**《软件工程》**

**第1部分 软件工程基础**

1. **软件及软件工程介绍**
   1. 软件与软件危机
      1. 软件的作用

计算机软件已经成为世界舞台上最为重要的科技领域，商业、科学和工程都离不开软件技术。现在的软件技术具有产品和产品生产载体的双重作用。

* + 1. 软件的概念及特性

1. 软件的概念

一般可以将软件划分为系统软件、应用软件和介于这两者之间的中间件。

计算机软件的传统定义为：软件是计算机系统中与硬件相依存的另一部分，软件包括程序、数据及其相关文档的完整集合。

程序是按事先设计的功能和性能要求执行的指令序列；数据是使程序能正常操纵信息的数据结构；文档是与程序开发、维护和使用有关的图文材料。

1. 软件的特性
2. 形态特性。软件是无形的、不可见的逻辑实体。
3. 智能特性。
4. 开发特性。
5. 质量特性。
6. 软件的需求在软件开发之初常常是不确切的；
7. 软件测试技术存在不可或缺的局限性；
8. 已经长期使用或多次反复使用的软件没有发现问题，但这并不意味着今后的使用总不会出现问题。
9. 生产特性。
10. 管理特性。
11. 环境特性。
12. 维护特性。
13. 废弃特性。
14. 应用特性。
    * 1. 软件危机

软件开发周期长、成本高、质量差、维护困难，导致60年代末软件危机的爆发，其突出的实例是IBM公司在1963年-1966年开发的IBM 360机的操作系统。

软件危机发生的原因主要有以下方面：

1. 缺乏软件开发的经验和有关软件开发数据的积累，使得开发工作的计划很难制订；
2. 软件人员与用户的交流存在障碍；
3. 软件开发过程不规范，缺少方法论和规范的指导；
4. 随着软件规模的增大，其复杂性往往会呈指数型增长；
5. 缺少有效的软件评测手段，提交用户的质量差。
   1. 软件工程及其基本原理
      1. 软件工程的概念

软件工程是指导计算机软件开发和维护的工程学科。

采用工程的概念、原理、技术和方法来开发和维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，以经济地开发出高质量的软件并有效地维护它，这就是软件工程。

* + 1. 软件工程的目标

软件工程的目标是运用先进的软件开发技术和管理方法来提高软件的质量和生产率，也就是要以较短的周期、较低的成本生产出高质量的软件产品，并最终实现软件的工业化生产。

衡量软件质量的6个特性：功能性、可靠性、可使用性、效率、可维护性和可移植性。

* + 1. 软件工程的基本原理

1983年美国TRW公司的B.W.Boehm概括出著名的软件工程7条基本原理：

1. 按软件生存周期分阶段制订计划并认真实施；
2. 坚持进行阶段评审；
3. 坚持严格的产品控制；
4. 使用现代程序设计技术；
5. 明确责任；
6. 用人少而精；
7. 不断改进开发过程。
   1. 软件生命周期

软件有一个孕育、诞生、成长、成熟和衰亡的过程，称此过程为软件生命周期或软件生存期。软件生存期由软件定义、软件开发和运行维护3个时期组成。

软件定义时期的主要任务是解决“做什么”的问题，它通常又分为3个阶段：问题定义、可行性研究和需求分析。

软件开发时期的主要任务是解决“如何做”的问题，通常由概要设计、详细设计、编码和测试4个阶段组成。

软件运行维护时期的主要任务是使软件持久地满足用户的需要，通常有四类维护活动：改正性维护、适应性维护、完善性维护和预防性维护。里程碑对于软件开发的管理非常重要。

开发过程中的典型文档包括：

1. 软件需求规格说明书；
2. 项目计划；
3. 软件测试计划；
4. 软件设计说明书；
5. 用户手册。

各个阶段所要完成的基本任务有：

1. 问题定义和可行性研究

本阶段要回答的关键问题是“到底要解决什么问题？在成本和时间的限制条件下能否解决问题？是否值得做？”

1. 需求分析

本阶段要回答的关键问题是“目标系统应当做什么？”

1. 软件设计

设计是软件工程的核心技术。本阶段要回答的关键问题是“如何做出目标系统？”

1. 程序编码和单元测试

本阶段要解决的问题是“正确地实现已做的设计”，即“如何编写正确的、可维护的程序代码？”

1. 集成和系统测试

测试是控制软件质量的重要手段，本阶段的主要任务是做集成测试和系统测试。

1. 软件运行和维护

已交互的软件投入正式使用，便进入运行阶段。

通常有4种类型的维护：改正性维护、适应性维护、完整性维护和预防性维护。

* 1. 软件工程方法学

把在软件生命周期全过程中使用的一整套技术的集合称为方法学(methodology)，也称为范型(paradigm)。

软件工程方法学包含3个要素：方法、工具和过程。

* + 1. 结构化方法学

结构化方法学也称为传统方法或生命周期方法。它采用结构化技术来完成软件开发的各项任务。这种方法学将软件生命周期的全过程分为若干个阶段，然后顺序地逐步完成每个阶段的任务。

* + 1. 面向对象方法

面向对象方法把数据和行为看成同等重要，是将数据和对数据的操作紧密地结合起来的方法。

* 1. 软件工程知识体系及知识域介绍

软件工程教育分为3个历史时期：

1. 1978年以前：软件工程的教育以计算机专业的一门孤立的课程形式存在；
2. 1978-1988年期间：早期的研究生教育；
3. 1988年以后：快速发展的研究生学科教育。

“SWEBOK指南”项目：

1. 软件工程知识体系指南的目标：
2. 促使软件工程本体知识成为世界范围的共识；
3. 澄清软件工程与其它学科的关系，并且确定软件工程学科的范围；
4. 反映软件工程学科内容的特征；
5. 确定软件工程本体知识的各个专题；
6. 为相应的课程和职业资格认证材料的编写奠定基础。
7. 软件工程知识体系指南的内容：

10个知识域(KA)，分为两类过程，一类是开发和维护过程，包括：软件需求、软件设计、软件构造、软件测试和软件维护；另一类是支持和组织过程，包括：软件配置管理、软件工程管理、软件工程过程、软件工程工具与方法和软件质量。

* 1. 软件产业的形成与发展
     1. 我国软件产业的形成

主要经历了四个阶段：

* 第一阶段：萌芽期(20世纪70年代至80年代初)
* 第二阶段：起步期(20世纪80年代初至80年代末)
* 第三阶段：进入期(20世纪90年代初至2000年初)
* 第四阶段：发展期(网络软件时期，2000年至今)
  + 1. 全球软件产业的发展

已经经历了比较完整的5代：

* 第一代：早期专业的服务公司(1949年-1959年)
* 第二代：早期软件产品公司(1959年-1969年)
* 第三代：强大的企业解决方案提供商的出现(1969年-1981年)
* 第四代：客户大众市场软件(1981年-1994年)
* 第五代：互联网增值服务(1994年至今)
  + 1. 软件产业的发展模式

1. 美国模式—技术与服务领导型
2. 印度模式—国际加工服务型
3. 爱尔兰模式—生产本地化型
4. 日本和欧洲模式—嵌入式系统开发型
5. 德国模式—企业级应用及自主研发型
   * 1. 软件工程在软件产业中的应用

一方面，软件产业离不开软件工程理论及其标准的指导；另一方面，软件产业的发展需要大量的软件工程人才。

1. **软件需求获取与确认**
   1. 软件需求获取的任务

软件需求是为了解决用户的问题和实现用户的目标，用户所需的软件必须满足的能力和条件。

业务需求、用户需求、系统需求、功能需求和非功能需求。

* 1. 软件需求的获取与确认过程

1. 获取
2. 协商
3. 说明与定义
4. 确认
   1. 快速原型化方法

原型法是这样一种软件开发技术，通过开发软件的初期版本让用户进行反馈来确认软件的可行性，研究开发技术或开发过程支持的其它问题。

快速原型法的重点是在开发过程的早期就开发出原型，使用反馈和分析提前，以支持开发过程。

使用原型可达到3个主要目的：明确并完善需求，研究技术选择方案，把原型发展为最终产品。

* 1. 基于用况的方法

基于用况的方法通过建立用况模型来获取与确定需求。

* + 1. 系统边界

系统边界是系统的所有内部成分与系统以外各种事物的分界线。

现实世界中的事物与系统的关系包括以下几种情况：

1. 某些事物位于系统边界以内，作为系统成分；
2. 某些事物将是与系统进行交互的参与者，系统中没有相应的成分作为它们的抽象表示，它们位于系统边界以外；
3. 有的事物可能既在系统内部有一个对象作为其抽象描述，而事物本身又在系统边界以外与系统进行交互；
4. 某些事物属于问题域，但与系统责任没有关系。
   * 1. 参与者
5. 概念与表示法

一个参与者(actor)定义了一组在功能上密切相关的角色，当一个事物与系统交互时，该事物可以扮演这样的角色。

参与者的标准图符是一个“人型符号”，参与者的名字放在图符的下方。

1. 识别参与者

参与者是在系统之外的与系统进行交互的任何事物。从3个方面识别参与者：

* 人员：直接使用系统的人员是参与者；
* 外部系统：所有与本系统交互的外部系统都是参与者；
* 设备：识别所有与系统交互的设备。

识别与组织参与者的指导策略：

1. 首先将精力集中于启动系统的参与者；
2. 考虑如何使用系统；
3. 对识别出来的参与者，记录其责任；
4. 若参与者存在着继承关系，则要予以建立。
   * 1. 用况
5. 概念与表示法

用况(use case)是描述系统的一项功能的一组动作序列，这样的动作序列表示参与者与系统间的交互，系统执行该动作序列要为参与者产生结果。

上述定义具有如下含义：

1. 使用用况来可视化、详述、构造和文档化所希望的系统行为；
2. 用况描述中的一个动作应该描述参与者或系统要完成的交互中的一个步骤；
3. 在用况中只描述参与者与系统彼此为对方直接做了些什么事，不描述怎么做，也不描述间接地做了些什么；
4. 系统所产生的结果，是指系统对参与者的动作要做出响应；
5. 用况描述的是参与者所使用的一项系统功能，该项功能应该相对完整，也即应该保证用况是某一项功能的完整的规格说明，而不能只是其中的一个片段；
6. 在用况描述中，由参与者首先发起交互的可能性较大，但有些交互也可能是由系统首先发起的。

在图形上，把用况表示成一个包含用况名字的椭圆。对用况的描述，可使用自然语言、活动图和伪码，也可使用用户自己定义的语言。

1. 用况与参与者之间的关系

在用况图中，一个参与者可以同多个用况交互，一个用况也可以同多个参与者交互。

把参与者与用况间的交互关系称为关联，用一条实线连起来。

1. 用况之间的关系

3种关系：包含、扩展和继承。

1. 包含

在两个或多个用况中经常存在着重复行为，为了避免重复，把重复行为放在一个用况中，原有的用况(基用况)再引入该用况(供应者用况)，这样就在用况间建立了包含关系。

用一个带箭头的虚线表示用况之间的包含关系。

1. 扩展

在一个或几个用况描述中，有时存在着可选的描述系统行为的片段。若存在这种情况，可以从用况中把可选的行为描述部分抽选出来，放在另一个用况(扩展用况)中，原来的用况(基用况)再用它扩展自己，以此来解决候选路径的复杂性。这样在描述基本动作序列的基用况和描述可选动作序列的扩展用况之间就建立了扩展关系。

用虚箭线表示用况之间的扩展关系。

1. 继承

用况之间的继承关系的含义和同类之间或参与者之间的继承关系一样。

用指向一般用况的带有封闭的空心箭头的实线来表示用况之间的继承关系。

1. 获取用况

可以从以下几个方面来获取用况：

1. 从参与者的角度获取用况；
2. 从系统功能的角度获取用况；
3. 利用场景获取用况。
   * 1. 用况图

用况图是一幅由参与者、用况以及这些元素之间的关系组成的图。这些关系是参与者和用况之间的关联、参与者之间的继承，以及用况之间的包含、扩展和继承。

* 1. 需求管理

需求管理要标识、控制和跟踪需求。

1. 需求标识和分类

最简单的方式是为每个需求赋予一个唯一的序列号。也可以按需求的抽象层次来进行数字编号。若需求量较大，则应进行分类组织，以便编写文档和使用。

1. 变更管理
2. 需求跟踪

进行需求跟踪的一个常见做法是建立需求矩阵。可根据需要定义需求矩阵中的需求关系，如依赖关系、从属关系、精化关系、实现关系等。

**第二部分 结构化软件开发方法**

1. **结构化分析建模**

深入描述软件的功能和性能，确定软件设计的约束和软件同其它系统元素的接口细节，定义软件的其它有效性分析。

**3.1 软件需求分析阶段的任务**

分为四个步骤：获取需求、分析需求、定义需求和验证需求：N、R1、R2和R3。

1. 获取需求

通过启发、引导从客户(或用户)那里得到原始需求是他们的业务要求(needs)，简称为N。

1. 需求分析

需求分析人员考虑以下几个方面：

1. 完整性；
2. 正确性；
3. 合理性；
4. 可行性：
5. 技术可行性；
6. 经济可行性；
7. 社会可行性。
8. 充分性。
9. 需求定义

作为软件开发的依据，必须将已经分析的需求清晰、全面、系统准确地描述为正式的文档。

1. 需求验证

为了确保已经定义的需求R2(需求规格说明)准确无误，并能为客户(或用户)理解或接受，需要对其进行严格的评审。这一评审一定要有客户(或用户)参加，并充分听取他们的意见。

**3.2 结构化分析方法简介**

传统的分析建模方法称为结构化分析(SA)方法。它是一种建模技术，在模型的核心是数据字典，它描述了所有的在目标系统中使用的和生成的数据对象。围绕这个核心有3种图：数据流图(DFD)，实体-关系图(ER图)和状态-迁移图(STD)。

**3.3 功能建模**

其思想就是用抽象模型的概念，按照软件内部数据传递、变换的关系，自顶向下逐层分解，直到找到满足功能要求的所有可实现的软件为止。功能模型用数据流图来描述。

3.3.1 数据流图的基本图形符号

数据源或数据汇点表示图中要处理数据的输入来源或处理结果要送往何处。

数据流表示数据沿箭头方向的流动。

加工是对数据对象的处理或变换。

数据存储在数据流图中起保存数据的作用，可以是数据库文件或任何形式的数据组织。

3.3.2 环境图

环境图(context diagram)也称为顶层数据流图(或0层数据流图)，它仅包括一个数据处理过程，也就是要开发的目标文件。

3.3.3 数据流图的分层

按照系统的层次结构进行逐步分解，并以分层的数据流图反映这种结构关系，能清楚地表达和容易理解整个系统。

画数据流图的基本步骤概括地说，就是自外向内，自顶向下，逐层细化，完善求精。

3.3.4 功能建模的方法和过程

(1)识别外部实体及输入输出数据流

(2)画出环境图(顶层数据流图)

(3)画出一层数据流图

(4)画出二层数据流图

**3.4 数据建模**

在结构化分析方法中，使用实体-关系建模技术来建立数据模型，也称为ER图。图中仅包含3种相互关联的元素：数据对象(实体)、描述数据对象的属性及数据对象彼此间相互连接的关系。

3.4.1 数据对象

数据对象是目标系统所需要的复合信息的表示，所谓复合信息是具有若干不同属性的信息。

3.4.2 属性

属性定义数据对象的特征，在ER图中用椭圆或圆角矩形表示属性。

3.4.3 关系

不同数据对象的实例3之间是有关联关系的，在ER图上用无向边表示，在无向边上可以标明关系的名字，也可以不标名字。可以将实例的关联分为三种：一对一关联、一对多关联、多对多关联。

**3.5 行为建模**

在需求分析过程中，应该建立起软件的行为模型。状态转换图(简称转换图)通过描绘系统的状态及引起系统状态转换的事件来表示系统的行为。

3.5.1 状态

状态是任何可以被观察到的系统行为模式，一个状态代表系统的一种行为模式，状态规定了系统对事件的响应方式。

在状态图中定义的状态可能有：初态、终态和中间态。初态用实心圆表示，终态用牛眼圆形表示，中间态用圆角矩形表示。

中间态可能包括三个部分：第一部分为状态的名称，第二部分为状态变量的名字和值，第三部分为活动表。第二、三部分可选。

活动部分的语法如下：

事件名(参数表)/动作表达式

3.5.2 状态转换

状态图中两个状态之间带箭头的连线称为状态转换。

3.5.3 事件

事件是在某个特定时刻发生的事情，它是对引起系统做动作或从一个状态转换到另一个状态的外部事件的抽象。

事件表达式的语法如下：

事件说明(守卫条件)/动作表达式

事件说明的语法如下：

事件名(参数表)

守卫条件是一个布尔表达式，动作表达式是一个过程表达式。

3.6 数据字典

数据字典以词条方式定义在数据模型、功能模型和行为模型中出现的数据对象和控制信息的特征，给出它们的准确定义，包括数据流、加工、数据文件、数据元素，以及数据点、数据汇点等。

3.6.1 词条描述

对于在数据流图中每一个被命名的图形元素均加以定义，其内容包括图形元素的名字，图形元素的别名或编号，图形元素的类别(如加工、数据流、数据文件、数据元素、数据源点或数据汇点等)、描述、定义、位置等。具体如下：

1. 数据流词条；
2. 数据元素词条；
3. 数据存储文件词条；
4. 加工词条；
5. 数据源点及数据汇点词条。

3.6.2 数据结构描述

有定义式、Warnier图等。

1. 定义式
2. Warnier图

3.7 加工规格说明

在对数据流图的分解中，位于层次树最低层的加工也称为基本加工或原子加工，对于每一个基本加工都需要进一步说明，这称为加工规格说明。

它应满足如下要求：

1. 对数据流图的每一个基本加工，必须有一个加工规格说明。
2. 加工规格说明必须描述基本加工如何把输入数据流变换为输出数据流的加工规格。
3. 加工规格说明必须描述实现加工的策略而不是实现加工的细节。
4. 加工规格说明中包含的信息应是充足的，完备的，有用的，没有重复的多余信息。

加工规格说明的内容可以包含叙述性正文、数学方程、图表等，也可以使用决策表和决策树。

3.7.1 决策表

在某些数据处理问题中，某个加工的执行需要依赖于多个逻辑条件的取值，此时可以用决策表(decision table)来描述。它由4个部分构成：左上部分是条件(condition stub)、左下部分是动作(action stub)、右上部分是条件项(condition entry)、右下部分是动作项(action entry)。

通常，将每一列条件项和动作项称作一条处理规则，它包含了一个条件取值组合和相应要执行的一组动作。

建立决策表的步骤如下：

1. 列出与一个具体过程(或模块)有关的所有操作。
2. 列出过程执行期间的所有条件(或所有判断)。
3. 将特定条件取值组合与特定的处理相匹配，消去不可能发生的条件取值组合。
4. 将右部每一纵列规定为一个处理规则，即对于某一条件取值组合将有什么动作。

3.7.2 决策树

它也是用来表达加工逻辑的一种工具。在构造决策树时，首先应从文档叙述中分清哪些是判定条件，哪些是判定结果。然后，从文档叙述中的一些连接词中，找出判定条件的从属关系、并列关系、选择关系等。

3.8 需求规格说明

按照国家标准GB/T8567-2006《计算机软件文档编制规范》。

3.8.1 软件需求规格说明

SRS描述了计算机软件配置项的需求以及为确保需求得到满足所使用的方法。

3.8.2 数据需求说明

DRD描述了在整个开发过程中所需处理的数据，以及采集数据的要求等。

1. **总体设计**

在软件生存期中，软件设计处于需求分析阶段及软件构造阶段(或编码阶段)的中间位置。

* 1. 软件设计的概念和目标
     1. 软件设计的概念

软件设计包括一套原理、概念和实践，可以指导高质量的系统或产品开发。设计原理建立了最重要的原则，用以指导设计人员工作。

* + 1. 软件设计的目标

软件设计的目标涉及性能、可靠性、成本、维护等多个方面的目标。一些可能的准则如下：

1. 性能准则
2. 可靠性准则
3. 成本准则
4. 维护准则
5. 最终用户准则
   1. 软件设计的任务

软件设计的主要任务是要解决如何做的问题，要在需求分析的基础之上，建立各种设计模型，并通过对设计模型的分析和评估，来确定这些模型是否能够满足需求。

* + 1. 软件设计的阶段与任务

从工程管理的角度，分为两个阶段：概要设计阶段和详细设计阶段。传统的结构化方法将软件设计划分为体系结构设计、数据设计、接口设计和过程设计4部分；而面向对象方法将软件划分设计划分为体系结构设计、类设计/数据设计、接口设计和构件级设计4部分。

* + 1. 结构化设计与结构化分析的关系

软件设计必须依据对软件的需求来进行，结构化分析的结果为结构化设计提供了最基本的输入信息。

* 1. 模块结构与数据结构

软件的结构包括两个部分，一部分为软件的模块结构，另一部分为软件的数据结构。

* + 1. 模块结果及表示

1. 模块

一个软件系统通常由很多模块组成，模块用矩形框表示，并用模块的名字标记它。可以按照在软件系统中的功能将模块划分为四种类型：

1. 传入模块
2. 传出模块
3. 变换模块
4. 协调模块
5. 模块结构
6. 树状结构
7. 网状结构
8. 结构图

结构图(SC)是精确表达模块结构的图形表示工具。

1. 模块的调用关系和接口；
2. 模块间的信息传递；
3. 两个辅助符号；
4. 结构图的形态特性。
   * 1. 数据结构及表示

典型的数据结构：标量项，链表，顺序向量，树状结构，网状结构，n维空间。

* 1. 创建良好设计的原则
     1. 分而治之和模块化
     2. 模块独立性

1. 松散耦合：非直接耦合，数据耦合，标记耦合，控制耦合，外部耦合，公共耦合，内容耦合。
2. 高度内聚：巧合内聚，逻辑内聚，时间内聚，过程内聚，通信内聚，信息内聚，功能内聚。
   * 1. 提高抽象层次：在最高的抽象层次上，可以使用问题所处环境的语言概括地描述问题的解决方案；在较低的抽象层次上，采用更过程化的方法，将面向问题的术语和面向实现的术语结合起来描述问题的解法；在最低的抽象层次上，则用某种程序设计语言来描述问题的解法。过程抽象和数据抽象是两种常用的抽象手段。
     2. 复用性抽象

复用是指同一实体不做修改或稍加修改就可以多次重复使用，将复用的思想用于软件开发，称为软件复用。将软件的重用部分称为软构件。

* + 1. 灵活性设计

保证软件灵活性设计的关键是抽象。

在设计中引入灵活性的方法如下：

1. 降低耦合并提高内聚(易于提高替换能力)；
2. 建立抽象(创建有多态操作的接口和父类)；
3. 不要将代码写死(消除代码中的常数)；
4. 抛出异常(由操作的调用者处理异常)；
5. 使用并创建可复用的代码。
   * 1. 预防过期
     2. 可移植性设计
     3. 可测试性设计
     4. 防御性设计
   1. 面向数据流的设计

面向数据流的设计方法也称为过程驱动的设计方法。

* + 1. 设计过程

1. 复查并精化数据流图；
2. 确定数据流图中数据流的类型，典型的数据流类型有变换型数据流和事务型数据流；
3. 导出初始的软件结构图；
4. 逐级分解；
5. 精化软件结构；
6. 导出接口描述和全局数据结构。
   * 1. 典型的数据流类型和系统结构
7. 变换型数据流与变换型系统结构图

工作过程大致分为3步：获取数据、变换数据和给出数据。

1. 事务型数据流与事务型系统结构图
   * 1. 变换型映射方法

是一系列设计步骤的总称。变换分析方法由4步组成：重画数据流图；区分有效(逻辑)输入，有效(逻辑)输出和中心变换部分；进行一级分解，设计上层模块；进行二级分解，设计输入、输出和中心变换部分的中、下层模块。

* + 1. 事务型映射方法

在很多软件应用中，存在某种作业数据流，它可以引发一个或多个处理，这些处理能够完成该作业要求的功能，这种数据流就叫做事务。

事务分析方法的步骤如下：

1. 识别事务流；
2. 规定适当的事务型结构；
3. 识别各种事务和它们定义的操作；
4. 注意利用公共模块；
5. 对每一事务，或对密切联系的一组事务，建立一个事务处理模块；
6. 对事务处理模块规定它们全部的下层操作模块；
7. 对操作模块规定它们的全部细节模块。
   * 1. 软件模块结构的改进方法
8. 模块功能的完善化；
9. 消除重复功能，改善软件结构；
10. 模块的作用范围应在控制范围之内；
11. 尽可能减少高扇出结构，随着深度增大扇入；
12. 避免或减少使用病态连接；
13. 模块的大小要适中。
    1. 接口设计
       1. 接口设计概述

接口设计的依据是数据流图中的自动化系统边界。

接口设计主要包括3个方面：模块或软件构件间的接口设计；软件与其它硬件系统之间的接口设计；软件与人(用户)之间的交互设计。

* + 1. 人机交互界面

1. 用户界面应具备的特性
2. 可使用性；
3. 灵活性；
4. 可靠性。
5. 用户类型

4种类型：

1. 外行型；
2. 初学型；
3. 熟练型；
4. 专家型。
5. 界面设计类型

从以下几个方面考察：

1. 使用的难易程度；
2. 学习的难易程度；
3. 操作速度；
4. 复杂过程；
5. 控制；
6. 开发的难易程度。
7. 设计详细的交互

若干准则：

1. 一致性；
2. 操作步骤少；
3. 不要“哑播放”；
4. 提供Undo功能；
5. 减少人脑的记忆负担；
6. 提高学习效率。

设计HCI(人机交互)类应注意以下几点：

1. 窗口需要进一步细化；
2. 设计HCI类；
3. 每个类包括窗口的菜单条、下拉菜单和弹出菜单的定义；
4. 每个类负责窗口的实际显示。
   1. 数据设计
      1. 文件设计
5. 顺序文件；
6. 直接存取文件；
7. 索引文件；
8. 分区文件；
9. 虚拟存储文件。
   * 1. 数据库设计

将数据库分为：网状数据库、层次数据库、关系数据库、面向对象数据库、文档数据库和多维数据库。

映射规则：

1. 数据对象(实体)的映射：采用横切和竖切的方法。
2. 关系的映射：
3. 一对一关系的映射；
4. 一对多关系的映射；
5. 多对多关系的映射。
   1. 软件设计规格说明

国家标准GB/T 8567-2006《计算机软件文档编制规范》中有关软件总体设计的文档是《系统/子系统设计(结构设计)说明(SSDD)》。

1. **详细设计与编码**
   1. 结构化程序设计

20世纪70年代，Dijkstra提出了程序要实现结构化的主张，并将这一程序设计称为结构化程序设计(structured programming)。

结构化程序需要符合两个条件：

1. 一个程序的代码块仅仅通过顺序、选择和循环3种基本控制结构进行连接；
2. 每个代码块只有一个入口和一个出口。

进行结构化程序设计时，需要遵循以下原则：

1. 使用语言中的顺序、选择、重复等有限的基本控制结构表示程序逻辑；
2. 使用的控制结构只准许有一个出口和一个入口；
3. 程序语句组成容易识别的块(block)，每块只有一个入口和一个出口；
4. 复杂结构应该用基本控制结构进行组合嵌套来实现；
5. 语言中没有的控制结构，可用一段等价的程序段模拟，但要求该程序段在整个系统中应前后一致；
6. 严格控制GOTO语句；
7. 在程序设计过程中，尽量采用自顶向下(top-down)，逐步细化(stepwise refinement)的方法，由粗到细，一步步展开。

5.1.2 自顶向下、逐步细化的设计过程

包括两个方面：一是将复杂问题的解法分解和细化成由若干个模块组成的层次结构；二是将每个模块的功能逐步分解细化为一系列的处理。

* 1. 过程设计的工具

表达过程规格说明的工具称为过程描述工具，分为以下三类：

1. 图形工具；
2. 表格工具；
3. 语言工具。

5.2.1 程序流程图

5种基本的控制结构：顺序型、选择型、先判定(while)型循环、后判定(until)型循环、多情况型(case)选择。

5.2.2 N-S图

为表示五种基本控制结构规定了五种图形构件。

5.2.3 PAD图

是日本日立公司提出的，设置了五种基本控制结构的图式，并允许递归使用。

5.2.4 伪代码

是一种介于自然语言和形式化语言之间的半形式化语言，是一种用于描述功能模块的算法设计和加工细节的语言，也称为程序设计语言(PDL)。

一般地，伪码的语法规则分为“外语法(Outer Syntax)”和“内语法(Inter Syntax)”。

* 1. 程序设计语言

程序编码阶段的任务是将软件的详细设计转换成用程序设计语言实现的程序代码。

5.3.1程序设计语言的性能

(1) 软件心理学的观点

软件心理学研究的任务是：关注人们对软件所关心的某些方面，如使用方便、简明易学、可靠性、减少故障发生率，以及使用户更加满意等。同时对机器功效、软件能力以及硬件限制也予以注意。

影响程序员心理的语言特性有如下5种：

1. 一致性；
2. 二义性；
3. 简洁性；
4. 局部性；
5. 传统性。

(2)软件工程的观点

1) 详细设计应能直接地容易地翻译成代码程序；

2) 源程序应具有可移植性；

3) 编译程序应具有较高的效率；

4) 尽可能应用代码生成的自动工具；

5) 可维护性。

5.3.2 程序设计语言的分类

分为4类：

1. 从属于机器的语言-第一代语言
2. 汇编语言-第二代语言
3. 高级程序设计语言-第三代语言
4. 第四代语言

5.3.3 程序设计语言的选择

1) 应用领域；

2) 系统用户的要求；

3) 编程语言自身的功能；

4) 编码和维护成本及开发环境；

5) 编程人员的技能；

6) 软件可移植性。

* 1. 程序设计风格
     1. 源程序文档化

1. 标识符的命名
2. 程序的注释
3. 视觉组织-空格、空行和移行
   * 1. 数据说明标准化
4. 数据说明的次序应该规范化，使数据属性容易查找，也有利于测试、排错和维护；
5. 当多个变量名用一个语句说明时，应当对这些变量按字母顺序排列；
6. 对于复杂的数据结构，应当使用注释对其进行说明。
   * 1. 语句结构简单化
7. 在一行内只写一条语句，并且采用适当的移行格式，使程序的逻辑和功能变得更加明确；
8. 程序编写首先应该考虑清晰性，不要刻意追求技巧性，使得程序编写得过于紧凑；
9. 程序编写得要简单，写清楚，直截了当地说明程序员的用意；
10. 除非对效率有特殊的要求，程序编写要做到清晰第一，效率第二；
11. 避免使用临时变量使得可读性下降；
12. 让编译程序做简单的优化；
13. 尽可能使用库函数；
14. 避免不必要的转移；
15. 尽量只使用3种基本的控制结构来编写程序；
16. 避免使用空的else语句和if…then if…语句；
17. 避免采用过于复杂的条件测试；
18. 尽量减少使用“否定”条件的条件语句。
    * 1. 输入/输出规范化

考虑以下原则：

1. 对所有的输入数据都进行检验；
2. 检查输入项的各种重要组合的重要性，必要时报告输入状态信息；
3. 使得输入的步骤和操作尽可能简单，并保持简单的输入格式；
4. 输入数据时，应允许使用自由格式输入；
5. 应允许默认值；
6. 输入一批数据时，最好使用输入结果标志，而不要由用户指定输入数据数目；
7. 在以交互式输入/输出方式进行输入时，要在屏幕上使用提示符明确提示交互输入的请求，指明可使用选择项的种类和取值范围；
8. 当程序设计语言对输入/输出格式有严格要求时，应保持输入格式与输出语句要求的一致性；
9. 给所有的输出加注释，并设计输出报表格式。
   1. 程序复杂程度度量
      1. McCabe方法

先画出控制流图，再计算控制流图中环路的个数。

1. 控制流图

可以将控制流图看做是退化的程序流程图。

1. 程序的环路复杂度
   * 1. Halstead方法

是根据程序中运算符和操作数的总数来度量程序的复杂度。

1. 实际的Halstead长度

 (其中，为程序中运算符出现的总次数，为操作数出现的总次数)

1. 预测的Halstead长度
2. 程序的潜在错误
3. **软件测试**

6.1软件测试的基本概念

6.1.1什么是软件测试

是在软件投入生产性运行之前，对软件需求分析、设计规格说明和编码的最终复审，是软件质量控制的关键步骤；软件测试是为了发现错误而执行程序的过程；或者说，软件测试是根据软件开发各阶段的规格说明和程序的内部结构而精心设计的一批测试用例(即输入数据及其预期的输出结果)，并利用这些测试用例去运行程序，以发现程序错误的过程。

6.1.2 软件测试的目的和原则

软件测试的目的提出以下观点：

1. 测试是程序的执行过程，目的在于发现错误；
2. 一个好的测试用例在于能发现至今未发现的错误；
3. 一个成功的测试是发现了至今未发现的错误的测试。

原则如下：

1. 应当把“尽早地和不断地进行软件测试”作为软件开发者的座右铭；
2. 测试用例应当由测试输入数据和与之对应的预期输出结果这两部分组成；
3. 程序员应避免检查自己的程序；
4. 在设计测试用例时，应当包含合理的输入条件和不合理的输入条件；
5. 充分注意测试中的群集现象；
6. 严格执行测试计划，排除测试的随意性；
7. 应当对每一个测试结果作全面检查；
8. 妥善保存测试计划、测试用例、出错统计和最终分析报告，为维护提供方便。

6.1.3 软件测试的对象

需求分析、概要设计、详细设计、程序编码等各阶段所得到的文档资料，包括需求规格说明、概要设计规格说明、详细设计规格说明以及源程序，都应该成为软件测试的对象。

确认(validation)：是一系列的活动和过程，其目的是想证实在一个给定的外部环境中软件逻辑的正确性；

验证(verification)：试图证明在软件生存期的各个阶段，以及阶段间的逻辑协调性、完备性和正确性。

确认和验证工作都属于软件测试。

6.1.4 测试信息流

测试过程需要如下3类输入：

1. 软件配置；
2. 测试配置；
3. 测试工具：回归测试。

6.1.5 测试与软件开发各阶段的关系

6.1.6 白盒测试与黑盒测试

1) 黑盒测试

已知产品的功能设计规格，可以进行测试证明每个实现了的功能是否符合要求，这就是黑盒测试。

2) 白盒测试

是对软件的过程性细节做细致的检查。

6.2 白盒测试的测试用例设计

6.2.1 逻辑覆盖

是以程序内部的逻辑结构为基础的设计测试用例的技术。

6.2.2 语句覆盖

是设计若干测试用例，运行被测程序，使得每一个可执行语句至少执行一次。

6.2.3 判定覆盖

就是设计若干个测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次。

6.2.4 条件覆盖

就是设计若干个测试用例，运行被测程序，使得程序中每个判断的每个条件的可能取值至少执行一次。

6.2.5 判定-条件覆盖

就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，同时每个判断本身的所有可能判断结果至少执行一次。

6.2.6 条件组合覆盖

就是设计足够的测试用例，运行被测程序，使得程序中每个判断的所有可能条件取值组合至少执行一次。

6.2.7 路径覆盖

就是设计足够的测试用例，覆盖程序中所有可能的路径。

6.3 基本路径测试

此法适用于模块的详细设计及源程序，其主要步骤如下：

1. 以详细设计或源代码作为基础，导出程序的控制流图；
2. 计算得到的控制流图G的环路复杂性；
3. 确定线性无关的基本路径集；
4. 生成测试用例，确保基本路径集中每条路径的执行。

6.4 黑盒测试的测试用例设计

6.4.1 等价类划分

(1)划分等价类

(2)确定测试用例

(3)用等价类划分法设计测试用例的实例

6.4.2 边界值分析

(1)边界值分析方法的考虑

(2)选择测试用例的原则：6个。

6.5 软件测试的策略

6.5.1 单元测试

又称模块测试，是针对软件设计的最小单元-程序模块，进行正确性检验的 测试工作。其目的在于发现各模块内部可能存在的各种差错。

1. 单元测试的内容
2. 模块接口测试
3. 局部数据结构测试
4. 路径测试
5. 错误处理测试
6. 边界测试
7. 单元测试的步骤

辅助模块分为两类：

1. 驱动模块(driver)
2. 桩模块(stub)

6.5.2组装测试

1. 一次性组装方式
2. 增值式组装方式：自顶向下，自底向上，混合增值式
3. 组装测试的组织和实施

6.5.3 确认测试

又称为有效性测试。

1. 进行有效性测试(黑盒测试)
2. 软件配置复查
3. 测试和测试

前者是由一个用户在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的测试。

后者是由软件的多个用户在一个或多个用户的实际使用环境之下进行的测试。应尽可能由主持产品开发的人员管理。

1. 验收测试
2. 确认验收的结果：分为与预期相符和不符。

6.5.4 系统测试

是将通过确认测试的软件，作为整个计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据、人员等其它系统元素结合走在一起，在实际运行环境下，对计算机系统进行一系列的组装测试和确认测试。

6.5.5 测试的类型

功能测试、回归测试、可靠性测试、强度测试、性能测试、恢复测试、启动/停止测试、配置测试(配置命令测试、循环配置测试、修复测试)、安全性测试、可使用性测试、可支持性测试、安装测试、互连测试、兼容性测试、容量测试、文档测试。

6.6 人工测试

不要求在计算机实际执行被测程序，而是以一些人工的模拟技术和一些类似动态分析所使用的方法对程序进行分析和测试。

6.6.1 静态分析

要对源程序进行静态检验。主要方法如下：

1. 生成各种引用表：标号交叉引用表，变量交叉引用表，子程序、宏和函数表，等价表，常数表。
2. 静态错误分析
3. 类型和单位分析；
4. 引用分析；
5. 表达式分析；
6. 接口分析。

6.6.2 人工测试的几种形式

可以有效地发现30%~70%的逻辑设计和编码错误。

1. 桌前检查：由程序员自己检查自己编写的程序。
2. 代码评审：是由若干个程序员和测试员组成一个评审小组，通过阅读、讨论和争议，对程序进行静态分析的过程。
3. 走查：
4. 把材料先发给走查小组每个成员；
5. 借助于测试用例的媒介作用，对程序的逻辑和功能提出各种疑问。

6.7 调试

也称为排错或纠错，其任务在于为错误确切地定位，找出出错的根源，并且通过修改程序将其排除。可按如下步骤进行：

1. 针对测试提供的信息，分析错误的外部表现形式，确定程序出错的位置；
2. 研究程序的相关部分，找出导致错误的内在原因；
3. 修改相关的程序段，如果是设计导致的错误，则需修改相关的设计，以排除错误；
4. 重复执行以前发现错误的测试；
5. 如果重新测试表明修改无效，发生错误的现象仍然出现，则要撤销上述修改，再次进行信息分析，实施上述过程，直至修改有效为止。

**第3部分 面向对象软件开发方法**

1. **面向对象方法概述**
   1. 面向对象的基本思想

是一种运用对象、类、继承、聚合、关联、消息、封装等概念和原则来构造软件系统的开发方法。其基本思想包括以下内容：

1. 此方法要求从现实世界中客观存在的事物出发来建立软件系统，强调直接以问题域(现实世界)中的事物为中心来思考问题和认识问题，并根据这些事物的本质特征和系统责任，把它们抽象地表示为系统中的对象；
2. 用对象的属性表示事物的数据特征，用对象的操作表示事物的行为特征；
3. 通过抽象对事物进行分类；
4. 复杂的对象可以由简单的对象构成；
5. 运用抽象的原则，可以得到较一般的类和较特殊的类；
6. 对象之间通过消息进行通信，以实现对象之间的动态联系；
7. 用关联表达类之间的静态关系。
   1. 面向对象的主要概念和基本原则
      1. 主要概念
8. 对象
9. 类
10. 继承
11. 消息：把向对象发出的操作请求称为消息；
12. 关联：是两个或多个类之间的一个静态关系；
13. 聚合：一个对象由其它若干个对象作为其构成部分。
    * 1. 基本原则

主要有：抽象、分类、封装、消息通信、多态性、复杂性控制等。

* 1. 面向对象方法的发展史和现状简介

1. 雏形阶段

20实际60年代的Simula67，是面向对象语言的先驱。1972年发布了Smalltalk-72。

1. 完善阶段

1981年推出了Smalltalk-80。

1. 繁荣阶段

20世纪80年代中期到90年代。UML已成为世界性的建模语言。

* 1. 关于统一建模语言

定义了建立系统模型所需要的概念并给出了表示法。

UML 2.0规范由4部分组成：基础结构、上层结构、对象约束语言和图交换。

1. 面向对象分析

8.1 什么是面向对象分析

OOA是软件生命周期的一个阶段，具有一般分析方法所共有的内容、目标及策略。

1. 问题域与系统责任

问题域：被开发系统的应用领域，即在现实世界中这个系统所涉及的业务范围；

系统责任：被开发系统的应该具备的职能。

1. OOA模型

分为3个层次：对象层、特征层和关系层。

这3个层次分别描述了：系统中应设立哪些类对象，每一类对象的内部构成，每类对象与外部的关系。

1. OOA过程

在青鸟面向对象软件开发规范中总结出来的。

8.2 建立基本模型-类图

8.2.1 对象与类

(1)概念和表示法

1)概念

对象是具有明确语义边界并封装了状态和行为的实体，由一组属性和作用在这组属性上的一组操作构成，它是构成系统的一个基本单元。

类是对具有相同属性和操作的一组对象的抽象描述，即它为属于该类的全部对象提供了统一的抽象描述。

2)表示法

用一个由水平线划分为3个分栏的实线矩形表示类。在最上面的分栏放类名，中间的分栏放属性列表，最下面的分栏放操作列表。

(2)识别对象与类

1)考虑问题域

2)考虑系统边界

3)考虑系统责任

(3)审查与筛选

1)舍弃无用的对象

2)精简对象

3)推迟到OOD考虑的对象

(4)抽象出类并进行整理

1)对象分类

2)对类进行调整

A. 类的属性或操作不适合该类的全部对象

B. 属性及操作相同的类

C. 属性及操作相似的类

D. 对同一事物的重复描述

(5)认识对象的主动行为并识别主动对象

主动对象是具有主动行为的对象，在设计阶段是拥有线程或进程并能够启动控制活动的对象。

主动类是它的实例为主动对象的类。

4个识别主动对象的策略：

1. 从问题域和系统责任考虑，哪些对象将在系统中呈现主动行为；
2. 重点考虑直接与系统边界以外的参与者直接进行交互的对象，这些对象可能是主动对象；
3. 根据系统责任观察系统功能的构成层次，重点考虑完成最外层功能的对象是否应该定义为主动对象；
4. 通过逆向追踪操作的执行路径来发现主动对象。

(6)类的命名

遵循以下原则：

1. 应恰好符合这个类及它的特殊类的每一个对象；
2. 类的名字应反映每个对象个体，而不是整个群体；
3. 采用名词或带有定语的名词，并使用规范的词汇对类命名；
4. 考虑使用恰当种类的语言文字对类命名。

(7)建立类图的对象层

根据已经找到并确认的类，建立类图的对象层：

1. 用类符号表示每个类，应该使用建模工具把它们绘制出来，形成OOA的基本模型中的对象层；
2. 在类详细说明中填写各类的详细信息；
3. 在发现对象的活动中认识到属性和操作要随时加入类符号中，对于认识到的符号也是如此。

8.2.2 定义属性与操作

(1)属性

事物的静态特征用类的属性来表示。定义属性的活动是要把一个类的所有对象所共有的每一个静态特征抽出来，分别作为该类的一个单独的属性。

1. 概念

属性是用来描述对象静态特征的一个数据项。基本格式为：

[可见性] 属性名[:类型][=初始值]

1. 表示法
2. 识别属性

一些启发性策略：

1. 按照一般常识这个对象应该具有哪些属性？
2. 在当前的问题域中，这个对象应该有哪些属性？
3. 根据系统责任的要求，这个对象应该有哪些属性？
4. 建立这个对象是为了保存和管理哪些信息？
5. 为了在对象的操作中实现特定功能，需要增设哪些属性？
6. 对象有哪些需要区别的状态？是否需增加一个属性来记录这些状态？
7. 审查和筛选
8. 属性的定位

是指把属性放置到拥有它的那个对象所属的类的符号中。

1. 描述属性

包括对属性的命名和对属性的详细描述。

(2)操作

抽象为类的事物往往具有一定的行为。

1. 概念

操作是类的对象被要求执行的服务。基本格式为：

[可见性] 操作名[(操作列表)] [:返回类型]

1. 表示法

把操作的可见性表示为+(共有的)、#(受保护的)、-(私有的)或~(包范围的)。

1. 识别操作
2. 考虑系统责任；
3. 考虑问题域；
4. 分析对象状态；
5. 追踪操作的执行路线。
6. 审查与调整
7. 检查每个操作是否真正有用；
8. 检查每个操作是不是高内聚的。
9. 操作的定位

把操作放置在哪个对象中，应该与问题域中拥有这种行为的实际事物相一致。

1. 描述操作

包括对操作的命名和对操作的详细描述。

8.2.3 建立关系

继承、关联、聚合和依赖。

8.3 建立行为模型

UML中的一些图可用于建立行为模型。

8.3.1 顺序图

(1)概念及表示法

1)顺序图概述

是一种详细描述对象之间以及对象与参与者之间交互的图，它由一组相互协作的对象(或参与者)以及它们之间可发送的消息组成，它强调消息之间的顺序。

2)对象的生命线

表示对象(object lifeline)在一段时间内的存在。

3)执行规则说明

是一个对象执行一个操作的时期。

4)消息

是对象之间通信的规格说明，这样的通信用于传输将要发生的活动所需要的信息。分为同步消息和异步消息。

(2)建立顺序图

8.3.2 活动图

(1)概念与表示法

活动图是描述动作、动作的执行顺序以及动作的输入与输出的图，它由结点和边这两种基本元素构成。

1. 动作(action)

是规格说明行为的基础单元，用以描述系统中的活动，它是原子的和即时的。

活动(activity)是由一系列的动作构成的，也称它为动作表达式，用于描述系统的行为。

1. 控制流

是指当一个动作结束时，马上进入下一个动作。

1. 对象流

对象可以出现在控制流，用以描述在动作间输入与输出的数据。

1. 泳道

在对业务过程建模时，有时要把动作分组，每组由特定的履行者来执行。

(2)建立活动图

通常用活动图对业务过程和操作的算法建模。

1. 对业务过程建模

业务过程描述了工作的流程以及贯穿于其中的业务对象。

1. 对操作建模

用活动图对操作建模是指把活动图作为流程图来对操作的算法细节建模。

8.3.3 状态机图

(1)概念与表示法

它描述了一个对象在其生命期内因响应事件所经过的状态序列以及对这些事件所做出的反应。

1. 事件

是对一件事情的规格说明，它的发生可能引发对象的状态转移。

1. 信号事件

信号(signal)是对象之间的异步通信的规格说明。按照如下格式定义信号：

信号名’(‘用逗号分隔的参数列表’)’

1. 调用事件

对操作的调用的接收，导致一个调用事件，这样的操作由接收事件的对象实现。

1. 时间事件

在指定事件发生后，经过了一段时间或到了指定时间，就导致一个时间事件。

1. 改变事件(条件变为真事件)

用布尔表达式描述的指派条件变为真，就导致了一个改变事件。按如下格式定义事件：

事件名(用逗号分隔的参数列表)

参数的格式：参数名：类型表达式。

1. 状态(state)

是对象在其生存周期中满足某一条件、进行某种活动或等待某一事件的条件或状况。

1. 转移(transition)

分为状态内转移和状态间转移。格式如下：

时间触发器[(用逗号分隔的参数表)][监护条件][/动作表达式]

1. 状态内转移

是指在一个状态内由事件引起的动作或活动执行后，对象仍处于该状态的情形，即引发状态内转移的事件的发生不会导致状态的改变。

1. 状态间转移

是两个状态之间的关系，表示当一个特定事件出现时，且满足一定的条件，对象就从第一个状态(源状态)进入第二个状态(目标状态)，并执行一定的动作或活动。

1. 组合状态

是由两个或多个子状态构成的状态，其中的子状态还可以是组合状态。

(2)建立状态图

8.4 建立组织模型-包图

8.4.1 概念与表示法

包(package)是对模型元素分割的机制。

1. 包的层次型
2. 包中元素的命名：C:A:B
3. 包中元素的可见性

通过在元素名字前加上一个可见性符号来指示。分为：公共的(+)、私有的(-)、受保护的(#)或包范围的(~)。

1. 包间的关系

包之间除了可以具有拥有关系(包内有包)外，包之间还可以具有引入依赖和访问依赖。

引入依赖是两个包之间的一种许可依赖关系，一个包中的可见性为公共的模型元素，可以在指定的包中被引用。

访问依赖是两个包之间的一种许可依赖关系。

8.4.2 如何划分与组织包

基本策略：

1. 识别低层包

把在语义上接近的并倾向于一起变化的元素放在一个包中。

1. 合并或组织包

如果低层包数量过多，则考虑把它们合并成一个包，或用高层包组织它们。

1. 组织包的层次

包的层次不要过深。

1. 标识包中的模型元素的可见性
2. 建立包间的关系
3. **面向对象设计与测试**

9.1 什么是面向对象设计

从正面观察OOD模型，它包括一个核心部分，即问题域部分，还包括3个外围部分：人机交互部分、控制驱动部分和数据管理部分。

OOD过程由4项活动组成：问题域部分设计、人机交互部分设计、控制驱动部分设计和数据管理部分设计。

9.2问题域部分的设计

9.2.1 为复用类而增加结构

9.2.2 提高性能

9.2.3 增加一般类以建立共同协议

9.2.4 按编程语言调整继承

(1)对多继承的调整

(2)取消继承

方法1：把继承结构展平；

方法2：采用聚合的方法

9.2.5 对复杂关联的转化并决定关联的实现方式

(1)把关联类和N元关联转化为二元关联

(2)关联的实现方式：聚合、关联

9.2.6 调整与完善属性

按照语法：[可见性] 属性名[‘: ’类型][‘=’ 初始值]对属性的定义进行完善。

9.2.7 构造及优化算法

格式：[可见性] 操作名[‘(‘参数列表’)’][‘:’ 返回类型]

9.2.8 决定对象间的可访问性

9.2.9 定义对象

按照如下方式：

1. 用相应的类定义内存中的对象，包括静态声明和动态创建两种方式；
2. 当系统需要通过从外存读取数据来创建对象时，就先创建该对象，再从外存中读取这个对象数据，把数据赋值给对象的相应属性。

9.3 人机交互部分的设计

此部分是人和计算机之家交互信息的媒介，对它的设计涉及计算机科学、心理学、艺术学、认知科学、人机工程学等学科。

9.3.1 什么是人机交互部分

是OOD模型的组成部分之一，要解决人如何命令系统以及系统如何向用户提供信息等问题。

9.3.2 如何分析人机交互部分

(1)分析与系统交互的人员参与者

(2)从用况分析人机交互

9.3.3 如何设计人机交互部分

(1)设计输入与输出

1)设计输入

A. 确定输入设备

B. 设计输入界面

C. 输入步骤的细化

2)设计输出

A. 确定输出设备

B. 确定输出的形式与内容

C. 输出步骤的细化

(2)命令的组织

两种措施：分解、组合。

基本命令：使用一项独立的系统功能的命令；

命令步：它是执行一条基本命令的交互过程中所包含的具体输入步骤；

高层命令：如果一条命令是在另外一条命令的引导下被选用的，则后者称为前者的高层命令。

1. 用OO概念表达所有的界面成分

9.4 控制驱动部分的设计

9.4.1 什么是控制驱动部分

控制流是软件在处理机上顺序执行的一个动作序列。

9.4.2 控制流

在面向对象方法中，用一个主动对象表示一个独立的控制流，该对象驱动进程或线程，也即每个控制流都以一个表示独立的进程或线程的主动对象为根。

9.4.3 如何设计控制驱动部分

(1)识别控制流

(2)审查

(3)定义控制流

1) 控制流的描述

2) 控制流的表示

(4)进程间和线程间的通信

控制流间常用的通信机制主要有4种：操作调用、邮箱、共享存储器、远程过程调用。

对进程间和线程间的通信建模：

1. 对线程间的通信建模

同一个进程中的跨线程的对象间可以通过邮箱、共享存储器或操作调用进行通信。

1. 对进程间的通信建模

可以使用邮箱、远程过程调用和共享存储器进行进程之间的通信。

1. 控制流间的同步

9.5 数据管理部分的设计

9.5.1 什么是数据管理部分

永久对象(persistent object)是需要长期存储的对象，数据管理部分负责存储和检索永久对象。它还要封装对这些对象的查询和检索机制。

9.5.2 数据库和数据库管理系统

(1)关系数据库：是采用关系模型的数据库。

(2)面向对象数据库：是采用面向对象模型的数据库。

9.5.3 如何设计数据管理部分

(1)针对关系数据库的数据存取设计

1)对永久类的存储设计

2)定义数据库表

3)对关系的存储设计

A. 对关联的存储设计

B. 对聚合的存储设计

C. 对聚合的存储设计。

4)设计数据管理部分的类并修改问题域部分

(2)针对面向对象数据库的数据存取设计

(3)针对文件的数据存取设计。

9.6 面向对象测试的概念

9.6.1 面向对象软件测试的问题

(1)封装

(2)继承

(3)测试组织

9.6.2 面向对象软件测试的参考过程模型

|  |  |
| --- | --- |
| 面向对象的分析测试 | |
| 面向对象的设计测试 | |
| 面向对象的程序测试 | 面向对象的单元测试 |
| 面向对象的集成测试 |
| 面向对象的系统测试 | |

图 面向对象测试的参考过程模型

9.7 面向对象测试方法

9.7.1 面向对象的分析与设计测试

通常以文档审查的方式进行。

9.7.2 面向对象的程序测试

主要是要对可执行代码进行测试。

1. 单元测试：主要是考虑对类和类族的测试。
2. 对操作的测试
3. 对类层次的测试：不变式边界测试、模态类的测试、非模态类的测试。
4. 对类树的测试

类树是指一般类和特殊类形成的类结构。

1. 多态操作的调用；
2. 多态消息的测试；
3. 展平测试。
4. 类集成测试
5. 协作集成：就是集成测试时针对系统完成的特定功能，将相互协作的类集成在一起进行测试；
6. 基于事件(或消息)的集成；
7. 基于使用的集成；
8. 客户机/服务器的集成；
9. 分布式集成；

9.7.3 面向对象的系统测试

(1)功能测试：是用于确认软件系统是否满足了软件需求。

(2)其它。

还有可能进行性能测试、安装测试、易用性测试、兼容性测试和文档测试等。

**第4部分 软件生存期模型和软件体系结构**

1. **软件生存期模型**

也称为软件过程模型，是从软件项目需求定义直至软件运行维护为止。

10.1 软件过程框架的内容

(1)过程框架：定义了若干框架活动，这些活动适用于所有软件项目；

(2)普适应活动(Umbrella Activities)：如软件质量保证，软件配置管理和测量；

(3)框架活动：框架中的每一个活动都由一组软件工程动作组成，每一个动作又包含一个任务集合；

(4)任务集合：每一个集合由软件工程工作任务、项目里程碑、软件工程产品以及质量保证点组成。

10.1.2 通用过程框架

由5部分组成：沟通、策划、建模、构造和部署。

10.1.3 典型的普适应活动

包括以下几个方面：

1. 软件项目跟踪和控制；
2. 风险管理；
3. 软件质量保证；
4. 正式技术评审；
5. 测量；
6. 软件配置管理；
7. 可复用管理；
8. 工作产品的准备和生产。

10.2 传统软件过程模型

10.2.1 瀑布模型

特点如下：

1. 阶段间具有顺序性和依赖性；
2. 推迟实现的观点；
3. 质量保证的观点；

10.2.2 快速原型模型

快速原型是快速建立起来的可以在计算机上运行的程序。

10.2.3 增量模型

也称为渐进模型，把软件产品作为一系列的增量构件来设计、编码、集成和测试。

10.2.4 螺旋模型

最初是由Boehm于1988年提出来的，基本思想是：使用原型以及其它方法来尽量降低风险。

螺线上的每一个循环可以划分为4个象限，分别表达了4个方面的活动：目标设定、风险估计与弱化、开发与验证和计划。

10.2.5 喷泉模型

软件过程各个阶段之间的迭代或一个阶段内各个工作步骤之间的迭代。

10.3 现代软件过程模型

10.3.1 基于构件的开发模型

基于构件的软件工程(CBSE)是强调使用可复用的软件“构件”来设计和构造基于计算机的系统的过程。

典型的构件模型有3种：OMG/CORBA、Microsoft COM、Sun JavaBean构件。

10.3.2 形式化方法模型

10.3.3 面向方面的软件开发

如果某个关注点涉及系统多个方面的功能、特性和信息时，这些关注点通常称为横切关注点(crosscutting concerns)。方面的需求(aspectual requirements)定义了那些对整个软件体系结构产生影响的横切关注点。面向方面的软件开发(AOSD)通常称为面向方面编程(AOP)。

10.3.4 统一过程：UML

(1)统一过程的工作流：需求工作流、分析工作流、设计工作流、实现工作流和测试工作流。

(2) 统一过程的阶段：初始阶段、细化阶段、构造阶段和移交阶段。

10.3.5 敏捷过程模型

是一种新的软件开发管理模式，用以代替以文档驱动开发的瀑布开发模式。也称为轻量级开发方法。它强调以下方面：

1. 以人为本，注重编程中人的自我特长发挥；
2. 强调软件开发的产品是软件，而不是文档；
3. 客户与开发者的关系是协作，而不是合约；
4. 设计周密是为了最终软件的质量，但不表明设计比实现更重要，要适应客户需求的不断变化，设计也要不断跟进，所以设计不能“闭门造车”。
5. 极限编程

XP使用面向对象方法作为推荐的开发范型。包含了策划、设计、编码和测试4个框架活动规则和实践。

1. 自适应软件开发(ASD)

其生命周期包括思考、协作和学习3个阶段。

ASD团队通过3种方式学习：焦点组、正式的技术评审和事后剖析。

1. **软件体系结构**

11.1 软件体系结构的基本概念

11.1.1 什么是体系结构

体系结构设计是一系列决策和基本原理的集合，这些决策的目标在于开发高效的软件体系结构。

软件体系结构是具有一定形式的结构化元素，即构件的集合，包括处理构件、数据构件和连接构件。处理构件负责对数据进行加工，数据构件是被加工的信息，连接构件把体系结构的不同部分组合连接起来。

11.1.2 体系结构模式、风格和框架的概念

1. 模式

每个模式都描述了一个在我们的环境中不断出现的问题及该问题解决方案的核心。划分为3类：

1. 体系结构模式(architectural pattern)

表达了软件系统的基本结构组织形式或者结构方案，包括一组预定义的子系统，规定了这些子系统的责任，同时还提供了用于组织和管理这些子系统的规则和向导。如OSI参考模型。

1. 设计模式(design pattern)

为软件系统的子系统、构件或构件之间的关系提供一个精炼之后的解决方案，描述了在特定环境下，用于解决通用软件设计问题的构件以及这些构件相互通信时的各种结构。

1. 惯用法(idiom)

是与编程语言相关的低级模式，描述如何实现构件的某些功能，或者利用编程语言的特性来实现构件内部要素之间的通信功能。

2. 风格

体系结构风格定义了一个系统家族，即一个体系结构定义一个词汇表和一组约束。

3. 框架

是特定应用领域问题的体系结构模式，定义了基本构成单元和关系后，开发者可以集中精力解决业务逻辑问题。

11.1.3 体系结构的重要作用

以下几方面：

1. 其表示有助于风险承担者(项目干系人)进行交流；
2. 突出了早期设计决策；
3. 软件体系结构是可传递和可复用的模型。

11.2 典型的体系结构风格

11.2.1 数据流风格

管道/过滤器、批处理序列都属于数据流风格。

11.2.2 调用-返回风格

3种子风格：

1. 主程序/子程序风格
2. 面向对象风格：对象管理体系结构OMA、公共对象请求代理体系结构
3. 层次结构

11.2.3 仓库风格

数据库系统、超文本系统和黑板系统都属于仓库风格。

黑板系统由3部分组成：知识源、黑板数据结构和控制。

11.3 特定领域的软件体系结构

11.3.1 类属模型(generic model)

是从许多实际系统中抽象出来的一般模型，它封装了这些系统的主要特性。

11.3.2 参考模型

源于对应用领域的研究，它描述了一个理想化的包含了系统应具有的所有特征的软件体系结构。

11.4 分布式系统结构

11.4.1 多处理器体系结构

11.4.2 C/S体系结构：服务器、客户机和网络

两层C/S体系结构有两种形态：瘦客户机模型和胖客户机模型。

三层C/S体系结构将整个系统分为表示层、应用逻辑层和数据层。

B/S体系结构。

11.4.3 分布式对象体系结构

分布式对象的实质是在分布式异构环境下建立应用系统框架和对象构件，它将应用服务分割成具有完整逻辑含义的独立子模块(构件)。

当前主流的分布式对象技术规范有OMG的CORBA、微软公司的.NET和SUN公司的J2EE。

11.4.4 代理

代理可以用于构建带有隔离组件的分布式软件系统，该软件通过远程服务调用进行交互。

对象代理请求(ORB)有4个对象接口：对象服务、公共设施、领域接口和应用接口。

11.5 体系结构框架

11.5.1 模型-视图-控制器

即MVC框架，一个交互式应用系统由模型、视图和控制器3个部件组成。

模型对象：独立于外在显示内容和形式，代表应用领域中的业务实体和业务规则，是整个模型的核心。

视图对象：代表GUI对象，并且以用户需要的格式表示模型状态，是交互系统与外界的接口。

控制器对象：代表鼠标和键盘事件。

11.5.2 J2EE体系结构框架

是分层结构，中间的3层(表示层、业务层、集成层)包含应用程序构件，客户层和资源层处于应用程序的外围。

客户层：用户通过客户层与系统交互；

资源层：可以是企业数据库，电子商务解决方案中的外部企业系统，或者是外部SOA服务；

表示层：也称为Web层或服务器端表示层，用户通过表示层来访问应用程序；

业务层：包含表示层中的控制器构件没有实现的一部分应用逻辑；

集成层：负责建立和维护与数据源的连接。

11.5.3 PCMEF和PCBMER框架

1. PCMEF框架

表示-控制-中介者-实体-基础(PCMEF)是一个垂直层次的分层体系结构框架。包括四层：表示层、控制层、领域层和基础层。

表示层：包含定义GUI对象的类；

控制层：处理表示层的请求；

领域层：其实体包处理控制请求；

基础层：负责与数据库及Web服务的所有通信。

2. PCBMER框架

表示-控制器-Bean-中介者-实体-资源(PCBMER)。

Bean层：表示那些预先确定要呈现在用户界面上的数据类和值对象；

表示层：表示屏幕以及呈现Bean对象的UI对象；

控制器层：表示应用逻辑；

实体层：响应控制器和中介者；

中介者层：建立了充当实体类和资源类媒介的通信管道；

资源层：负责所有与外部持久数据资源(数据库、Web服务等)的通信。

11.6 体系结构建模

体系结构设计描述了建立计算机系统所需要的数据结构和程序构件。

一个好的体系结构设计要求为：

1. 软件模块的分层；
2. 编程标准的执行。

体系结构设计是一种管理模块依赖性的实践，它使用一种主动方法来管理软件中的依赖性。

对已实现软件的依赖性进行测量和管理的方法称为被动方法。

11.6.1 类及其依赖性

1. 类

2. 继承依赖性：多态继承、无多态继承、扩展继承和约束继承。

3. 交互依赖性

也称为方法依赖性，是通过消息连接产生的。

11.6.2 接口及其依赖性

1. 接口：是不可直接实例化的特殊集合的声明；

2. 实现依赖性：每一个类及该类实现的接口之间的依赖性；

3. 使用依赖性：通过类(接口)和它所需接口之间的依赖关系来说明需求接口。

11.6.3 包及其依赖性

1. 包：

有时也可将之称为层或子系统，是表示组织类的一种方式，用于划分应用程序的逻辑模型。

2. 包的依赖性

来自两个包中类之间的依赖性。

11.6.4 构件及其依赖性

传统的构件也称为模块，它承担如下3个重要角色之一：控制构件、问题域构件和基础设施构件。

为了支持复用，软件构件应具有以下特性：

1. 独立部署单元；
2. 作为第三方的组装单元；
3. 一个构件不能有任何可见状态；

软件构件是一种组装单元，它具有规范接口规格说明和显示的语境依赖，软件构件可以被独立部署，并被第三方任意组装。

11.6.5 结点和部署图

在UML中，分布式物理体系结构或系统任何其它体系结构都被描述为部署图，部署图中的计算资源(运行时的物理对象)被称为结点。将结点形象化地描述为立方体。

**第5部分 软件维护与软件管理**

1. **软件维护**

12.1 软件维护的概念

12.1.1 软件维护的定义

称在软件运行/维护阶段对软件产品所进行的修改就是所谓的维护。分为以下四种类型：

1. 改正性维护：诊断和改正错误的过程；
2. 适应性维护：为了使软件适应外部环境或数据环境可能发生的变化，而修改软件的过程称为适应性维护；
3. 完善性维护：

修改或再开发软件，以扩充软件功能、增强软件性能、改进加工效率、提高软件的可维护性，这种情况下进行的维护活动称为完善性维护。

1. 预防性维护

为了提高软件的可维护性、可靠性等，为今后的进一步改进软件打下良好的基础。

11.1.2 影响维护工作量的因素：系统规模、程序设计语言、系统年龄大小、数据库技术的应用水平、所采用的软件开发技术及软件开发工程化的程度，其它。

11.1.3 软件维护的策略

1. 改正性维护

2. 适应性维护

3. 完善性维护。

12.2 软件维护活动

12.2.1 软件维护申请报告

所有软件维护申请应按规定的方式提出。软件维护组织通常提供维护申请报告(maintenance request form, MRF)，或称软件问题报告，由申请维护的用户填写。

内部人员填写软件修改报告(software change report, SCR)。

12.2.2 软件维护工作流程

第1步是先确认维护要求。

“救火式”的紧急维护，是指如果发生的错误非常严重，不马上解决可能会导致重大事故，这样就必须紧急修改。

12.2.3 维护档案记录

12.2.4 维护评价：7种度量值

12.3 程序修改的步骤和修改的副作用

12.3.1 分析和理解程序

采用如下几种方法：

1. 分析程序结构图
2. 数据跟踪
3. 控制跟踪
4. 在分析过程中，应充分阅读和使用源程序清单和文档，分析现有文档的合理性
5. 充分使用由编译程序或汇编程序提供的交叉引用表、符号表，以及其它有用的信息
6. 如有可能，争取参加开发工作。

12.3.2 修改程序

1. 设计程序的修改计划

修改计划的内容包括以下几项：规格说明信息、维护资源、人员、提供。

采用自顶向下的方法，在理解程序的基础上做如下工作：

1. 研究程序的各个模块、模块的接口及数据库，从全局的观点提出修改计划；
2. 依次把要修改的、以及那些受修改影响的模块和数据结构分离出来；
3. 详细地分析要修改的，以及那些受变更影响的模块和数据结构的内部细节，设计修改计划，标明新逻辑以及要改动的现有逻辑；
4. 向用户提供回避措施；

2. 修改代码，以适应变化

12.3.3 修改程序的副作用以及其控制

所谓副作用是指因修改软件而造成的错误及其它不希望发生的情况，有以下3种副作用：

1. 修改代码的副作用
2. 修改数据的副作用
3. 修改文档的副作用

12.3.4 重新验证程序

1. 静态确认

2. 确认测试

(1)确认测试程序)

先对修改部分进行测试，然后隔离修改部分，测试程序的未修改部分，最后再把它们集成起来进行测试，这种测试称为回归测试。

(2)准备标准的测试用例

(3)充分利用软件工具帮助重新验证过程

(4)在重新确认过程中，需邀请用户参加

3. 维护后的验收

12.4 软件的可维护性

12.4.1软件可维护性的定义

所谓软件可维护性，是指纠正软件系统出现的错误和缺陷，以及为满足新的要求进行修改、扩充或压缩的容易程度。

可维护性、可使用性、可靠性是衡量软件质量的几个主要质量特性。

12.4.2 可维护性的度量

1. 可理解性

表明人们通过阅读源代码和相关文档，了解程序功能及其如何运行的容易程度。一个可理解的程序主要应具备以下一些特性：模块化，风格一致性，不使用令人捉摸不定或含糊不清的代码，使用有意义的数据名和过程名、结构化、完整性等。

2. 可靠性

表明一个程序按照用户的要求和设计目标，在给定的一段时间内正确执行的概率。度量的标准主要有：平均失效间隔时间(MTTF)，平均修复时间(MTTR)，有效性A()。

度量可靠性的方法主要有如下两类：根据程序错误统计数字，进行可靠性预测；根据程序复杂性，预测软件可靠性。

3. 可测试性

表明验证程序正确性的容易程度。

1. 可修改性

表明程序容易修改的程度。设C是程序中各个模块的复杂性，n是必须修改的模块数，A是要修改的模块的复杂性，则要修改的难度D由下式计算：，若，则说明该程序修改困难。

1. 可移植性

表明程序转移到一个新的计算环境的可能性大小。或者它表明程序可以容易地、有效地在各种各样的计算环境中运行的容易程度。

1. 效率

表明一个程序能执行预定功能而又不浪费机器资源的程度。这些机器资源包括：内存容量、外存容量、通道容量和执行时间。

1. 可使用性

从用户的观点出发，把可使用性定义为程序方便、实用和易于使用的程度。一个可使用的程序应是易于使用的、能允许用户出错和改变，并尽可能不使用户陷入混乱状态的程序。

1. 其它间接定量度量可维护性的方法

12.5 提高可维护性的方法

12.5.1 建立明确的软件质量目标和优先级

12.5.2 使用提高软件质量的技术和工具

1. 模块化

2. 结构化程序设计

3. 使用结构化程序设计技术，提高现有系统的可维护性

(1)采用备用件的方法；

(2)采用自动重建结构和重新格式化点的工具(结构更新技术)；

(3)改进现有程序的不完善的文档；

(4)使用结构化程序设计方法实现新的子系统；

(5)采用结构化小组。

12.5.3 质量保证审查

1. 在检查点进行审查

2. 验收检查

是一个特殊的检查点检查，是交付使用前的最后一次检查，是软件投入运行之前保证可维护性的最后机会。

验收检查必须遵循的最小验收标准：

1. 需求和规范标准
2. 设计标准
3. 源代码标准
4. 文档标准

3. 周期性地维护审查

4. 对软件包进行检查

软件包是一种标准化了的、可为不同单位、不同用户使用的封装软件。

12.5.4 改进文档

程序文档是对程序总目标、程序各组成部分之间的关系、程序设计策略、程序实现过程的历史数据等的说明和补充。

历史文档有如下3种：

1. 系统开发日志：

它记录了项目的开发原则、开发目标、优先次序、选择某种设计方案的理由、决策策略、使用的测试技术和工具、每天出现的问题、计划的成功和是失败之处等。

1. 错误记载

它把出错的历史情况记录下来，对于预测今后可能发生的错误类型及出错频率有很大的帮助。

1. 系统维护日志

记录了在维护阶段有关系统修改和修改的目的的信息。包括修改的宗旨、修改的策略、存在的问题、问题所在的位置、解决问题的方法、修改要求和说明、注意事项、新版本说明等信息。

1. **软件项目管理**

13.1 软件项目管理概述

13.1.1 软件项目管理的目标

包括以下几个方面：

1. 达到项目预期的软件产品功能和性能要求，使用户认为这样的软件产品正是自己所期待的；
2. 时限要求；
3. 项目开销限制在预算之内。

13.1.2 软件项目管理涉及的几个方面

人员、产品、过程和项目(4P)。

1. 人员管理：共利益者、团队负责人和团队集体；
2. 产品管理
3. 过程管理
4. 项目管理

其任务是如何利用已有的资源，组织实施既定的项目，提交给用户适合的产品。

主要工作分为3类：计划及计划管理，资源管理和成果要求管理。

13.2 项目估算

13.2.1 项目策划和项目估算

项目策划是在项目开始初期阶段点的重要工作，其主要目标是得到项目计划，或者说计划是策划的结果。

项目策划中需要开展的活动有如下几项：

1. 确认并分析项目的特征；
2. 选择项目将遵循的生存期模型，确定各阶段的任务；
3. 确定应得到的阶段性产品以及最终产品；
4. 开展项目估算；
5. 制订项目进度计划；
6. 对项目风险进行分析；
7. 制订项目计划。

3个量的估算：规模、工作量和成本。

进行成本分析后就能够得到自己的生产率数值和人工价格：

* 生产率是平均每个人月完成的源程序行数，可记为KLOC/人月或FP/人月；
* 人工价为每人月的价值。

工作量=规模/生产率，成本=工作量X人工价

13.2.2 软件规模估算的功能点方法(function point)

简称FP方法，该方法克服了项目开始时无法得知源程序行数的实际困难，从软件产品的功能度(functionality)出发估算出软件产品的规模。

1. 功能度

建立了应用系统边界的概念，有5种类型的功能。

1. 外部输入

处理那些进入应用系统边界的数据或是控制信息。

1. 外部输出

处理离开应用系统边界的数据或是控制信息。

1. 内部逻辑文件

是用户可识别的逻辑相关数据或控制信息组，它可在应用系统边界之内使用。

1. 外部接口文件

是用户可识别的逻辑相关数据或控制信息构成的集合，该控制信息为应用系统所使用却被另一应用系统所支持。

1. 外部查询

是唯一的输入/输出组合，它为实现即时输出引起所需数据的检索。

1. 功能复杂性

将其分为简单的、中等的和复杂的3个等级。

1. 未调节功能点



其中是第i类功能度和第j级复杂性的影响参数；是第i类功能度和第j级复杂度功能点的个数。

1. 调节因子
2. 影响因子：共14个类型
3. 影响级：分为6级

复杂度调节因子为CAF为：。

1. 交付功能点：
2. 交付功能点与软件规模

软件的规模可用交付的源代码行数(DLOC)来表示。

1. 功能点方法的优点
2. DFP只与由规格说明得到的信息相关；
3. 与实现软件的语言无关。
4. 功能点方法的不足之处
5. 针对需求规格说明进行分析时，主观因素难以完全排除；
6. 非数据处理问题，如实时软件、系统软件、科学计算软件等功能点的上述计算方法并不适用；
7. DFP的计算目前尚不能借助工具自动完成。

13.2.3 软件开发成本估算

1. 专家判定-Delphi方法

(1)组织者发给每位专家一份软件系统的规格说明书和一张记录估算值的表格，请他们进行估算；

(2)专家详细研究软件规格说明书的内容，然后组织者召集小组会议，在会上，专家们与组织者一起对估算问题进行讨论；

(3)各位专家对该软件提出3个软件规模的估算值，即： (最小源代码行数)， (最小可能的源代码行数)，(最多源代码行数)。

(4)组织者对各位专家在表中填写的估算值进行综合和分类；

(5)组织者召集会议，请专家们对其估算值有很大差异之处进行讨论；

(6)在综合专家估算结果的基础之上，组织专家再次无记名地填写表格。

2. COCOMO模型

软件估算模型层次结构，称为构造式成本模型COCOMO。

1. 3种类型的软件：固有型项目、嵌入型项目和半独立型项目。
2. COCOMO的3级模型
3. 基本COCOMO模型
4. 中级COCOMO模型
5. 高级COCOMO模型

3. COCOMO II模型

其估算过程反映了任一开发项目的3个主要阶段：

1. 在开发的第一阶段，通常要建立一个原型来解决包括用户界面、软件与系统间的交互作用、软件的性能或是技术成熟性等高风险问题；
2. 在第二阶段开发工作进入了初步设计，设计人员要比较和选择软件的整体结构和运行方案；
3. 第三阶段是结构设计之后的工作。

其基本模型为：

式中：前者为基于规模的初始估算，后者为调节向量。

13.3 风险管理

13.3.1 什么是软件风险

1. 软件风险

把软件工程过程中可能出现的那些影响软件目标实现，或是可能造成重大损失的事件称为软件风险。

2. 风险的特点

(1)可能发生的事件；

(2)会给项目带来损失的事件；

(3)可能对其进行干预，以期减少损失。

3. 风险的分类

(1)依据危害性：成本风险、绩效风险和进度风险。

(2)从风险涉及的范围上考虑：项目风险、技术风险和商业风险(市场风险、策略风险、管理风险和预算风险)。

(3)其它风险：已知风险、可预知风险和不可预知风险。

13.3.2 风险管理的任务

1. 风险管理点的目标和策略

(1)目标：识别风险、采取措施

(2)策略：回避风险、转移风险和承受风险。

2. 风险管理活动

围绕风险评估和风险控制开展活动。

13.3.3 风险评估

1. 风险识别

检查单(checklist)是识别风险的有力工具。

2. 风险分析

任务是分析每个风险可能造成的影响，给出风险大小的量值。

3. 风险排序

风险高低是以风险显露(risk exposure)造成损失的大小来衡量。注意两个重要因素：风险概率和风险影响。

风险排序的步骤：4个。

13.3.4 风险控制

1. 风险管理策划

风险控制从风险管理策划开始，风险管理策划则是要针对每个已经识别和分析认为应该受控的风险制定风险管理计划。

2. 风险化解

是要实际消除风险或减轻风险。

把风险显露的损失差和风险减轻成本的比例称为风险杠杆(risk leverage)。

3. 风险监控

(1)随时监控的必要性

(2)跟踪监控

13.3.5 做好风险监控的必要性：12个。

13.4 进度管理

13.4.1 进度控制问题

1. 值得重视的现象

软件项目延期提交的原因：8个。

2. 制订项目进度安排的条件

有以下7条：

1. 项目分解：工作分解结构WBS；
2. 确定各部分之间的相互关系；
3. 时间分配；
4. 确定投入的工作量；
5. 确定人员的责任：任务责任矩阵；
6. 规定工作成果；
7. 规定里程碑。

13.4.2 甘特图(Gantt chart)

是表示工作进度计划以及工作实际进度情况最为简明的图式方法。

13.4.3 时标网状图(time scalar network)

13.4.4 PERT图

活动赋值与复审方法(PERT)也称网络图方法，也称关键路径法CPM。

13.5 需求管理

13.5.1 系统需求和软件需求

1. 系统和系统需求分配

(1)系统：基于计算机的系统或计算机控制的系统；

(2)系统需求分配

分配给软件的系统需求和分配的需求。

2. 软件需求

(1)软件需求的定义

按IEEE-STD-610标准的定义，软件需求是用户为解决某个问题或是为实现某个目标，要求软件必须满足的条件或能力。

软件需求的3个层次：业务需求、用户需求、功能和非功能需求(过程需求和外部需求)。

(2)质量功能展开QFD

是将客户的要求(needs)转化为软件技术需求的质量管理方法，于1970年在日本推出，其目标在于是软件工程过程最大程度地让客户满意。

分为3类：常规需求、期望需求和兴奋需求。

13.5.2 需求工程

是系统工程或软件工程中解决需求问题的一个崭新领域，其目标在于使工程得到的产品能够准确、真实地体现客户的需求，令客户满意。包括2个方面：

1. 需求开发

是在开发人员和客户双方密切配合下完成的，任务是得到需求规格说明。需求开发工作包括以下4个方面：获取需求、分析需求、定义需求和验证需求。

1. 需求管理

是要解决在需求开发之后，也即得到需求规格说明之后，在后继的开发过程中用户提出的新的需求或变更了原定的需求，在这种情况下开发工作应如何处理。

需求管理会涉及：变更控制、需求状态跟踪、需求跟踪和需求文档版本控制。

13.5.3 需求变更

1. 需求变更难于完全避免

2. 需求变更原因分析

(1)单纯的用户原因；

(2)市场形势变化引起的需求变更；

(3)系统因素；

(4)工作环境因素；

(5)需求开发工作缺陷。

3. 需求变更对软件开发工作的影响

(1)使得变更前的开发工作和成果失效；

(2)使得返工成为不得不采取的对策；

(3)势必带来软件开发计划的相应变更。

4. 需求变更失控可能导致的后果

得到的产品并不是用户所需要的产品。

1. 降低需求变更风险的策略
2. 在需求开发工作中要与客户充分交流；
3. 与用户共同确定需求，吸引用户参与需求开发十分重要，并且开发人员一定要认真听取客户意见；
4. 开发组织和用户双方签署的项目开发合同中应包括开发中对出现需求变更的应对条款；
5. 如果项目本身具有需求不易确定的特点，在项目启动时最好采用快速原型方法(rapid prototyping)或螺旋模型(spiral model)，以便在确认需求的基础上开发产品；
6. 在项目开始时，如估计到需求可能有变，则可在开发计划中适当留有余地，以防变更需求造成的被动；
7. 严格实施变更控制，使产品质量不致因需求变更受到影响。

13.5.4 需求变更控制

1. 需求变更控制要求

(1) 变更控制的策略：6个

(2) 需求变更影响的控制

(3) 变更控制的步骤：提出变更请求，审理变更请求、进行变更影响估计，批准变更请求，取得用户的认可，修订项目计划，实施变更，验证变更。

2. 需求变更控制实施

(1) 需求变更请求

(2) 需求控制流

13.5.5 可追溯性管理

1. 需求可追溯性与需求变更控制

2. 可追溯性管理的目标

使每一项需求均能追溯到：对应的设计、实现该项需求的代码以及测试此项实现的用例。

1. 两类不同的追溯：向前追溯和向后追溯。
2. 可追溯矩阵。

13.6 配置管理

13.6.1 什么是软件配置管理

其目的是为某个过程或某个项目的软件项建立和保持完整性，以便相关方能够使用它。要开展的活动包括：配置标识、配置控制、配置状态报告、配置评价以及发布管理、交互等。

把软件配置管理的对象称为软件配置项(software configuration item)，它包括：与合同、过程、计划和产品相关的文档及数据，源代码、目标代码和可执行代码，相关的产品。

软件配置管理的主要任务是：

1. 制订软件配置管理计划；
2. 实施变更管理，防止项目进行中因变更导致的混乱；
3. 实施版本管理和发布管理。

13.6.2 软件配置标识

制定适当的命名规则是配置标识的第一步，命名要求具有：唯一性，可追溯性。

通常需标识两类型对象：基本对象和复合对象。基本对象是由软件工程师在分析、设计、编码和测试时所建立的“文档单元”。复合对象是基本对象和其它复合对象的一个集合。

每个对象可用一组信息来唯一地标识，这组信息包括：(名字、描述、一组“资源”、“实现”)。

13.6.3 变更管理

1. 变更不可避免

变更管理的任务如下：

1. 分析变更，确定是否实施变更；
2. 记录变更信息，并追踪变更信息，
3. 确保变更在受控条件下进行。

2. 配置数据库

可分为开发库、受控库和产品库3类：

3. 基线和变更控制

基线(baseline)是软件生存期各开发阶段末尾的特定点，也被称为里程碑(milestone)。它的作用是把各阶段的开发工作划分得更加明确。

4. 变更管理过程

*提交变更请求表(CRF)*

*分析变更请求*

*如果 变更请求可接受 则*

*估计变更如何实现*

*估算变更成本*

*将CRF送交变更控制委员会(CCB)*

*如果 变更获准 则*

*重复*

*实施变更*

*记录变更*

*将变更的软件提交质量保证人员审查*

*直到 软件质量达到要求*

*由配置管理人员生成系统的新版本*

*否则 拒绝变更请求*

*否则 拒绝变更请求*

软件的变更通常有两类不同的情况：

1. 为改正小错误需要的变更；
2. 为了增加或删除某些功能，或者为了改变完成某个功能的方法而需要的变更。

13.6.4 版本控制

1. 版本管理和发行管理

(1) 版本管理(version management)

是对系统不同版本进行标识和跟踪的过程。

(2) 系统发行(system release)

是分配给客户的一个版本，每次系统发行都应有新的功能或是针对不同的系统运行环境。

2. 版本标识(version identification)

是由版本的命名规则决定的。

1. 号码顺序型版本标识
2. 符号命名版本标识
3. 属性版本标识

3. 发行管理

13.6.5 系统建立

是将系统的组件组合成完整的程序以执行某一特定目标配置的过程。

13.6.6 配置审核(configuration audit)

其目的是要证实整个软件生存期中各项产品在技术上和管理上的完整性。

13.6.7 配置状态报告(configuration status accounting)

任务：为了清除、及时地记载软件配置的变化，不致于到后期造成贻误，需要对开发的过程做出系统的记录，以反映开发活动的历史情况。

它提高了所有开发人员之间的通信能力，避免了可能出现的不一致和冲突。

1. **软件工程标准及软件文档**

14.1 标准的意义

是人们为在一定范围内获得最佳秩序，经协商一致制定，并由公认机构批准，共同使用和重复使用的一种规范性文件。

所谓的标准化是是指围绕着标准的制定与贯彻实施等方面的工作开展的一系列活动。

14.2 软件标准化的意义

14.3 标准的分类和分级

大致可分为推荐性标准和指导性技术文件两类。

按照标准的制定机构和适用范围，分为：

1. 国际标准：ISO、IEC(国际电工委员会)。
2. 国家标准：GB标准，ANSI标准(美国国家标准协会)，FIPS(NBS)(美国商务部国家标准局联邦信息处理标准)，BS(英国国家标准)，DIN(德国标准化协会)，JIS(日本工业标准)。
3. 行业标准
4. SJ：我国信息行业标准；
5. IEEE软件标准分技术委员会(SESS)；
6. GJB：中华人民共和国国家军用标准；
7. DOD-STD：美国国防部标准；
8. MIL-S：美国军用标准。
9. 地区标准
10. 企业标准(或企业规范)
11. 项目标准(或项目规范)

将国际标准转换为我国标准有三种情况：等同采用(IDT)、等效采用(EQV)、参照使用(NEQ)。

14.4 软件工程标准的制定与实施

建议、开发、咨询、审批、公布、培训、实施、审核和修订。

14.5 软件组织内的标准化工作

14.6 软件文档的作用和分类

(1) 什么是文档(document)

是指某种数据媒体和其中所记录的数据。

(2) 软件文档的作用

1) 提高软件开发过程的能见度；

2) 提高开发效率；

3) 作为开发人员在一定阶段的工作成果和结束标志；

4) 记录开发过程的相关信息；

5) 提供对软件的运行、维护和培训的相关信息；

6) 便于潜在用户了解软件的功能、性能等各项指标。

(3) 文档的分类

大致分为两类：工作表格和文档或文件。

按照文档产生和使用的范围，可分为3类：

1. 开发文档
2. 管理文档
3. 用户文档
4. 文档的内容

GB/T8567-2006《计算机软件文档编制规范》中给出了25种软件文档的编制要点。一般的工程项目利用其中的17种基本文档。

14.7 软件基本文档的内容要求

对17种基本文档的内容要求做简要的说明：

1. 可行性分析(研究)报告：该报告是项目初期所作项目策划的结论，报告应注重分析项目的要求；
2. 软件(或项目)开发计划：本计划描述的是软件开发人员要实施的开发工作计划；
3. 软件需求规格说明：该说明描述的是对软件配置项的需求，其目的在于每项需求均在项目实施中得到满足；
4. 接口需求规格说明：本文档描述为实现一个或多个系统、子系统、硬件配置项、软件配置项、手工操作、其它系统部件之间的一个或多个接口而施加在这些实体上的需求；
5. 系统/子系统设计(结构设计)说明：本文档描述的是系统或子系统的系统级或子系统级设计与体系结构设计；
6. 软件(结构)设计说明：本文档说明应描述软件配置项的设计；
7. 接口设计说明：该文档描述的是一个或多个系统或子系统，硬件配置项、软件配置项、人工操作或其它系统部件的接口特性；
8. 数据库(顶层)设计说明：该文档提到的数据库是指存储在一个或多个文件中的相关数据集合，其数据可由用户或计算机程序通过数据库管理系统访问；
9. 用户手册；
10. 测试计划；
11. 测试报告；
12. 软件配置管理计划；
13. 软件质量保证计划；
14. 开发进度月报；
15. 项目开发总结报告；
16. 软件产品规格说明；
17. 软件版本说明。

上述所有的17个文档，最终要向软件管理部门，或向用户回答下列问题：What、Where、When、Who、How、Why。

14.8 对文档编制的质量要求

体现在以下方面：

1. 针对性：文档编制以前应分清读者对象；
2. 精确性：文档的行文应当十分确切，不能出现多义性的描述；
3. 清晰性：文档编写应力求简明；
4. 完整性：任何一个文档都应当是完整的、独立的，它应自成体系；
5. 灵活性：
6. 应根据具体的软件开发项目，决定编制的文档种类；
7. 当所开发的软件系统非常大时，一种文档可以分成几卷编写；
8. 文档的详细程度；
9. 对规范条款可以扩展，进一步细分；
10. 程序的设计表现形式；
11. 建立一些特殊的文档种类要求；
12. 对于文档的表现形式，没有规定或限制。
13. 可追溯性。

14.9 文档的管理和维护

(1) 软件开发小组应设一位文档保管员，负责集中保管本项目已有的文档的两套主文本；

(2) 软件开发小组的成员可根据工作需要在自己手中保存一些个人文档；

(3) 开发人员个人只保存主文本中与其工作相关的部分文档；

(4) 在新文档取代旧文档时，管理人员应及时注销旧文档；

(5) 项目开发结束时，文档管理人员应收回开发人员的个人文档；

(6) 在软件开发过程中，可能发现需要修改已完成的文档，按照：-提议-评议-审核-批准-实施-的步骤加以严格的控制。

软件产品(包括文档和程序)在开发的不同时期具有不同的组合，这个组合随着软件开发工作的进展而在不断变化，这就是软件配置的概念。

1. **软件过程和软件过程改进**

影响提高软件产品质量和提高软件项目生产率的主要因素：人员、技术与设备以及过程。

15.1 软件过程概述

1. 过程概念

分解为两类子过程：一类是直接子过程(也称为基本过程)，一类是间接子过程(也称为支持过程)。

2. 过程要素：输入、输出、活动或进一步被分解的任务或作业、资源、测量与验证、过程目标。

3. 过程思维(process thinking)：

用对过程的共同思考去考虑问题。

4. 软件过程

以过程思维来重新认识软件业存在的问题。

15.2 软件生存期过程国际标准

软件生存期过程指的是软件生存期中可能出现的过程。

1. 结构

该标准分为两部分：与系统相关的过程和软件特有过程。

与系统相关的过程包括4类：协议过程、组织的项目实施过程、项目过程和技术过程。

软件特有的过程包括3类过程：软件实现过程、软件支持过程、软件复用过程。

1. 过程简述

7类43个过程：

1. 协议过程类
2. 组织的项目实施过程类
3. 项目过程类
4. 技术过程类
5. 软件实现过程类
6. 软件支持过程类
7. 软件复用过程类

15.3 软件过程成熟度

15.3.1 什么是软件过程成熟度

1. 软件过程成熟度(software process maturity)的概念

是软件过程改进的一个重要概念，它是指一个特定软件过程得到清晰的定义、管理、测量、控制以及有效的程度。

2. 不成熟的软件过程和成熟的软件过程的对比

15.3.2 过程制度化

1. 过程认同与过程制度化

当一个规范化过程已经渗入组织的日常生活之中，过程的要求已经变成全体员工的自觉行动，得到大家的认同和坚持遵循时，过程便成为制度化的(process institutionalization)。

2. 过程文化(process culture)

过程文化是指人们的习惯和行为受到过程思维和过程管理原则的影响。

1. 过程基础设施(process infrastructure)

包含了组织管理基础设施和技术基础设施两个方面：

1. 组织管理基础设施

组织和管理基础设施包括建立、监控和推进过程活动的岗位及其职责。如软件工程过程组(SPEG)和软件过程改进组(PIT)。

1. 技术基础设施

是支持SPEG和PIT的技术平台、计算机设施和工具。

15.4 软件能力成熟度模型(CMM/CMMI)

15.4.1 CMM与SEI

1. 什么是CMM

是能力成熟度模型(Capability Maturity Model)。划分为5个等级：1级被认为是成熟度最低，5级则是成熟度最高。

CMM给出了从混乱的个别过程达到成熟的规范化过程的一个框架。

2. SEI的软件过程研究

(1) 是软件工程研究所(Software Engineering Institute)。该机构隶属于卡内基-梅隆大学，于1984年成立。

SEI的任务是在软件工程领域中努力提高依赖软件的系统质量，促进软件开发和维护的工程化管理，为军方服务。

(2) CMM项目的主要负责人是Watts Humphrey、Mark Paulk等。

(3) 包括以下一些领域：CMM，基于CMM的过程评估，软件过程定义，个人软件过程(PSP)，团队软件过程(TSP)，软件工程测量与分析。

15.4.2 CMM的演化

15.4.3 CMM族与CMMI

1. 基于CMM的模型

* P- CMM
* SA- CMM
* IPD- CMM
* SE- CMM
* SSE- CMM

2. CMMI

将已有的几个CMM模型结合在一起，使之构成“集成模型”。

15.4.4 CMMI 1.2简介

1. CMMI模型的两种表示

分级表示和连续表示。

2. 两种表示的对比

3. 成熟度等级与能力等级

(1) 成熟度等级：初始级(ML1)、已管理级(ML2)、已定义级(ML3)、已量化管理级(ML4)、优化级(ML5)。

(2) 能力等级：不完备级(CL0)、已实施级(CL1)、已管理级(CL2)、已定义级(CL3)、定量管理级(CL4)、优化级(CL5)。

4. 过程域(process area)

是CMMI为实施软件过程改进的组织提出的若干个值得重视的软件过程。

5. 两种目标、两种实践

(1) 模型部件(model components)：也称为过程域部件。

包含3个模型部件：必需的部件、期望的部件和资料性部件。

(2) 共用目标和共有实践(GP)：5个共用目标(generic goals，GG)。

(3) 共用目标和共有实践的要点：

* GG 1达到专用标准
* GG 2将已管理过程制度化
* GG 3将已定义过程制度化
* GG 4将已量化管理级制度化
* GG 5将优化过程制度化

(4) 制度化要求

涉及从2级到5级的每个等级。

1. 专用目标和专用实践

专用目标(SG)给出了满足某个过程域应该达到的一组特定目标；

专用实践(SP)则规定了为达到过程域的专用目标需要开展的一些活动或者说是一组最佳实践。

15.4.5 CMMI评估

1. 标准评估方法：SCAMPI

2. 评估原则

3. 评估实施

考虑以下问题：

* 确定评估的范围
* 选定评估实施的等级A、B或C
* 确定评估组成员
* 确定被评组织参加访谈的人员
* 确定评估要得到的结论形式
* 制订评估的约束条件
* 制订评估实施计划

15.5 软件过程改进

15.5.1软件过程改进的IDEAL模型

经历5个阶段：

1. 启动阶段(initiating)

主要工作包括：激励变更、确定变更范围、着手发动、建立基础设施。

1. 诊断阶段(diagnosing)

主要工作包括：表征现行的过程状态和期望的状态，提出过程改进的建议。

1. 建立阶段(establishing)

包括：安排优先顺序、确定改进途径和策划行动。

1. 行动阶段(acting)

包括：提出改进方案，试行解决方案，改进解决方案，正式实施解决方案。

1. 提高阶段(leveraging)

包括：分析和确认已实施方案，提出后续行动建议。

15.5.2软件过程改进框架

(1) 过程改进框架

对其4个方面给予说明：

1. 过程改进基础设施：组织管理基础设施和技术基础设施。
2. 过程改进路线图
3. 软件过程评估方法
4. 软件过程改进计划

(2) 软件过程改进循环

4个步骤构成的循环：评估、计划、改进、监控。

有经验表明，整个循环需要12~30个月的时间。

15.5.3 有效的软件过程

(1) 有效的软件过程应具备的条件

1) 过程得到遵循

2) 过程受到督促检查

3) 过程要有测量

4) 以过程要求为内容进行培训

5) 明确过程所有者

6) 管理者对过程的有效支持

7) 把对员工激励和过程目标的实现集合起来

8) 新员工接受过程培训

9) 员工对过程的意见受到鼓励、分析和引导

10) 过程得到技术的适当支持

(2) 建立使软件过程更为有效的机制

1) 明确过程的所有者

2) 组织培训

3) 对过程实施情况进行测量，收集过程实施的反馈意见

4) 吸收过程使用者的反馈意见

5) 吸收来自企业外部的意见

6) 实施过程的督促与检查