软件维护
software maintenance

内容提要

- 软件维护的定义
- 软件维护的类型
- 结构化维护VS非结构化维护
- 影响软件维护工作量的因素
- 软件维护的过程
- 可维护性
- 软件维护的管理

软件维护的定义

- 软件维护是指软件系统交付使用以后，为了改正错误或满足新的需要而修改软件的过程。

- 一般来说，要求进行维护的原因大致有以下几种：

（1）改正程序中的错误和缺陷。

（2）改进设计以适应新的软、硬件环境。

（3）增加新的应用范围。

软件维护的类型

- 根据软件维护的不同原因，软件维护可以分成4种类型：
  - 改正性维护
  - 适应性维护
  - 完善性维护
  - 预防性维护

改正性维护

- 在软件交付使用后，因开发阶段的问题以及测试得不彻底、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。

适应性维护

- 在使用过程中，
  - 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）

  可能发生变化。
- 为使软件适应这种变化，而去修改软件的过程就叫做适应性维护。

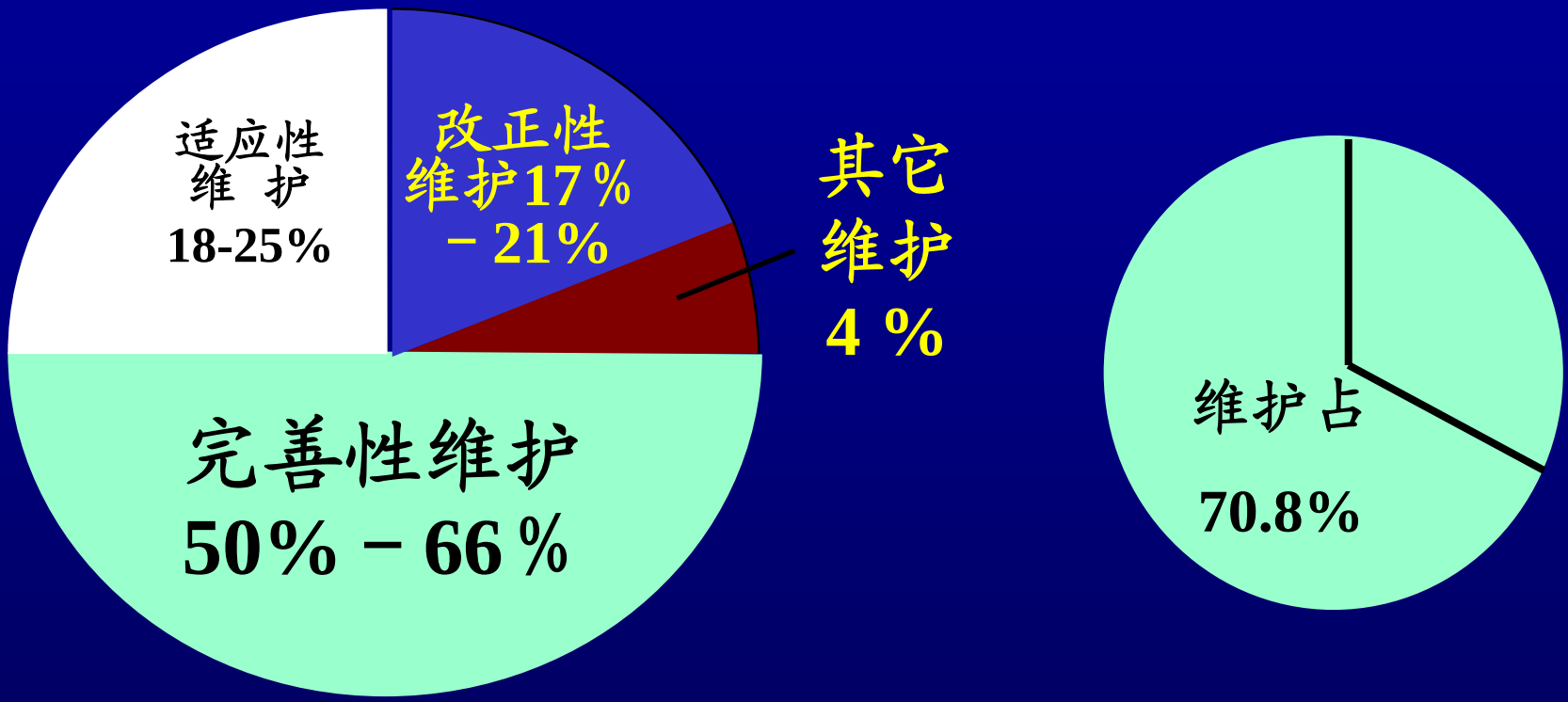
完善性维护

- 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。

预防性维护

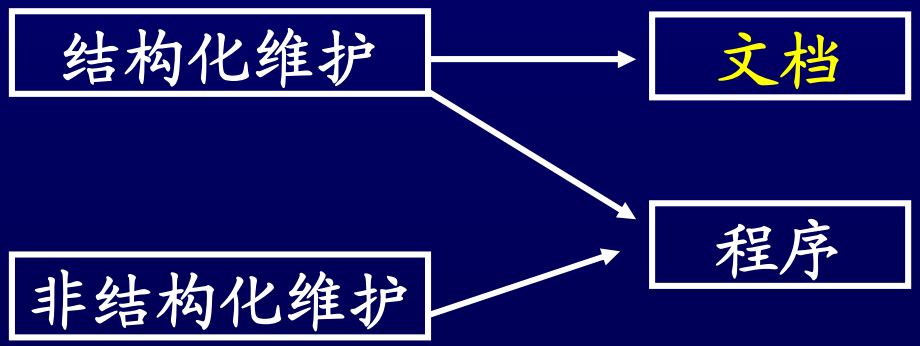
- 预防性维护即软件再工程，是为了提高软件的可维护性、可靠性等，为以后进一步改进软件打下良好基础。
- 采用先进的软件工程方法对需要维护的软件或软件中的某一部分（重新）进行设计、编制和测试，称为预防性维护。

各种维护类型和维护工作量的比例



结构化维护VS非结构化维护

- 软件的开发过程对软件的维护产生较大的影响。
  - 如果采用软件工程的方法进行软件开发，保证每个阶段都有完整且详细的文档，这样维护会相对容易，被称为结构化的维护。
  - 反之，如果不采用软件工程方法开发软件，软件只有程序而欠缺文档，则维护工作变得十分困难，被成为非结构化的维护。



非结构化维护

- 在非结构化维护过程中，开发人员只能通过阅读、理解和分析源程序来了解系统功能、软件结构、数据结构、系统接口和设计约束等，这样做是十分困难的，也容易产生误解。
- 要弄清楚整个系统，势必要花费大量的人力和物力，对源程序修改产生的后果难以估计。
- 在没有文档的情况下，也不可能进行回归测试，很难保证程序的正确性。

结构化维护

- 在结构化维护的过程中，所开发的软件具有各个阶段的文档，它对于理解和掌握软件的功能、性能、体系结构、数据结构、系统接口和设计约束等有很大的作用。
- 维护时，开发人员从分析需求规格说明开始，明白软件功能和性能上的改变，对设计说明文档进行修改和复查，再根据设计修改进行程序变动，并用测试文档中的测试用例进行回归测试，最后将修改后的软件再次交付使用。
- 这种维护有利于减少工作量和降低成本，大大提高软件的维护效率。

软件维护的代价高昂

- 下述表达式给出了维护工作量的一个模型:

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

其中，M是维护的总工作量，P是生产性工作量，K是经验常数，c是复杂程度，d是维护人员对软件的熟悉程度

- 上述模型表明，如果软件开发没有运用软件工程方法学，而且原来的开发人员未能够参与到维护工作之中，则维护工作量和费用将指数增加。

软件维护的问题

- 下面列出了和软件维护有关的部分问题:
  - 理解别人的代码通常是非常困难的，而且难度随着软件配置成分的缺失而迅速增加;
  - 需要维护的软件通常往往没有合格的文档，或文档资料显然不足。--认识到文档仅仅是第一步，容易理解且和程序保持一致的文档才是真正具有价值的;
  - 当软件要求维护时，不能指望开发人员给我们仔细说明软件。由于维护持续时间很长，因此当需要解释软件时候，往往开发人员已经不在附近了;
  - ……

上述种种问题在现有没有采用软件工程思想开发出来的软件中，都或多或少存在。

影响软件维护工作量的因素

- 在软件维护中，影响维护工作量的因素主要有以下六种：
  - 系统的大小

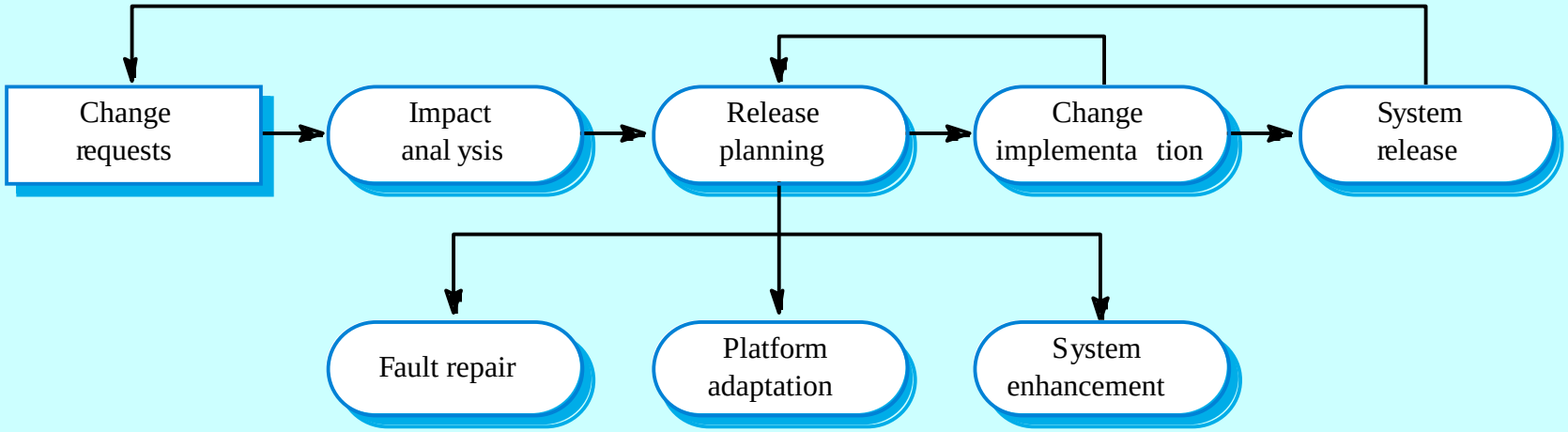
    系统规模越大，其功能就越复杂，软件维护的工作量也随之增大。
  - 程序设计语言

    使用强功能的程序设计语言可以控制程序的规模。语言的功能越强，生成程序的模块化和结构化程度越高，所需的指令数就越少，程序的可读性越好。

软件维护

- 软件维护的定义
- 软件维护的类型
- 结构化维护VS非结构化维护
- 影响软件维护工作量的因素
- 软件维护的过程
- 可维护性
- 软件维护的管理

The system evolution process



- Change implementation
  - 修改软件需求说明
  - 修改软件设计
  - 设计评审
  - 对源程序做必要的修改
  - 单元测试
  - 集成测试（回归测试）
  - 确认测试
  - 软件配置评审等。

- 维护申请提交给维护管理员，他把申请交给某个系统监督员去评价。
- 一旦做出评价，由修改负责人确定如何进行修改。
- 在修改程序的过程中，由配置管理员严格把关，控制修改的范围，对软件配置进行审计。
- 在维护之前，就把责任明确下来，可以减少维护过程中的混乱。

维护请求报告

- 应该用标准的格式来表达维护要求。软件维护人员通常提供给用户空白的维护请求表（报告）即软件问题报告，该报告（表）由要求一项维护活动的用户填写。
  - 如遇到什么错误，用户需要详细描述错误出现的现场信息(包括输入数据、列表文件和其他有关信息);
  - 对适应性维护、完善性维护应该给出一个简短的需求规格说明书。最终由维护管理员和系统管理员评价用户提出的维护请求表。

一个维护申请被核准后，维护请求表就成为外部文档，视作规划本次维护任务的依据。

软件修改报告

- 依据维护请求表，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:
  - 满足维护请求表中提出的要求所需的工作量;
  - 维护要求的性质;
  - 维护要求的优先次序;
  - 与修改有关的背景数据。
- 在拟定进一步维护计划前，把软件修改报告提交控制决策机构审查批准。

程序修改的步骤

- 在软件维护时，必然会对源程序进行修改。
- 通常对源程序的修改不能无计划地仓促上阵，为了正确、有效地修改，需要经历以下三个步骤。
  - 分析和理解程序
  - 修改程序
  - 重新验证程序

2. 修改代码，以适应变化

在修改时，要求：

(1) 正确、有效地编写修改代码；

(2) 要谨慎地修改程序，尽量保持程序的风格及格式，要在程序清单上注明改动的指令；

(3) 不要删除程序语句，除非完全肯定它是无用的；

(4) 不要试图共用程序中已有的临时变量或工作区，为了避免冲突或混淆用途，应设置自己的变量；

(5) 插入错误检测语句；

(6) 在修改过程中做好修改的详细记录，消除变更中任何有害的副作用（波动效应）；

软件维护

- 软件维护的定义
- 软件维护的类型
- 结构化维护VS非结构化维护
- 影响软件维护工作量的因素
- 软件维护的过程
- 可维护性
- 软件维护的管理

- 目前广泛使用的是用如下的七个特性来衡量程序的可维护性。

  可理解性　可使用性

  可测试性　可移植性

  可修改性　效率

  可靠性

- 而且对于不同类型的维护，这七种特性的侧重点也不相同。

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. 可靠性

- 可靠性表明一个程序按照用户的要求和设计目标，在给定的一段时间内正确执行的概率。
- 关于可靠性，度量的标准主要有：
  - 平均失效间隔时间MTTF
  - 平均修复时间MTTR
  - 有效性A = MTBD/(MTBD+MDT)

系统平均不工作间隔时间MTBD
(Mean Time Between System Downs)

平均停机时间MDT(Mean Down Time)。

3. 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。
- 用于可测试性度量的检查项目如下：
  - 程序是否模块化？结构是否良好？
  - 程序是否可理解？程序是否可靠？
  - 程序是否能显示任意中间结果？
  - 程序是否能以清楚的方式描述它的输出？
  - 程序是否能及时地按照要求显示所有的输入？
  - 程序是否有跟踪及显示逻辑控制流程的能力？
  - 程序是否能从检查点再启动？
  - 程序是否能显示带说明的错误信息？

4. 可修改性

- 可修改性表明程序容易修改的程度。
- 一个可修改的程序应当是可理解的、通用的、灵活的、简单的。
- 通用性（可配置性）是指程序适用于各种功能变化而无需修改。
- 灵活性是指能够容易地对程序进行修改。

5. 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大小。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。
- 用于可移植性度量的检查项目如下:

6. 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。
- 用于效率度量的检查项目如下:
  - 程序是否模块化? 结构是否良好?
  - 是否消除了无用的标号与表达式，以充分发挥编译器优化作用?

7. 可使用性

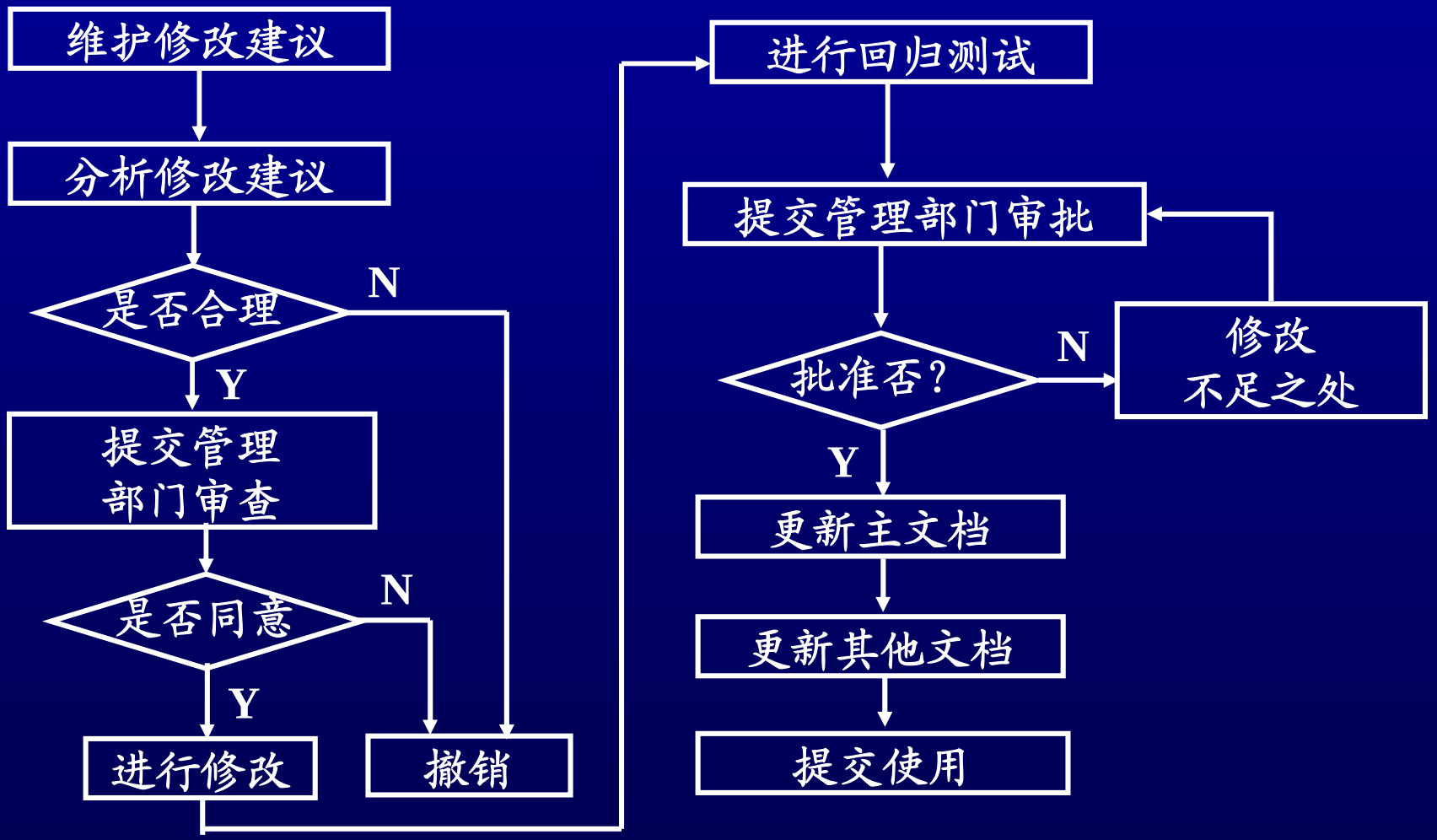
- 从用户观点出发，可使用性定义为程序方便、实用、及易于使用的程度。一个可使用的程序应是易于使用的、能允许用户出错和改变，并尽可能不使用户陷入混乱状态的程序。

软件维护

- 软件维护的定义
- 软件维护的类型
- 结构化维护VS非结构化维护
- 影响软件维护工作量的因素
- 软件维护的过程
- 可维护性
- 软件维护的管理

软件维护的管理

- 软件维护的管理流程如下：



软件维护的管理

- 由谁来承担软件维护管理工作是维护管理的另一个重要问题。
  - 一般认为应该有开发人员来维护，应为他们对软件最熟悉，维护起来最方便
  - 另一种做法是安排专职维护人员负责维护工作，而非开发人员。这样的好处是开发人员可以集中精力做好开发工作，有利于坚持实施开发标准，有利于保证文档的编制质量。同时专职的维护人员可以深入透彻的分析软件，从而更有力于维护的开展。
  - 还有一种较好的做法，安排软件人员开发任务和维护任务的定期轮换。这样可以使软件人员体会到开发和维护工作的具体要求、开发和维护的关系，有利于软件人员的技术水平和软件系统的质量。