软件工程
第二部分 结构化方法学
软件维护

7.1 软件维护概述
7.2 软件维护过程
7.3 软件的可维护性
7.4 软件再工程

7.1 软件维护概述

- 软件维护：在软件交付使用之后，为了改正错误或满足新的需要而修改软件的过程。软件维护的基本任务是保证软件在一个相当长的时期能够正常运行。
- 软件维护需要的工作量很大，平均说来，大型软件的维护成本高达开发成本的4倍左右。提高软件的可维护性，减少软件维护所需工作量，降低软件系统的总成本。
- 维护活动都必须应用于整个软件配置。维护软件文档和维护软件代码是同样重要的。

- 不同类型的维护活动：
  - 改正性维护：在任何大型程序的使用期间，用户必然会发现程序错误，并且把他们遇到的问题报告给维护人员，为此而进行的诊断和改正错误的过程。
  - 适应性维护：为了适应环境的变化而进行的修改软件的活动。如操作系统的变更导致的维护。
  - 完善性维护：在使用软件的过程中用户往往提出增加新功能或修改已有功能的建议。通常占软件维护工作的大部分。

  - 预防性维护：为了改进未来的可维护性或可靠性，或为了给未来的改进奠定更好的基础而修改软件。

- 经验数据表明：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%，预防性维护占4%左右。

软件维护的特点

- 非结构化维护需要付出很大代价

如果软件配置的唯一成分是程序代码，维护活动从评价程序代码开始，而且常常由于程序内部文档不足而使评价更困难。

- 结构化维护能提高维护的总体质量

维护工作从评价设计文档开始，可以充分地估量要求的改动将带来的影响，首先修改设计后在源程序代码中实现，复用测试用例进行回归测试。该维护要求系统地应用软件工程方法学。

- 维护的代价高昂

1970年用于维护已有软件的费用只占软件总预算的35%～40%，1980年上升为40%～60%，1990年上升为70%～80%。

美国空军飞行软件的例子。

开发成本：75美元/指令

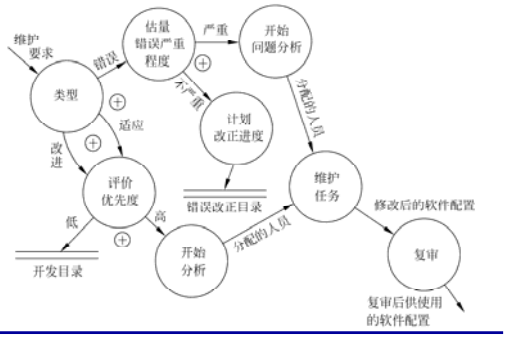
维护成本：4000美元/指令

- 软件维护的问题多数归因于软件定义和软件开发的方法有缺点

7.2 软件维护过程

- 维护过程本质上是修改和压缩了的软件定义和开发过程。维护过程中的几个要素:
  - ✓必须建立一个维护组织;
  - ✓确定维护报告和评价维护的过程;
  - ✓为每个维护要求规定一个标准化的事件序列;
  - ✓建立一个适用于维护活动的记录保管过程，并且规定复审标准。
- 维护报告,又称软件问题报告表，应该用标准化的格式表达所有软件维护要求。

- 维护的事件流



- 保存维护记录(Swanson的提议):

1)程序标识;
2)源语句数;
3)机器指令条数;
4)使用的程序设计语言;
5)程序安装的日期;
6)自从安装以来程序运行的次数;
7)自从安装以来程序失效的次数;
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。

7.3 软件的可维护性

- 可维护性：维护人员理解、改正、改动或改进这个软件的难易程度。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。
- 决定软件可维护性的主要因素:
  - ✓可理解性：理解软件的结构、功能、接口和内部处理过程的难易程度。

    模块化、详细的设计文档、结构化设计、程序内部的文档等，都对提高软件的可理解性有重要贡献。

- ✓可修改性：软件容易修改的程度。耦合、内聚、信息隐藏、局部化、控制域与作用域的关系等等，都影响软件的可修改性。
- ✓可测试性：诊断和测试的容易程度，取决于软件容易理解的程度。

  良好的文档对诊断和测试是至关重要的；软件结构、可用的测试工具和调试工具，以前设计的测试过程也都是非常重要的；对于程序模块来说，可以用程序复杂度来度量它的可测试性。

- ✓可移植性：把程序从一种计算环境（硬件配置和操作系统）转移到另一种计算环境的难易程度。

  与硬件、操作系统以及其他外部设备有关的程序代码集中放到特定的程序模块中有助于提高可移植性。C语言的打印设计例子。

  尽量选择Java语言。

- ✓ 可重用性：重用是指同一事物不做修改或稍加改动就在不同环境中多次重复使用。

大量使用可重用的软件构件对可维护性有助于：

1）可重用的软件构件在开发时经过很严格的测试，可靠性比较高，改正性维护需求越少；

2）可重用的软件构件易于修改，很容易应用到新环境中。软件中使用的可重用构件越多，适应性和完善性维护也就越容易。

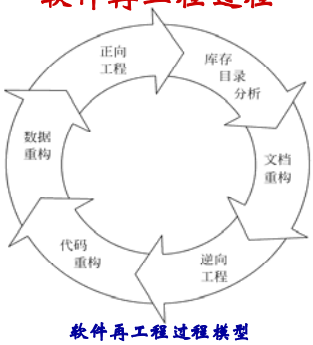
3）提高维护的效率。

- 文档是影响软件可维护性的决定因素。
  - ✓ 由于长期使用的大型软件系统在使用过程中必然会经受多次修改，所以文档比程序代码更重要。
  - ✓ 软件系统的文档可以分为用户文档和系统文档两类。用户文档主要描述系统功能和使用方法，并不关心这些功能是怎样实现的；系统文档则描述系统设计、实现和测试等各方面的内容。

7.4 软件再工程

- 软件维护面临的一个非常有挑战性问题是如何维护与修改“老程序”（软件资产问题）。预防性维护是比较有效的做法。
- 预防性维护方法是由Miller提出来的，“把今天的方法学应用到昨天的系统上，以支持明天的需求”。
- 软件再工程：以软件工程方法学为指导，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，对程序全部重新设计、重新编码和测试。

软件再工程过程



软件再工程过程模型

- 库存目录分析

保存所有应用系统的库存目录，该目录包含关于每个应用系统的基本信息。仔细分析库存目录，选出再工程的候选者，分配再工程所需要的资源。

通常应考虑如下应用程序进行预防性维护：

(1) 预定将使用多年的程序；

(2) 当前正在成功地使用着的程序；

(3) 在最近的将来可能要做重大修改或增强的程序。

- 文档重构

按照如下原则处理各种情况下的文档重构：

- ✓ 如果一个程序是相对稳定的，正在走向其有用生命的终点，而且可能不会再经历什么变化，那么保持现状。
- ✓ 为了便于今后的维护，必须更新文档，但限于有限资源，只针对系统中当前正在修改的那些部分建立完整文档。
- ✓ 如果某应用系统是完成业务工作的关键，而且必须重构全部文档，则仍然应该设法把文档工作减少到必需的最小量。

- 逆向工程

  通过分析程序，在比源代码更高的抽象层次上创建出程序的某种表示的过程。逆向工程是一个恢复设计结果的过程，逆向工程工具从现存的程序代码中抽取有关数据、体系结构和处理过程的设计信息。

- 代码重构

  首先用重构工具分析源代码，标注出和结构化程序设计概念相违背的部分；然后重构有问题的代码；最后复审和测试生成的重构代码并更新代码文档。

- 数据重构

  数据重构发生在相当低的抽象层次上，一种全范围的再工程活动。通常，数据重构始于逆向工程活动，分解当前使用的数据体系结构，必要时定义数据模型，标识数据对象和属性，并从软件质量的角度复审现存的数据结构。当数据结构较差时，应该对数据进行再工程，可能影响体系结构和代码层的改变。

- 正向工程

  正向工程过程应用软件工程的原理、概念、技术和方法来重新开发某个现有的应用系统。

小结

- 维护是软件生命周期的最后一个阶段，也是持续时间最长、代价最大的一个阶段。软件工程学的主要目的就是提高软件的可维护性，降低维护的代价（影响各个阶段活动）。
- 软件维护包括改正性维护、适应性维护、完善性维护、预防性维护。软件再工程是完成预防性维护的主要手段。
- 影响软件的可维护性的基本因素包括可理解性、可测试性、可修改性、可移植性和可重用性。文档是影响软件可维护性的决定因素。