第九章　软件实现

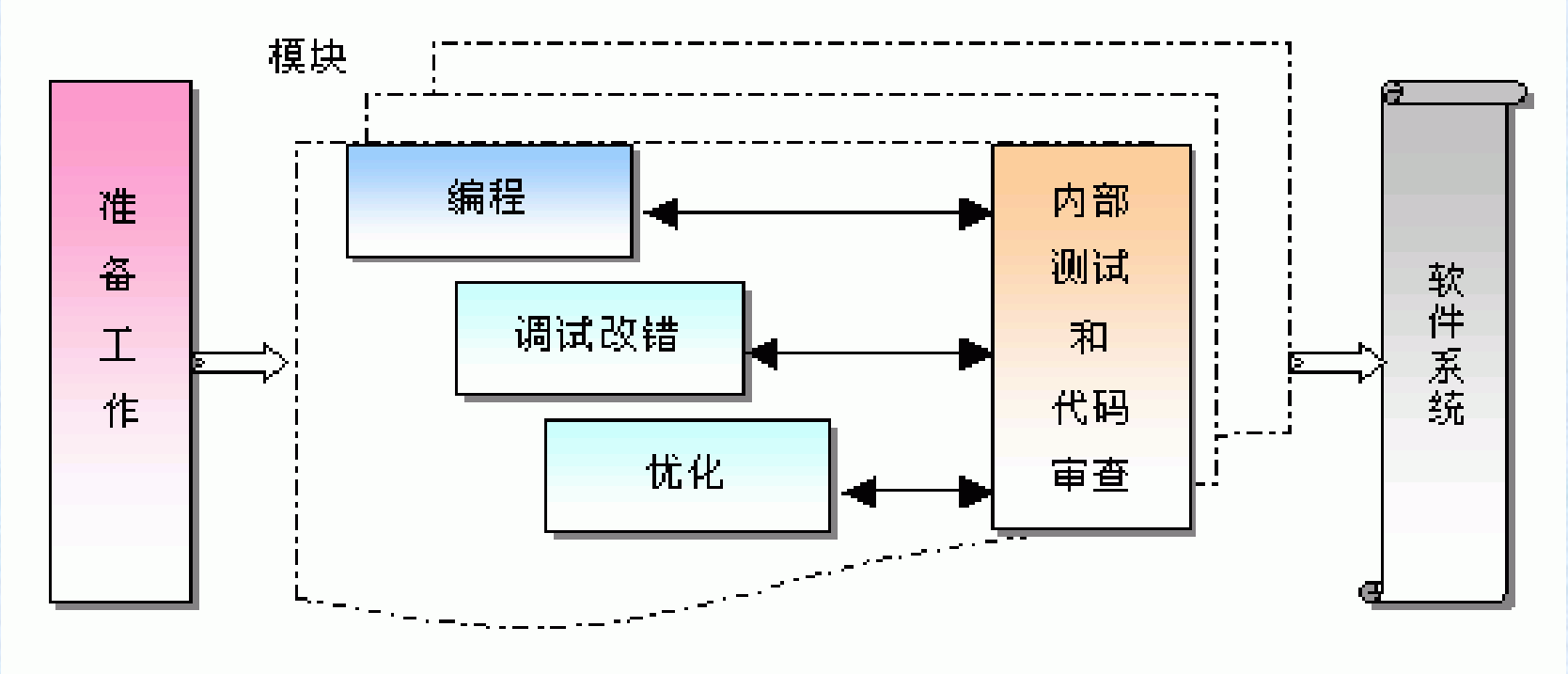
一、软件实现的流程

什么是软件实现（Software Implementation）？

软件编码和软件测试通常统称为软件实现，软件实现不等同于纯粹的编程，它是“编程、内部测试、代码审查、调试改错、优化”的综合表述。

需不需要“高手”？

目前软件开发，不能完全依赖目一“高手”。因为软件实现阶段“人多、活儿多、问题更多”，天生就有混乱倾向，因此必须制定软件实现的规范，让所有人员都按照规范执行，才可能顺利地完成任务。


模块
准备工作
编程
调试改错
优化
内部测试和代码审查
软件系统

二、 准备工作

1、准备什么？

- 开发小组制定计划。
- 开发小组确定编程、代码审查、内部测试等规范。
- 开发小组构建编程与测试环境
- 确定是否安排相应的培训。

2、制定计划

- 编程计划最重要是主线

  代码审查计划和测试计划可以根据编程计划推演出来。

- 编程计划应当根据产品的功能特征和开发人员的技能来制定。

3. 制定编程规范

- 如果没有统一的编程规范，放任程序员按照自己的风格编程的话，那么代码风格将五花八门，可能潜伏许多Bug，即使没有Bug也让别人难以理解

- 编程规范的主要用途是统一编程风格、提高代码质量，是给已经懂得编程的人用的，所以不要把编程规范写成入门教科书。
  - Internet上有许多公开的C++、Java编程规范，人们根据企业的需求适当裁剪就可以了，不必彻头彻尾地自己撰写。

一般编程规范包含以下主题：

- 文件版式、命名规则、
- 基本语句、函数设计、
- 内存管理、错误处理

- 通常每个主题都有若干的规则、建议和示例，其中“规则”是必须要遵守的，“建议”不是强制的但是推荐采用的，“示例”用来解释“规则”和“建议”。
- 制定编程规范后，应当马上组织程序员们学习，促使大家按照该规范编写应用程序，逐步形成统一的风格。千万不可让编程规范成为一种摆设。

4 技术预研

技术预研是指对项目将采用的关键技术提前学习和研究，以便尽可能早地发现并解决开发过程中将会遇到的技术障碍。

主要步骤：

- 项目经理或技术负责人识别项目中的技术难题，指定技术人员攻克该问题。
- 技术攻关人员制定简要的计划，设定技术预研的内容和目标，预计应递交的工作成果，制定进度表。
- 按照计划进行技术攻关。在任务结束时，技术攻关人员撰写《技术攻关报告》，并向相关人员介绍工作成果。

三. 编程注意事项

1、 尽可能采用成熟可靠的技术

- 应当尽可能采用成熟可靠的技术来开发软件。
- 尽量体现软件复用这种思想，
- 不要急于从零开始编程，应当先调查是否有现成的程序库可以使用
- 在编程的时候尽量少用技巧。

为什么少用技巧？

- 技巧的优点在于能另辟蹊径地解决一些问题，缺点是技巧并不为人熟知。
- 若在程序中使用太多的技巧，可能会留下错误隐患，别人也难以理解。
- 一个局部的优点对整个系统而言是微小的，而一个错误则可能对整个系统是致命的。

2 单步跟踪调试

大多数程序员有这样的习惯，等别人发现Bug后自己才去调试改错。

- 理由是：程序中正确的代码远比错误的代码多，单步跟踪调试简直就是大海捞针，还不如等别人发现Bug后再改错来得方便。

《Writing Clean Code》中极力提倡程序员应当养成单步跟踪调试的习惯。

- 即当程序员编写完成一个和几个相关程序之后，不必等别人测试，自己马上对代码进行单步跟踪调试。
- 单步跟踪调试能够发现数据溢出、内存泄漏、野指针等仅靠黑盒测试难以察觉的Bug，无疑大大地提高了程序的质量和开发、测试效率。

假设我们设计和编写200行C++代码要花费一天时间（8小时），那么对这200行代码进行单步跟踪调试大约会花费10分钟。这10分钟的单步跟踪调试不会让我们很劳累，它带来的好处是：

- （1）减少了后继的测试和改错代价（远远不止10分钟的工作量）；
- （2）让你对自己的程序更有信心，不再为未知的Bug提心吊胆

从记忆规律来说立即进行单步跟踪要比事后检测效果好的多

3、 及时写编程日记

随时随地记录你在工作中遇到的问题，以及你产生的灵感。不要等到将来再靠回忆来写总结，那时候你可能想不起来了，岂非浪费了一笔“财富”？

建议准备一个笔记本随时记录所遇到的问题。

4、 对代码进行配置管理

选择好的适合的配置管理工具

5 作息时间

编程和调试有这样的特点，干活要一鼓作气完成，不要中途停下来做其它事情之后再接着编程调试，否则思路丢失，重新捡起来很费劲。所以程序员经常在夜里干活，这样效率比较高。

注意休息，身体是革命的本钱！

四. 内部测试与代码审查

1 内部测试

- 对于严格系统，开发人员编写完成某个程序之后，先进行单步跟踪调试，然后请同伴进行代码审查和内部测试。
- 对于非严格系统，那么代码审查和内部测试的力度可以适当减弱一些。
- 如果发现缺陷，则记录在缺陷跟踪工具中，开发者应当及时修正程序，消除缺陷。

2 代码审查

- 代码审查通常在开发人员之间开展，用眼睛检查代码是否符合编程规范。
- 开发小组在执行代码审查之前要制定“代码审查表”，从编程规范中提取最重要的规则。
- 为了提高代码审查效率，检查者不必给每一个检查项填写结论，凡是正确的就跳过去，仅仅记录缺陷就行了。

为什么有了软件测试，还要代码审查呢？

（1）软件测试不能发现代码风格不统一的问题，而代码审查则很容易做到；

（2）有经验的人可以一目十行地审查代码，很快就能抓住一些Bug（主要是常见的Bug）。

五、 如何改错

- 主动
- 对症下药
  - 改错的第一步是找出错误的根源
  - 根据软件错误的症状推断出根源并不是件容易的事，因为
    - 症状和根源可能相隔很远
    - 症状可能在另一个错误被纠正后暂时性消失
    - 症状可能不是某个程序错误直接引发的，如误差累积。
    - 症状可能时隐时现，如内存泄漏。
    - 很难重新产生一样的输入条件，难以恢复“错误的现场”
    - 症状可能分布在许多不同的任务中，难以跟踪。
  - 人们把寻找错误根源的过程称为调试（debugging）。

●硬件调试改错方法

据说继承了中医的“望闻听切”诊断方法：

- 望，即用眼睛查看哪些地方是否有破损。
- 闻，即用鼻子闻哪些地方是否有烧焦的味道。
- 听，即用耳朵听哪些地方是否有异常的噪声。
- 切，即用手触摸哪些地方是否异常发烫。

●软件调试

- ◆软件调试的基本方法是“粗分细找”。
  - 对于隐藏得很深的Bug，我们应该运用归纳、推理、“二分”等方法先“快速、粗略”地确定错误根源的范围，然后再用调试工具仔细地跟踪此范围的源代码。
  - 如果没有调试工具，那么只好用“土办法”：在程序中插入打印语句如printf(...)，观看屏幕的输出。
- ◆世界上最好的调试工具恐怕是那些有经验的人。

六. 完善性工作

1 优化

- 软件的优化是指优化软件的各个质量属性，如提高运行速度，提高对内存资源的利用率，使用户界面更加友好等等。
- 想做好优化工作，首先要让开发人员都有正确的认识：优化工作不是可有可无的事情，而是必须要做的事情。

2、 写文档

- 编程工作结束后，程序员至少要撰写一些重要的文档。
- 补写或修改需求和设计文档。
- 总结编程、测试、改错的经验教训，可以写出不少专题报告，积累知识财富。
- 项目经理要有优化软件、写文档的强烈意识，然后监督组员们执行，才能把完善性工作做好。

七、程序设计风格

程序实际上也是一种供人阅读的文章，有一个文章的风格问题。应该使程序具有良好的风格

- 源程序文档化
- 数据说明
- 语句结构
- 输入 / 输出方法

1、源程序文档化

标识符的命名
安排注释
程序的视觉组织

符号名的命名

符号名即标识符，包括模块名、变量名、常量名、标号名、子程序名、数据区名以及缓冲区名等。

这些名字应能反映它所代表的实际东西，应有一定实际意义。

例如，表示次数的量用Times，表示总量的用Total，表示平均值的用Average，表示和的量用Sum等。

统一命名风格：如匈牙利命名法

程序的注释

- 夹在程序中的注释是程序员与日后的程序读者之间通信的重要手段。
- 注释决不是可有可无的。
- 一些正规的程序文本中，注释行的数量占到整个源程序的1／3到1／2，甚至更多。

注释分为序言性注释和功能性注释。

2、数据说明

在设计阶段已经确定了数据结构的组织及其复杂性。在编写程序时，则需要注意数据说明的风格。

为了使程序中数据说明更易于理解和维护，必须注意以下几点。

1. 数据说明的次序应当规范化

2. 说明语句中变量安排有序化

例：integer size, length, width, cost, price

改为

integer cost, length, price , size, width

3. 使用注释说明复杂数据结构

3、语句结构

在设计阶段确定了软件的逻辑流结构，但构造单个语句则是编码阶段的任务。语句构造力求简单，直接，不能为了片面追求效率而使语句复杂化。

- 在一行内只写一条语句
- 程序编写首先应当考虑清晰性eg:交换变量
- 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。
- 首先要保证程序正确,然后才要求提高速度。

- 尽量少用使用临时变量
- 尽可能使用库函数
- 避免不必要的转移。同时如果能保持程序可读性，则不必用 GO TO语句。
- 少用复杂的控制结构来编写程序。
- 尽量减少使用“否定”条件的条件语句
- 尽可能用通俗易懂的伪码来描述程序的流程，然后再翻译成必须使用的语言
- 不要修补不好的程序，要重新编写。也不要一味地追求代码的复用，要重新组织

4、输入和输出

●输入和输出信息是与用户的使用直接相关的。输入和输出的方式和格式应当尽可能方便用户的使用。一定要避免因设计不当给用户带来的麻烦

●在软件需求分析阶段和设计阶段，就应基本确定输入和输出的风格。系统能否被用户接受，有时就取决于输入和输出的风格。

一些原则和经验

- 数据录入的有效性检查
- 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。Eg： TAB，日期
- 应允许缺省值
- 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
- 使用图形化报表