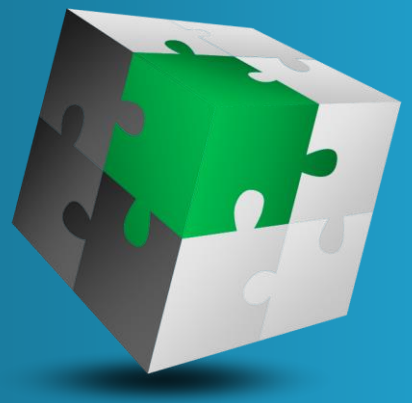
软件工程

Software Engineering



第七章 软件维护

穆海伦

计算机学院 智能与软件研究所

E-mail: helen_se@163.com QQ: 1055874556 PH: 13750802617(612617)

1 软件维护的定义

2 软件维护的特点

3 软件维护过程

4 软件的可维护性

引言

在软件产品被开发出来并交付用户使用之后，就进入了软件的运行维护阶段。这个阶段是软件生命周期的最后一个阶段，其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。目前国外许多软件开发组织把60%以上的人力用于维护已有的软件，而且随着软件数量增多和使用寿命延长，这个百分比还在持续上升。将来维护工作甚至可能会束缚住软件开发组织的手脚，使他们没有余力开发新的软件。

软件工程的目的是要提高软件的可维护性，减少软件维护所需要的工作量，降低软件系统的总成本。

1 软件维护的定义

2 软件维护的特点

3 软件维护过程

4 软件的可维护性

1 软件维护的定义

所谓软件维护就是在软件已经交付使用之后，为了改正错误或满足新的需要而修改软件的过程。

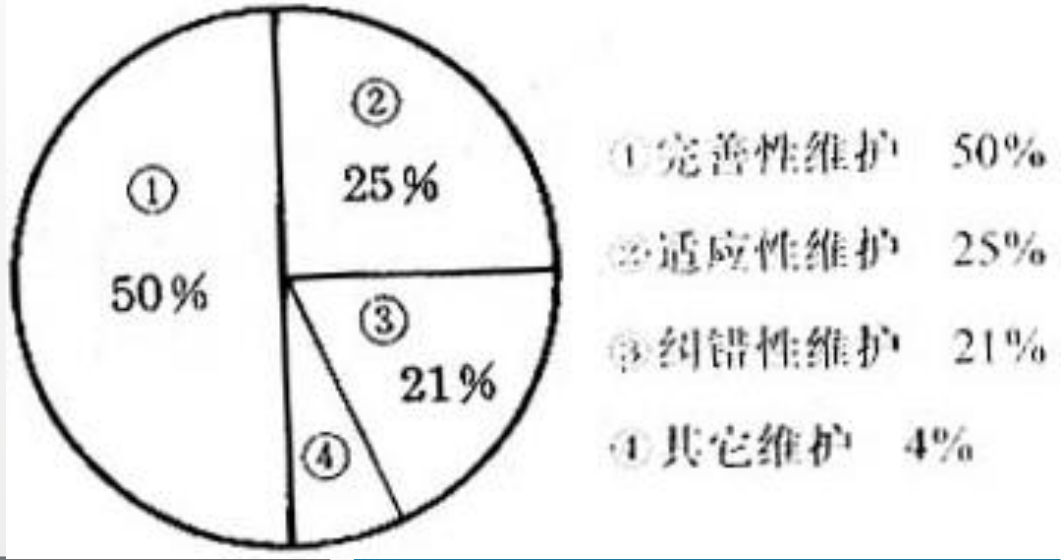
软件维护的目的

维护的目的：是满足用户对已开发产品的性能与软件环境不断提高的需求，进而达到延长软件的寿命。

软件维护的分类

可以通过描述软件交付使用后可能进行的4项活动，对软件维护进行分类：

- 完善性维护：在软件使用过程中，为了满足用户对软件的功能与性能提出新的需求而进行的维护。(50%)
- 适应性维护：使软件适应运行环境(包括软，硬件环境及数据环境)的变化而进行的维护。(25%)
- 纠错性维护：为纠正在开发期间未能发现的错误而进行的维护。(21%)
- 其他维护：（如：预防性维护）为改善软件的可维护性，可靠性等，以减少今后对其进行维护所需的工作量的工作而进行的维护。(4%)



1 软件维护的定义

2 软件维护的特点

3 软件维护过程

4 软件的可维护性

2 软件维护的特点

软件维护的特点：

- 结构化维护与非结构化维护差别巨大
  - **非结构化维护：**如果软件配置的惟一成分是程序代码，那么维护活动从艰苦地评价程序代码开始，而且常常由于程序内部文档不足而使评价更困难，对于软件结构、全程数据结构、系统接口、性能和(或)设计约束等经常会产生误解，而且对程序代码所做的改动的后果也是难于估量的：因为没有测试方面的文档，所以不可能进行回归测试(即指为了保证所做的修改没有在以前可以正常使用的软件功能中引入错误而重复过去做过的测试)。非结构化维护需要付出很大代价(浪费精力并且遭受挫折的打击)，这种维护方式是没有使用良好定义的方法学开发出来的软件的必然结果。
  - **结构化维护：**如果有一个完整的软件配置存在，那么维护工作从评价设计文档开始，确定软件重要的结构特点、性能特点以及接口特点；估量要求的改动将带来的影响，并且计划实施途径。然后首先修改设计并且对所做的修改进行仔细复查。接下来编写相应的源程序代码；使用在测试说明书中包含的信息进行回归测试；最后，把修改后的软件再次交付使用。
- 维护的代价高昂

维护费用只不过是软件维护的最明显的代价，其他一些现在还不明显的代价将来可能更为人们所关注。因为可用的资源必须供维护任务使用，以致耽误甚至丧失了开发的良机，这是软件维护的一个无形的代价。其他无形的代价还有：

- 当看来合理的有关改错或修改的要求不能及时满足时将引起用户不满；
- 由于维护时的改动，在软件中引入了潜伏的错误，从而降低了软件的质量；
- 当必须把软件工程师调去从事维护工作时，将在开发过程中造成混乱。
- 软件维护的最后一个代价是生产率的大幅度下降，这种情况在维护旧程序时常常遇到。

- 维护的问题很多

与软件维护有关的绝大多数问题，都可归因于软件定义和软件开发的方法有缺点。在软件生命周期的头两个时期没有严格而又科学的管理和规划，几乎必然会导致在最后阶段出现问题。下面列出和软件维护有关的部分问题：

- 理解别人写的程序通常非常困难，而且困难程度随着软件配置成分的减少而迅速增加。如果仅有程序代码没有说明文档，则会出现严重的问题。
- 需要维护的软件往往没有合格的文档，或者文档资料显著不足。认识到软件必须有文档仅仅是第一步，容易理解的并且和程序代码完全一致的文档才真正有价值。
- 当要求对软件进行维护时，不能指望由开发人员给我们仔细说明软件。由于维护阶段持续的时间很长，因此，当需要解释软件时，往往原来写程序的人已经不在附近了。
- 绝大多数软件在设计时没有考虑将来的修改。除非使用强调模块独立原理的设计方法学，否则修改软件既困难又容易发生差错。
- 软件维护不是一项吸引人的工作。形成这种观念很大程度上是因为维护工作经常遭受挫折。

1 软件维护的定义

2 软件维护的特点

3 软件维护过程

4 软件的可维护性

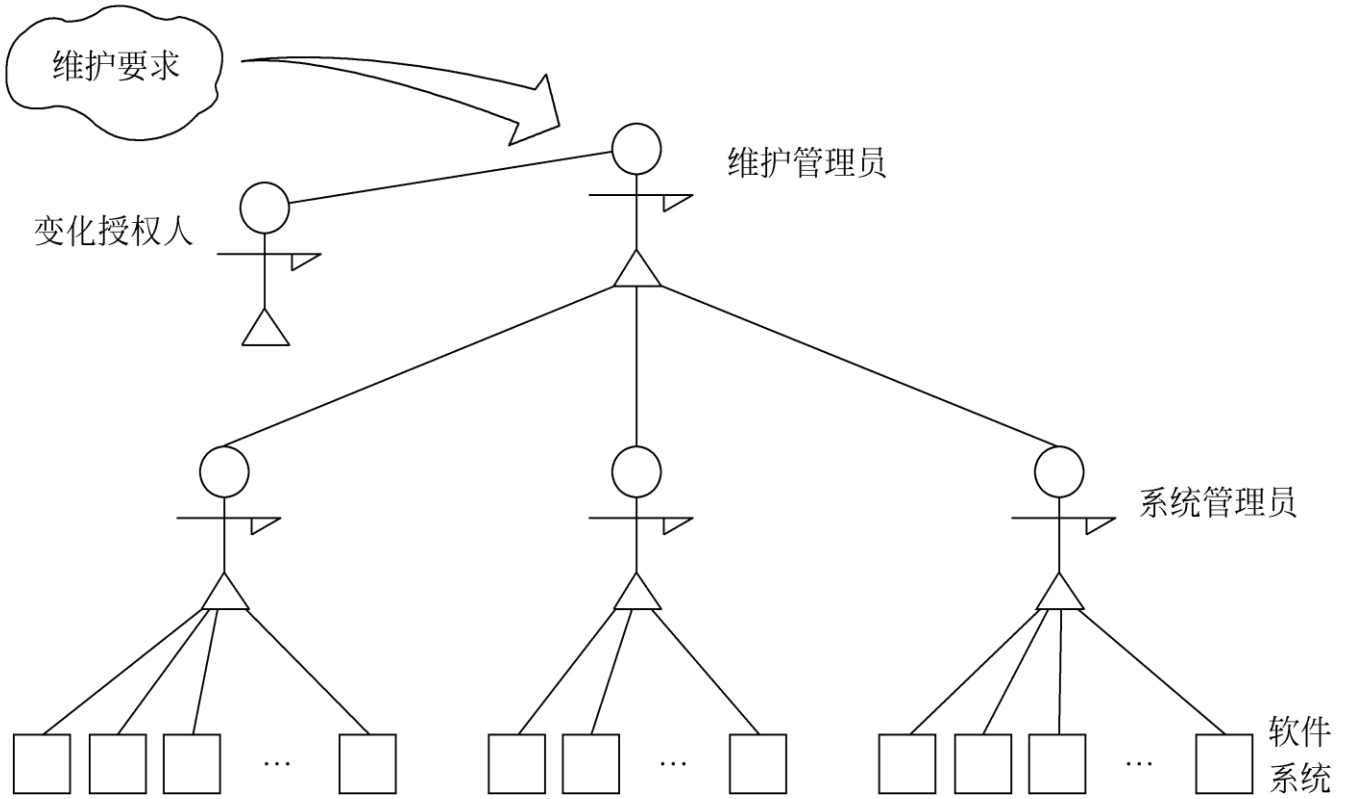
3 软件维护过程

维护过程本质上是修改和压缩了的软件定义和开发过程，而且事实上远在提出一项维护要求之前，与软件维护有关的工作已经开始了。首先必须建立一个维护组织，随后必须确定报告和评价的过程，而且必须为每个维护要求规定一个标准化的事件序列。此外，还应该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。

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。每个维护要求都通过维护管理员转交给相应的系统管理员去评价。系统管理员是被指定去熟悉一小部分产品程序的技术人员。系统管理员对维护任务做出评价之后，由变化授权人决定应该进行的活动。下图描绘了上述组织方式。

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

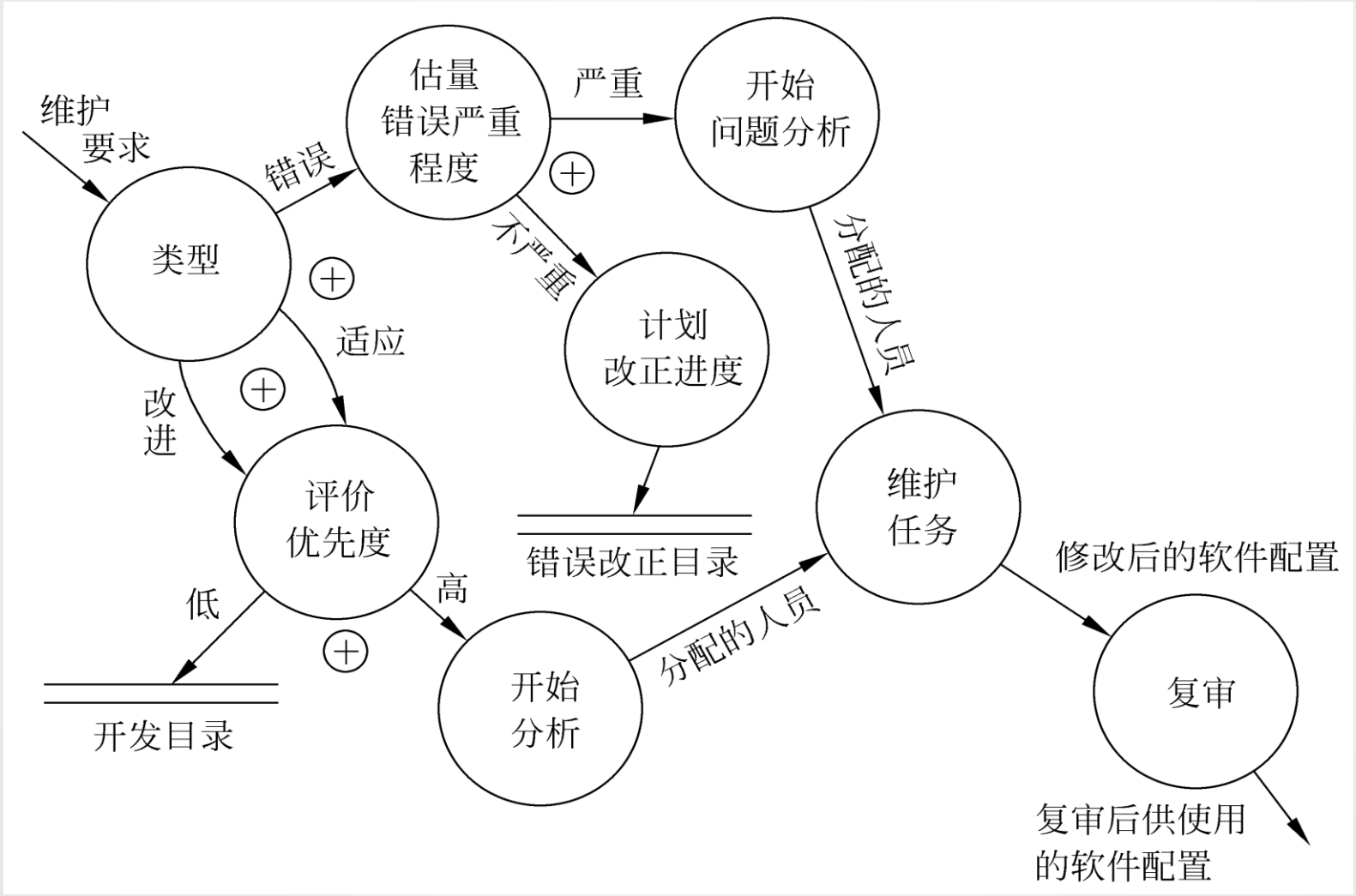
应该用标准化的格式表达所有软件维护要求。软件维护人员通常给用户提供空白的维护要求表——有时称为软件问题报告表，这个表格由要求一项维护活动的用户填写。如果遇到了一个错误，那么必须完整描述导致出现错误的环境(包括输入数据、全部输出数据以及其他有关信息)。对于适应性或完善性的维护要求，应该提出一个简短的需求说明书。如前所述，由维护管理员和系统管理员评价用户提交的维护要求表。

维护要求表是一个外部产生的文件，它是计划维护活动的基础。软件组织内部应该制定出一个软件修改报告，它给出下述信息：

- (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。

3）维护的事件流

下图描绘了由一项维护要求而引出的一串事件。



4）保存维护记录

对于软件生命周期的所有阶段而言，以前记录保存都是不充分的，而软件维护则根本没有记录保存下来。由于这个原因，往往不能估价维护技术的有效性，不能确定一个产品程序的“优良”程度，而且很难确定维护的实际代价是什么。

- 保存维护记录遇到的第一个问题就是，哪些数据是值得记录的?Swanson提出了下述内容：

  程序标识； 源语句数； 机器指令条数； 使用的程序设计语言； 程序安装的日期； 自从安装以来程序运行的次数；自从安装以来程序失效的次数；程序变动的层次和标识；因程序变动而增加的源语句数；因程序变动而删除的源语句数；每个改动耗费的人时数； 程序改动的日期；软件工程师的名字；维护要求表的标识；维护类型；维护开始和完成的日期；累计用于维护的人时数；与完成的维护相联系的纯效益。

- 应该为每项维护工作都收集上述数据。可以利用这些数据构成一个维护数据库的基础，并且像下面介绍的那样对它们进行评价。

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。至少可以从下述7个方面度量维护工作：

- (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、每种语言、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。

根据对维护工作定量度量的结果，可以做出关于开发技术、语言选择、维护工作量规划、资源分配及其他许多方面的决定，而且可以利用这样的数据去分析评价维护任务。

1 软件维护的定义

2 软件维护的特点

3 软件维护过程

4 软件的可维护性

4 软件的可维护性

是指纠正软件系统出现的错误或缺陷，以及为满足新的要求进行修改，扩充或压缩的容易程度，即衡量维护容易程度的一种属性。

可以把软件的可维护性定性地定义为： 维护人员理解、改正、改动或改进这个软件的难易程度。

影响软件可维护性的因素

维护就是在软件交付使用后进行的修改，修改之前必须理解待修改的对象，修改之后应该进行必要的测试，以保证所做的修改是正确的。如果是改正性维护，还必须预先进行调试以确定错误的具体位置。因此，决定软件可维护性的因素主要有下述5个：

- 1. 可理解性

  软件可理解性表现为外来读者理解软件的结构、功能、接口和内部处理过程的难易程度。模块化（模块结构良好，高内聚，松耦合）、详细的设计文档、结构化设计、程序内部的文档和良好的高级程序设计语言等等，都对提高软件的可理解性有重要贡献。

- 2. 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，此外，软件结构、可用的测试工具和调试工具，以及以前设计的测试过程也都是非常重要的。维护人员应该能够得到在开发阶段用过的测试方案，以便进行回归测试。在设计阶段应该尽力把软件设计成容易测试和容易诊断的。
  - 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。模块的环形复杂度越大，可执行的路径就越多，因此，全面测试它的难度就越高。

- 3. 可修改性

  软件容易修改的程度和本书第5章讲过的设计原理和启发规则直接有关。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。

- 4. 可移植性

  软件可移植性指的是，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。把与硬件、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，可以把因环境变化而必须修改的程序局限在少数程序模块中，从而降低修改的难度。

- 5. 可重用性

  所谓重用（reuse）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来开发软件，可以从下述两个方面提高软件的可维护性：
  - (1) 通常，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，且在每次重用过程中都会发现并清除一些错误，随着时间推移，这样的构件将变成实质上无错误的。因此，软件中使用的可重用构件越多，软件的可靠性越高，改正性维护需求越少。
  - (2) 很容易修改可重用的软件构件使之再次应用在新环境中，因此，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。

本章结束

ANY QUESTION