《代码大全》 全书内容摘要

Bear

2000/4/10----2000/4/27

一、创建的定义

Detail Design, Coding, Debugging, and Unit Testing

枢纽地位 不可缺少 代码往往是唯一精确描述

二、用隐喻理解

系统积累：只需要成为一个坚实的骨架结构，以便能承受将要在它之上发展的真实系统。

建造软件：如果样样都自己动手是很不明智的。Building Software 。建筑工程方法。

智能工具箱：如果你拥有的唯一工具就是一把锤子，那么你就会把整个世界都当作一个钉子。

三、软件创建的先决条件

1、重要性

优秀程序员的一个突出特点是他们采用高质量的过程来创建软件。这种过程在计划的开始，中间和末尾都强调高质量。

论据：计划是弄清楚你要干什么。食物链。一次完成是最好的选择，不必要的修改是非常昂贵的。过分的使用计算机往往与低生产率紧密相连。

2、问题定义的先决条件

弄清楚要解决的问题是什么。问题定义应该从用户的观点出发，使用用户的语言进行定义。一般来说，它不应该用计算机术语进行定义，因为最好的解决方法可能不是一个计算机程序。

3、需求分析的先决条件

需求详细描述了一个软件系统要解决的问题。

明确的需求可以保证是由用户而不是程序员决定系统的功能。

4、需求变动控制

用户对自己想要的东西，也是随着项目的进行而越来越清楚的。
以原型对付变化。
让每个人都知道由于变更需求所付出的代价。
建立一套更改控制过程。
放弃项目。

5、结构设计先决条件

软件结构设计是较高意义上的软件设计，它是支持详细设计的框架。

典型的结构要素

A、程序的组织形式
定义主要模块：一个模块是能完成某一高级功能的子程序的组合。 模块化
定义模块做什么：一个模块应该只完成一项任务而且圆满完成。 高内聚
定义模块间的界面：调用关系。 低耦合

B、变动策略

C、购买而不是建造的决定

D、主要的数据结构----存取控制，信息隐蔽

E、关键算法

F、主要对象

G、通用功能
用户界面
输入输出
内存管理
字符串存储

H、错误处理

I、 坚固性， 裕度设计， 断言， 容错性

J、 性能

好的结构设计特征：

A、对于系统中模块的讨论

B、对于模块中隐含的信息

C、选用何不选用某方案的原因

6、选择编程语言的先决条件

各语言比较

使用熟悉的语言比使用不熟悉的语言效率要高的多

7、编程约定

在高质量的软件中，你可以发现结构设计的概念完整性域较低层次实现之间的密切联系。这种联系必须与指导它的结构设计保持一致，而且，这种一致应该是内在的。这就需要实现时：

给变量和子程序命名

进行格式约定

注释约定

四、建立子程序的步骤

1、设计、工作任务

3、命名

4、决定如何测试

5、考虑效率 优化的收益主要来自高层设计,而不是个别子程序

6、研究算法或数据结构 拿来主义

7、编写 PDL 用自然语言

8、编写工作应该从抽象到具体

9、考虑数据

10、检查 PDL

11、逐步细化 --直到这样做是在浪费时间

12、编码

13、书写子程序说明

14、把 PDL 变成高层注释 用{}

15、在每一行注释下面填上代码

16、非正式地检查代码

17、进行收尾工作

18、按需要重复步骤

19、检查

20、在心里检查

21、编译

22、调试

23、除错

五、高质量子程序的特点

1、何时用子程序：

降低复杂性

避免代码段重复

限制改动带来的影响

隐含顺序

改进性能

进行集中控制

隐含数据结构

隐含全局变量

隐含指针操作

重新使用代码段

计划开发一个程序族

提高部分代码的可读性

提高可移植性

分隔复杂操作

独立非标准语言函数的使用

简化复杂的布尔测试

是出于模块化的考虑吗？绝不是，有些工作更适合放在一个大程序中

2、高质量子程序的特点：

子程序命名恰当 ---见名知意

强内聚性 功能，顺序，通讯，临时

松耦合性 参数（简单数据，数据结构，控制码），全局变量，全局数据

子程序长度 10--150 行

防错性编程 使用断言，检查输入参数，处理异常，检查返回值，防错适度

子程序参数

确保实际参数与形式参数匹配

按照输入--修改--输出的顺序排列参数

如果几个子程序中使用了相似的参数，应按照不变的顺序排列这些参数

使用所有参数

把状态和错误变量放在最后

不要把子程序中的参数当作工作变量

说明参数的接口假设

个数<=7

仅传递需要的那部分结构变量

函数是否返回了值：工作数据 or 状态

六、模块化设计

模块是指数据及作用于数据的子程序的集合

模块化设计的目标是使每个子程序都成为一个黑盒子，你知道进入盒子和从盒子里出来的是什么，却不知道里边发生了什么。它的接口非常简单，功能明确，对于任何一个特定的输入，你都可以精确地预测它相应的输出结果。

1、模块化：强内聚--松耦合

2、信息隐蔽：保密代理

3、常见的需要隐含的信息：

A、容易被改动的区域

对硬件有依赖的地方

输入和输出

非标准语言特性

难于设计和实现的域

状态变量

数据规模限制

商业规则

预防到改动

B、复杂的数据

C、复杂的逻辑

D、在编程语言层次上的操作

一般来说，在设计一组在程序语言层次上操作数据的子程序时，应该把对数据操作隐含在子程序组中，这样程序的其余部分就可能在比较抽象的层次上处理问题了。

如果不利用模块数据，就不能了解由其带来的巨大收益。那只是一个子程序。

4、建立模块的理由 模块化是面向对象的核心

用户接口

对硬件有依赖的区域

输入与输出

操作系统依赖部分

数据管理

真实目标与抽象数据类型

可再使用的代码

可能发生变动的相互联系的操作

互相联系的操作

5、检查表： 对象化

七、高级结构设计

结构化设计的层次：

1、划分子系统

2、划分成模块

3、划分成子系统

4、子程序内部的设计

结构化设计的方法：

自顶向下分解--擅长，但要求系统功能层次化，现代事件驱动，难

自底向上合成--识别细节，难于独立使用

面向对象

关键思想：一个程序模型越是真实地反映了实际问题，那么，由此产生的程序质量越好，在多数情况下，关于项目的数据定义要比功能稳定得多，因此应用面向对象设计，根据数据来进行设计，这可以使设计更稳定。

抽象：在更高层次上对问题进行考虑

封装：信息隐蔽

模块化

层次结构和继承性

四个要素：问题域，用户接口域，任务管理域，数据管理域

往返设计

在从事低层次问题时所获得的细节将为你对高层次的总体理解和做出总体设计决定奠定下良好的基础。这种在高层次和此层次之间的往返设计思维过程时非常有益的，由此而产生的结构要比..稳定得多。

设计是一个启发的过程。-- 多种方法！

受欢迎的设计的特点：

1、智力上的可管理性。

2、低复杂性。

3、维护的方便性。

4、最小的联系性。

5、可扩充性。

6、高扇入。

7、低或中等程度扇出。

8、可移植性。

9、简练性。

10、成层设计。

11、标准化技术。

八、生成数据

在你的工具箱中需要一张全部数据结构的清单，以便用最合适的方法处理每一种问题。

建立自己的指向现实实体的数据结构，已增加程序的可变动性，并使其成为自说明的。

初始化：临近初始化，再次初始化，用子程序初始化。

九、数据名称

见名知意：变量名称完整而准确地描述了变量所代表的实体。

命名约定：区分各种不同类别的数据

十、变量

尽量减少变量的作用域

使每个变量有且只有一个功能

如果全局数据不可避免的话，应通过存取子程序来对其操作

把对数据的所有存取保持在统一抽象水平

十一、基本数据类型

常数：避免奇异数（程序代码中 Hard Coding 的数字）

浮点数：避免大小加减，比较

逻辑变量：简化条件判断

枚举： 代替命名常量改善可读性，可靠性，易改动性。

指针：释放后置为空。万不得已才使用。

整数

字符串

数组

命名常量

十二、复杂数据类型

记录与结构：简化操作，参数，维护

表驱动：枚举下标多维数组

抽象数据类型：实体对象＝数据＋操作

十三、顺序程序语句

组织顺序时代码的原则是整理出依赖关系。

用合适的子程序名，参数表，注释来标明依赖关系。

如果代码没有明显依赖关系，把相关语句组织在一起，特别是使用同一参数的那些语句。

十四、条件语句

注意 if 和 else 的顺序，特别是在处理好多异常情况时，务必使正常情况流向清晰。

组织好 if-then-else 和 case 语句中的几种情况，使可读性最好。

在 case 语句中用缺省值，在 if-then-else 中的最后一个 else 中获得意外错误。

各种控制结构并不都同样有用，在编码时选用最合适的控制结构。

十五、循环语句

循环很复杂，使其简化有利于阅读。

简化循环的技巧有：避免使用怪样子循环、使循环次数最小、使进出口清楚、把内务代码放在一个地方。

循环控制变量不可滥用，应给它起一个有含义的名字并让它只有一个用途。

仔细考虑一下整个循环，保证循环在各种情况和终止条件下都能照常运行。

十六、少见的控制结构

有些情况下，goto 是编出易读易维护程序的最好方法。

问题较简单时，递归调用能把问题很巧妙地解决。要慎用递归。

十七、常见的控制问题

降低复杂性

简化判断表达式：布尔变量，子程序，决策表数组！

括号，空格，数轴

十八、布局和风格

布局原则：显示程序的逻辑结构。

评价标准：准确性，连续性，可读性，易维护性。

十九、文档

非源码+源码

对编码层文档的主要贡献不是注释，而是好的程序风格。

注释原则：好的注释是在意愿层次上进行的，解释为什么而不是是什么。

好的代码是自说明的。

做法：使用 PDL 代码流程

文件，模块，子程序，控制结构，数据说明，语句和段落

二十、编程工具

好的工具使编程更容易

你可以编写大多数你需要的工具

二十一、项目大小如何影响创建

1-3 人 占 50% 4-8 人 占 33%

在一些小项目中的活动并不能想当然地用于大项目中，你应当仔细计划它们。随着项目增大，创建作用减弱。

随着项目的增大，交流方法应简化。

相同条件下，大项目的生产率比小项目低一些。

相同条件下，大项目的每行错误数比小项目的每行错误数多。

二十二、创建管理

使用好的代码

最好代码示例，强调代码是公共财产，奖赏好的代码，一个简单标准

配置管理：修改控制，版本控制

处理落后：三等优先级，较少项目功能范围

度量

程序员和管理者都是普通人，当他们收到礼待时往往感觉更好。

合适的软件工程评估是软件开发管理最富挑战性的方面。

二十三、软件质量概述

软件外部特征：

正确性

可用性

效率

可靠性

完整性

适应性

精确性

坚固性

软件内部特征：

可维护性

灵活性

可移植性

可重用性

可读性

可测试性

可理解性

软件质量的一般原则是提高其质量并减少各种花费

IBM：如果不顾质量而只是想用最短时间将软件开发出来，往往很可能需要较长时间和花费超出。

并不是所有质量保证目标都能实现。确定你想实现的目标，并将其让你所在组的每一个人知道。

综合各种错误检查方法，尽早发现错误。

二十四、评审

总的说来，评审在发现错误方面较测试好。

评审侧重于错误发现而不是纠错。

检查表 P389

普查，代码阅读

二十五、单元测试

方法

统计规律

度量

二十六、调试

假设，定位，理解

二十七、系统集成

改进的公布法：本质是在完整连续的层次中首先并公布一个程序，并且每一层就是在一定程度上可用的程序版本。

优点：

继承照顾自己

缩短产品发货周期

在建立过程中，使客户满意

宜于获得工程状态

减少估计错误

更加平衡地分配开发、测试资源

增加工程完成的可能性

改进的方法全面提高代码质量

你能在早期发现程序是否支持修改

编码需要很少的文件编制

二十八、代码调整策略

程序结构、具体设计数据结构或算法选择对于程序规模和执行速度的影响要比产生代码本身的工作大。

优化设计关键要用到定量测量。重要的是找到确实能改善程序性能的段落，在此就要强调优化的作用。

大多程序主要的时间花在小部分代码上。在执行程序及测试前不容易知道是哪个部分代码。

为写出优良程序最好是写出清晰易读和易修改的源程序来。

二十九、代码调试策略

开始设计代码时不必过多考虑执行速度，而应从可读性入手。

提高速度的技术：

将关键子程序翻成汇编程序

用表查找替代复杂逻辑

降低内层循环的运算强度

写子程序

预先计算结果

三十、软件优化

原则：多设计小程序

减少全局变量

改进你的编程风格

变更管理

重审修订后的程序

重测试

三十一、个人性格

谦虚，好奇心，诚实，创造性和纪律，还有文明的懒惰。

好的习惯。

抽出少量时间学习别人的编程经验，过一段时间后，你将在你的同行中脱颖而出。

大师有着专家那样的专业知识，并能意识到编程只是 15%和计算机交流，其余 85%是和人打交道。一般程序员只有 30%的时间甚至更少。大师所编写的代码与其说是给计算机看到不如说是给人看的。真正的大师级程序员所编写的代码是十分清晰易懂的，而且他们注意建立有关文档。

三十二章、软件开发方法的有关问题

克服复杂性：计算机科学的核心是减少复杂性。层次，抽象。

精选开发过程。

首先为人编写程序，其次才是计算机。

注意约定使用。

根据问题范围编程。

高级问题范围

低级问题范围

计算机科学结构

高级语言结构

操作系统操作和机器指令

重视警告。

重构。重构有助于提高产品的描述，计划，设计，代码质量和其性能。

不要固执己见。开放的思想

精神：开放，学习，归纳，验证