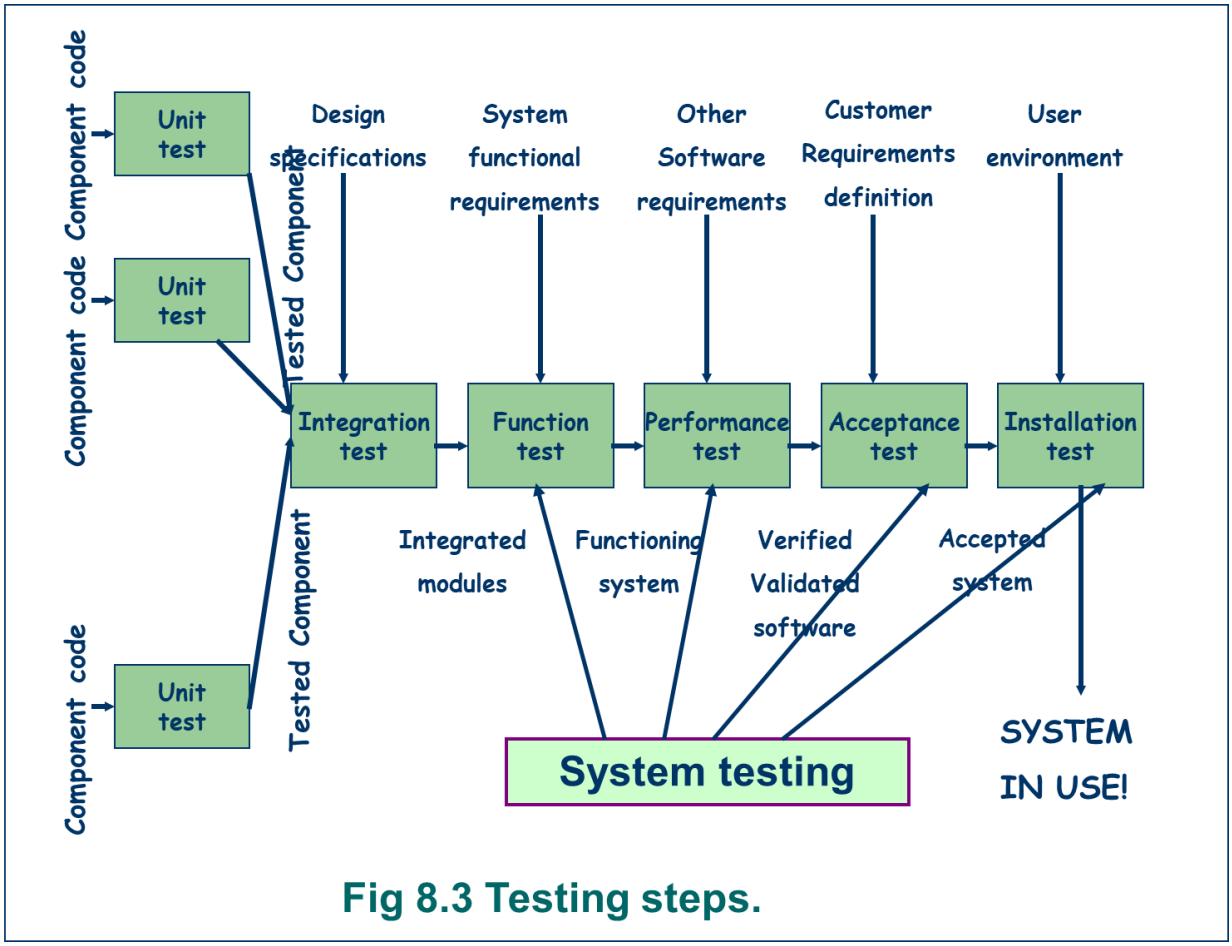


Fig 8.3 Testing steps.

5.掌握测试的方法----黑盒、白盒的概念？

1. 黑盒：测试人员在完全不了解程序内部的逻辑结构和内部特性的情况下，只根据程序的需求规格及设计说明，检查程序的功能是否符合它的功能说明。
2. 白盒：以测试对象的内部结构为基本依据，手工或自动的展开各种测试。

6.什么是单元测试？

1. 静态检查代码，与需求及设计进行比较：代码复审。
2. 编译并调试代码。
3. 开发测试用例并且测试。

7.黑盒、白盒方法各自的分类？测试用例的设计方法和给出方法。

1. 黑盒方法：
    a. 等价分类法：将输入域划分为若干等价类。每□个测试用例都代表了一类与它等价的其他例子。如果测试用例没有发现错误，那么对应的等价例子也不会发生错误。有效等价类的测试用例尽量公用，以此来减少测试次数，无效等价类必须每类一个用例，以防止漏掉可能发现的错误
    b. 边界值分析法：在等价分类法中，代表一个类的测试数据可以在这个类的允许范围内任意选择。但如果把测试值选在等价类的边界上，往往有更好的效果，这就是边界值分析法的主要思想。
    c. 错误猜测法：猜测程序中哪些地方容易出错，并据此设计测试用例。更多的依赖于测试人员的直觉和经验。
    d. 因果图法：适用于被测试程序有很多输入条件，而对于程序的输出又依赖输入条件的各种组合的情况。
2. 白盒测试方法：
    a. 语句覆盖：给出的测试用例能使模块中每一个语句至少执行一遍。
    b. 分支测试。
    c. 路径测试。
    d. 条件覆盖：要求判定中的每个条件均按照“真”、“假”两种结果至少执行一次。
    e. 条件组合覆盖：要求所有条件结果的组合都至少出现一次(比如 A&&B，两个条件，
    那么就有四种条件的组合)。

8.黑盒、白盒方法的分类原则，各种覆盖方法等。（课件等）

9.面对一个命题，设计和给出测试用例的问题

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/29c66ae1-7e29-4b78-b674-cd2a0d636c3a/Untitled.pptx

10.集成测试及其主要方法的分类？（驱动模块、桩模块的概念）

1. 自底向上集成测试：

   从模块结构图的最底层开始，从下而上按调用关系逐步添加新模块，组成子系统并分别测试，重复执行直到全部模块组装完毕。

   驱动模块：代替上级模块传递测试用例的程序

2. 自顶向下集成测试：

   从顶层控制组件开始，首先对它本身进行测试，然后将被测组件调用的下级组件组合起来，对这个更大的子系统进行测试，反复采用这种组装方法，知道包含了所有组件为止。

   桩模块：代替下级模块的仿真程序。

11.传统测试和OO测试有何不同？OO测试有何困难？

1. 不同：

   (1)测试用例的充分性：对过程语言而言，当系统改变时，我们可以针对改变测试是否
   正确，并使用原有的测试用例来验证剩余的功能是否同原来一致。但是面向对象的测试中，
   我们可能需要编写不同的测试用例。
   (2)面向对象趋向于小粒度，并且平常存在于构件内的复杂性常常转移到构件之间的接
   口上。这意味着，其单元测试较为容易，但是集成测试涉及面变得更加广泛。
   (3)传统测试和面向对象的测试主要集中在：需求分析和验证、测试用例生成、源码分
   析和覆盖分析

2. OO测试的困难：

   a. 需求文档的验证缺乏工具支持，很多时候依赖人工。

   b. 测试工具生成的测试用例处理OO模型中的对象和方法时，针对性不强。

c. 源代码分析：传统的测试方法在评价OO系统的规模和复杂性时，还不是很有效，或没有太强的针对性。

d. 覆盖分析：对象的交互是OO系复杂性的根源，传统的覆盖分析等测试方法作用有限。

第九章 系统测试

1.系统测试的主要步骤及各自含义？（图9.2）

1. 函数功能测试：通过SRS软件需求规格说明书检查函数。
2. 性能测试：通过SRS检查性能。
3. 验收测试：检查客户的需求定义。
4. 安装测试：检查在实际环境中的系统。

2.什么是回归测试？

应用于新的版本的测试，用来验证新旧功能的正确性。

3.功能测试的含义及其作用？

含义：根据SRS软件需求规格说明书测试函数的功能。

作用：有极大的可能性发现缺陷。

4.软件功能测试的基本指导原则？

1. 较高的差错概率
2. 独立的测试团队
3. 了解预期的输出结果
4. 对合法与非法的输入都予以测试
5. 绝不能仅仅为了测试的方便而修改系统
6. 停止测试应该有前提条件

5.性能测试的含义与作用？

含义：根据SRS测试非功能需求

作用：性能测试所针对的是非功能需求。它需要确保这个系统的可靠性、可用性与可维护性。性能测试由测试小组进行设计和执行并将结果提供给客户。

6.性能测试的主要分类？

1. 压力测试/强度测试：在短时间内加载极限负荷，以验证系统能力。
2. 巨量数据测试/容量测试：验证系统处理巨量数据的能力。
3. 配置测试：构建测试用例对系统软硬件的各种配置从最小到最大进行测试。
4. 兼容性测试：测试接口。
5. 回归测试：通过测试旧和新的测试用例。
6. 安全性测试。
7. 时序测试。
8. 环境测试。
9. 质量测试。
10. 恢复性测试
11. 人为因素测试

7.确认测试概念，确认测试分类？（基准测试和引导测试）

确认测试（验收测试）：

用户检查是否系统符合需求定义的要求。

分类：

1. 基准测试：用户准备测试用例，由用户准备典型测试用例，在实际安装后的系统运作并由用户对系统执行情况进行评估。
2. 引导测试：用户和开发者安装运作系统。

   在假设系统已经永久安装的前提下执行系统。它依赖系统的日常工作进行测试，相对基
   准测试不是非常的正式与结构化
3. 并行测试：新老版本并行运转，使客户逐渐适应新版软件系统。

8.什么是alpha测试？β测试？

α测试：

通过开发者在组织中的试点测试。

β测试：

用户作为试点在工作场所测试。

9.什么是安装测试？

在用户的环境中进行测试，用来解决由于开发环境和用户所处环境不同而导致的问题。