

期末考试形式为 10 道判断 (1), 10 道单选 (1), 10 道问答 (3), 还有 4 道应用题 (需求分析 (画图)、建模分析 (总体设计)、测试、压轴是 2 选 1 (面向对象、都做加分))。

其中问答和应用主要包括

需求工程方法,
软件测试方法,
UML 的几个主要的图的绘制。

问答题中是基本的

软件生命周期, ok
测试方法,
敏捷方法,
软件能力成熟模型 CMM。

细化:

- 1 软件危机; ok
 - 2 软件工程的主要开发模型及区别; ok
 - 3 需求抽取的技术 (需求工程方法); what?
 - 4 UML, 需要掌握主要的几个图: 用例、类图、状态机图、时序图; ok
 - 5 软件设计中的体系结构有哪些; ok 模块的内聚和耦合; ok
面向对象的设计原则; ok 结构化软件设计和数据流的基本概念; ok
 - 6 软件测试方法, 需要掌握并且会应用; ok
 - 7 软件的演化主要包括什么 (软件维护); ok
 - 8 如何度量软件的规模 ok、质量 ok; 软件能力成熟度模型 CMM 是什么; ok
 - 9 甘特图的画法。ok
-

1. 0226-引入

1.1. 软件

软件=程序+数据+文档

程序: 程序是按事先设计好的功能和性能要求执行的指令序列。

数据: 数据是指程序能正常处理信息的数据和数据结构。

文档: 文档是与软件开发、设计、运行、维护有关的图文资料。

1.2. 程序的形式

面向机器的程序=二进制代码

面向过程的程序=算法+数据结构

面向对象程序=对象+消息

面向组件的程序=组件+基础设施

1.3. 文档的作用

记录; 交流工具; 阶段性成果; 便于管理; 软件质量保障; 参照; 便于维护;

1.4. 软件特点

本质: 复杂性、一致性、可变性、不可见性

1.5. 软件分类

系统软件、实时软件、商业管理软件、工程与科学计算软件、嵌入式软件...

1.6. 发展趋势

网络化、服务化、全球化、智能化

1.7. 软件标准

满足用户需求；可维护性；可靠性；效率；可用性；

1.8. 软件危机

是什么：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。

两方面：

其一：如何开发软件，以满足不断增长、日趋复杂的需求；（开发）

其二：如何维护数量不断膨胀的软件产品。（维护）

现象：

- 1 进度迟滞
- 2 成本高
- 3 质量差
- 4 难以维护
- 5 低效

为什么（怎么办）：

- 1 不可见性 —— 文档
- 2 系统庞大
- 3 系统复杂度高
- 4 内部灵活多变 —— 追求编码技巧
- 5 需求不清晰 —— 个体化、轻视协作

1.9. 什么是软件工程？

v1：软件工程-是为了**经济**地获得**可靠**的和能在实际机器上**高效**运行的软件-而确立和使用的健全的工程原理（方法）。

v2：软件工程是开发、运行、**维护和修复**软件的系统方法。

book：软件工程是采用工程的观念、原理、技术和方法来开发与维护软件，结合正确的管理方式和优质的技术，经济地开发出高质量的软件并维护它的方法。

1.10. 软件工程的基本目标

较低的开发成本

按时完成开发任务

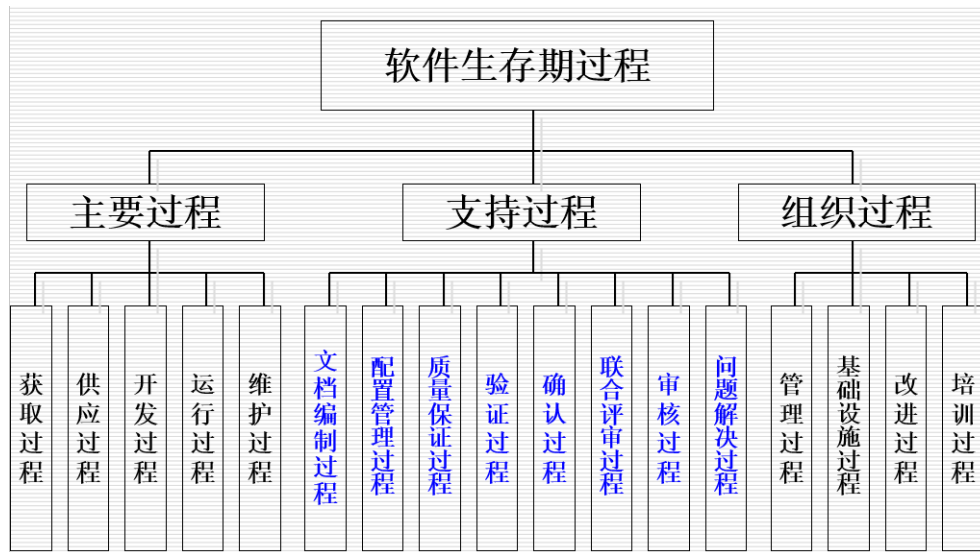
良好的性能

较高的可靠性、可扩展性、可移植性

软件维护费用低

1.11. 软件生命周期

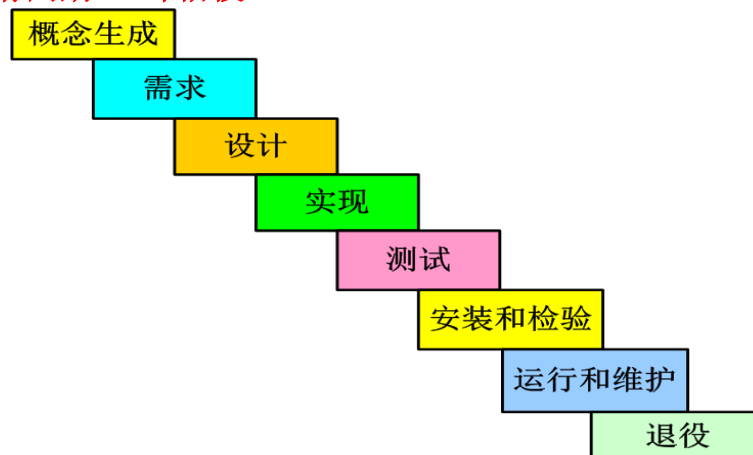
软件产品或软件系统，从设计、投入使用、到被淘汰的全过程。



软件工程的 8 个过程：

- (1) 问题定义
- (2) 可行性分析
- (3) 需求分析
- (4) 总体设计
- (5) 详细设计
- (6) 编码实现
- (7) 测试
- (8) 运行和维护

现代软件生存周期（8 个阶段）：



2. 0228-过程模型

2.1. 线性顺序过程模型

分析-涉及-编码-测试

2.2. 循环过程模型

需求-编码-测试-反馈-编码-测试...

2.3. 增量过程模型

阶段 1: 分析-涉及-编码-测试

阶段 2: 分析-涉及-编码-测试...

2.4. 螺旋过程模型、并行过程模型、喷泉模型...

软件开发的主要过程模型和区别:

*瀑布模型（线性）

缺: 推迟实现; 文档驱动; 不能并行; 不能逆转; 错误放大; 需求变更;

优: 容易管理;

场景: 需求明确; 需求不变; 面向结构方法;

*快速原型模型（迭代、效果预览）

缺: 可能忽视设计; 用户期望管理; 资源消耗;

优: 用户满意度高; 需求验证; 缩短开发周期; 降低成本;

场景: 需求不明确或易变; 用户界面设计; 新产品开发;

*增量模型（迭代、多版本）

缺: 每个增量需要完整的开发过程; 管理多版本的复杂性; 集成问题;

优: 早期交付部分功能; 降低失败风险; 更容易管理变更;

场景: 大型项目; 长期项目; 需求稳定但复杂;

*螺旋模型（迭代、风险驱动）

缺: 复杂性高; 对风险分析依赖大;

优: 风险管理好; 灵活应对需求变更; 成本可控;

场景: 高风险项目; 需求易变; 大型复杂系统;

敏捷过程和极限编程模型...

2.5. 软件设计与开发方法

结构化方法: process function

面向对象: object class...

3. 0304-可行性分析

3.1. 目的

用最小的代价在尽可能短的时间内确定问题是否能解
确定能否解决问题, 同时确定问题是否值得去解决
不是解决问题, 而是确定问题是否值得解!

3.2. 任务

技术可行性

技术、资源、风险、困难

经济可行性

成本与收益

操作可行性

使用、组织、文化

社会可行性

包括法律、人权、市场、政策……

决策(方案的选择)

…

4. 0311-需求分析 1

4.1. 用户需求

4.2. 软件需求

功能需求、性能需求、可靠性需求、约束性需求

4.3. 系统需求

4.4. 软件需求项的类别

功能需求：

描述系统必须执行的功能和服务，通常与用户直接交互，具体化为系统的行为和操作。例如，用户登录、数据查询、报表生成等。

非功能需求：

描述系统的质量属性和运行特性，通常不涉及具体的用户操作，但对系统的性能、可用性、安全性等方面提出要求。非功能需求可以进一步细分为：

- (1) 性能需求：系统在处理速度、响应时间、吞吐量等方面的要求。
- (2) 可用性需求：系统的可靠性、易用性、维护性等方面的要求。
- (3) 安全性需求：系统在数据保护、访问控制、认证等方面的要求。
- (4) 兼容性需求：系统在与其它软件或硬件兼容方面的要求。
- (5) 可移植性需求：系统在不同平台上的可移植性要求。

4.5. 如何描述需求

自然语言

图文结合

功能：输入+输出+约束

非功能：简洁、清晰、肯定的自然语言

需求获取的内容（12 点）

5. 0313-需求分析 2（需求获取）

5.1. 需求获取方法

访谈

不利于理解、难表达

讲故事
观点视角
仿真
调查表
网络搜索

5.2. 结构化分析的组成

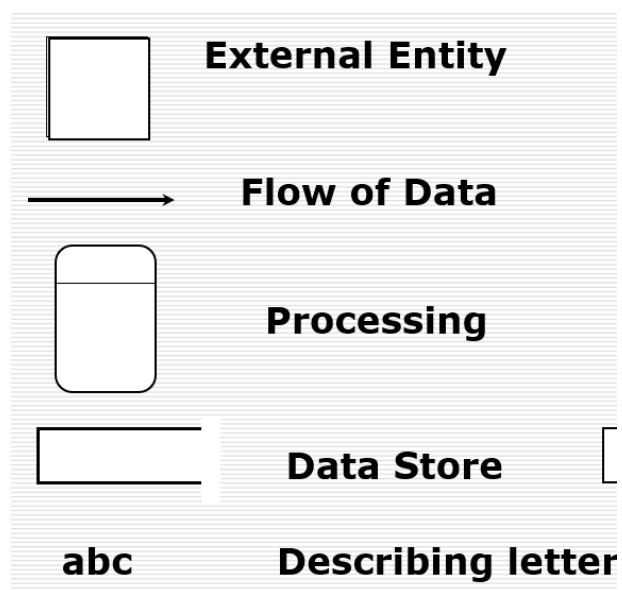
ER 图、DFD、状态转移图

6. 0318-需求分析（DFD）

6.1. DFD



描述信息流动（非流程）



processing – 数据转化 至少有一个输入和一个输出

data store - database

eg1-p23 eg2-p29

注意：输入输出不同、对象唯一、db 和 process 联系
分级、注意命名、从左到右

eg-p48

缺点：

(1) 无法提供精确和详细的定义

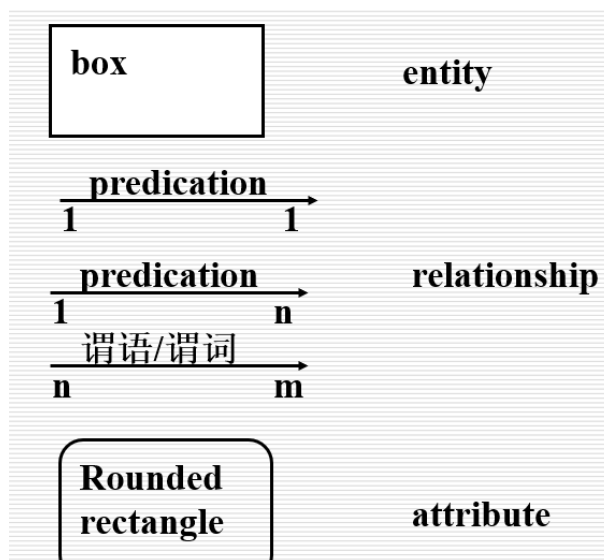
(2) DFD 只是一个静态的表示方法，用于描述系统的状态和数据流动。它不能用于动态模拟系统的行为或性能，也无法进行运行时的验证。

6.2. 数据字典

和数据流图共同构成系统的逻辑模型，是“需求说明书”的主要组成部分

<u>Notation</u>	<u>Meaning</u>
=	is composed of
+	and
[]	either-or
{ } ⁿ	n repetitions of
(...)	optional data
* ... text ...*	delimits a comment

6.3. ER 图



属性：椭圆；联系：菱形

7. 0325-需求分析 4 (UML)

面向对象的分析方法

7.1. 和结构化的区别

数据/函数 是否分离

面向功能 面向实体关系

7.2. UML – 0325 基础教程

用例图 egp9

小人（用户）+要做的事（用例）

关系		解释	图
参与者与用例之间的关系	关联	表示参与者与用例之间的交互，通信途径。 (关联有时候也用带箭头的实线来表示，这样的表示能够显示地表明发起用例的是参与者。)	
用例之间的关系	包含	箭头指向的用例为被包含的用例，称为包含用例；箭头出发的用例为基用例。包含用例是必选的，如果缺少包含用例，基用例就不完整；包含用例必须被执行，不需要满足某种条件；其执行并不会改变基用例的行为。	
	扩展	箭头指向的用例为被扩展的用例，称为扩展用例；箭头出发的用例为基用例。扩展用例是可选的，如果缺少扩展用例，不会影响到基用例的完整性；扩展用例在一定条件下才会执行，并且其执行会改变基用例的行为。	
参与者之间的关系	泛化	发出箭头的事物“is a”箭头指向的事物。泛化关系是一般和特殊关系，发出箭头的一方代表特殊的一方，箭头指向的一方代表一般一方。特殊一方继承了一般方的特性并增加了新的特性。	

类图 egp21

反映类的构成

类名+数据成员+成员函数

菱形箭头-聚合、组合

实心箭头-继承、接口

虚线箭头-绑定、友元

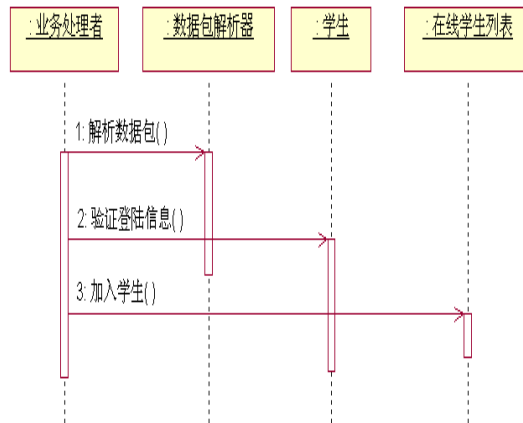
状态图 egp37

类似 DFA

状态	上格放置名称，下格说明处于该状态时，系统或对象要做的工作(见可选活动表)	
转移	转移上标出触发转移的事件表达式。如果转移上未标明事件，则表示在源状态的内部活动执行完毕后自动触发转移	
开始	初始状态(一个)	
结束	终态(可以多个)	

定义状态、定义转移条件、初始和终态、拒绝（出错）

时序图（顺序图） egp30



事物名称	解释	图
参与者	与系统、子系统或类发生交互作用的外部用户(参见用例图定义)。	
对象	顺序图的横轴上是与序列有关的对象。对象的表示方法是：矩形框中写有对象或类名，且名字下面有下划线。	
生命线	坐标轴纵向的虚线表示对象在序列中的执行情况(即发送和接收的消息，对象的活动)这条虚线称为对象的“生命线”。	
消息符号	消息用从一个对象的生命线到另一个对象生命线的箭头表示。箭头以时间顺序在图中从上到下排列。	同步 异步 简单 同步/异步销毁

横轴（下划线）：参与者（1个）+对象、类（若干）

纵向虚线：生命线

横向箭头：消息符号（上下有先后顺序）

8. 0327-需求分析 5（面向对象）

8.1. UML 分析模型

功能模型（做什么）

用例建模——**用例图**（角色+关联+用例）

egp13

对象模型（结构）

egp29

类图（完整的动作、聚合组合关系）

动态模型（什么时候做什么）

顺序图+状态图 egp61

编写脚本

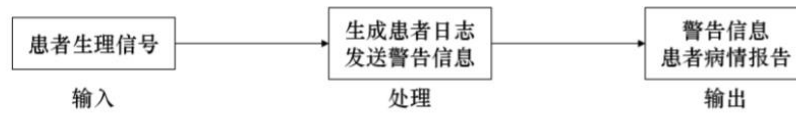
9. 0401-需求分析 6（规范）

9.1. IPO 图

IPO 图是输入加工输出（Input Process Output）图的简称

在系统的模块结构图形成过程中，产生了大量的模块，在进行详细设计时开发者应为每一个模块写一份说明。IPO 图就是用来说明每个模块的输入、输出数据和数据加工的重要工具。

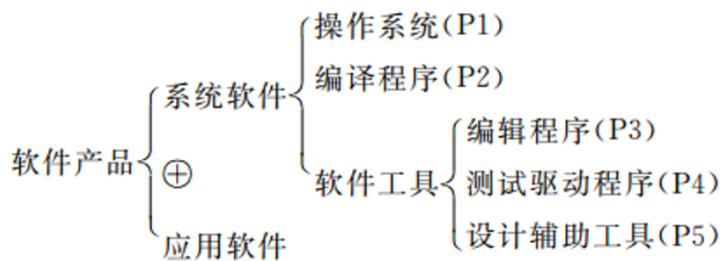
IPO 图:



9.2. Warnier 图

Warnier图^Q

Warnier图用树形结构描绘信息



10. 0408-结构设计 1

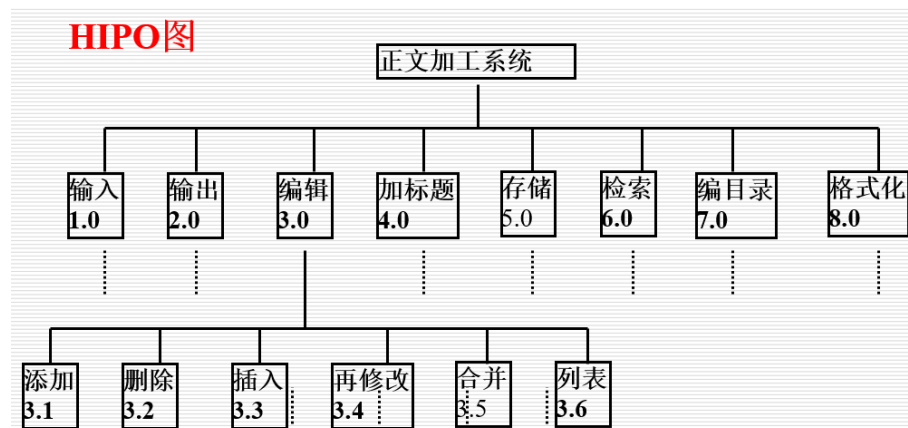
总体设计

分模块、设计功能、明确调用关系、传递的数据、评价

软件体系结构定义了软件局部和总体计算部件的构成。

10.1. HIPO 图

分层级描述功能



10.2. 经验方法

划分子系统

函数导向、组织管理导向...

广播、中断

体系结构风格、设计模式、框架

10.3. 客户机/服务器体系结构

应用系统分层

表示层——交互——显示逻辑

功能层——运算——应用处理
数据层——增删改——数据管理

举例

胖瘦客户机

3 层 BS 硬件结构

10.4. 分布式对象体系结构

基本的系统组件是对象

对象根据情况扮演客户机和服务器

10.5. DFD 导出总体结构

精细化 DFD（第二级分解）

确认输入输出边界（第一级分解）

逐层绘制分解结果图（软件结构）

egp71

核心：图转树

注：UML 关键类图就是总体结构

11. 0410-结构设计 2（模块）

模块化原理、独立性度量、设计经验

11.1. 模块设计原理

模块化

不复杂、易理解、重用、易实现

抽象

集中考虑和当前问题有关的主要方面，而忽略和当前问题无关的方面

信息隐蔽

模块的数据对外不可见

模块独立

追求高内聚低耦合

耦合：模块间相互依赖的程度、互连程度

调用、返回、进入、跳出

- (1) 无直接耦合：联系通过主模块的调用
- (2) 数据耦合（尽量）：相互传参，松散，eg 函数返回
- (3) 控制耦合（少）：向下传参；更复杂：上移判定、分解功能
- (4) 公共耦合（限制）：全局数据共享
- (5) 内容耦合（避免）：模块间相互访问内部数据，最差

内聚：模块内结合的紧密程度、功能唯一程度

- (1) 偶然（避免）：无联系
- (2) 逻辑（少）：类似功能放一起 eg 公用代码段
- (3) 时间：一起执行的放一起 eg initial
- (4) 过程（尽量）：顺序依赖
- (5) 通信：输入同/输出同
- (6) 功能（优先）：一个功能

11.2. 软件结构度量

- 深度：表示控制的层数。
- 宽度：表示控制（同一层次）总跨度。
- 扇出数：指由一模块直接控制的其他模块的数目。
- 扇入数：指有多少个模块直接控制一个给定的模块。
- 上级模块
- 下级模块

独立性

分解或合并审查——高内聚低耦合
模块过多会导致接口复杂

12. 0415-详细设计

详细设计阶段的根本目标是如何实现所要求的系统，也即要对目标系统进行精确描述。

定义：系统流程图、DFD、结构化语言、形式化软件设计语言

总体设计：... / 详细设计：程序流程图、伪码

12.1. 结构程序设计

清晰第一；标准、规范的控制结构；逐步细化(1)；

自顶向下(2)；不使用 GOTO，单入单出(3)

缺点：存储容量、运行时间；

12.2. 面向对象设计

分析模型 — 设计模型

软件重用

知识、方法、软件成分

类重用：继承、多态

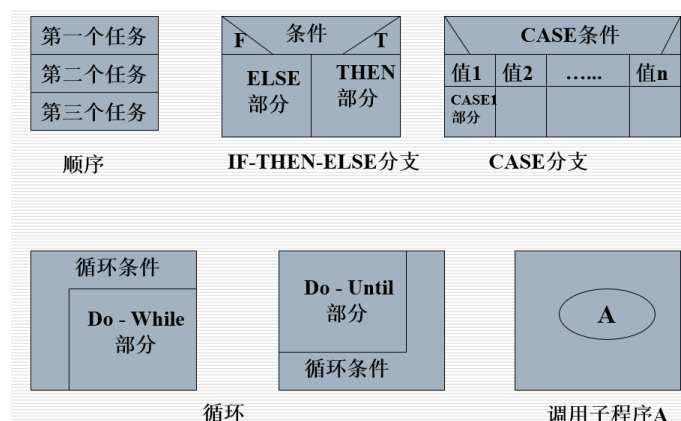
特点：对象是实体的抽象；封装；数据不共享；

程序流程图

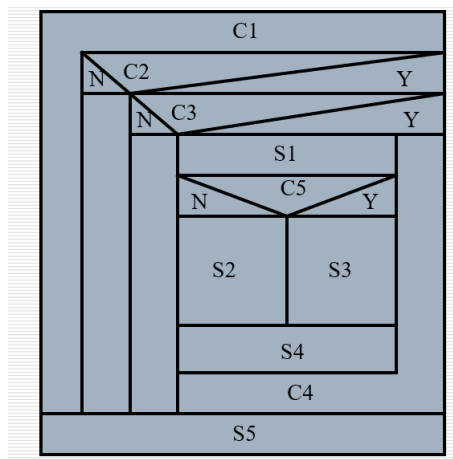
三种基本控制结构是：顺序、选择、循环

过早考虑流程，而非全局结构

N-S 盒图



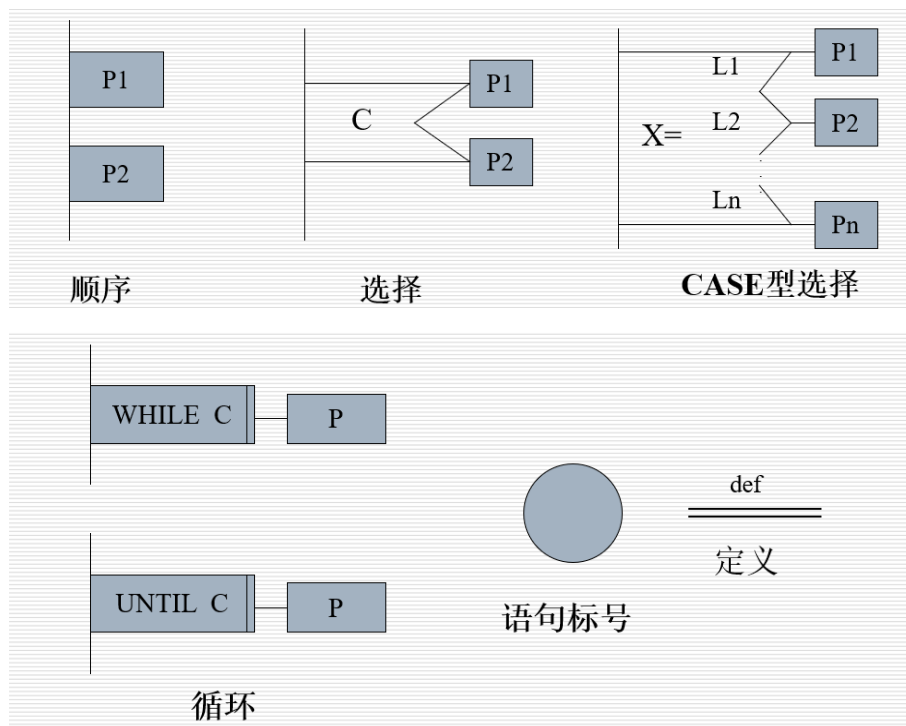
盒图转伪码 egp43:



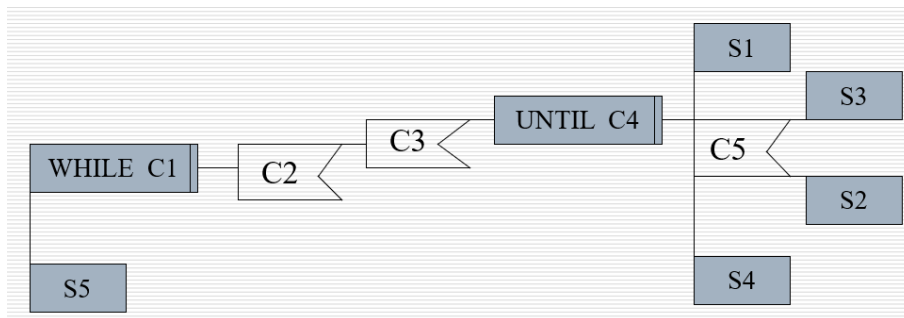
```
while c1:
    if c2:
        if c3:
            while !c4:
                s1
                if c5: s3
                else s2
                s4
            else break
        else break
```

s5

PAD 图



egp48:



判：使用表示结构化控制结构的 PAD 符号所设计出的程序必然是结构化程序；

- ✓ 既可用于表示程序逻辑，也可用于描述数据结构。
- ✓ PAD 描述的算法易译为高级程序语言。

判定表
决策树

PDL 伪码语言

PDL 虽然不是程序设计语言，但是它与高级程序设计语言非常类似
容易实现自顶向下逐步求精的设计原则
同自然语言很接近，易于理解
可以直接作为注释插在源程序中，成为程序的内部文档
半结构化的描述

13. 0422-编码与实现

13.1. 程序开发环境应该具备的特性

- ✓ 通用性：适用于不同的语言、不同的应用领域和开发方法；
- ✓ 适应性：通过开关设置，能配制出不同需要的程序设计支撑环境实例；
- ✓ 开放性：能方便地增加新工具；
- ✓ 支持复用：能支持可复用模块的存储、索引和查找；
- ✓ 自控性：保证自身操作的正确与协调；
- ✓ 自带数据库：提供数据库机制，存储、管理已开发的软件产品；
- ✓ 保证质量：有助于提高所开发软件的质量；
- ✓ 吸引用户：用户愿意使用；
- ✓ 具有市场竞争力：能真正提高软件生产力。

13.2. 程序复杂性 / 算法复杂性

行度量

v1：程序出错率的估算范围是从 0.04%~7%

v2：对于小程序，每行代码出错率为 1.3%~1.8%；对于大程序，每行代码的出错率增加到 2.7%~3.2%之间

McCabe 度量

基于程序流——环路个数

添一条有向边从出口指向入口，然后计算环的个数

环路的复杂度可加

出错率跃变复杂度值 10

Halstead 的软科学法

运算符个数、对象个数 — 程序长度

程序长度预测：

n1 不同运算符(包括保留字)的个数，

n2 不同运算对象的个数，

H 表示“程序长度”，则有

$$H = n1 \times \log_2 n1 + n2 \times \log_2 n2$$

H 是程序长度的预测值

程序潜在的错误数：

$$B = (N1 + N2) \times \log_2(n1 + n2) / 3000$$

N1：运算符数量总和

N2：对象运算数量总和

14. 0429-软件测试 1

考点：测试的方法！！

软件产品最大的成本是检测软件错误、修正软件错误的成本。

在整个软件开发中，测试工作量一般占 30%~40%，甚至≥50%。

14.1. 定义

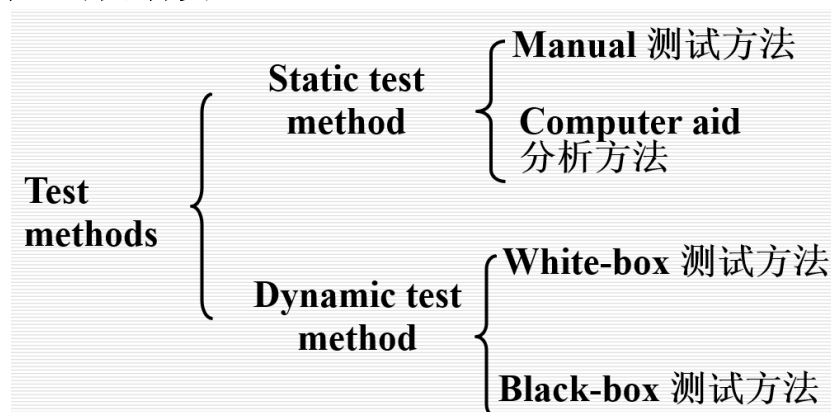
（软件）测试是带着找出错误的目的执行程序的过程。

穷尽条件、输入

14.2. 程序测试 / 软件测试

软件测试含需求分析+设计...(64%)

14.3. 如何——方法分类



静——瞪眼法

动——运行 debug

要素(2)：程序、数据

15. 0506-软件测试 2（白盒）

所有的语句和条件至少执行过一次
内部测试、结构测试、基于覆盖的测试

15.1. 选择性测试

用例设计原则

语句覆盖（点覆盖）

所有**语句**至少被执行一次（并非所有分支）

判断覆盖（分支覆盖、边覆盖）

所有分支被执行过

！区别路径覆盖

条件覆盖

每个**逻辑表达式**（一个分支判断可能含多个条件）

每个条件表达式（+取反）在所有用例中**出现 1 次**就算覆盖

判：判定覆盖和条件覆盖相互均不包含（T）

条件组合覆盖

所有**逻辑表达式的组合**均执行一次

某一条分支判断的所有组合

不同分支判断间不算组合

不能覆盖全部路径

路径覆盖

跟着流程图盖

16. 0508-黑盒测试

不看内部细节，通过外部条件完整地测试

input – component - output

输入输出测试、外部测试、功能测试、数据驱动测试

16.1. 等价类划分法

代表性数据

不同输入对于揭示错误是等效的，**暴露错误的能力是等价的**

有效等价类：指那些对于软件的规格说明书而言，是合理的、有意义的输入数据所构成的集合。

无效等价类：指那些对于软件的规格说明书而言，是不合理的、无意义的输入数据所构成的集合。

如何划分：

- （1）取值范围内、外
- （2）输入值类别(n)有限 —— 若干等价类(n)+非法类(1)
- （3）输入规则 —— 合规(1)+不合规(n)
- （4）等价类处理方式不同 —— 再分

16.2. 边界测试

针对等价类的每一个边界
需要考虑输出空间

16.3. 错误推测

经验
一定要有缺省等价类

16.4. 因果图法

组合测试
个人信息的多个复选框
适合多输入
生成判定表

因：边界界定，多因组合往后指
中间节点：为了与或非不混淆
连线：T / 连线+波浪线：F

把因果图转换为判定表							
组合条件		1	2	3	4	5	6
条件 (原因)	1	1	1	0	0	0	0
	2	0	0	1	1	1	1
	3	1	0				
	4			1	0	1	0
	5			0	0	1	1
动作 (结果)	A	1	0	0	0	0	0
	B	0	1	1	0	0	0
	C	0	0	0	1	1	0
	D	0	0	0	0	0	1
测试用例							

17. 0513-软件测试（集成）

17.1. 单元测试

以模块为单位的黑白盒
局部的数据结构——类型、未初始化、缺省、写错...
路径测试——关键的执行路径、循环控制...
模块接口——传参、调用、数据交换...
边界——边界数据、运行时间...
错误处理——定位、叙述、处理...

单元测试环境

辅助模块——模拟

驱动模块——接数据、喂数据

桩模块（存根）——替换被测模块的子模块

17.2. 集成测试

一次性组装

单元 xx 后组装

增量组装

慢慢装

自顶向下——较早地验证了主要的控制和判断点

自底向上——不需要桩

混合增量

模块 – 自底向上子模块 – 自顶向下主模块

17.3. 确认测试

有效性测试

黑盒

软件配置复查

α 测试（开发者，编码结束/模块测试后开始）

β 测试（用户，α 测试到一定阶段才开始）

17.4. 系统测试

实际的环境下

目的在于通过与系统的需求定义作比较，发现软件与系统的定义不符合或与之矛盾的地方。

功能测试

规定时间、所有功能

非功能测试

性能

是否满足在**需求**规格说明中**规定的性能或效率**

需要借助工具

强度测试

服务压力

界面测试

系统兼容

可靠性测试

...（后面都是屁话）

18. 0520-软件测试（断点调试）

18.1. 何时停止

都是屁话

18.2.可靠性分析

利用测试的统计数据来估算软件的可靠性，以控制**软件的质量**

- ✓ 推测错误的产生频度
- ✓ 推测残留在程序中的错误数
- ✓ 评价测试的精确度和覆盖率

➤ 定义: usability of software

程序在给定的**时间点**，按照规格说明书的规定，成功运行的概率。

➤ 公式

$$A_{use} = T_{up} / (T_{up} + T_{down})$$

$$A_{use} = MTTF / (MTTF + MTTR)$$

MTTF: 平均无故障时间, MTTR: 平均修复时间

MTTF 计算

ET——测试前错误

IT——软件大小

t——测试时长

Ec(t)——[0,t]纠正错误数

假设:

(1) $0.005 \leq ET / IT \leq 0.02$

(2) $MTTF \propto 1 / \text{hidden bugs}$

公式:

$$MTTF = 1 / [K(ET / IT - Ec(t) / IT)]$$

其中: K=200

$$Ec = ET - IT / (K \times MTTF) \text{——(stop rule)}$$

植入法

植入法是一种通过故意在软件中植入已知错误来估计软件总错误数的方法。

具体步骤和公式如下:

Np: 测试前故意植入的错误数

np: 在测试过程中发现的植入错误数

n: 在测试过程中发现的新错误数

N: 预测的总错误数

植入法的核心公式是: $N/n = Np/np$

从公式推导出总错误数的计算方法:

$$N = Np/np * n$$

debug

阶段上, 处于**进行了成功的测试之后**

进一步诊断和改正潜在的错误

定位+修改

分类: 强行排错法(全部 print)、回溯调试法(回退步骤)、归纳调试法(推断)、演绎调试法(瞪眼+测试)

19. 0522-软件维护

阶段：软件开发完成后

不涉及结构变化

占大头、更多创造性

19.1. 软件维护是什么？

软件维护是指在软件系统投入使用后，对其进行的各种修改、修正和改进。

具体包括以下几个方面：

程序修改：在软件投入使用后对程序进行的修改。

错误修正：在软件发布给客户后进行的修改或纠正。

系统变更：在软件运行过程中对系统进行的变更。

适应性进化：为了适应不断变化的业务条件和用户需求，对软件系统进行的演变和改进。

变更管理：管理系统变更的过程。

19.2. 为什么？

(1) 系统不断变化，需求也不断变化；

(2) 软件错误需要修正；

(3) 扩展功能需要添加；

19.3. 维护分类(4)

改正（18%）

软件缺陷、使用时的问题、潜在错误

适应（18%）

新平台、新环境、新版本(windows 10 - 11)

完善（60%）

加功能、优化用户体验

预防（4%）

可靠性、可维护性

19.4. 维护成本

全是屁话

20. 0527-软件管理 1

20.1. Gantt 图

横轴表示时间，纵轴表示活动

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。

20.2. 网络工程图

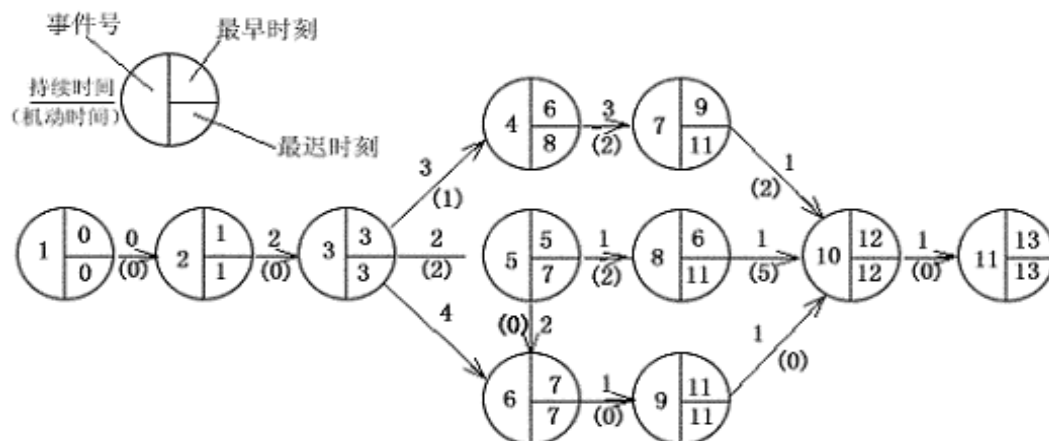


图12-2 工程网络图示例

注意是时刻

依赖关系通过表格可以表达

关键路径（最早=最迟）

所有的最早 - 所有的最迟 - 关键路径 - slack（浮动时间）

21. 0603(05)-软件管理 2(3)（团队、标准）

21.1. CMM

是什么？

- CMM 是用于衡量软件过程能力的事实上的标准，同时也是目前软件过程改进最好的参考标准。

CMM 重要概念

5 个成熟度等级：Initial, Repeatable, Defined, Managed, Optimizing

初始级（不可预测）

没有提供开发和维护软件的稳定的环境；当发生危机时，项目通常放弃计划的过程，回复到编码和测试。

可重复级（严格）

将软件项目的有效管理过程制度化，这使得组织能够重复以前项目中的成功实践；配备了基本的软件管理控制。

已定义级（标准化）

开发、维护、管理过程被文档化；对组织的标准软件过程进行裁剪，来开发它们自己的定义软件过程。

已管理级（可预测）

为软件产品和过程都设定了量化的质量目标；项目减小过程性能的变化性，使其进入可接收的量化边界，从而达到对产品和过程的控制。

持续优化级（持续改善）

关注于持续的过程改进；软件过程被评价，以防止过失重复发生，从中获得的教训散布给其它项目。