软件工程背诵

【精讲】软件工程用图的各个阶段及其应用（详细）系统流程图、数据流图、数据字典、ER图、状态转换图、层次方框图、Warnier图、IPO图、层次图、HIPO图、结构图、程序流程图、盒图等_流程图和ipo的应用-CSDN博客

这次考试考了的内容大概有：

1、选择题基本和平时测验的题目一致，甚至是原题。

2、简答题

第一题：画工程网络图 （要会画）然后写几个最早最晚以及机动时间
第二题：好像是使用面向对象分析的好处？对维护有什么好处？
后面的顺序忘了，大概有：画状态图、基本路径测试、面向数据流的设计方法 映射过程（这个上课联系过，但要记得步骤，考试好像是要写出完整的步骤加画图）、自底向下和三明治集成的画图、人机界面要注意的点？好的人机界面？、

3、分析题 画用例图，写用例规约、顺序图（基本事件流和备选事件流两个）、画参与类图。

第一章 概述

1、什么是软件？软件的特点？与硬件相比？

软件是包括**程序**、**数据**及其**相关文档**的完整集合

特点
抽象性、没有明显的制造过程、与硬件相比没有机械磨损、老化问题。

软件危机：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。包括**如何开发**即**如何维护**

应该推广使用在实践中总结出来的开发软件的成功的技术和方法，并且研究探索更有效的技术和方法，尽快消除在计算机系统早期发展阶段形成的一些错误观念和做法。 吸取和借鉴人类长期。。。。各种工程。。。所积累。。。。。方法原理。。。经验教训。

2、软件工程的三要素？

软件工程方法学包含三个要素：方法、工具和过程。

方法：完成软件开发的各项任务的技术方法，回答“怎样做”的问题；
工具：软件工具为软件工程方法提供了自动或半自动的软件支撑环境；
过程：是为了获得高质量软件所需要完成的一系列任务的框架，它规定了完成各项任务的步骤，将软件工程的方法和工具综合起来以达到合理、及时地进行软件开发的目的。

生命周期：
软件定义时期：问题定义阶段、可行性研究阶段、需求分析阶段
软件开发时期：总体设计阶段、详细设计阶段、编码和单元测试阶段、综合测试阶段
软件维护时期：改正性维护、适应性维护、完善性维护、预防性维护

第二章 软件过程

☆1、 几个经典的软件过程（优点和缺点、会选择）

1、**瀑布模型** 需求分析-规格说明-设计-编码-综合测试-维护
*文档驱动模型
特点：阶段间有顺序性和依赖性（上一阶段结束才能进行下一阶段）
要求用户不经实践提出**完整准确的需求，不切实际！！**

优点：
1、可强迫开发人员采用规范的方法（例如，结构化技术）
2、严格地规定了每个阶段必须提交的文档
3、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、瀑布模型的成功在很大程度上是由于它基本上是一种文档驱动的模型

缺点：
1、要求用户不经过实践就提出完整准确的需求，在许多情况下都是不切实际的
2、仅仅通过写在纸上的静态的规格说明，很难全面正确地认识动态的软件产品
3、将本来非线性的软件开发过程人为地加以线性化，不符合实际中的软件开发情况
4、软件开发耗时长，可运行版本要等到项目后期才能得到，一旦在后期发现错误，付出的代价将是巨大的。
5、“由文档驱动”的这个事实也是瀑布模型的一个主要缺点,这可能导致最终开发出的软件产品不能真正满足用户的需要

2、V**模型** 改进瀑布模型 **强调测试活动和分析设计之间的关联**
需求分析-系统设计-程序设计-编码-单元测试和集成测试-系统测试-验收测试-运行和维护
活动驱动模型

优点：
1、本质是把瀑布模型中一些隐含的**迭代**过程明确出来，使开发活动和验证活动的相关性更加明显；
2、V模型使抽象等级的概念也更明显：所有从需求到实现部分的活动关注的是建立更多的系统详细表述，而所有从实现到交付运行的活动关注的是对系统的验证和确认。
缺点：
和瀑布模型一样，都是对软件开发过程过于简单、理想化的抽象，对需求变化的适应性差。

3、**快速原型模型** 快速分析-快速构造原型-运行原型-评价原型（-快速修改原型）-形成最终系统

特点

1、利用原型能统一客户和开发人员对软件项目需求的理解，**有助于需求的定义和确认**；
2、可以考虑结合瀑布模型，二者互补性强。用快速原型做为需求分析的一种技术，用于收集客户的真实需求，然后把客户满意了的原型再作为瀑布模型的输入，从而达到优势互补。

4、增量模型

增量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。
*第一个增量构件往往实现软件的基本需求，提供最核心的功能。

特点
用户能在**较短时间内**使用上部分功能；
优点
1、适用于人员配备不充裕、不能在软件项目期限之前实现一个完全版本的软件的情况；
2、能有计划地管理技术风险；
3、每个增量都发布了一个高质量的可操作的版本，用户能在较短时间内使用上部分功能；
4、逐步增加产品功能可以使用户有较充裕的时间学习和适应新产品，减少一个全新的软件可能给客户带来的冲击。
缺点：
1、软件**体系结构必须是开放的**，使得每个新的增量构件能够**无缝地集成**到现有的软件体系结构中，**增加了设计阶段的投入**；

2、本身具有矛盾性，一方面要求开发人员把软件看作一个整体，另一方面要求开发人员把软件看作构件序列，且构件间彼此独立。需要开发人员协调这一矛盾。

5、螺旋模型 快速原型-规格说明-设计-编码-综合测试-维护
制定计划、风险分析、实施工程、客户评估

*基本思想：使用原型及其他方法来尽量**降低风险。**
每个阶段之前都增加了风险分析过程的快速原型模型。

优点：
1、螺旋模型是对瀑布模型的发展，并由客户对阶段性产品做出评审，这对保证软件产品质量十分有利；
2、由于引入风险分析等活动，测试活动的确定性增强了；
3、螺旋模型最外层代表维护，开发与维护采用同样方式，使维护得到与开发同样的重视。、

缺点：
1、主要适合内部开发，否则风险分析必须在签订合同前完成，或者争取客户的最大理解；
2、只适合大型软件项目的开发，否则，每个阶段的风险分析将占用很大一部分资源，增加成本；
3、对开发人员的风险分析能力是极大的考验，否则，模型将退化到瀑布模型，甚至更糟。

6、喷泉模型

体现面向对象

“喷泉”体现了面向对象软件开发过程**迭代**和**无缝**的特性

RUP：迭代式开发

敏捷过程的一些关键词：适应而非预测变化、价值驱动、极限编程、使用用户素材
微软过程：

第三章 可行性研究

目的和实质

1、可行性研究的**目的**：
用最小的代价，在尽可能短的时间内确定问题是否能够解决。
可行性研究的**实质**：
就是一次压缩、简化了的系统分析和设计的过程。
需要导出**逻辑模型**

包括技术可行性、经济可行性、操作可行性、法律可行性、开发方案的选择性研究

最根本的任务：对以后的行动方针提出建议。

2、成本效益分析

i—年利率；
P—现在存入的钱数；
n—年数；
n年后可以获得的钱数为F=P(1+i)n（n次方）；
反之，如果n年后能收入F元，这些钱的现在价值为P=F/(1+i)n?

第四章需求分析

1、为什么需求分析?

1、在需求过程中会产生很多错误（事实3和4）
2、许多错误并没有在早期被发现（事实2）
3、这样的错误是能够在产生的初期被检查出来的（事实5）
4、如果没有及时检查出来这些错误，软件费用会直线上升（事实1）

☆2、需求满足的特征

正一完现实验回

正确性：要确保需求的表达中没有引入错误（faults）。

一致性：确保没有互相冲突、矛盾的需求；确保没有不确定的需求。

完整性：如果需求描述了所有可能的状态，以及状态的变化、输入、过程和约束，那么这组需求就是完整的。

现实性：确保客户要求系统做的事真的能做到。

实用性：确保需求和要解决的问题有直接关系。

可检验性：必须能写出测试来说明需求已被满足。

可回溯性：保证每个系统功能都能追溯到一组要求它的需求；确保很容易找到处理系统特定方面的某一组需求。

3、需求分析原则

☆**操作性原则**

1、必须理解并描述问题的信息域，根据这条准则应该建立**数据模型**；

2、必须定义软件应完成的功能，这条准则要求建立**功能模型**；

3、必须描述作为外部事件结果的软件行为，这条准则要求建立**行为模型**；

4、必须对描述信息、功能和行为的模型进行**分解**，用**层次**的方式**展示细节**

指导性原则
在你开始建立分析模型前先理解问题
开发使用户能够了解人机交互将如何发生的原型
记录每个需求的起源及原因
使用多个需求视图
给需求赋予优先级
努力删除歧义性

4、非功能性需求

性能、可靠性、安全/保密性、运行限制、物理限制、用户界面友好性

☆5、自顶向下逐层画数据流图的步骤

1. 首先建立**顶层的数据流图**（基本系统模型）其中只含有一个代表目标软件系统整体处理功能的转换。
2. 画系统的内部：将顶层图中的处理分解成若干个处理，并用数据流将这些处理连接起来，使得顶层图中的输入数据流经一连串的加工处理后变换成顶层图中的输出数据流。这张图称为1层数据流图。又称为**系统功能级数据流图。**

3. 画处理的内部：把每一个处理看作一个小系统，该处理的输入输出数据流看成小系统的输入输出数据流，于是可以用画1层图同样的方法画出每个处理的DFD子图。
4. 对第3步分解出来的DFD子图中的每个处理重复第3步的分解，直至图中尚未分解的处理都足够简单为止。到此得到了一套分层的数据流图。

☆6、数据字典的作用功能

1、数据词典与数据流图**共同**构成系统的逻辑模型

2、数据字典是对数据流图中包含的所有元素的定义的集合

第五章 总体设计

☆1、设计原理

（1）**模块化**：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，把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。

（2）**抽象**：抽出事物的本质特性而暂时不考虑它们的细节。软件设计过程应当是在不同抽象级别考虑和处理问题的过程。

（3）**逐步求精**：逐步求精可视为一种自顶向下的设计策略。按照这种设计策略，程序的体系结构是通过逐步精化处理过程的层次而设计出来的。

(4)**信息隐藏和局部化**:信息隐藏原理指出：模块应该设计成其中包含的信息（过程和数据）对不需要这些信息的其他模块来说是不可访问的。　局部化：指把一些关系密切的软件元素物理地放得彼此靠近 。隐藏的是模块的实现细节。

(5)**模块独立性**:每个模块完成一个相对独立的子功能，并且和其他模块之间的关系很简单。 它是模块化、抽象、信息隐藏和局部化概念的直接结果。

模块独立的理由：
第一，有效的模块化（即具有独立性的模块）的软件比较容易开发出来；
第二，独立的模块比较容易测试和维护。

2、内聚和耦合

☆**(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。**

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

降低耦合的方法?

☆（2）内聚（Cohesion）标志一个模块内各个元素彼此结合的紧密程度，是模块功能强度的度量，用来量化表示一个模块在多大程度上专注于一件事情。一个模块内部各个元素彼此结合得越紧密，内聚度就越高，模块独立性就越强

偶然内聚
逻辑内聚
时间内聚
过程内聚
通信内聚
顺序内聚
功能内聚

3、启发式原则

1．改进软件结构提高模块独立性：

2．模块规模应该适中：

3．深度、宽度、扇出和扇入都应适当：

4．模块的作用域应该在控制域之内 ：

5．力争降低模块接口的复杂程度：

6．设计单入口单出口的模块：

7．模块功能应该可以预测 ， 避免对模块施加过多限制：

☆4、面向数据流的设计方法 映射过程（步骤）

第1步 复查基本系统模型

第2步 复查并精化数据流图

第3步 确定数据流图具有变换特性还是事务特性

第4步 确定输入流和输出流的边界，从而孤立出变换中心

第5步 完成“第一级分解(first level factoring)”。
分解出模块：Cm,Ca,Ct,Ce，其中
Cm：协调下述从属的控制功能；
Ca：输入信息处理控制模块，协调对所有输入数据的接收；
Ct：变换中心控制模块，管理对内部形式的数据的所有操作；
Ce：输出信息处理控制模块，协调输出信息的产生过程。

第6步 完成“第二级分解”
即把数据流图中的每个处理映射成软件结构中一个适当的模块。
从变换中心的边界开始沿着输入通路向外移动，把输入通路中每个处理映射成软件结构中Ca控制下的一个低层模块；
沿输出通路向外移动，把输出通路中每个处理映射成直接或间接受模块Ce控制的一个低层模块；
把变换中心内的每个处理映射成受Ct控制的一个模块。

第7步 使用设计度量和启发式规则对第一次分割得到的软件结构进一步精化

由事务流映射成的软件结构包括一个接收分支和一个发送分支；

映射接收分支：从事务中心的边界开始，把沿着接收流通路的处理映射成模块；
发送分支的结构包含一个调度模块，它控制下层的所有活动模块；
把数据流图中的每个活动流通路映射成与它的流特征相对应的结构。

第六章 详细设计

结构化程序设计技术是详细设计的逻辑基础。

☆、不同的过程设计工具的优缺点

程序流程图

优点：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精神，随意转移控制。

缺点：1、程序流程图本质上**不是逐步求精的好工具**，它诱使程序员过早地考虑程序的控制流程，而不去考虑程序的全局结构。

2、程序流程图**不易表示数据结构。**

盒图

优点：

1、**功能域**(即某个特定控制结构的作用域)**明确**，可以从盒图上一眼就看出来。

2、很容易确定局部和全程数据的**作用域。**

3、很容易表现**嵌套关系**，也可以表示模块的层次结构。

缺点：

1、不可能任意转移控制。

PAD（Problem Analysis Diagram）图

用二维树形结构的图来表示程序的控制流, 设置了五种基本控制结构的图式，并允许递归使用。

PAD图的主要优点:
1、使用表示结构化控制结构的PAD符号所设计出来的程序**必然是结构化程序。**
2、PAD图所描绘的程序结构十分**清晰。**
3、用PAD图表现程序逻辑，**易读、易懂、易记。**
4、容易将PAD图**转换成高级语言源程序**，这种转换可用软件工具自动完成。
5、既可用于表示程序逻辑，也可用于描绘数据结构。
6、PAD图的符号**支持自顶向下、逐步求精方法**的使用。

判定表
1、当算法中**包含多重嵌套的条件选择（注意是条件不是循环！）时**，使用判定表能够清晰地表示复杂的条件组合与应做的动作之间的对应关系。
2、判定表用于表示程序的**静态逻辑。**
3、在判定表中的条件部分给出所有的两分支判断的列表，动作部分给出相应的处理。
4、要求将程序流程图中的多分支判断都改成两分支判断。

优点：能够简洁、无二义性地描述所有的处理规则。
缺点：判定表表示的是静态逻辑，是在某种条件取值组合情况下可能的结果，它**不能表达加工的顺序**，也**不能表达循环结构**，因此判定表不能成为一种通用的设计工具。

判定树
判定树是判定表的变种。
优点：形式简单，比判定表更直观
缺点：简洁性不如判定表
简洁性不如判定表。
画判定树时分枝的次序可能对最终画出的判定树的简洁程度有较大影响 。

过程设计语言（PDL）

伪代码

PDL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；另一方面，PDL表示实际操作和条件的内部语法通常又是灵活自由的，可以适应各种工程项目的需要。

优点：
1、可以作为注释直接插在源程序中间。这样做能促使维护人员在修改程序代码的同时也相应地修改PDL注释，因此有助于保持文档和程序的一致性，提高了文档的质量
2、可以使用普通的正文编辑程序或文字处理系统，很方便地完成PDL的书写和编辑工作。
3、已经有自动处理程序存在，而且可以自动由PDL生成程序代码。

PDL的缺点是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。

☆2、人机界面设计的黄金规则

赋予用户控制权
提供灵活的交互；
允许用户交互可以被中断和撤销；

减少用户的记忆负担
保持界面一致

为什么ui要设计的更好?

人机界面可以看作是基于计算机的系统或产品的最重要的元素；
不好的人机界面将严重地阻碍用户使用系统的计算能力；
一个弱的界面可能导致一个很好设计和可靠实现的应用的失败。

3、人机界面设计过程

用户、任务和环境分析及建模
界面设计
界面构造
界面确认

4、McCaBe

主要看一下缺点

环形复杂度的用途
程序的环形复杂度取决于程序控制流的复杂程度，也即是取决于程序结构的复杂程度。
当程序内分支数或循环个数增加时，环形复杂度也随之增加，因此它是对测试难度的一种定量度量，也能对软件最终的可靠性给出某种预测。
环形复杂度是可加的。例如，模块A的复杂度为3，模块B的复杂度为4，则模块A与模块B的复杂度是7。

用来限制模块的最大行数。McCabe从大量的调查中发现，当V(G)等于或大于10时，对模块进行充分的测试将变得非常困难。他主张将10作为环域数的上限，并以此来限制模块的最大规模。

☆**缺点：**
1、对于不同种类的控制流的复杂性不能区分；
2、简单IF语句与循环语句的复杂性同等看待；
3、嵌套IF语句与简单CASE语句的复杂性是一样的；
4、模块间接口当成一个简单分支一样处理；
5、一个具有1000行的顺序程序与一行语句的复杂性相同。

“这里面啊但是我们还要知道这还复难度的定量度量的方法，就拿一比方法呢它有局限性啊，比如**他只考虑控制结构的复杂度，不考虑语句本身的复杂度**啊，如果上千条顺执行的语句可能就会跟一条语句的这个复杂度是一样的，用这个环形复杂度的不良方法，这就是他是考虑控制结构三度了，没有考虑具体的具体的三度，所以这也是三个一个局限性，这是我们的详细设计啊这一部分找到这种程度就可以了。‘

第七章 实现

1、测试的目的：发现错误

一个好的测试用例在于能发现至今未发现的错误

一个成功的测试是发现了至今未发现的错误的测试

2、黑盒和白盒

如果已经知道了产品应该具有的功能，可以通过测试检验是否每个功能都能正常使用。

黑盒测试：把测试对象看作一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能☆**是否符合它的功能说明。**
☆**黑盒测试又叫做功能测试或数据驱动测试。**

如果已经知道了产品的内部工作过程，可以通过测试 来检验产品内部动作是否按照规格说明书的规定正常进行。

白盒测试：把测试对象看作一个透明盒子，它允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序的☆**所有逻辑路径**进行测试。
☆**白盒测试又称为结构测试、玻璃盒测试、或逻辑驱动测试。**

☆3、软件测试步骤

模块测试（单元测试）： 检查 模块接口、局部数据结构、重要的执行通路、出错处理通路、边界条件
把每个模块作为一个单独的实体进行测试，发现的往往是编码和详细设计的错误。
子系统测试（集成测试）：
把经过单元测试的模块放在一起形成一个子系统进行测试，着重**测试模块的接口。**
系统测试（集成测试）：
把经过测试的子系统装配成一个完整的系统进行测试。发现的往往是软件设计中的错误，也可能发现需求说明中的错误。
验收测试（确认测试）： 确认测试也叫验收测试，其目标是验证**软件的有效性。**软件有效性的简单定义：如果软件的功能和性能如同用户所合理期待的那样，软件就是有效的。
把软件系统作为单一的实体进行测试，用户积极参与，主要使用实际数据。
目的是验证系统确实能够满足用户的需要。
发现的往往是系统需求说明书中的错误。

单元测试的测试重点：
单元测试需要驱动程序(上层模块)、存根程序（下层其所调用的模块）

集成测试重点：
集成测试分为非渐增式和渐增式。

确认测试重点：功能

4、几种集成测试的策略和步骤

1、一次性集成

缺点：
需要编写大量存根程序和驱动程序。
所有组件一次进行合并，很难找出所有错误的原因。
不容易区分接口错误与其他类型的错误。

2、自顶向下集成

从主控制模块开始，沿着程序的控制层次向下移动，逐渐把各个模块结合起来。（深度优先、宽度优先）
优点：
自顶向下的结合策略能够**在测试的早期对主要的控制或关键的抉择进行检验。**
自顶向下测试不需要驱动程序。
选择深度优先的结合方法，可以在早期实现软件的一个完整的功能并且验证这个功能。
缺点：
需要编写存根程序。
为了充分地测试软件系统的较高层次，需要在较低层次上的处理。
可能需要很多存根程序

3、自底向上集成

从“原子”模块(即在软件结构最低层的模块)开始组装和测试，不需要存根程序。 适用于原子模块是公共用例
优点：
不需要编写存根程序。
测试驱动程序的数目较少。
底层模块往往承担主要的计算和输入、输出任务，更容易出错，该方式可以在早期发现这类错误。
底层模块可以并行测试。
缺点：
对顶层组件测试进行得晚，**会推迟主要错误的发现。**
程序在最后一个模块加入时才具有整体形象。

☆4、三明治集成

将自顶向下策略与自底向上策略结合起来
关键是选取结构图某一层为**基准层**（或称目标层）
选取不同的层次为基准层，则整个集成测试的活动情况相应会有很大不同

优点：
允许在测试的早期进行集成测试
结合了自顶向下和自底向上测试的优点，在测试的最开始就对控制和公用程序进行测试
缺点：
在集成之前没有彻底地测试单独的组件

☆回归测试：回归测试（regression test）是指重新执行已经做过的测试的某个子集，以保证由于调试或其他原因引起的变化，不会导致非预期的软件行为或额外错误。

5、白盒测试

☆逻辑覆盖测试

逻辑覆盖是以程序**内部的逻辑结构**为基础设计测试用例的技术

语句覆盖
判定覆盖（每个判定的真值分支和非真分支都走一边）
条件覆盖（每个判定中的每个条件的可能取值都执行一边） 注意！！：虽然他更强，但存在满足条件覆盖但不满足判定覆盖的情况
判定/条件覆盖：存在条件隐藏问题
条件组合覆盖：必定满足2-3，是前述覆盖标准中最强的，但不一定使程序中的每条路径都执行到。
点覆盖：经过流图中的每个点，与语句覆盖相同
边覆盖：经过流图中的每个边
路径覆盖：每条可能的路径都走一次（可能相当多）

☆控制结构测试

基本路径测试

1、画流图
2、求环形复杂度
3、确定线性独立路径的基本集合

6、黑盒测试

黑盒测试法与白盒测试法不能互相代替，两者应互为补充。
白盒测试在测试过程的早期阶段进行，黑盒测试主要用于测试过程的后期

☆等价划分

等价划分方法把所有可能的输入数据，即程序的输入域划分成若干部分，据此导出测试用例，一个理想的测试用例能够独自发现一类错误。

有效等价类：是指对于程序的规格说明来说，那些合理的、有意义的输入数据构成的集合。
无效等价类：是指对于程序的规格说明来说，那些不合理的、无意义的输入数据构成的集合。

容易考：！！！！：

设计一个新的测试用例，使其**尽可能多地**覆盖尚未被覆盖的**有效等价类**，重复这一步，直到所有的有效等价类都被覆盖为止；
设计一个新的测试用例，使**其仅覆盖一个**尚未被覆盖的**无效等价类**，重复这一步，直到所有的无效等价类都被覆盖为止。

边界值分析

边界

错误推测法

经验和直觉

7、白黑对比

1、白盒与黑盒这两类测试是从完全不同的角度出发看待测试对象的。
2、白盒测试只考虑测试软件产品，不保证完整的需求规格是否被满足；而黑盒测试只考虑需求规格，不保证实现的所有部分是否被测试到。
3、黑盒测试会发现遗漏的缺陷，指出规格的哪些部分没有被完成；而白盒测试会发现提交的缺陷，指出哪些实现的部分是错误的。

4、白盒测试的成本比黑盒测试高得多。它需要在测试可以被计划之前先有源代码。
5、一个白盒测试的失败会导致一次修改，这需要所有的黑盒测试被重复执行并且重新决定白盒测试路径。

8、软件可靠性

软件可靠性

软件可用性

稳定可用性

MTTF：平均无故障时间
MTTR：平均修复时间

☆经验表明，平均无故障时间与单位长度程序中剩余的错误数成反比,即

Et： 测试之前程序总错误数 It：程序长度（机器指令总数）
Ec （t）：在t时间内改正的错误

9、调试

它与软件测试不同，调试的任务是进一步诊断和改正程序中潜在的错误。

蛮干法、回溯法、原因排除法。

第八章 软件维护

1、什么是软件维护 完善性维护、适应性维护、改正性维护，预测性维护

软件维护：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、决定软件可维护性的因素

可移植性
可重用性
可修改性
可测试性
可理解性

3、如何提高软件可维护性

1、需求分析复审
标注可能的改进和修改
软件可移植性
系统界面

设计复审：
评价软件的结构和过程
对将来可能修改的部分预作设计

代码复审：
编码风格、内部说明文档

**设计和编码：

配置复审：
审查软件配置成分使用可重用的软件构件

第九章 软件项目管理

1、甘特图的优缺点

优点：
Gantt图能很形象地描绘任务分解情况，以及每个子任务(作业)的开始时间和结束时间，因此是进度计划和进度管理的有力工具。

☆缺点
(1) **不能显式地描绘**各项作业彼此间的**依赖关系**；
(2) 进度计划的关键部分不明确，**难于判定**哪些部分应当是**主攻和主控的对象**；
(3) 计划中有潜力的部分及潜力的大小不明确，往往造成潜力的浪费。

工程网络图

2、人员组织机构以及适合的项目

（1）民主制程序员组
小组成员完全平等，享有充分民主，通过协商做出技术决策。

主要优点：
组员们对发现程序错误持积极的态度 。
组员们享有充分民主．小组有高度凝聚力，组内学术空气浓厚，有利于攻克技术难关。
主要问题：
软件开发人员多数比较缺乏经验；
程序设计过程中有许多事务性的工作，例如，大量信息的存储和更新；
多渠道通信很费时间，将降低程序员的生产率。

（2）主程序员组
主程序员组的两个重要特性： 专业化和层次性。

主程序员很难找到：成功的管理员、高度熟练的程序员的结合体；

（3）现代程序员组—主程序员小组的变化形式

技术组长，行政组长

（4）大型项目的技术管理组织结构

项目领导人—小组领导—程序员

扩展：包含分散决策的组织方式

各种制度的适用情况：

集中式 大短简
简单、重复的问题
模块化程度高的问题
大项目
周期固定、较短的项目

分散式 小长难
复杂、创新的问题
模块化程度低的问题
小项目
周期长的项目

3、需要重点理解的概念

软件配置?
SCI?
基线？基线的作用?

简言之：软件配置管理是软件系统发展过程中管理和控制变化的规范。

软件配置项（Software Configuration Item）：为了配置管理而作为单独实体处理的一个工作产品或一段软件，简称SCI。即软件过程输出的全部计算机程序、文档、数据。

基线（baseline）：已经**通过了正式复审**的规格说明或中间产品，它可以作为进一步开发的基础，并且只有通过正式的变化控制过程才能改变它。**基线就是通过了正式复审的软件配置项。**

项目数据库：
一旦一个SCI成为基线，就被存放到项目数据库中；
项目数据库和变更控制规程相结合，保护着项目的基线；

软件配置管理主要有5项任务：
标识
版本控制
变化控制
配置审计
配置状态报告

第十章 面向对象

需要掌握的内容：

分析阶段（问题空间）

1.根据问题的描述，用用例的方法去识别系统的的主要的功能性需求

2、创建合理规范的用例模型 识别参与者、画用例图、完善用例规约文档（有基本事件流也有备选事件流）

3、进行用例分析，提取分析类、转述usecase（用顺序图来刻画）

4、整理分析类、画出类图（参与类），用VOPC类图画出来，表达出各个类的职责和他们之间的关系。

将分析类映射成设计元素（求解空间）即识别设计元素

从分析类映射的设计类、从分析机制映射的设计类

将用例模型（顺序图和参与类图）中的分析类用设计类替换

用例规约样例：

用例名称	系统配置
相关参与者	老人家属
简要描述	家属进行看护系统的系统配置，设置机器人与老人安全距离、机器人移动速度、用户账号密码等。
前置条件	无
后置条件	系统配置成功
基本事件流	1. 家属设置机器人与老人安全距离2. 家属设置机器人移动速度3. 家属设置用户账号密码
备选事件流	A-1 1. 配置完成2. 设置距离超出范围A-2 1. 设置完成2. 设置速度过快或过慢（超出范围）
补充说明	非功能需求：无 设计约束：无 数据需求：1.密码为6-12位 阿拉伯数字+英文字母的组合 未解决的问题：无