

软件工程导论

第8章 维 护

主讲教师：欧毓毅 副教授

第8章：维护

- 软件维护是软件生命周期的最后一个阶段。
- 它的任务是：维护软件的正常运行，不断改进软件的性能和质量，为软件的进一步推广應用和更新替换做积极工作。
- 软件维护所需的工作量非常大，一般说来，大型软件的维护成本高达开发总成本的四倍左右。目前，软件开发组织把60%以上的工作量用于维护自己的软件上。

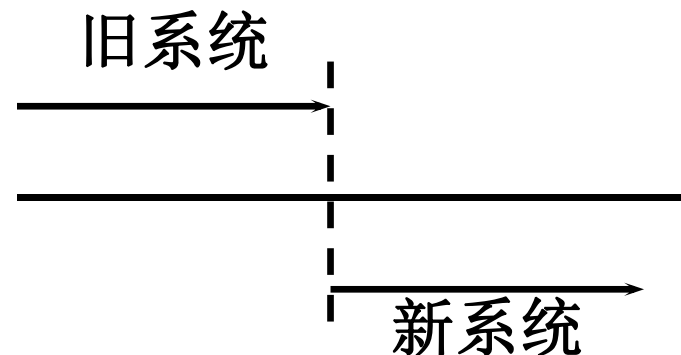
问题：软件交付使用

- 软件验收测试以后，就标志着软件设计开发阶段的结束。
- 而软件交付用户使用，才真正标志漫长的**维护**阶段的开始。
- **软件交付**使用就是新系统和旧系统的转换。
- 旧系统可能是人工作业系统，也可能是某个旧的计算机系统。
- 软件交付应该是一个**过程**，而不是一个突然事件，软件的交付使用应尽可能平稳过渡，不影响生产或工作，新系统逐步安全地取代旧系统。

- **一、软件交付使用的工作**
- **1) 将旧系统的数据转换到新系统（如数据库数据）；**
- **2) 新系统调试完成并加载入机器，准备运行；**
- **3) 将有关资料（如使用说明）转交给用户；**
- **4) 对用户做适当的培训。**

• 二、软件交付使用的方式

• 1) 直接方式



(a) 直接方式

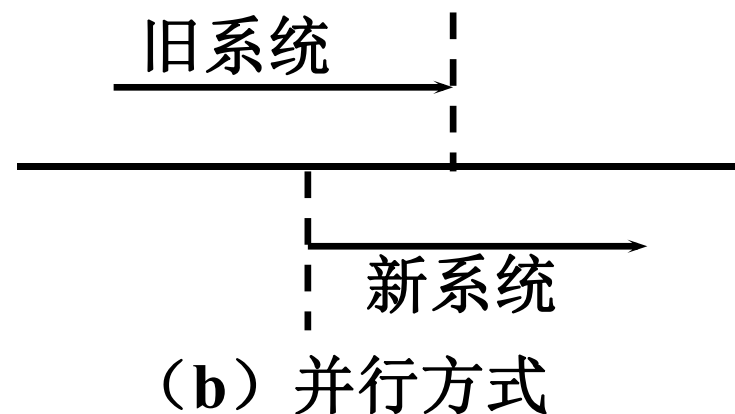
直接方式是用新系统直接替换旧系统，没有过渡。

优点：转换简单，费用最省。

缺点：风险大。

- 由于新系统没有承担过实际工作，可能会出现意想不到的问题，甚至出现程序设计错误。
- 因此，实际应用时，采取一些措施，以便新系统一旦出错，旧系统能够恢复运行。
- 直接方式不适用于一些关系重大的系统。

• 2) 并行方式



• 优点:

- A. 可以对系统进行全面测试，减少了新系统失灵带来的风险，因为旧系统也仍然存在；
- B. 用户也能够有一段熟悉新系统的时间。

• 缺点:

- 所需费用较高，双系统要投入更多的人力财力。

•3) 逐步方式

- 逐步方式是将软件**分期**，部分地交付使用。这种方式克服了上面两种方式的缺点，既能防止直接转换产生的危险性，又能减少并行方式的费用。
- 但是这种方法使得整个系统中一部分是旧系统，一部分是新系统，所以必须考虑好它们的相互**配合问题和接口问题**。
- 实际应用中，常常是**混合**以上几种方法。对系统不重要的部分采用直接方式，对系统重要部分采用并行方式，使系统平稳交付使用。

8.1 软件维护的定义

8.1.1 软件维护的原因

- 通常要求进行软件维护的原因有三种：
- 1) 改正在**特定使用条件**下暴露出来的一些潜在程序错误或设计缺陷；
- 2) 因在软件使用过程中**数据环境**发生变化（如所要处理的数据发生变化）或**处理环境**发生变化（如硬件或软件操作系统等发生变化），需要修改软件，以适应这种变化；
- 3) 用户和数据处理人员在使用时常**提出改进**现有功能、增加新功能、以及改善总体性能的要求，为满足这些要求，需要修改软件。

8.1.2 软件维护的类型

•1) 改正性维护

•交付给用户使用的软件，即使通过严格的测试，仍可能有一些**潜在的错误**在用户使用的过程中发现和修改。诊断和改正错误的过程称为改正性维护。



•2) 适应性维护

•随着计算机的飞速发展，新的硬件系统和外部设备时常更新和升级，一些数据库环境、数据输入/输出方式、数据存储介质等也可能发生**变化**。为了使软件适应这些环境变化而修改软件的过程叫做适应性维护。

•3) 完善性维护

•在软件投入使用过程中，用户可能还会有**新的**功能和性能要求，可能会提出增加新功能、修改现有功能等要求。为了满足这类要求而进行的维护称为完善性维护。

•4) 预防性维护

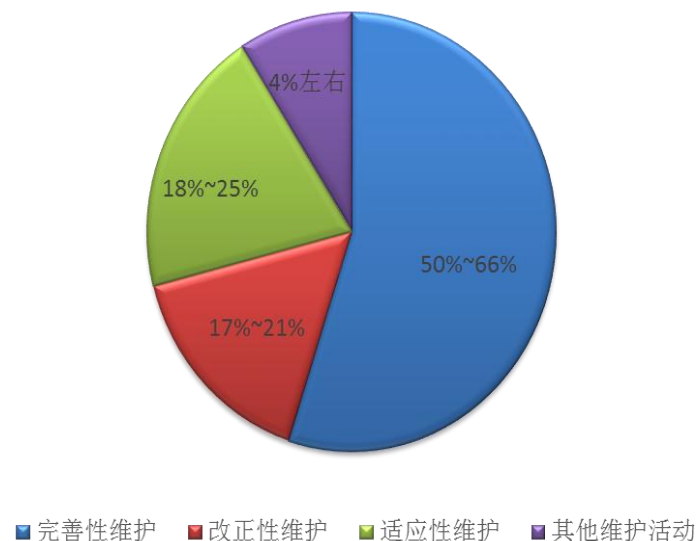
•为了改进软件未来的可维护性或可靠性，或者为了给未来的改进奠定更好的基础而进行的修改，称为预防性维护。

•这种维护活动在实践中比较少见。

• 在各类维护中，完善性维护占软件维护工作的大部分。

• 根据国外的数据统计表明，完善性维护占全部维护活动的50%~66%，改正性维护占17%~21%，适应性维护占18%~25%，其它维护活动占4%左右。

软件维护所占百分比



8.2 软件维护的特点

• 8.2.1 结构化维护与非结构化维护的差别

• 1.非结构化维护

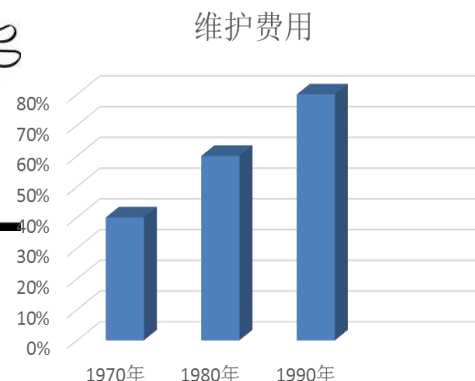
- 软件配置的唯一成分是**代码**，维护从评价程序代码开始，对软件结构、数据结构、系统接口、设计约束等常产生误解，不能进行回归测试，维护代价大。

• 2.结构化维护

- 有**完整的**软件配置，维护从评价设计文档开始，确定软件结构、性能和接口特点，现修改设计，接着修改代码，再进行回归测试。

8.2.2 软件维护的代价

1. 有形代价与无形代价



- 软件维护的代价表现为有形代价和无形代价。

- **有形代价指软件维护的费用开支。**

- 70年代，用于软件维护的费用只占软件总预算的30% ~ 40%，80年代上升到60%左右，90年代许多软件项目的维护经费预算达到了80%。

这是不是唯一的代价呢？

- **无形代价：**

- 1) 当一些看起来合理的要求不能及时满足时，会引起用户的不满；
- 2) 改动软件可能会引入新的错误，使软件质量下降；
- 3) 把许多软件工程师调去从事维护工作，势必影响开发工作。

2. 软件维护工作量模型

- 软件维护所花费的工作量，一部分用于**生产性**活动，如分析、评价、修改设计、编写程序等；
- 另一部分用于**非生产性**活动，如理解代码的含义、解释数据结构和接口特点等。（经验，文档，熟悉程度等）
- **如何构造这个工作量模型？**

- Belady和Lehman提出了一种维护工作量模型：

- $$M = P + Ke^{(c-d)}$$

- 其中：

- M：用于维护工作的总工作量；

- P：生产性工作量；

- K：经验常数；

- c：因缺乏好的设计和文档而导致软件复杂性的度量；

- d：维护人员对软件熟悉程度的度量。

- 上述模型指出：如果使用了不好的软件开发方法，原来参加开发的人员或小组不能参加维护，则工作量和成本将按**指数级**增加。

• 8.2.3 软件维护的典型问题(注：都是常见的问题)

- 别人的程序很难读懂。说明性文档不可缺少!
- 文档与代码不一致。那是给谁看呢?
- 开发人员往往不参加维护。工资不一样嘛!
- 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。

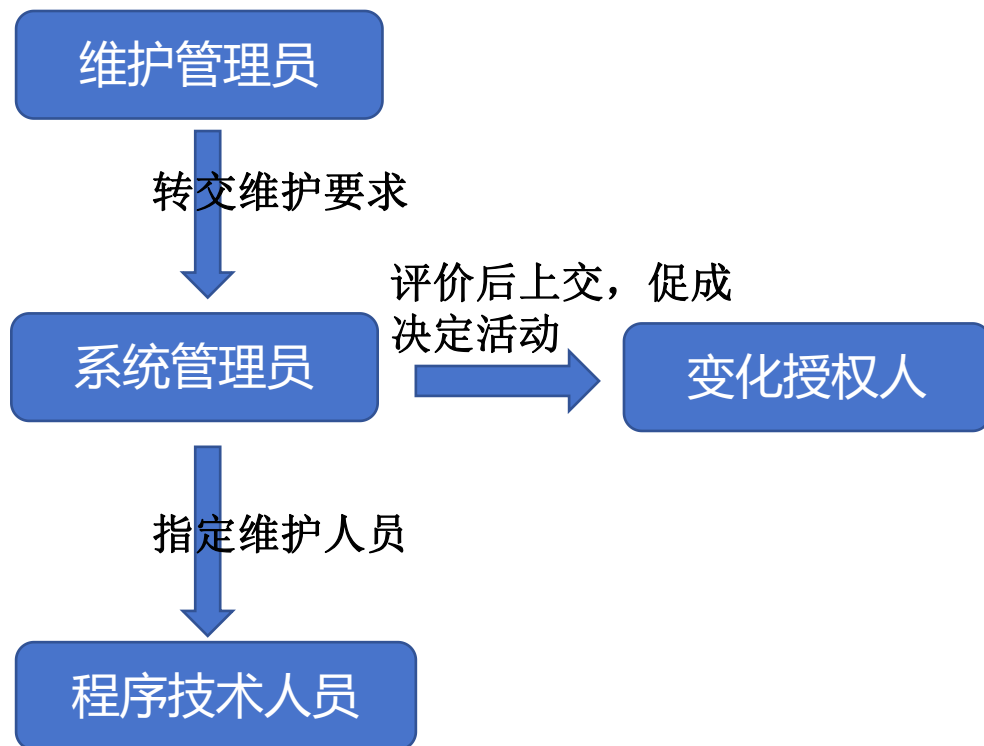
所以不是人人能发财

8.3 软件维护过程

• 1. 维护组织

虽然通常并不需要建立正式的维护组织，但是，即使对于一个小的软件开发团体而言，非正式地**委托责任**也是绝对必要的。

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



2. 维护报告

- 根据软件问题报告（维护要求），作出的软件修改报告包含的信息主要有：
 - 1) 满足维护要求表中提出的要求所需要的工作量；
 - 2) 维护要求的性质；
 - 3) 这项要求的优先次序；
 - 4) 与修改有关的事后数据（如测试数据等）。

•3. 维护的事件流



• 4. 保存维护记录

在维护时哪些数据是值得记录的？

Swanson提出了：

- 1) 程序标识；
- 2) 源语句数；
- 3) 机器指令数；
- 4) 使用的程序设计语言；
- 5) 程序安装的日期；
- 6) 自安装以来程序运行次数；
- 7) 自安装以来程序失效次数
- 8) 程序变动的层次和标识；
- 9) 因程序变动而增加的源语句数；
- 10) 因程序变动而删除的源语句数；
- 11) 每个改动耗费的人时数；
- 12) 程序改动的日期；
- 13) 软件工程师的名字；
- 14) 维护要求表的标识；
- 15) 维护类型；
- 16) 维护开始和完成的日期；
- 17) 累计用于维护的人时数；
- 18) 与完成的维护相联系的纯效益。

可以利用这些数据构成一个维护数据库的基础。


5. 评价维护活动

缺乏有效的数据就无法评价维护活动。如果已经开始保存维护记录了，则可以对维护工作做一些定量度量。

可以从以下方面度量维护工作：

- 1) 每次程序运行**平均失效的次数**；
- 2) 用于每一类**维护活动的总人时数**；
- 3) 平均每个程序、每种维护类型所做的**程序变动数**；
- 4) 维护过程中**增加或删除一个源语句**平均花费的**人时数**；
- 5) **维护每种语言**平均花费的人时数；
- 6) 一张**维护要求表**的平均周转时间；
- 7) 不同维护类型所占的**百分比**。

8.4 软件的可维护性



提高可维护性是支配软件工程方法学所有步骤的关键目标!

8.4.1 决定软件可维护性的因素

- **软件可维护性是：维护人员理解、改正和改进软件的难易程度。**
- **一个软件的可维护性，主要由5个因素决定：**
- **1. 可理解性**

可理解性表明人们通过阅读源代码和相关文档，了解程序功能及其如何运行的容易程度。

一个可理解的程序应具备以下一些特性：**模块化，风格一致性，不使用令人捉摸不定或含糊不清的代码，使用有意义的数据名和过程名，结构化，完整性等。**

2. 可测试性

- 可测试性表明论证程序正确性的容易程度。程序越简单，证明其正确性就越容易。而且设计合用的测试用例，取决于对程序的全面理解。
- 一个可测试的程序应当是可理解的，可靠的，简单的。

•3. 可修改性

- 软件的可修改程度与软件设计阶段采用的原则和策略是直接相关的。如：模块的耦合、内聚、控制范围和作用范围、局部化程度都直接影响软件的可修改性。

4. 可移植性

- 可移植性表明程序转移到一个新的计算环境的可能性的。或者它表明程序可以容易地、有效地在各种各样的计算环境中运行的容易程度。
- 一个可移植的程序应具有结构良好、灵活、不依赖于某一具体计算机或操作系统的性能。

5. 效率

- 效率表明一个程序能执行预定功能而又不浪费机器资源的程度。
- 这些机器资源包括内存容量、外存容量、通道容量和执行时间。

6. 可使用性

- 从用户观点出发，可使用性定义为程序方便、实用、及易于使用的程度。
- 一个可使用的程序应是易于使用的、能允许用户出错和改变，并尽可能不使用户陷入混乱状态的程序。

• 8.4.2 文档

- 1. 用户文档
 - 1) 功能描述;
 - 2) 安装文档;
 - 3) 使用手册;
 - 4) 参考手册;
 - 5) 操作员指南;
- 2. 系统文档

- SVN 软件:
- 配置修改记录、
代码版本管理。

- 从问题定义、需求说明到验收测试计划这样一系列和系统实现有关的文档。、系统文档的结构应该能表达系统概貌，引导到对系统每个方面每个特点的更形式化更具体的认识。

• 8.4.3 可维护性复审

- **测试结束时进行正式的可维护性复审，称为配置复审，目的是：保证软件配置的所有成分是完整的、一致的和可理解的。**

• 8.4.4 影响维护工作量的因素

- **在软件的维护过程中，花费的大量工作量会直接影响软件的成本。**
- **因此，应当考虑有哪些因素会影响软件维护的工作量，应该采取什么维护策略，才能有效地维护软件并控制维护的成本。**

- **影响软件维护工作量的因素有：**
- 1) **系统大小**。系统越大，功能越复杂，理解掌握起来就越困难，需要的维护工作量越大。
- 2) **程序设计语言**。使用功能强的程序设计语言可以控制程序的规模。语言的功能越强，生成程序所需的指令数就越少；语言的功能越弱，实现同样功能所需的语句就越多，程序就越大，维护起来就越困难。
- 3) **系统年龄**。老系统比新系统需要更多的维护工作量。许多老系统在当初并未按照软件工程的要求进行开发，没有文档，或文档太少，或者在长期的维护中许多地方与程序不一致，维护起来困难较大。

- 4) **数据库技术的应用**。使用数据库工具，可有效地管理和存储用户程序中的数据，可方便地修改、扩充报表。数据库技术的使用可以减少维护工作量。
- 5) **先进的软件开发技术**。在软件开发时，如果使用能使软件结构比较稳定的分析与设计技术（如面向对象分析、设计技术），可以减少一定的工作量。
- 6) **其它**。如，应用的类型、数学模型、任务的难度、IF嵌套深度等等都会对维护工作量产生一定的影响。

8.5 预防性维护

怎样满足用户对老程序的维护要求？

(1) 反复多次地做修改程序的尝试，与不可见的设计及源代码“顽强战斗”，以实现所要求的修改。（盲目）

(2) 通过仔细分析程序尽可能多地掌握程序的内部工作细节，以便更有效地修改它。

(3) 在深入理解原有设计的基础上，用软件工程方法重新设计、重新编码和测试那些需要变更的软件部分。（局部再工程）

(4) 以软件工程方法学为指导，对程序全部重新设计、重新编码和测试，为此可以使用CASE工具（逆向工程和再工程工具）来帮助理解原有的设计（软件再工程）

什么是老程序呢？

程序的体系结构和数据结构都很差，文档不全甚至完全没有文档，对曾经做过的修改也没有完整的记录。

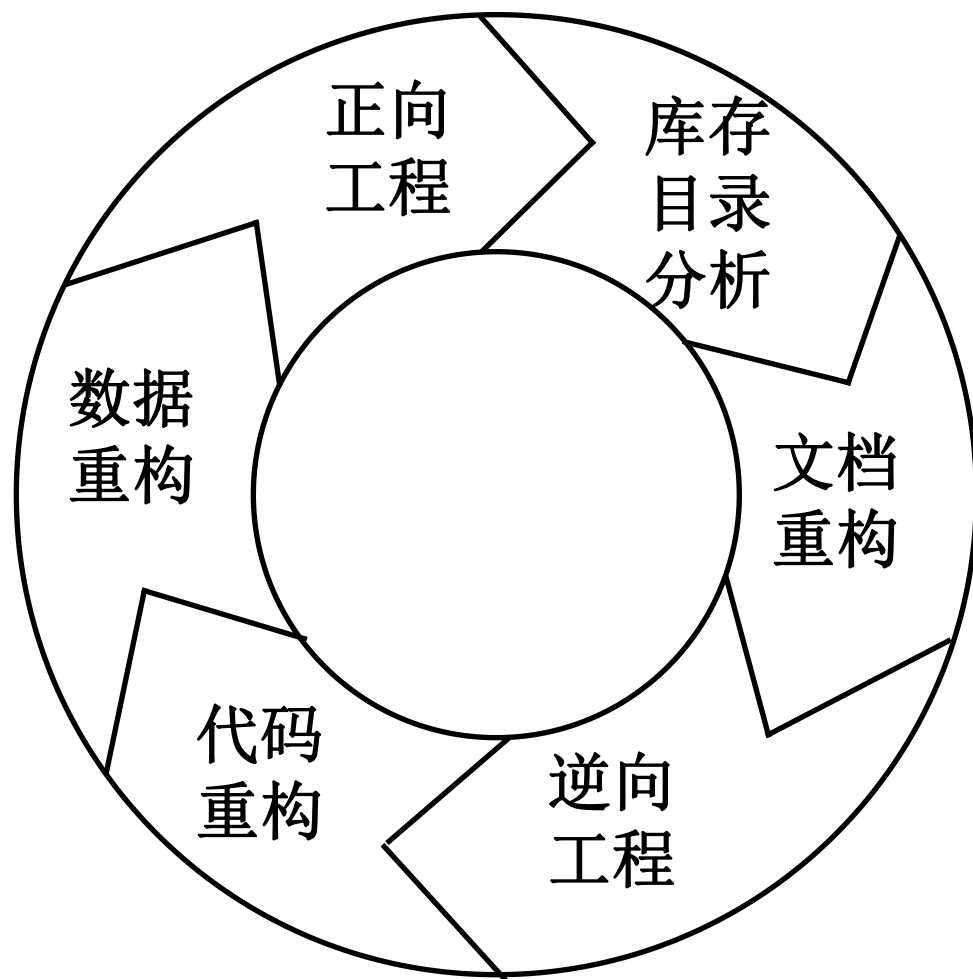
预防性维护方法是由Miller提出来的，他把这种方法定义为：“把今天的方法学应用到昨天的系统上，以支持明天的需求。”

粗看起来，在一个正在工作的程序版本已经存在的情况下重新开发一个大型程序，似乎是一种浪费。其实：

- (1) 维护一行源代码的代价可能是最初开发该行源代码代价的14~40倍。
- (2) 重新设计软件体系结构（程序及数据结构）时使用了现代设计概念，它对将来的维护可能有很大的帮助。
- (3) 由于现有的程序版本可作为软件原型使用，开发生产率可大大高于平均水平。
- (4) 用户具有较多使用该软件的经验，因此，能够很容易地搞清晰的变更需求和变更的范围。
- (5) 利用逆向工程和再工程的工具，可以使一部分工作自动化。
- (6) 在完成预防性维护的过程中可以建立起完整的软件配置。

占比很小，但是不应该忽视

8.6 软件再工程过程



- 逆向工程是指分析已有的程序，寻求比源代码更高一级的抽象形式。
- 再生工程，也称为修复和改造工程，它是在逆向工程所获信息的基础上修改或再生已有的系统，产生系统的一个新版本。
- 相关概念：
设计恢复指借助工具从已有的程序中抽象出有关数据设计、总体结构设计和过程设计的信息。

一、逆向工程恢复信息的级别

(1) 实现级: 程序的抽象语法树、符号表等信息

(2) 结构级: 如调用图、结构图

(3) 功能级: 反映程序段功能和段间关系的信息

(4) 领域级: 反映程序分量与应用领域概念间对应关系的信息。

低

抽象级别

高

信息的抽象级别越高, 它与代码距离越远, 通过逆向工程恢复的难度越大, 自动工具支持的可能性变小。

二、逆向工程的方法

反汇编、反编译

程序分析技术：程序结构分析工具
程序功能分析工具

源程序



目标代码

概要设计
详细设计



源程序

需求分析



概要设计

维护的文档

- ◇ **软件维护手册**

- 主要包括软件系统说明、程序模块说明、操作环境、支持软件的说明、维护过程的说明，便于软件的维护。

- ◇ **软件问题报告**

- 指出软件问题的登记情况，如日期、发现人、状态、问题所属模块等，为软件修改提供准备文档。

- ◇ **软件修改报告**

- 软件产品投入运行以后，发现了需对其进行修正、更改等问题，应将存在的问题、修改的考虑以及修改的影响作出详细的描述，提交审批。

小结

- 软件维护的定义（原因，类型）
- 软件维护的特点
- 维护的过程
- 软件的可维护性
- 预防性维护
- 软件再工程