软件维护的概念

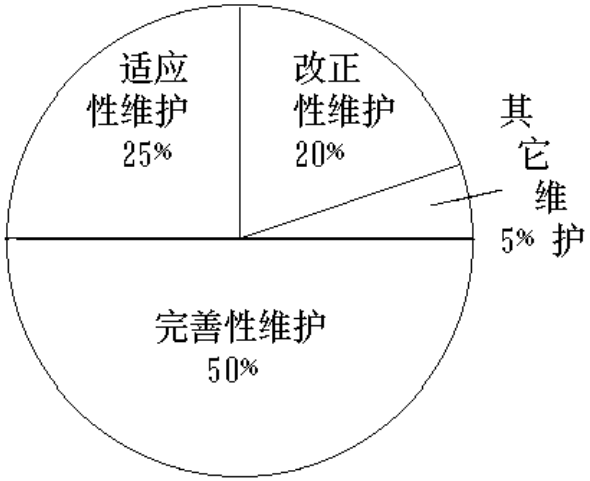
- 软件维护的定义
  - 软件运行或维护阶段对软件产品所进行的修改。
- 改正性维护
  - 在软件交付使用后，因开发时测试的不彻底、不完全，必然会有部分隐藏的错误遗留到运行阶段。
  - 这些隐藏下来的错误在某些特定的使用环境下就会暴露出来。
  - 为了识别和纠正软件错误、改正软件性能上的缺陷、排除实施中的误使用，应当进行的诊断和改正错误的过程就叫做改正性维护。

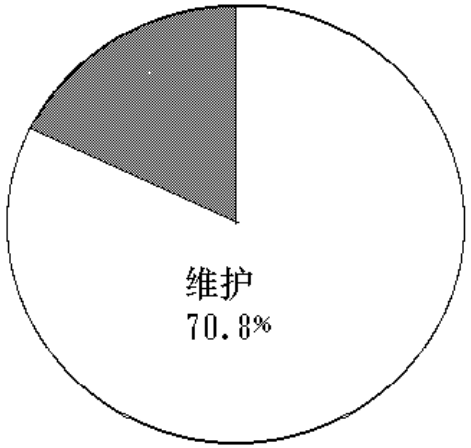
软件维护的概念

- 适应性维护
  - 在使用过程中，有可能发生以下变化：
    - 外部环境（新的硬、软件配置）
    - 数据环境（数据库、数据格式、数据输入/输出方式、数据存储介质）
  - 为使软件适应这种变化而修改软件的过程就叫做适应性维护。
- 完善性维护
  - 在软件的使用过程中，用户往往会对软件提出新的功能与性能要求。
  - 为了满足这些要求，需要修改或再开发软件，以扩充软件功能、增强软件性能、改进加工效率、提高软件的可维护性。
  - 这种情况下进行的维护活动叫做完善性维护。

软件维护的概念

- 实践表明，在几种维护活动中，完善性维护所占的比重最大。即大部分维护工作是改变和加强软件，而不是纠错。
- 完善性维护不一定是救火式的紧急维修，而可以是有计划、有预谋的一种再开发活动。
- 事实证明，来自用户要求扩充、加强软件功能、性能的维护活动约占整个维护工作的50%。



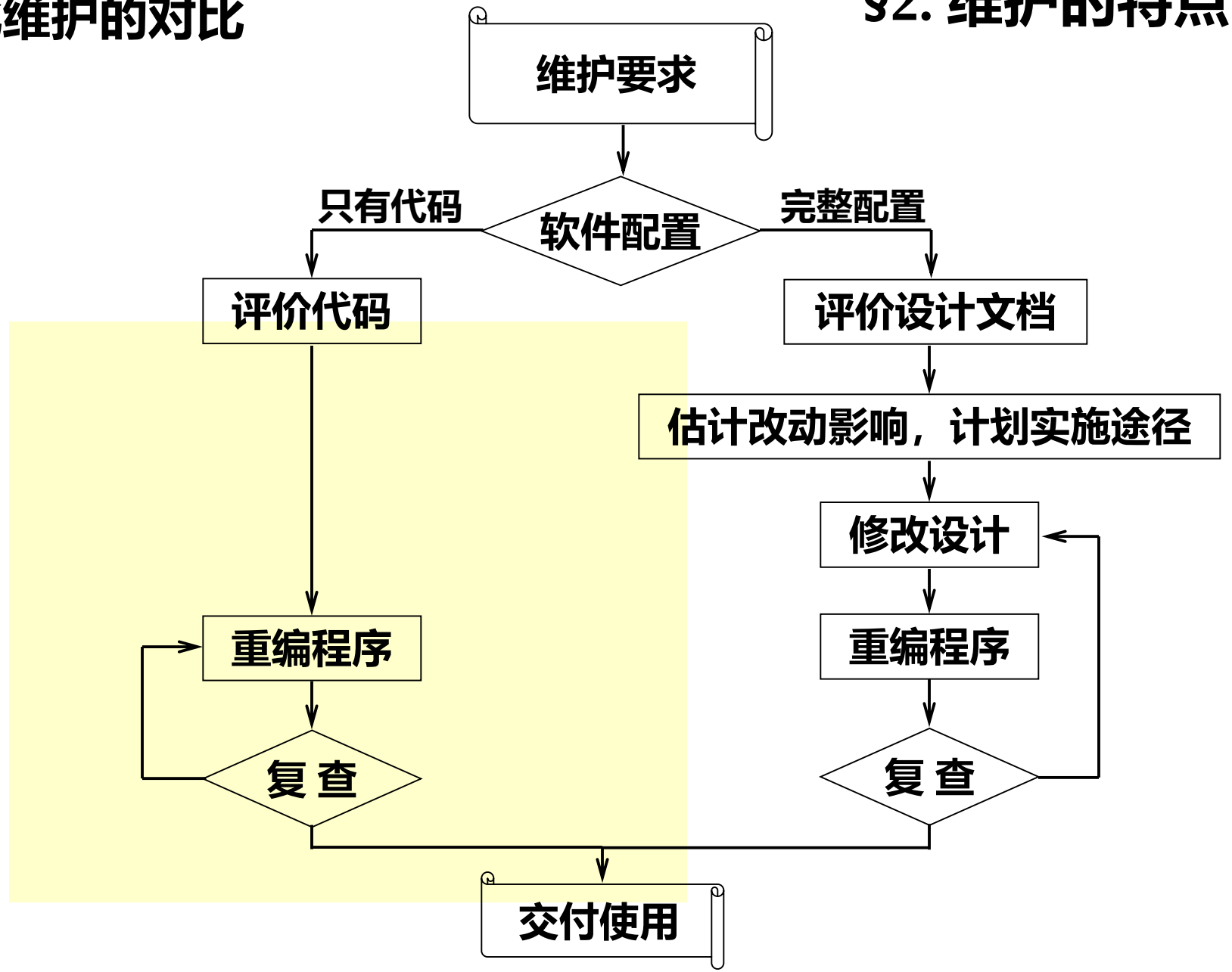


软件维护的概念

- 预防性维护
  - 预防性维护是为了提高软件的可维护性、可靠性等，为以后进一步改进软件打下良好基础。
  - 预防性维护定义为：采用先进的软件工程方法对需要维护的软件或软件中的某一部分（重新）进行设计、编制和测试。
- 在整个软件维护阶段所花费的全部工作量中，完善性维护占了几乎一半的工作量。
- 软件维护活动所花费的工作占整个生存期工作量的70%以上，这是由于在漫长的软件运行过程中需要不断对软件进行修改，以改正新发现的错误、适应新的环境和用户新的要求，这些修改需要花费很多精力和时间，而且有时会引入新的错误。

§2. 维护的特点

1、结构化维护与非结构化维护的对比



2、维护的代价

- 有形代价：费用已上升至总预算的80%；
- 无形代价：
  - 占用资源以致延误开发；
  - 修改不及时引起用户不满；
  - 维护引入新错误，降低了软件质量；等等。
- 维护工作量的经验模型：

$$M = P + K e^{c-d}$$

其中：M = 维护用的总工作量（Total maintenance effort）
P = 生产性工作量（Productive efforts (e.g. Analysis evaluation, design, coding, and testing)）
K = 经验参数（Empirical constant ）
c = 复杂度（Complexity (caused by the lack of structured design and documentation)）
d = 维护人员对软件的熟悉程度（Degree to which the maintenance team is familiar with the software.）

3、维护的问题

说明性文档不可缺少!

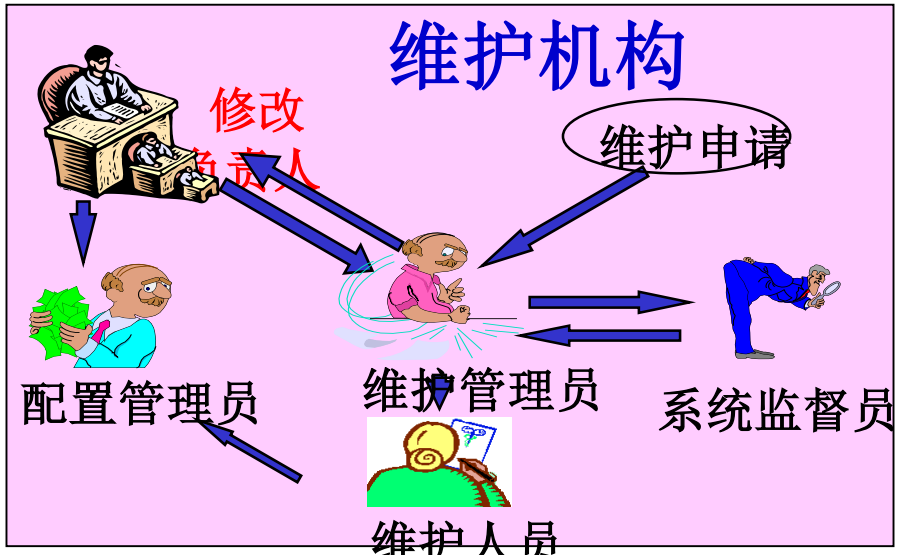
- 别人的程序很难读懂
- 文档与代码不一致
- 开发人员往往不参加维护
- 大多数软件在设计时没有考虑将来的修改

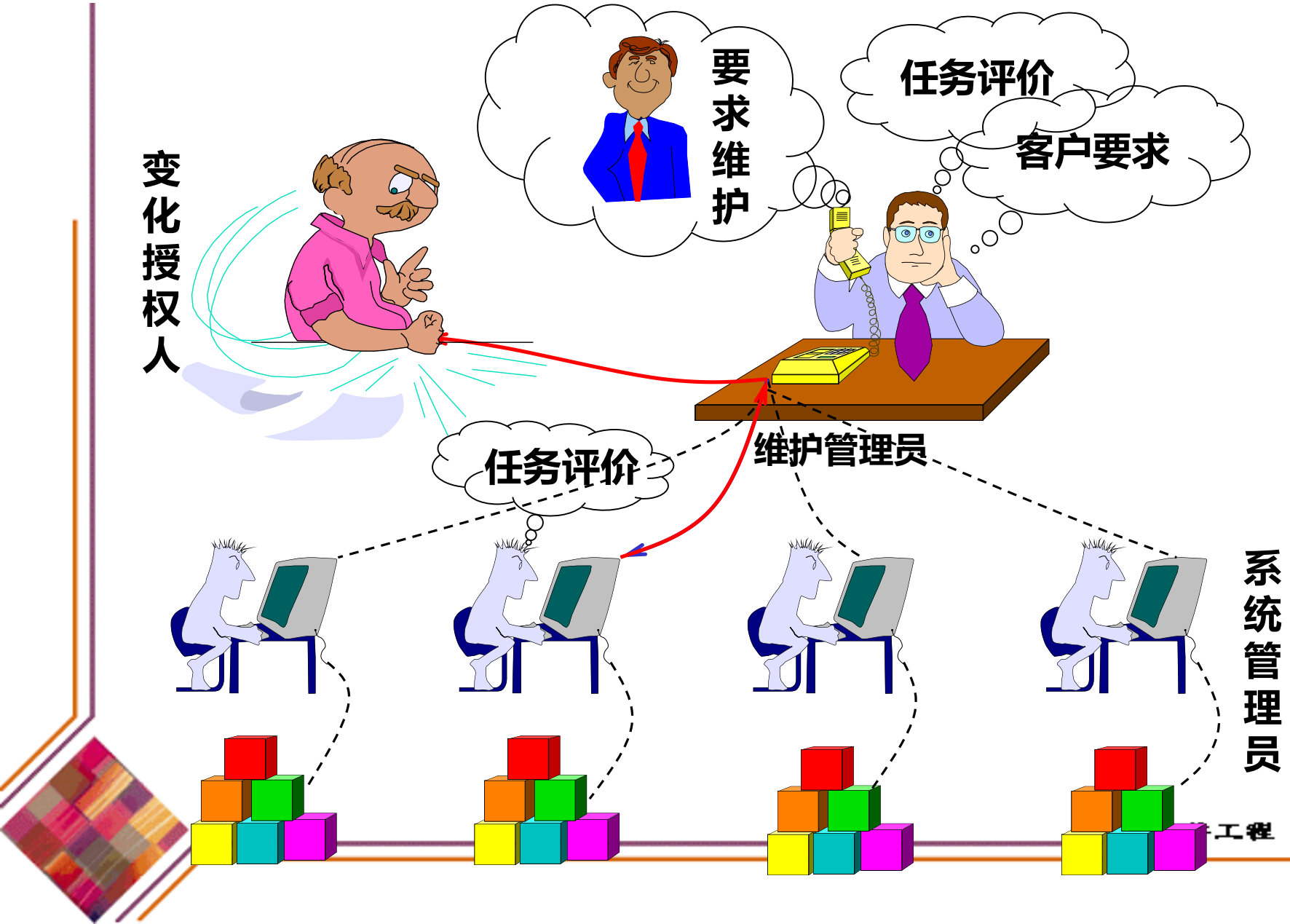
软件工程的思想至少部分地解决了与维护有关的每一个问题。

§3. 维护过程 —— 本质上是修改和压缩了的软件定义和开发过程

1、建立维护组织(maintenance team)：

在维护活动开始之前就明确维护责任是十分必要的，这样可以大大减少维护过程中可能出现的混乱




变化授权人
要求维护
任务评价
客户要求
维护管理员
任务评价
系统管理员

2、维护报告

(1) 维护申请报告(MRF, Maintenance Request Form)

由用户填写的外部文件，提供错误情况说明（输入数据，错误清单等），或修改说明书等。

(2) 软件修改报告(Software Change Report)

与MRF相应的内部文件，要求说明：

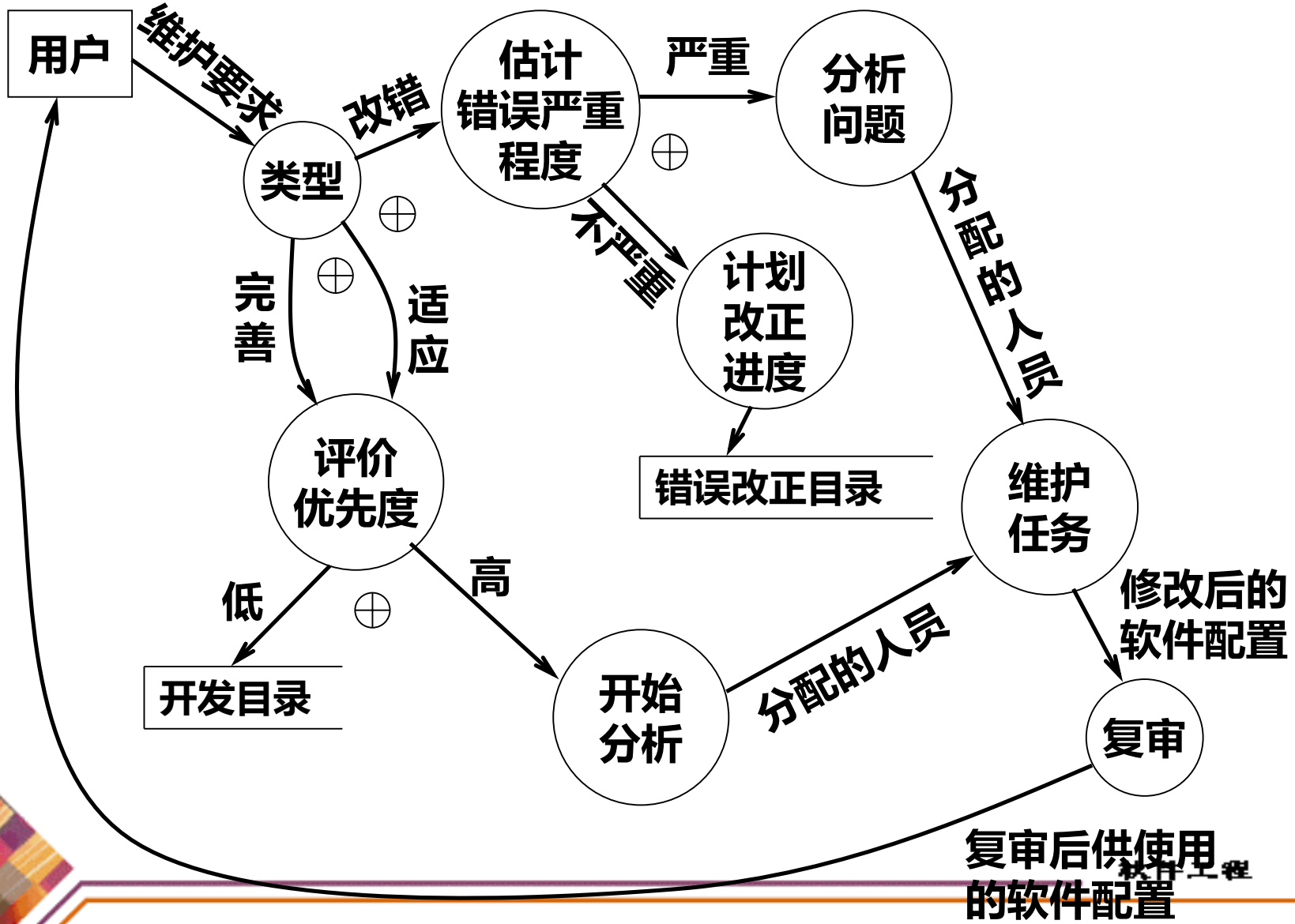
①所需修改变动的性质；

②申请修改的优先级；

③为满足某个维护申请报告，所需的工作量；

④预计修改后的状况。

3、维护的事件流


用户
维护要求
类型
改错
估计
错误严重
程度
严重
分析
问题
不严重
计划
改正
进度
分配的人员
完善
适应
评价
优先度
错误改正目录
维护
任务
低
高
修改后的
软件配置
开发目录
开始
分析
分配的人员
复审
复审后供使用
的软件配置

4、存档及评估

Q：哪些数据是值得记录的?

A： Swanson 提议的18项内容。

Q：维护工作一般从哪几方面定量度量?

A： 七个方面。

§4.可维护性(Maintainability)的度量
—— 软件度量学(Software Measurement)

软件可维护性可定性地定义为：维护人员理解、改正、改动和改进这个软件的难易程度。

1、用于衡量可维护性的软件特性：

	改正性维护	运行性维护	完善性维护
1. 可理解性	√		
2. 可测试性	√		
3. 可修改性	√	√	
4. 可靠性	√		
5. 可移植性		√	
6. 可使用性		√	√
7. 效　率			√

(1) 可理解性(Understandability)

是指由文档代码理解功能运行的容易程度。

好程序的特征：

模块化、结构化、代码与设计风格一致，高级语言实现。

度量方法：

90 - 10 Test ——读源程序10分钟，能否默写出90%？

(2)可测试性(Testability)

是指论证程序正确性的容易程度。

好程序的特征：可理解、可靠、简单。

度量方法：　　程序复杂度

(3) 可修改性(Reparability)

是指程序容易修改的程度。

好程序的特征：可理解、简单、通用。

度量方法：$D = \frac{A}{C}$

其中：D = 修改难度； A = 要修改的模块的复杂度；

C = 所有模块的平均复杂度。

$D > 1$表示修改很困难。

(4) 可靠性

(5) 可移植性(Portability)

是指程序被移到一个新环境的容易程度。

好程序的特征：结构好，不特别依赖于某一具体的计算机或操作系统。

(6) 可使用性

(7) 效率(Efficiency)

是指程序能执行预定功能，而又不浪费机器资源（包括内存、外存、通道容量、执行时间等等）的程度。

2、文档 —— 影响可维护性的决定因素，比代码更重要。

(1) 用户文档：

①功能描述 —— 说明系统能做什么；

②安装文档 —— 说明安装系统的方法及适应特定的硬件配置的方法；

③使用手册 —— 说明使用方法以及错误挽救方法；

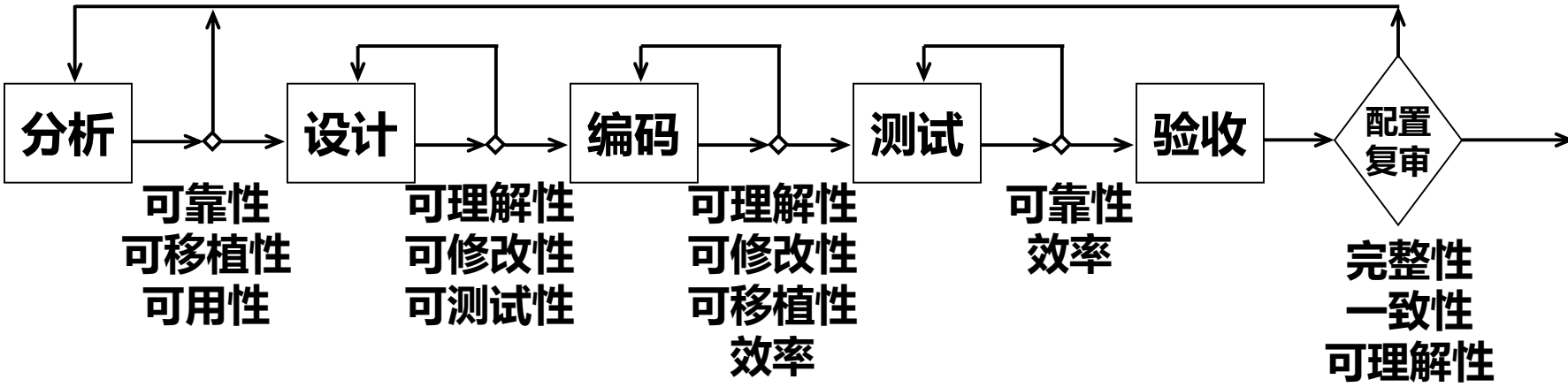
④参考手册 —— 详尽描述用户可使用的所有系统设施以及它们的使用方法；给出错误信息注解表；

⑤操作员指南（如果需要有系统操作员的话）—— 说明操作员处理使用中出现的各种情况的方法。

(2)系统文档：即软件生产过程中每一步产生的文档。

3、复审

各阶段复审重点：


分析
设计
编码
测试
验收
配置
复审
可靠性
可移植性
可用性
可理解性
可修改性
可测试性
可理解性
可修改性
可移植性
效率
可靠性
效率
完整性
一致性
可理解性

维护的副作用

- 维护的副作用指在维护过程中，因修改软件而引起的错误或其它不希望发生的行为。
- 维护的副作用大致可分为三类：
  - 代码副作用
  - 数据副作用
  - 文档副作用

(1) 修改代码的副作用

- 每次代码修改都可能引入潜在的错误。下列修改更易出错：
  - 删除或修改一个子程序
  - 删除或修改一个标号
  - 删除或修改一个标识符
  - 为提高程序的执行效率而做的修改
  - 修改文件的打开或关闭操作
  - 修改逻辑运算符
  - 由设计修改引起的代码修改
  - 修改边界条件

- 代码副作用可以通过回归测试发现，此时应立即采取补救措施。

(2) 修改数据的副作用

- 容易引起数据副作用的修改有：
  - 重新定义局部的或全局的常量
  - 重新定义记录或文件的格式
  - 增大或减小一个数组或其它复杂数据结构的体积
  - 修改全局或公共数据
  - 重新初始化控制标志或指针
  - 重新排列输入／输出表或子程序的参数表

- 数据副作用可以通过交叉访问表加以控制。交叉访问表把数据与引用它们的模块一一对应起来。

(3) 文档的副作用

- 维护时应考虑整个软件配置，在修改源程序的同时应及时修改相应的文档。如果文档未能准确反映代码的修改情况就导致文档副作用。
- 若使用手册未反映修改后的状况，那么用户在这些问题上必定会出错。
- 一次维护完成后，再次交付前应仔细复审整个配置，这可有效地较少文档副作用。