**《软件工程导论》期末知识点复习**

**前言**：软件工程知识点详解，是在。本书参考《软件工程导论》第六版，张海藩，红色的那本。带\*不重要了解一下即可,黑体重点部分,需记忆。  
<--more-->

**第一章 软件工程概论**

1. \*软件：是计算机程序、方法、规则、相关的文档以及运行计算机系统时所必需的数据的总和(狭义定义：软件＝程序+数据+文档)
2. \*软件的特性：软件是复杂的、软件是不可见的、软件是不断变化的和软件质量难以稳定。
3. \*软件的质量特性：功能性、可靠性、易用性、效率、维护性、可移植性。
4. 软件危机：指在计算机软件的开发和维护过程中所遇到的一系列严重问题。
5. **软件危机的主要表现：**

* 对软件开发成本和进度估计常常很不准确
* 用户对"已完成"的系统不满意的现象经常发生
* 软件产品的质量往往靠不住
* 软件常常是不可维护的
* 软件成本在计算机系统总成本所占的比例逐年上升

1. **软件危机产生的主要原因:**

* 软件日益复杂和庞大
* 软件开发管理困难和复杂
* 软件开发技术落后
* 生产方式落后
* 开发工具落后
* 软件开发费用不断增加

1. 软件危机如何解决：既要有技术措施（方法和工具），又要有必要的组织管理措施。
2. **软件工程**：是指导计算机软件开发和维护的一门工程学科。采用工程的概念、原理、技术和方法来开发与维护软件，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。
3. 软件工程以关注软件质量为目标，包括方法、过程、工具、范式四个要素。
4. **软件工程方法学**：把软件生命周期全过程中使用的一整套技术方法的集合成为方法学(也称范型paradigm),包括三个要素:方法,工具和过程.目前使用最广泛的是传统方法学和面向对象方法学

* 传统方法学(也称生命周期方法学或结构化范型):采用结构化技术(结构化分析,结构化技术,结构化实现)来完成软件开发的各项任务,并使用适当的软件工具或软件环境来支持结构化技术的运用...略太过了
* 面向对象方法学:有4个要点;它是一个多次反复迭代的演化的过程;特有的继承性和多态性进一步提高了面向对象软件的可重用性

1. **软件生命周期**

软件生命周期

* 问题定义:确定要解决的问题是什么(通过客户的访问调查,系统分析员写出问题的性质,工程目标和工程规模的书面报告,并得到客户的确认)
* 可行性研究:不是具体解决问题,而是研究问题的范围,探索问题是否值得去解,是否有可行的解决方法
* 需求分析:准确确定"为了解决这个问题,目标系统必须做什么",主要是确定目标系统必须具备哪些功能
* 总体设计:设计出目标系统的多种方案;设计程序的体系结构
* 详细设计:详细的设计每个模块,确定要实现模块功能所需要的算法和数据结构
* 编码和单元测试:写出正确的容易理解,容易维护的程序模块
* 综合测试:通过各种类型的测试(及相应的的调试)使软件达到预定的要求
* 软件维护:通过各种必要的维护活动使系统持久地满足用户的需要

1. 软件过程:指为了获得高质量软件所需完成一系列任务框架,它规定了完成各项任务的工作步骤;通常使用生命周期模型简洁地描述软件过程
2. **生命周期模型**:也称过程模型规定了把生命周期划分成哪些阶段及各个阶段的执行顺序
3. **瀑布模型**

①瀑布模型特点：

* 阶段间具有顺序性和依赖性:必须等前一阶段的工作完成后之后,才能开始后一段的工作;前一段的输出文档就是后一阶段的输入文档
* 推迟实现的观点:在编码之前设置了系统分析和系统设计的各个阶段,分析与设计阶段的基本任务规定,这两个阶段主要考虑目标系统的逻辑模型,不涉及物理实现
* 质量保证的观点:每个阶段必须完成规定的文档;每个阶段结束前都要对完成的文档进行评审,以便尽早发现问题

②瀑布模型适用：在开发的早期阶段软件需求被完整确定

③瀑布模型的优点: 可强迫开发人员采用规范的方法;严格规定了每个阶段必须提交的文档;要求每个阶段交出的所有产品都必须经过质量保证小组的仔细验证

④瀑布模型缺点：瀑布模型是由文档驱动;最终产品不能真正满足客户的需求

1. **快速原型模型**：首先建立一个反映用户主要需求的原型系统,让用户体验,之后提出意见,随之进行修改,再让用户体验...直至用户完全满意,据此写出规格说明文档
2. **增量模型**：也称渐增模型，融合了瀑布模型的基本成分（重复应用）和原型实现的迭代特征，该模型采用随着日程时间的进展而交错的线性序列，每一个线性序列产生软件的一个可发布的“增量”。

* 增量模型优点:人员分配灵活,刚开始不用投入大量人力资源；可先发布部分功能给客户，对客户起到镇静剂的作用；增量能够有计划地管理技术风险。
* 增量模型缺点:需要软件具备开放式的体系结构；容易退化为边做边改模型，从而使软件过程的控制失去整体性；增加系统内部的耦合复杂性。

1. **螺旋模型**：把它看作在每个阶段之前增加了风险分析的快速原型模型。
2. \*螺旋模型与增量模型的区别：（1）两者迭代层级不同：增量模型在活动级迭代；螺旋模型在过程级迭代；（2）两者需求分析的时间不同：增量模型常常是先做总体需求分析和设计，然后在编码和测试中逐个增量开发；螺旋模型在开发周期内采用简化瀑布模型或快速模型；(3)两者提交软件的方式不同：增量开发在上次增量的基础上提交新的一部分软件；螺旋模型每次迭代都提交一个新的完整的软件版本；(4)两者减少风险的方式不同：增量模型使用未成熟技术和经常的客户反馈等方法减少风险；螺旋模型中直接加进风险识别，风险分析、风险控制，计划性较强.
3. **喷泉模型**:典型的具有面向对象软件开发过程迭代和无缝的特性
4. **Rational 统一过程**(Rational Unified Process Rational，RUP): RUP核心：RUP核心是解决可操作性问题，帮助开发人员尽可能少地依赖那些“不可描述的经验”。RUP特点：用例驱动；以体系结构为中心（高内聚低耦合）；增量和迭代开发。
5. RUP最佳实践

* 迭代式开发:允许每次迭代过程中需求都可以有变化;采用可验证的方法来减少风险
* 管理需求:RUP采用用例分析来捕获需求,并由它们驱动设计和实现
* 使用基于构件的体系架构:使用现有的或新开发的构件定义体系结构的系统化方法,从而降低软件开发的复杂性,提高软件重用率
* 可视化建模:建立软件系统的可视化模型,提高管理复杂软件的能力,如UML
* 验证软件质量:软件质量评估贯穿整个开发过程,由全体成员参与
* 控制软件的变更:RUP描述了如何控制,跟踪和监控修改,以确保迭代开发的成功

1. RUP软件开发生命周期

①核心工作流 (纵轴代表核心工作流,横轴代表时间) 前6个为核心过程工作流, 后3个为核心支持工作流

* 业务建模:深入了解使用目标系统的机构及商务运作评估目标系统对使用它的机构的影响
* 需求:捕获客户的需求并且使开发人员和用户达成对需求描述的共识
* 分析与设计:把需求分析的结果转化为分析模型和设计模型
* 实现:把设计模型转化为实现结果
* 测试:检查各子系统的交互与集成,验证所有需求是否都被正确实现,识别,确认缺陷并确保在软件部署之前消除缺陷
* 部署:成功生成目标系统的可运行版本,并将软件移交给用户
* 配置与变更管理:跟踪并维护在软件过程中产生的所有制品的完整性和一致性
* 项目管理:提供项目管理框架,为软件开发制定计划,人员配备,执行和监控等方面的使用准则,并为风险管理提供框架
* 环境:向软件开发机构提供软件开发环境,包括过程管理和工具支持

②工作阶段

* 初始阶段:建立业务模型,定义最终产品视图,并确定项目的范围
* 精化阶段:设计并确定系统的体系结构,制定项目计划确定资源需求
* 构建阶段:开发出所有构件和应用程序,把它们集成客户需要的产品,并且详细地测试所有功能
* 移交阶段:把开发出的产品提交给用户使用

③RUP迭代式开发

**第二章 可行性研究**

1. **可行性研究的目的**不是为了解决问题，而是确定问题是否值得去解决
2. **可行性**：技术可行性、经济可行性、操作可行性、运行可行性、法律可行性。
3. **可行性研究过程**：

* 复查系统规模和目标
* 研究正在使用的系统
* 导出新系统的高层逻辑模型
* 进一步定义问题
* 导出和评价供选择的解法
* 推荐行动方针
* 草拟开发计划
* 书写文档提交审查

1. \*系统流程图：是概括地描绘物理系统的传统工具。用图形符号以黑盒子形式描绘组成系统的每个部件（程序，文档，数据库，人工过程等）。表达的是数据在系统各部件之间流动的情况

| **符号** | **名称** | **说明** |
| --- | --- | --- |
| 矩形 | 处理 | 能改变数据值或数据位置的加工或部件。如程序 、处理机、人工加工等都是处理 |
| 平行四边形 | 输入输出 | 表示输入或输出，是一个广义的不指名具体设备的符号 |
| 圆形 | 连接 | 指出转到图的另一部分或从图的另一部分转来，通常在同一页上 |
| 矩形下面 加个三角形 | 换页连接 | 指出转到另一页图上或另一页图转来 |
| 箭头 | 数据流 | 用来连接其他符号，指明数据流的方向 |

1. \*数据流图表示方法：实线表示数据流；虚线表示控制流；圆框代表处理数据的过程；矩形框表示产生与接收数据的对象；平行线表示数据存储区。
2. 数据字典定义组成：数据流；数据流分量（即数据元素）；数据存储；处理
3. 数据字典定义数据的方法（对数据自顶向下分解）：

| **符号** | **含义** |
| --- | --- |
| = | 等价于或定义为 |
| + | 和(连接两个分量) |
| [] | 或，多个用|隔开 |
| {} | 重复 |
| () | 可选 |
| 标识符 | 字母字符+字母数字串 |
| 字母数字串 | 0{字母或数字}7 |
| 字母或数字 | [字母字符|数字字符] |

1. **成本效益分析的方法及如何运算:**  
   第一步估计开发成本、运行费用和新系统将带来的经济效益。需从货币时间价值,投资回收期,纯收入,投资回收率  
   方法有三种：

* 代码行技术：软件成本=每行代码的平均成本\*估计的源代码总行数
* 任务分解技术：单本任务成本=任务所需人力估计值\*每人每月平均工资；软件开发项目总成本=每个单独任务成本估计总和
* 自动估计成本技术：采用自动估计成本的软件

**第三章 需求分析**

1. **需求分析的任务**

* 确定对系统的综合要求
* 分析系统的数据要求
* 导出系统的逻辑模型
* 修正系统开发计划

1. **分析建模与规格说明**  
   模型: 就是为了理解事物而对事物作出的一种抽象,是对事物的一种无歧义的书面描述。通常，模型由一组图形符号和组织这些符号的规则组成  
   需要建立的三种模型：数据模型、功能模型和行为模型

* 实体-联系图：描绘数据对象、数据对象的属性及数据对象之间的关系，用于建立数据模型
* 数据流图：描绘当数据在软件系统中流动和被处理的逻辑过程，是建立功能模型的基础
* 状态转换图：描绘了系统的状态及引起状态转换的事件，是建立行为模型的基础

1. **状态转换图符号含义及怎么画** P65，P67
2. 层次方框图：是用树形结构的一系列多层次的矩形框描绘数据的层次结构。最顶层矩形框：代表完整的数据结构；下面各层的矩形框代表数据的子集；最底层的矩形框代表实际数据元素

**第四章 形式化说明技术**

1. 按形式化程度分为三类：

* 非形式化，如用自然语言描述规格说明
* 半形式化，如用数据流图或实体-联系图建立模型
* 形式化，如描述系统性质是基于数学的技术

1. 非形式化的缺点

* 矛盾性：在需求规格说明书中对同一问题前后存在不同的描述
* 二义性：读者可以用不同的方式理解的陈述
* 含糊性：需求规格说明书对某一问题描述不清晰，不可理解
* 不完整性：需求规格说明书对某一问题只说明了局部，没说明整体
* 抽象层次混乱：指在非常抽象的陈述中混入了关于低层次的细节陈述

1. 形式化的优点：

* 能够简洁准确的描述物理现象、对象或动作的结果
* 在不同的软件工程活动之间平滑过渡
* 提供了高层确认的手段

1. 应用形式化准则

* 选用适当的表示方法
* 应该形式化，但不要过分形式化
* 应该估算成本
* 应该有形式化顾问随时提供咨询
* 不应该放弃传统的开发方法
* 应该建立详尽的文档
* 不应该放弃质量标准
* 应该测试，测试再测试
* 应该重用

**第五章 总体设计**

1. **设计过程**：**2个阶段**(系统设计阶段:确定系统的具体实现方案 和 结构设计阶段:确定软件结构); **9个步骤**：

* 设想供选择的方案
* 选取合理的方案
* 推荐最佳方案
* 功能分解
* 设计软件结构
* 设计数据库
* 制定测试计划
* 书写文档
* 审查和复审

1. **设计原理的相关概念**

* 模块化：就是把程序划分成独立命名且可独立访问的模块，每个模块完成一个子功能，这些模块集成起来构成一个整体，可以完成指定的功能满足用户的需求
* 抽象：抽出事物本质特性而暂时不考虑细节
* 逐步求精：为了能集中精力解决最主要问题而尽量推迟对问题细节的考虑。逐步求精是人类解决复杂问题时采用的基本方法，也是许多软件工程技术的基础
* 信息隐藏：应该这样设计和确定模块，使得一个模块内包含的信息对于不需要这些信息的模块来说是不能访问的
* 局部化：是指把一些关系密切的软件元素物理地址放得彼此靠近
* 模块独立：是模块化、抽象、信息隐藏和局部化的概念的直接结果。独立的程度测量标准：内聚、耦合

1. **耦合**：是对一个软件结构内不同模块之间互连程度的度量。耦合强弱取决于模块间接口的复杂程度，进入或访问一个模块的点，以及通过接口的数据。耦合程度强烈影响着系统的可理解性、可测试性、可靠性、可维护性。设计原则：尽量使用数据耦合，少用控制耦合和特征耦合，限制公共环境的耦合的范围，完全不用内容耦合。

* 数据耦合：如果两个模块彼此间通过参数交换信息，而且交换的信息仅仅是数据。数据耦合是低耦合
* 控制耦合：传递的信息中有控制信息。中等耦合，增加了系统的复杂性
* 特征耦合：当整个数据结构作为参数传递而被调用的模块只需要使用其中一部分数据元素时
* 公共环境耦合：当两个或对个模块通过一个公共数据环境互相作用时。公共环境可以是全程变量、共享通信区、内存的公共覆盖区、任何存储介质的文件、物理设备等。
* 内容耦合：如果发生之一就是①一个模块访问另一个模块的内部数据,②一个模块不通过正常入口而转到另一个模块的内部,③两个模块有一部分程序代码重叠,④一个模块有多个入口

1. **内聚**：标志着一个模块内各个元素彼此之间结合的紧密程度，它是信息隐藏和局部化概念的扩展。设计原则：力求高内聚，通过提高模块间的内聚降低耦合从而使模块获得较高的独立性。高内聚意味着低耦合
2. **低内聚**

* 偶然内聚：如果一个模块完成一组任务，这些任务彼此之间有关系，关系也是很松散的。如在一个程序内有一组语句在两处或多处出现，于是把这些语句作为一个模块以节省内存
* 逻辑内聚：如果一个模块完成的任务在逻辑上属于相同或相似的一类。如一个模块产生各种类型的全部输出
* 时间内聚：如果一个模块包含的任务必须在同一时间内执行。如模块完成各种初始化工作

1. **中内聚**

* 过程内聚：如果一个模块内的处理元素是相关的，且必须以特定次序执行。如流程图确定模块的划分，得到的往往是过程内聚的模块
* 通信内聚：如果一个模块中所有元素都是用同一个输入数据和产生同一个输出数据

1. **高内聚**

* 顺序内聚：如果一个模块内的处理元素和同一个功能密切相关，且这些处理必须顺序执行。如一个处理元素的输出数据作为下一个处理元素的输入数据，根据数据流图划分模块得到往往是顺序内聚模块
* 功能内聚：如果模块内的所有处理元素属于一个整体，完成单一的功能

1. 7种内聚的优劣评分

| **名称** | **得分** |
| --- | --- |
| 功能内聚 | 10 |
| 顺序内聚 | 9 |
| 通信内聚 | 7 |
| 过程内聚 | 5 |
| 时间内聚 | 3 |
| 逻辑内聚 | 1 |
| 偶然内聚 | 0 |

1. **启发规则**

* 改进软件结构提高模块的独立性
* 模块规模应该适中
* 深度、宽度、扇出和扇入都应适当
* 模块的作用域应该在控制域内
* 力争降低模块接口的复杂程度
* 设计单入口单出口的模块
* 模块的功能应该可预测

1. 描绘软件结构的图形工具（例子见P102,P103）

* 层次图：描绘软件的层次结构。一个矩形框代表一个模块，方框间的连线表示调用关系
* HIPO图：“层次图加输入/处理/输出图”，就是在层次图的每个方框加编号
* 结构图：每个方框代表一个模块，框内注明模块的名字或主要功能，方框间的箭头（或直线）代表模块的调用关系，注释表示来回传递的信息【尾部空心圆表示传递数据，实心圆代表传递控制信息】

**第六章 详细设计**

1. 人机界面设计指南：P123,P124

* 一般交互指南
* 信息显示指南
* 数据输入指南

1. 程序流程图：

* 优点：对控制流程的描绘直观，初学者很容易掌握
* 缺点：①程序流程图不是精益求精的好工具吗，它诱使程序员过早地考虑程序的控制流程，而不去考虑全局结构②程序流程图中用箭头代表控制流 ，因此程序员不受任何约束，可以完全不顾结构程序设计的思想，随意转移③程序流程图不易表示数据结构

1. 盒图（N-S图）的特点：

* 功能域明确
* 不可能任意转移控制
* 很容易确定局部和全局数据的作用域
* 很容易表现嵌套关系，也可以表示模块的层次结构

1. 问题分析图（PAD图）：使用二维结构的图来表示程序的控制流。其优点：

* 使用PAD符号设计出来必然是结构化程序
* PAD图描绘的程序结构十分清楚
* PAD图表现程序的逻辑，易读、易懂、易记
* 很容易将PAD图转化为高级语言程序
* 即可表示程序逻辑，也可表示数据结构
* PAD符号支持自动向下，逐步求精

1. 判定表：当算法中含有多重嵌套的条件选择时

* 优点：能清晰表示复杂的条件组合与应做的动作之间的关系
* 缺点：①判定表的含义不能一眼看出来②当数据元素多于两个时，判定表的简洁程度下降

1. 判定树：判定表变种

* 优点：一眼看出其含义，易于掌握，使用
* 缺点：①简洁性不如判定表，数据元素需重复写多遍②判定树的分支次序对画出的判定树的简洁程度有较大影响

1. PDL（过程设计语言）：也称伪码，具有严格的关键字外部语法，用于定义控制结构和数据结构，内部语法灵活自由，适应各种工程项目。

其优点：

* 可作为注释直接插在源程序中间
* 可使用普通的正文编辑程序或文字处理系统完成PDL的书写和编辑
* 已有自动处理PDL的程序，可自动生成程序代码

其缺点：

* 不如图形工具形象直观，描述复杂的条件组合与动作间的对应关系时，不如判定表清晰简单

1. **McCabe方法**：McCabe根据程序控制流的复杂程度度量 程序的复杂程度，这样度量出的结果称为程序的环形复杂度

①流图的表示：

* 结点：用圆表示，一个圆代表一条或多条语句
* 边：箭头线称为边，代表控制流
* 区域：由边和结点围成的面积 称为区域，当计算区域数时应该包括图外未被围起来的区域
* 判定结点：包含条件的结点

②计算环形复杂度的方法:

* 流图中线性无关的区域数等于环形复杂度
* 流图G的环形复杂度V(G)=E-N+2,其中E是流图中边的条数，N是结点数
* 流图G的环形复杂度V(G)=P+1,其中P是流图中判定结点的数目

**第七章 实现**

编码：把详细设计结果翻译成某种程序语言书写的程序  
软件测试：是保证软件质量的关键步骤，是对软件规格说明、设计和编码的最后复审

第七章实现(编码和测试).png

**补充**

测试用例：所谓测试用例是为特定的目的而设计的一组测试输入、执行条件和预期的结果；测试用例是执行测试的最小实体。

白盒测试主要用于对模块的测试，包括：程序模块中的所有独立路径至少执行一次；对所有逻辑判定的取值（“真”与“假”）都至少测试一次；在上下边界及可操作范围内运行所有循环；测试内部数据结构的有效性等。

黑盒测试可用于各种测试，它试图发现以下类型的错误：不正确或遗漏的功能；界面错误；数据结构错误或外部信息(如外部数据库)访问错误；性能错误；初始化和终止错误。

**第八章 维护**

1. **软件维护的定义**：就是在软件已经交付使用之后，为了改正错误或满足新的需要而修改软件的过程
2. **结构化维护和非结构化维护**

①非结构化维护

* 如果软件配置的唯一成分是程序代码，那么维护活动从评价代码开始，而且由于内部文档不足而使评价更困难
* 非结构化维护需要付出巨大代价，是没有使用良好定义的方法学开发出来的必然结果  
  ②结构化维护
* 如果有一个完整软件配置存在，那么维护从评价设计文档开始就很规范
* 减少精力的浪费，提高维护的总体质量

1. **决定软件可维护的因素**：

* 可理解性
* 可测试性
* 可修改性
* 可重用性

**第八章 面向对象方法学引论**

1. **面向对象方法学要点**：

①基本原则：尽可能模拟人类习惯的思维方式，是开发软件的方法和过程尽可能接近人类认识的世界解决问题的方法和过程  
②4个要点

* 软件是由对象组成的，任何元素都是对象，复杂软件对向由比较简单的软件对象组成
* 所有对象都划分成对象类，类都定义了一组数据和一组方法
* 若干对象类组成一个层次的系统
* 对象间仅能通过传递消息互相联系  
  ③面向对象方法学优点
* 与人类习惯的思维方法一致
* 稳定性好
* 可重用性好
* 较易开发大型软件产品
* 可维护性好

1. 对象：是描述该对象属性的数据以及对这些数据施加的所有操作封装在一起构成的统一体

①对象的定义

* 对象是具有相同状态的一组操作的集合
* 对象是对问题域中某个东西的抽象
* 对象::=<ID,MS,DS,MI>。ID是对象的标识或名字，MS操作集合，DS数据结构，MI对象受理的消息名集合  
  ②对象的特点
* 以数据为中心
* 对象是主动
* 实现了数据的封装
* 本质上具有并行性
* 模块独立性好

1. 其它概念

* 类：是一组具有相同属性和相同操作的对象的集合。
* 实例：由某个特定的类所描述的一个具体的对象。
* 消息：要求某个对象执行在定义它的那个类中所定义的某个操作的规格说明。组成部分：接收消息的对象；消息名；消息变元
* 方法：就是对象所能执行的操作，也就是类中所定义的服务。
* 属性：属性就是类中所定义的数据，它是对客观世界实体所具有的性质的抽象。
* 封装：对象封装了对象的数据以及对这些数据的操作。
* 继承：继承是子类自动地共享基类中定义的数据和方法的机制。
* 单重继承：子类仅从一个父类继承属性和方法
* 多重继承：子类可从多个父类继承属性和方法
* 多态性：指子类对象可以像父类那样使用
* 函数重载：指在同一作用域内的若干参数特征不同的函数可以使用相同的函数名
* 运算符重载：指在同一个运算符可以施加于不同类型的操作数上面

**第十章 面向对象分析**

1. **建立对象模型**

①三个子模型，按所解决的问题进行划分

* 静态结构（对象模型）
* 交互次序（动态模型）
* 数据变换（功能模型）

②5个层次

* 主题层
* 类与对象层
* 结构层
* 属性层
* 服务层

③对象模型创建的步骤

* 确定类与对象
* 确定关联
* 划分主题
* 确定属性
* 识别继承关系
* 反复修改

**第十一章 面向对象设计**

1. **面向对象设计准则**

* 模块化
* 抽象
* 信息隐藏
* 弱耦合
* 强内聚
* 可重用

1. **类构件的重用方式**：

* 实例重用
* 继承重用
* 多态重用