# 第 26 章　文档和配置管理

## 26.1　信息系统文档

每一个信息系统都会经历规划阶段、制订方案阶段、研制阶段、试运行阶段、安装调试阶段、运行阶段和更新阶段，每一阶段都有大量的文档产生。文档是记录系统的痕迹，是系统维护人员的指南，是开发人员与用户交流的工具，是系统相关人员对系统了解和使用的必须资料。健全规范的文档意味着系统是按照工程化的方法开发的，意味着系统的质量有了形式上的保证，而文档欠缺和文档的随意性和不规范性，极有可能导致开发人员流动后，系统不可维护，成了没有生命力的系统。

信息系统的文档，不但包括软件开发过程中产生的文档，还包括硬件采购和网络设计中形成的文档；不但包括上述有一定格式要求的规范文档，也包括系统建设过程中的各种来往文件、会议纪要、会计单据等资料形成的不规范文档，后者是建设过程中有各方谈判甚至索赔的重要依据；不但包括系统实施记录，也包括程序资料和培训教程等。

### 26.1.1　信息系统文档的种类

信息系统文档种类繁多，非常复杂，可以说是不胜枚举。信息系统中的文档的作用也就是系统中各种参与者之间交流沟通的工具。下面我们从用户、分析人员、开发人员、项目管理人员、测试人员、维护人员之间的交流沟通将这些文档做一个分类总结。

（1）用户和分析人员的沟通。

- 可行性研究报告。
- 总体规划报告。
- 系统开发合同。
- 系统方案说明书。

（2）开发人员与项目管理人员的沟通。

- 系统开发计划（包括计划相关的各种文档）。
- 系统开发月报。
- 系统开发总结报告。

- 开发人员间的交流。
- 系统方案说明书。
- 系统设计说明书。

（3）测试人员和开发人员间的沟通。

- 系统方案说明书。
- 系统开发合同。
- 系统设计说明书。
- 测试计划。
- 测试用例。
- 测试记录。
- 测试报告。

（4）系统开发人员和用户之间的沟通。

- 用户手册。
- 操作指南。

（5）系统开发人员和系统维护人员间的沟通。

- 系统设计说明书。
- 系统开发总结报告。
- 技术手册。

（6）用户与维护人员间的沟通。

- 系统运行报告。
- 维修修改建议。

### 26.1.2　信息系统文档的特点

系统文档往往是多人合作完成的，并且会传送给更多的人使用。它有两个重要的特性：变更性和共享性。

系统文档的形成并不是一蹴而就的，往往需要进行多次的修改，并且经常是在多人之间的合作成果。往往也需要经历开发、评审、修改的过程。系统文档通常有众多的使用者，文档开发者创建好文档后都需要经过一个有效的途径分发到使用者手上，通常是每个使用者都有一份文档的备份。

如何在系统文档的开发过程中进行有效的控制和管理，如何进行文档的分发并保证每个使用者都有相同的备份，这是文档管理中的一个重要课题，解决它的唯一办法就是配置管理。

## 26.2　配置管理的基本概念

### 26.2.1　配置项

信息系统中的文档和软件在其开发、运行、维护的过程中会得到许多阶段性的成果，并且每个文档、软件在开发和运行过程中还需要用到多种工具软件或配置。所有这些信息项都需要得到妥善的管理，决不能出现混乱，以便在提出某些特定的要求时，将它们进行约定的组合来满足使用的目的。

这些信息项是配置管理的对象，称为配置项。它们通常可以分成下面的 6 种类型。

**（1）环境类。**软件开发、运行和维护的环境，如编译器、操作系统、编辑软件、管理系统、开发工具、测试工具、项目管理工具、文档编制工具等。

**（2）定义类。**需求分析与系统定义阶段结束后得到的工件，如需求规格说明书、项目开发计划、设计标准或设计准则、验收测试计划等。

**（3）设计类。**设计阶段得到的工件，如系统设计说明书、程序规格说明、数据库设计、编码标准、用户界面设计、测试标准、系统测试计划、用户手册。

**（4）编码类。**编码及单元测试结束后得到的工件，如源代码、目标码、单元测试用例、数据及测试结果。

**（5）测试类。**系统测试完成后的工作，如系统测试用例、测试结果、操作手册、安装手册。

**（6）维护类。**维护阶段产品的工作，以上任何需要变更的软件配置项。

配置项是一个独立存在的信息项，我们可以把它看成一个元素，单独的一个元素发挥不了什么作用，但随着工作的进展，出于不同的要求，需要将这些元素进行不同的组合，这个组合称配置，配置是一个软件产品在生存期各个阶段的不同形式（记录特定信息的不同媒体）和不同版本的程序、文档及相关数据的集合，或者说是配置项的集合，它具有完整的意义。

系统需求是由很多需求描述文件和系统用例组成的，每一个文件是一个配置项，所有的配置项结合起来才能够形成一个完整的系统需求，系统需求就是一个配置。

### 26.2.2　配置管理

简单来说，配置管理简单说，就是对配置的管理。按国际标准 ISO9000：3.1997 的说法，配置管理是一个管理学科，它对配置项（包括软件项）的开发和支持生存期给予技术和管理上的指导。配置管理的应用取决于项目的规模、复杂程序和风险大小。软件工程专家 W.Babich 认为，软件配置管理能协调软件开发，使得混乱减少到最小。软件配置管理是一种标志。组织和控制修改的技术，目的是最有效地提高生产率。《软件工程术语》国家标准 GB/T 11457-1995，给配置管理下了定义，配置管理是标志和确定系

统中配置项的过程，在系统整个生存期内控制这些配置项的投放和更动，记录并报告配置的状态和变动要求，验证配置项的完整性和正确性。并对下列工作进行技术和行动指导与监督的一套规范：

（1）对配置项的功能特性和物理特性进行标志和文件编制工作。

（2）控制这些特性的变动情况。

（3）记录并报告这些变动进行的处理和实现的状态。

综合以上几种对配置管理的解释，可以把软件配置管理概括为：它是采用技术手段和行政手段进行管理和监督的一套规范化方法；对配置项的功能特性和物理特性加以标志，并将其文件化；控制这些特性的变更；报告变更进行的情况和变更实施的状态及验证与规定需求的一致性。

总之，配置管理主要是对软件生存期过程中的各种阶段产品和最终产品演化和变更的管理，它是软件质量管理的重要组成部分。如果从变更的意义讲，软件配置管理是要解决软件的变更标志、变更控制，以及变更发布的问题。

### 26.2.3　配置管理的意义

信息系统项目的对象是信息系统，它和传统的制造产品有着很大的差别，这些差别决定了信息系统项目必须相应地采取特殊的措施，否则将无法达其目标。信息系统是不可见抽象的智力产品，其规模日益扩大和复杂，参加的人员数量日益增多，沟通工作量也越来越大，并时时处于变化之中，并对系统开发人员的依赖相当大。由于这些原因，信息项目很容易造成信息的拥有者的版本不一致，或者需要的文件找不到，或者需求变化太快以致产品与需求不一致的现象。所有这些现象用配置管理的方法都是很容易实现的。

具体的配置管理能够解决以下问题：

**（1）多重维护问题。**一个文档的几个备份在不同的地方使用，或者若干个文档中都含有一些共同的内容。如果一个用户发现了一个文档出现了问题便直接进行修正，或几个用户发现了问题都各自做了修正，这样，文档就不一致了。

这是配置管理最容易解决的问题，用户需要修改某文档时，必须从配置库中检出该文档，修改后再检入，每个用户需要该文档时都从配置库中检出最新的文档。

**（2）同时修改问题。**多个用户对同一个文档进行修改，这时就有可能出现有的用户的变更消失了。要解决这个问题，有两个办法，一个是同一时间只允许一个人检出，另一个办法是将文档分成多个文档，避免编辑冲突。

**（3）丢失版本或不知版本。**这个问题的解决要明确规定保留哪个版本，销毁哪个版本；采用一种系统化的文档标志版本，并控制版本的变更采用统一的备份规程。

## 26.3　配置管理过程

配置管理是 CMM2 中的一个重要的 KPA，其作用就是建立和保证整个软件生命周期中产品的完整性。它是所有成熟的软件组织必需的一个管理过程。本节说明配置管理中的角色、流程及配置管理计划的制订。

### 26.3.1　配置管理中的角色和分工

要使配置管理活动在信息系统的开发和维护中得到贯彻执行，首先要明确确定配置管理活动的相关人员及其职责和权限。配置管理过程的主要参与人员如下：

**（1）项目经理（Project Manager，PM）。**项目经理是整个信息系统开发和维护活动的负责人，他根据配置控制委员会的建议，批准配置管理的各项活动并控制它们的进程。其具体工作职责如下：

- 制订项目的组织结构和配置管理策略；
- 批准、发布配置管理计划；
- 决定项目起始基线和软件开发工作里程碑；
- 接受并审阅配置控制委员会的报告。

**（2）配置控制委员会（Configuration Control Board，CCB）。**负责指导和控制配置管理的各项具体活动的进行，为项目经理的决策提供建议。其具体工作职责如下：

- 批准配置项的标志，以及软件基线的建立；
- 制订访问控制策略；
- 建立、更改基线的设置，审核变更申请；
- 根据配置管理员的报告决定相应的对策。

**（3）配置管理员（Configuration Management Officer，CMO）。**根据配置管理计划执行各项管理任务，定期向 CCB 提交报告，并列席 CCB 的例会，其具体工作职责如下：

- 软件配置管理工具的日常管理与维护；
- 提交配置管理计划；
- 各配置项的管理与维护；
- 执行版本控制和变更控制方案；
- 完成配置审计并提交报告；
- 对开发人员进行相关的培训；
- 识别开发过程中存在的问题并制订解决方案。

**（4）开发人员（Developer，Dev）。**开发人员的职责就是根据项目组织确定的配置管理计划和相关规定，按照配置管理工具的使用模型来完成开发任务。

## 26.3.2　配置管理流程

配置管理流程图如图 26-1 所示。

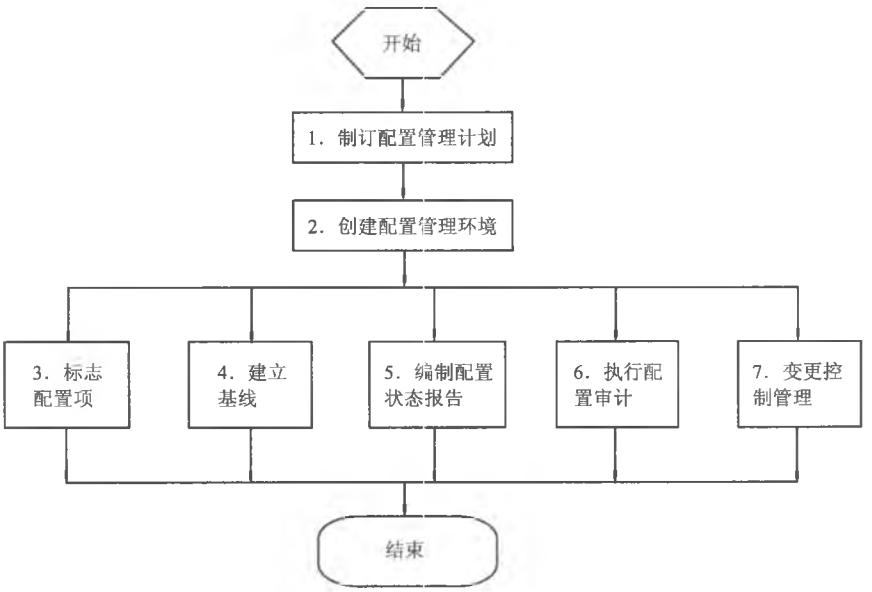


图 26-1　配置管理流程图

**（1）制订配置管理计划。**在项目启动阶段，项目经理首先要制订整个项目的开发计划，它是整个项目研发工作的基础。总体研发计划完成之后，配置管理的活动就可以展开了，如果不在项目开发之初制订配置管理计划，那么配置管理的许多关键活动就无法及时有序地进行，而它的直接后果就是造成项目开发状况的混乱，并注定使配置管理活动成为一种救火的行为。由此可见，在项目启动阶段制订配置管理计划是项目成功的重要保证。配置管理计划由 CMO 制订，主要内容是制订配置管理策略，制订变更控制策略，编写配置管理计划，评审配置管理计划。

**（2）创建配置管理环境。**创建配置管理环境主要是由 CMO 设置硬件环境、设置网络环境、设置软件环境、建立一个配置管理库，存储项目中定义的配置项，安装配置管理工具，例如 ClearCase，VSS 等，并提供配置管理培训。

**（3）配置管理计划的实施。**配置管理计划的实施由项目相关参与人员进行，主要是进行配置标志、建立配置基线、编制状态报告、招待配置审计和变更控制。

制订配置管理计划的过程包括以下主要工作流程：

- CCB 根据项目的开发计划确定各阶段里程碑和开发策略。
- CMO 根据 CCB 的规划，制订详细的配置管理计划，交 CCB 审核。
- CCB 审核通过配置管理计划后交项目经理批准，发布实施。

**（4）配置管理计划的执行。**执行阶段的配置管理活动主要分为 3 个层面：

- 由 CMO 完成日常管理和维护工作。
- 由 DEV 具体执行配置管理策略。
- 变更控制。

这 3 个层面彼此之间既相互独立、又互相联系。

在配置管理执行过程中，具体按照如下流程进行：

- CCB 设定研发活动的初始基线。
- CMO 根据软件配置管理规划设立配置库和工作空间，为执行配置管理人员做好工作准备。
- 开发人员按照统一的软件配置管理策略，根据获得授权的资源进行项目的研发工作。
- CCB 根据项目的进展情况，审核各种变更请求，并适时地划定新的基线，保证开发和维护工作有序地进行。

### 26.3.3　配置管理计划

原则上，配置管理计划是信息系统开发计划的一个组成部分。一个信息系统项目启动以后，要认真分析项目的要求和特点，精心地组织策划。在考虑制订进度安排计划、人员投入计划、质量保证计划、风险管理计划、文档编制计划等的同时，必须制订配置管理计划。

配置管理计划通常要涉及该项目对配置管理的要求，实施配置管理的责任人、责任组织及其职责，开展的配置管理活动、方法和工具等。

这里以 IEEE 的标准为例，介绍配置管理计划应包括的内容。

配置管理计划标准 IEEE 828-1990 Standard for Software Configuration Management Plan。

**（1）引言。**

- 配置管理计划的目的、适用范围、使用要求。
- 项目概述。
- 项目中需特别关注的配置管理问题和风险。
- 配置管理严格性要求的等级。
- 限制和假设。
- 术语。
- 参考文件。

**（2）配置管理。**

- 配置管理的组织结构。
- 职责和权限。
- 指令和方针。

- 参照的规程（组织的规程或客房的规程）。
- 遵循的标准。

**（3）配置管理活动。**

- 配置标志。
- 变更管理和配置控制。
- 配置状态说明。
- 配置审核。
- 接口和子合同方控制。

**（4）配置管理进度安排。**

- 配置管理重要事件的顺序。
- 配置管理各项活动间的依赖关系。
- 与其他重要项目里程碑的关系。

**（5）配置管理所需的资源。**

- 采用的工具。
- 使用的设备。
- 应用的技术。
- 所需的培训。
- 对其他人员的要求。

**（6）配置管理计划的维护。**

- 维护的责任。
- 计划更新的条件和审批。
- 计划变更的交流和通报。

## 26.4　配置管理中的活动

### 26.4.1　配置标志

配置标志是配置管理的基础性工作，是管理配置管理的前提。配置标志是确定哪些内容应该进入配置管理形成配置项，并确定配置项如何命名，用哪些信息来描述该配置项。

**1．确定配置项**

信息系统项目中形成的技术性文档和管理性文档，除一些临时性的文档外一般都应该进行配置管理。一般来讲，判定一个文档是否进行配置管理的标准应该是此文档是否有多个人需要使用，这些文档往往在项目的进程中不断地修正和扩展，要保证每个使用者都使用同一版本的文档，就必须将这些文档纳入配置管理，成为受控的配置项。

Roger S.Pressman 认为至少以下所列的文档应该成为配置项。

**（1）系统规格说明书。**

**（2）项目计划。**

**（3）需求规格说明书。**

- 图形分析模型。
- 处理规格说明。
- 原型。
- 数学规格说明。

**（4）用户手册。**

**（5）设计规格说明。**

- 数据设计描述。
- 体系结构设计描述。
- 模块设计描述。
- 对象描述。

**（6）源代码。**

**（7）测试规格说明。**

- 测试计划和步骤。
- 测试用例、记录和结果。

**（8）操作和安装手册。**

**（9）可执行程序。**

- 模块可执行代码。
- 链接的模块。

**（10）数据库描述。**

- 模式和文件结构。
- 初始内容。

**（11）联机用户手册。**

**（12）维护文档。**

- 软件问题报告。
- 维护请求。
- 工程变更指令。

**（13）软件工程标准和规程。**

2. 配置项命名

确定了配置项后，还需要对配置项进行合理、科学的命名。配置项的命名绝不能随意为之，必须满足以下两点。

**（1）唯一性：**要求在一个项目内不能出现重名现象，以避免混淆。

**（2）可追溯性：**名字应能体现相邻配置项之间的关系。

一个典型的实例是采用层次式的命名规则来反映树状结构，树状结结构上结点之间

存在着层次的继承关系。如图 26-2 所示为一个典型的信息系统工程的配置项目录结构。

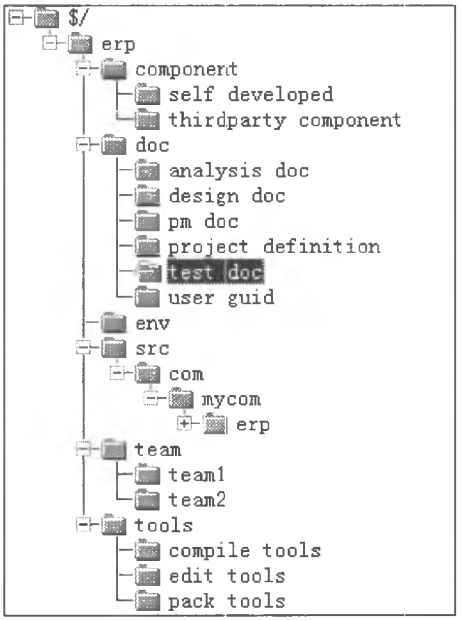


图 26-2 一个典型的信息系统工程的配置项目录结构

### 3. 配置项的描述

由于配置项除了名称外还有一些其他属性和与其他配置项的关系，因此它可以采用描述对象的方式来进行描述。

每个配置项用一组特征信息（名字、描述、一组资源、实现）唯一地标志。

**（1）名字：**确切标志对象的字符串。

**（2）描述：**数据项表，应包括对象表示的配置项类型（如文档、程序或数据）、项目标志符、变更和版本信息。

**（3）资源：**该对象所提供的、处理的、引用的或另外所需要的实体。例如，数据类型、特定函数、甚至是变更名。

**（4）实现：**基本对象是指向“文本单元”的指针；复合对象则为 null。

配置项间的关系有整体和部分的关系及层次关系，也有关联关系。

配置项间的关系可以用 MIL 语言（Module Interconnection Language）表示。MIL 描述的是配置项间的相互依赖关系，可自动构造系统的任何版本。

## 26.4.2 版本控制

### 1. 什么是版本

对于信息产品的版本有两个方面的意思，一是为满足不同用户的不同使用要求，如用于不同运行环境的系列产品。如适合 Linux、Windows、Solaris 用户的软件产品分别

称为 Linux 版、Windows 版和 Solaris 版。它们在功能和性能上是相当的，原则上没有差别，或者说，这些是并列的系列产品。对于这类差别很小的不同版本，互相也称为变体（Variant）。

另一种版本的含义是在软件产品投产使用后，产品经过一系列的变更，如纠错、增加功能、提高性能的更改，而形成的一系列的顺序演化的产品，这些产品也称为一个版本，每个版本都可说出它是从哪个版本导出的演化过程。

必须注意到，修正后的新版本往往不能完全代替老版本，尽管新版本有某些优越的特性。因为一些用户仍然使用着老版本，并且不容易立刻做到以旧换新，否则可能会打扰老版本原有的工作环境。显然，多个版本被多个用户同时使用的情况是不可避免的现实。这就要求多个版本共存，这也就是配置管理要解决的一个重要课题。

2．版本控制（Version Control）

版本控制用于管理信息工程中生成的各种不同的配置将规程和相关管理工具结合起来。按 G.M.Clemm 对版本管理的解释：配置管理使用户借助选择适用的版本来选定软件系统的配置，为此需要确定每个软件版本的属性，同时还考虑到同描述一些预期属性所构成的配置。

版本管理要解决的第一个问题是版本标志，也就是为区分不同的版本，要给它们科学的命名。通常有以下几种版本命名的方法。

**（1）号码版本标志。**以数字表示，如用 1.0、2.0、1.2、2.1.1 等表示版本号。一般认为 1.0、2.0 等为基础版本，1.1、1.2 则是对基础版本 1.0 的第一次修改和第二次修改。对有重大更动或因多次修改导致的全局性重要更动，则应该提高基础版本号，例如，上升到 2.0。

这种顺序号码的命名法被广泛地使用，它的突出优点就是简单直观。但如果版本多了，并且出现了非简单顺序的线型号码，就很难从号码上区分其前后的继承关系，无法体现命名的可追溯性原则。另外，只根据号码也不能看出更多的信息。

**（2）符号版本标志。**这种标志版本的命名方法是将重要的版本属性有选择地给出，如 Windows 98，Windows 2000，JBuilder 2005 将版本产生的时间给出。为了从版本标志上看到更多信息，可能给出更多的属性，如面向的客户群、开发语言、硬件平台、生成日期等。

配置管理中，版本包括配置项的版本和配置的版本，这两种的版本的标志应该各有特点，配置项的版本应该体现出其版本的继承关系，它主要是在开发人员内部进行区分。另外，还需要对重要的版本做一些标记，如对纳入基线的配置项版本应该做一个标志。对于配置来讲其版本号往往是在非常广的范围内使用，需要对其版本的重要属性进行标志，但同时，它还必须有一个内部的版本号，这个内部的版本号通常采用的号码版本标志方法。

上面讲的版本控制都是指配置管理的纵向生成的产品的版本控制，对于纵向的版本即适应不同运行环境的版本，在配置管理中应该采用增加一个配置项或配置的方式来进

行管理。配置管理本身就应该将信息系统的生成产品纳入配置库，既然该信息系统生成一系列应该用于不同环境的版本，当然每个版本都应该纳入配置管理，形成配置项。

### 26.4.3　变更控制

配置管理的最重要的任务就是对变更加以控制和管理，其目的是对于复杂，无形的软件，防止在多次变更下失控，出现混乱。

1．变更

**（1）变更是不可避免的。**

变更来源有两个方面，一是用户，他们是信息工程项目需求的提出者。一个十分常见的现象是用户提出需求以后，在开发过程中用户又改变了其需求，这只能迫使开发工作返工，丢弃一些无法修正的部分。无疑这会造成一定的损失，但却无法完全避免。要求用户一次性地把需求讲清楚，并且不允许此后做任何变更，这是不现实的。我们只能尽力减少变更，降低其影响。开发人员如何解决好自己的工作产品与变更的用户需求之间的一致性，正是CMM2级需求管理这个关键过程域的主要目标。

变更来源的另一个方面来自开发人员自身。他们在工作中可能发现前期工作中有些不妥当的地方，便要修改已经确定了的设计方案或是设计的细节。也许是项目管理人员提出要修订已经确定了的项目方案。由此所导致的返工甚至部分工作产品的报废也是在所难免的。

原则上说，随着工作的进展，无论是用户还是开发人员都将掌握更多的信息，对问题本身和设计方案有了更深入的认识，同时也会发现原来的设想有不充分，不完善甚至有不合理、不可行的成分。这时提出修正是完全合理的，是符合人们认识规律的。对于复杂而生疏的问题要求人们一次认识正确，其解决方案也要求一次设计完全无误都是不现实的。

变更出现的不可避免性决不意味着其可以任意修改，也不能以此作为软件产品质量达不到要求的借口。毫无疑问，信息工程过程中变更管理的责任重大，能否解决变更管理问题是成熟软件组织的一个明显的检验标志。

**（2）变更是复杂的。**

一个配置项出现变更，可能会涉及一些相关的部件和文档，这将影响到项目开发工作中的许多人员。例如，测试引发了需求的修改，那么很可能要涉及需求规格说明书、概要设计、详细设计和代码等相关文档，甚至测试计划随之变更。

如果是多个开发人员对同一部件做了修改，情况会更加复杂。例如，在测试中发现了两个故障。先指定甲去解决第一个故障，同时指定乙去解决第二个故障。尽管最初以为两个故障是无关的，但后来两人发现这两个故障引起的根本原因都是同一个部件的不同位置引起的。可是两人接受任务时还不了解这一情况。于是甲从配置库中取出该部件并做了修改，并送回配置库；此后，乙从库中取出了原始版，做了他的修改，放入库中时代替了甲修改后的版本。显然甲的工作白做了。在回归测试时，发现甲并没有做他的

修改工作，结果甲又得重做修改。

**（3）变更管理的任务。**

变更管理简单地说就是控制修改，使之不出现改错、改乱的现象。变更管理的任务如下：

- 分析变更：研究变更的必要性，经济可行性（是否合算）和技术可行性（能否实现）。
- 记录和追踪变更。
- 采取措施保证变更在受控状态下进行。

IEEE 解释变更管理时说，它是软件配置管理的一个重要的组成部分，涉及在给配置项建立了正式的配置标志后，变更的评价、审批与实现诸方面的活动。

2．配置库

配置库（Configuration Library）也称配置项库（Configuration Item Repository），是配置管理的有力工具。

**（1）配置库的作用。**

配置库的主要作用表现在：

① 记录与配置相关的所有信息，其中存放受控的配置项是很重要的内容。

② 利用库中的信息可评价变更的后果，这对变更控制有着重要的意义。

③ 从库中可提取各种配置管理过程的管理信息，可利用库中的信息查询回答许多配置管理的问题，例如：

- 哪些客户已提取了某个特定的系统版本？
- 运行一个给定的系统版本需要什么硬件和系统软件？
- 一个系统到目前已生成了多少个版本，何时生成的？
- 如果某一特定的构件变更了，会影响到系统的哪些版本？
- 一个特定的版本曾提出过哪几个变更请求？
- 一个特定的版本有多少已报告的错误？

利用配置库实现配置管理是非常有效的。如同一个大型工厂，生产出的许多零部件，以及许多成品需要在仓库里加以集中存放和保管一样，要依靠仓库的管理机制保证存放在其中的零部件和成品的安全和有序，不致发生混乱（例如，把外形相似或完全一样的两种产品混淆），也不致发生仓库存放的物品丢失现象。为了强化仓库的管理，要采取一些有力和有效的措施，例如，要严格坚持出入库检查制度。

与此相似，采用配置库实现软件配置管理，就可把软件开发过程的各种工作产品，包括半成品、阶段产品和最终产品管理得井井有条，使其不致管乱、管混、管丢。上述甲乙二人修改程序时出现的问题，正是要靠对配置库的“入库检查（Check In）”和“出库检查（Check Out）”来加以解决。同时，若配合有访问权限的措施就完全可以做到库内存放的产品什么人可以看，什么人可以取，什么人可改，什么人可以存入等的控制。在这种控制之下的库品产品，如果甲正对其修改，乙就无法拿到，因为他取出时，这个

产品被锁住了，所以不可以发生甲乙之间的问题。

**（2）配置库的分类。**

配置库有三类：

- 开发库（Development Library）。存放开发过程中需要保留的各种信息，供开发人员个人专用。库中的信息可能有较为频繁的修改，只要开发库的使用者认为有必要，无须对其做任何限制。因为这通常不会影响到项目的其他部分。
- 受控库（Controlled Library）。在软件开发的某个阶段工作结束时，将工作产品存入或将有关的信息存入。存入的信息包括计算机可读的，以及人工可读的文档资料。应该对库内信息的读/写和修改加以控制。
- 产品库（Product Library）。在开发的软件产品完成系统测试之后，作为最终产品存入库内，等待交付用户或现场安装。库内的信息也应加以控制。

作为配置管理的重要手段，上述受控库和产品库的规范化运行能够实现对软件配置项的管理。

**3. 配置基线**

**（1）基线。**

基线（Baseline）是软件生存期各开发阶段末尾的特定点，也称为里程碑（Milestone），在这些特定点上，阶段工作已结束，并且已经形成了正式的阶段产品。

建立基线的概念是为了把各开发阶段的工作划分得更加明确，使得本来连续开展的开发工作在这些点上被分割开，从而更加有利于检验和肯定阶段工作的成果，同时有利于进行变更控制。有了基线的规定就可以禁止跨越里程碑去修改另一开发阶段的工作成果，并且认为建立了里程碑，有些完成的阶段成果已被冻结。

如图 26-3 所示给出了配置基线的示意图。图中在每个开发阶段的末尾都标出了该阶段的基线，图的上部则给出了各开发阶段的工作成果。事实上，现在人们已经把这些成果称为基线了。例如，设计基线指的就是设计规格说明。

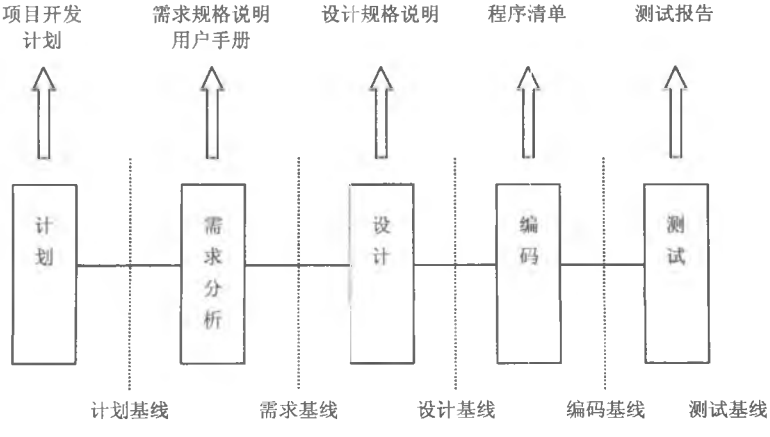


图 26-3　配置基线的示意图

作为阶段工作的正式产品，基线应该是稳定的，如作为设计基线的设计规格说明应该是通过评审的。如果还只是设计草稿，就不能作为基线，不能被冻结。

**（2）基线的种类。**

如果把软件看做系统的一个组成部分，以下 3 种基线是最受人们关注的。

- 功能基线。功能基线是指在系统分析和软件定义阶段结束时，经过正式评审批准的系统设计规格说明中对被开发软件系统的规格说明；或是指经过项目委托单位和项目承办单位双方签字同意的协议或合同中所规定的对被开发软件系统的规格说明；或是指由下级申请上级同意或直接由上级下达的项目任务中所规定的对待开发软件系统的规格说明。
- 分配基线。分配基线是指在软件需求分析阶段结束时，经正式评审和批准的软件需求规格说明。
- 产品基线。产品基线是指在软件组装与系统测试阶段结束时，经正式评审和批准的有关所开发的软件产品的全部配置项的规格说明。

**（3）基线与配置项。**

提出基线的概念本来是为了更好地实现变更控制，但如果把每个基线都当成一个整体来看待会造成麻烦。因为一个变更很可能只涉及基线的很小部分。例如，假定某个大型软件中的一个模块修改了，如果将这一变更当做整个软件产品基线的变更，就很不方便。

事实上，基线可由多个软件配置项组成，一个软件配置项可以是一个文档，或者是一个可直接放在配置控制之下的工作产品，能够作为一个独立的基本部件加以修改。文档通常已被认为是独立可修改的部件了，但如有必要还可将其再加以细分，把文档中的章、节甚至段当做配置项来看待。

以产品基线为例，它往往含有多个代码级的配置。代码的变更是频繁的，因为几乎所有的变更最后都要导致某些代码的变更。特别是在多个程序人员参与工作的情况下，每个人负责自己分工的那个模块，责任是清楚的，这就十分有利于变更的控制和追踪。

在定义软件配置项时有两种做法：一种做法是，把每个单独可编译的模块当做一个软件配置项，模块的名字就是软件配置项的名字；另一种做法是把每个文件当做一个配置项，文件名当做配置项名。

很明显，配置管理所管的配置项并不都是互相独立的，它们之间可能存在着某种相互依赖关系。如果说配置项 X 对配置项 Y 依赖，是指假如 Y 做变更，要求 X 也做变更，使 X 保持正确或者说使两个基线是一致的。不过除非是从配置项的性质导出的情况，这种依赖关系很难清晰地在文档中表达。例如，体现设计文档的配置项往往依赖于代表需求文档的配置项，那就要在设计文档中说明，每项设计对应了哪些需求的实现。如果一个设计基线是由许多配置项组成，我们可以据此理解设计的哪些项和需求的某些项之间有着依赖关系。在代码中的情况也是这样，代表一模块的配置项依赖于另一模块的配置项，这种依赖关系往往可从设计规格说明中得到，在实施变更控制时，依赖关系应在

变更请求中反映出来。这一点在后面的讨论变更请求中还会涉及，就是要明确变更的影响范围。

**4．变更控制**

**（1）变更控制委员会（Change Control Board，CCB）。**

也可称为（Configuration Control Board），配置控制委员会，是配置项变更的监管组织。其任务是对建议的配置项变更做出评价、审批，以及监督已批准变更的实施。

CCB 的成员通常包括项目经理、用户代表、软件质量控制人员、配置控制人员。这个组织不必是常设机构，完全可以根据工作的需要组成。例如，按变更内容和变更请求的不同，组成不同的 CCB。小的软件项目 CCB 可以只有 1 人甚至只是兼职人员。

如果 CCB 不只是控制变更，而是承担更多的配置管理任务，那就应该包括基线的审定、标志的审定，以及产品的审定，并且可能实际的工作需分为项目层、系统层和组织层来组建，使其完成不同层面的配置管理任务。

**（2）变更请求与变更控制。**

**① 利用配置库实现变更控制。**一般情况下，开发中的配置项尚未稳定下来，对于其他配置项来说是处于不处理工作状态下，或称自由状态下，此时它并未受到配置管理的控制，开发人员的变更并未受到限制。但当开发人员认为工作已告完成，可供其他配置项使用时，它就开始稳定。把它交出评审，就开始进入评审状态；若通过评审，可作为基线进入配置库（实施检入），开始冻结，此时开发人员不允许对其任意修改，因为它已处于受控状态。通过评审表明它确已达到质量要求；但若未能通过评审，则将其回归到工作状态，重新进行调整。可以通过图 26-4 看到上述配置项的状态变化过程。

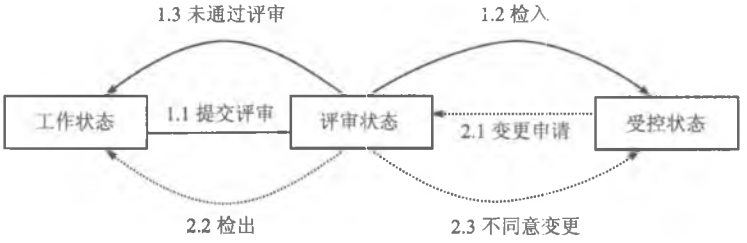


图 26-4　配置项的状态变化过程

处于受控状态下的配置项原则上不允许修改，但这不是绝对的，如果由于多种原因需要变更，就需要提出变更请求（Change Request）。在变更请求得到批准的情况下，允许配置项从库中检出，待变更完成，并经评审后，确认变更无误方可重新入库，使其恢复到受控状态。

**② 变更请求。**变更请求是实施变更控制的第一步，也是必不可少的一步。最常见的变更理由可能是消除缺陷，适应运行平台的变更；或是软件扩展提出的要求，例如，增加功能、提高性能等。

变更请求的主要内容有如下 3 个方面：

- 变更描述。包括变更理由、变更的影响、变更的优先级等，就是要申述做什么变更，为什么要做，以及打算怎么做的问题。
- 对变更的审批。对变更的必要性、可行性的审批意见，主要是由配置管理员和 CCB 对此项变更把关。
- 变更实施。变更实施的情况、质量保证审查和配置管理审查情况。
- 故障报告。提出变更请求最为常见的情况是已经入库的基线发现了新的缺陷，表现为故障，为了更好地实施变更和变更管理，有的软件组织要求在提出变更请求前提出故障报告（Fault Report，FR）。

有关变更实施的一些信息。如表 26-1 所示提供了变更请求表的实例。

**表 26-1　变更请求表的实例**

| 项目名称： | 变更请求标志： | |
|---|---|---|
| 变更请求<br>变更理由：<br>变更描述：<br>影响范围：<br>变更优先性考虑：<br>估计变更工作量： | | |
| 分析与评估：<br>分析评估意见 | 分析者： | 日期： |
| 审批：<br>CCB 审查意见 | CCB 负责人： | 日期： |
| 变更实施：<br>变更实施情况 | 实施负责人： | 日期： |
| 质量保证审查：<br>审查意见 | QA 负责人： | 日期 |
| 配置管理审查：<br>变更意见 | CM 负责人： | 日期： |

**③ 变更控制过程。**从表 26-1 所示的变更请求表中已能看出变更控制的大致过程，下面以变更请求表 CRF 为基础进一步给出其控制过程，如图 26-5 所示。

如图 26-5 所示，项目相关人员发现问题，提出变更，将变更申请提交到 CCB，CCB 对变更申请进行评审，如果有必要将变更交由专门的变更分析人员进行分析评估，那么由变更分析人员对变更进行分析评估，分析评估后意见交 CCB 作为变更评审的依据。

在分析和评估变更请求时主要考虑的是变更对成本、进度和质量等方面的影响。必要时配置管理人员、变更分析人员可能要与变更请求人交谈和商讨。

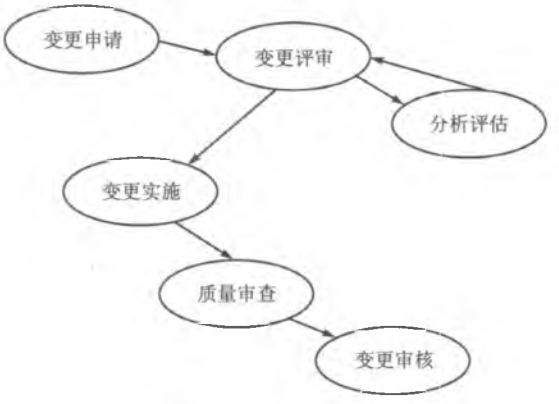


图 26-5　变更控制过程

在 CCB 批准后送交变更实施者，应该要求记录变更的情况。实际上，变更请求表上不仅记载了变更请求和变更审批的信息，而且还包含有关变更实施的信息，可见，可通过变更请求表了解到变更的实施状态。

关于变更实施情况还可通过状态说明了解。在变更实施后，质量保证人员和配置管理人员还将对变更的质量进行监控，保证变更的质量和变更的有效性。

附有故障报告的变更请求，特别在故障较为严重时，常常被当做高优先级的变更请求处理。事实上，故障报告还可用于追踪软件中缺陷清除的状态。

故障报告包含的内容有：

① **FR ID（故障标志）。**故障信息，包括故障描述、故障严重程度、怀疑有问题的部位、故障的影响、故障现象和环境信息、估计的故障原因、故障信息提供者等信息。

② **CCB 评估意见。**是否批准，优先级如何，相关说明等。

③ **故障修复信息。**要变更的部分和相关说明。

**（3）变更记录。**

按上述要求，尽管变更被置于控制之下，但便于长期保留变更的相关信息以备后用，需要把这些信息保存起来。

首先，应将变更请求表作为配置项在配置库中登录。其次，在变更的代码模块或文档内应记录有关变更的信息。

## 26.4.4　构造管理

### 1. 构造的定义

**（1）什么是构造。**由于人们在认识事物中不可能是一开始就完全认识清楚的。同样，在信息工程中，我们不可能一下子就把需求搞清楚，不可能一下就把这个信息系统

建立起来，所有这些，都是从表面到深入、从片面到全面、从模糊到清楚、从简单到复杂的过程。信息工程中的产品也不是一下子就出来的，而是每个阶段都有一个产品产生，这些产品是有连续性的，下一阶段的产品是上一阶段产品的延续，是逐步进化的过程。这些工作产品从表象上来看一般都是一组完整的相关的配置项进行整合或编译而得的。从这个意义上讲，每个阶段性的产品就称为构造（Build）。

**（2）构造的特点。**根据构造的定义，我们不难得出构造的以下 3 个特点：

- 构造的内容，构造都是由一组相关的配置项进行整合或编译生成的，这组配置项就是构造的来源。
- 构造的方法，每个构造都有一个生成方法，以及其生成的工具及环境。比如，一个普通的 Word 文档，它是将几个文档的内容进行编辑而生成的。生成环境和工具就是当前使用的 Office 软件及其版本。一个执行代码，则是由一个编译工具对一组代码按一定的步骤和方法进行编译的结果。
- 构造的版本，构造也是一系列顺序演化的产品，显然它也应该进行版本管理。

**（3）构造的要素。**由这几个特点，我们可以知道，要准确描述一个构造，我们得知道构造的名称、构造的应用、构造的功能、构造的内容、构造的方法，以及构造的版本。只有对这 6 方面进行准确的描述人们才能够对一个构造有清楚的认识，这 6 个方面就是构造的六要素。配置管理中显然应该将构造纳入配置管理，形成一个特殊的配置项，并且在配置管理中对构造的这 6 个方面进行描述，特别是应该将构造的功能，构造的内容和构造的方法形成几个文件一起纳入进行管理。

### 2. 构造的管理

对构造的管理，就是要在构造出现问题时，我们能根据构造的内容和构造的方法重新构造，确定问题所在。另外，我们应该对构造的功能和用法相配合，取一个构造，我们就得同时取其相配置的用法和功能，这样我们才能够正常地使用构造。

**（1）构造的标志。**构造的标志，也就是将构造添加到配置库形成配置项的过程。构造由于其应用环境的不同，其构造方法不同，构造的功能也略有差别。由此可见，构造往往也存在一些并行版本。比如，执行代码中应用于不同环境的版本，如 Linux 版本、Solaris 版本和 Windows 版本。还有从功能和性能上进行区别得到的普通版、高级版、企业版等。这些并行的版本实际上应该是不同的构造，每一个版本都应该单独纳入配置管理，形成不同的配置项。

在构造标志的过程中，我们必须将其功能、内容、构造方法、构造工具和构造结果一起纳入管理中，这几个方面是一个整体，构造的每一个版本都对应其相应的功能、内容、构造方法和构造工具。在构造的使用中，这几个方面的内容应该是一起配合才能正常使用的，不然就会出现问题，出现不一致的现象。

**（2）构造的版本管理。**构造的版本管理和普通配置的版本管理是一致的，它也适合普通配置项的版本标志方式。

产品因为每一次构造生成的结果都是不相同的，于是它还有一个 build 码来进行标

志，如 Windows 2000 5.00.2195，这其中的 5.00 是其内部版本号，2195 是一个 build 码，表示第 2195 次编译的结果，Windows 2000 是 Windows 的一个版本。

### 26.4.5　状态报告

#### 1．配置状态报告

配置状态报告（Configuration Status Reporting）也称为配置状态说明与报告（Configuration Status Accounting&Reporting），它是配置管理的一个组成部分，其任务是有效地记录报告管理配置所需要的信息，目的是及时、准确地给出配置项的当前状况，供相关人员了解，以加强配置管理工作。

在信息工程过程中，必须注意到它的动态特性。事实上，在信息工程管理过程中，配置项在不停地演化着。随着开发工作的进展，工作产品不断扩展，形式也不断变化，从需求规格说明、设计说明到源程序等。另一方面，由于各种原因（纠错只是其中的一个原因），设计说明本身也在演变着，版本在更新着，对于这种动态特性如果没有控制手段，其后果是不可想象的。

配置状态报告就是要对在某个特定的当时的配置状态进行报告，也就是要对动态演化着的配置项取瞬时的照片，以利于在状态报告信息分析的基础上，更好地进行控制。

需要跟踪捕捉的状态报告信息可以是配置项的当前或以前的配置，变更请求或问题报告的状态和已获准变更的状态。

#### 2．配置状态报告的内容

配置状态报告的内容一般包括以下各项。

- 各变更请求概要：变更请求号、日期、申请人、状态、估计工作量、实际工作量、发行版本、变更结束日期。
- 基线库状态。
- 发行信息。
- 备份信息。
- 配置管理工具状态。
- 配置管理培训状态。

#### 3．配置状态报告的利用

在配置状态报告中提到了许多有关配置的信息，应该充分利用这些信息实现配置的控制，以下给出利用这些信息可以解决一些需要澄清的问题。例如：

- 程序 p13 的 1.6 版在哪个备份中可以使用？
- 在发行 5.1 和发行 5.2 之间实现了哪些变更请求？
- 在发行 5.2 中哪些程序更改过了？
- 在变更请求 671 中要对哪些配置项进行更改？在变更前和变更后，这些程序单元的版本是什么？是否所有的变更都完成并入库了？

#### 4. 状态说明

在变更请求批准后，实施变更需要一段时间，要设置一种管理手段来反映变更所处的状态，这就是变更状态说明（Status Accounting），它可供项目经理和 CCB 追踪变更的情况。

要求状态说明回答的问题可以是：

- 某个变更请示是否已被批准？
- 已批准的变更请求目前处于什么状态？
- 已完成的变更投入了多少时间和工作量？
- 某个配置项与哪几个变更请求有关？

状态说明的信息可以通过变更请求（CR）和故障报告（FR）得到，变更状态可分为活动（正在实施变更）、完成状态（已完成变更）和未列入变更状态 3 种。

### 26.4.6　配置审核

#### 1. 什么是配置审核

关于配置标志、配置项的变更控制等方面应该如何按规定实施，前面已经给出了说明，但在具体的项目开发中是否得到了遵循，需要进行检查。配置审核的任务便是验证配置项对配置标志的一致性。软件开发的实践表明，尽管对配置项做了标志，实践了变更控制和版本控制，但如果不做检查或验证仍然会出现混乱。这种验证包括：

（1）对配置项的处理是否有背离初始的规格说明或已批准的变更请求的现象。

（2）配置标志的准则是否得到了遵循。

（3）变更控制规程是否已遵循，变更记录是否可供使用。

（4）在规格说明、软件产品和变更请求之间是否保持了可追溯性。

配置审核工作主要集中在两个方面，一是功能配置审核，即验证配置项的实际功效是与其软件需求是一致的；二是物理配置审核，即确定配置项符合预期的物理特性。这里所说的物理特性是指定的媒体形式。

#### 2. 配置审核的意义

配置审核的实施是为了确保软件配置管理的有效性，体现配置管理的最根本要求，不允许出现任何混乱现象，例如：

（1）防止出现向用户提交不适合的产品，如交付了用户手册的不正确版本。

（2）发现不完善的实现，如开发出不符合初始规格说明或未按变更请求实施变更。

（3）找出各配置项间不匹配或不相容的现象。

（4）确认配置项已在所要求的质量控制审查之后作为基线入库保存。

（5）确认记录和文档保持着可追溯性。

3．配置审核的实施

**（1）配置审核要选择适当的时机。**通常选择以下几种情况实施配置审核：

- 软件产品交付或是软件产品正式发行前。
- 软件开发的阶段工作结束之后。
- 在维护工作中，定期地进行。

**（2）配置审核的责任人。**实施配置审核的审核人员可以包括项目组人员及非项目组人员，例如，其他项目的配置管理人员、软件组织的内部审核员，以及软件组织的软件配置管理人员。

**（3）配置审核的工作开展。**

- 由项目经理决定何时进行配置审核工作。
- 质量保证组或软件组的配置管理组指定该项目的配置审核人员。
- 项目经理和配置审核员决定审核范围。
- 配置审核员准备配置审核检查单。
- 配置审核员安排时间审核文档和记录，审核活动可能涉及项目范围、配置项的入库及出库、评审记录、配置项的变更历史、测试记录、文件的命名、变更请求、版本的编号。
- 配置审核员在审核中发现不符合现象，并作记录。
- 由项目经理负责消除不符合现象。
- 配置审核员验证所有发现的不符合现象确定已得到解决。

## 26.5　配置管理的团队支持

### 26.5.1　大型信息系统项目的特点

随着信息化时代的到来，信息系统功能越来越多，集成度越来越高，信息系统项目也就越来越复杂，管理也越来越困难。今天的信息系统项目已经常是在时间，空间上的一个广泛的集成了。它往往有如下 3 个特点：

**（1）系统复杂产品众多。**随着 Internet 和高速上网的兴起，现在的 ERP 系统、电子商务系统、电子政务系统，都逐步向高度集成方向发展，一个系统往往和多个系统互连形成一个统一的整体。同时在每个信息系统的开发过程中，都产生大量的文档、源代码和交流信息，这些都是信息系统项目开发实施过程中的产品，这些产品数量众多，类别也五花八门。并且有的项目还要求产生运行于不同环境的最终产品。

**（2）地域分布广泛。**现在世界的生产已经是一个全球协作的大生产了，同样在信息系统的生产过程中也往往是多个企业多个地理位置组织的一个全球协作过程了。经常是一个项目的团队分布在多个地方。特别是一些 Open Source 的机构，其开发团队更是分布在世界各地，互相之间甚至都不认识。

**（3）参加人员众多。**信息系统项目的工作量越来越大，参与者也就越来越多，并且参与者越多，信息交流的花销更多，花在开发工作上的时间也就相应减少，这样使得人越多效率越低，效率越低要求的人工也越来越多。这也是现代信息系统项目管理上需要解决的一大课题。

## 26.5.2　信息系统项目中的配置管理的实施

### 1. 变更冲突问题

如果项目中存在同时需要维护的两个版本的产品，那就会存在某些配置项需要同时维护两个版本的情况，这时我们要构建产品时怎样才能找到其需要的配置项版本？

或者一个配置项是共有的程序文档，一个开发小组需要在此程序中加入或更改一些变量或函数，如果直接将其更改后，或对其他小组的工作产生巨大的影响，而此小组对这些文件的更改还不能完全确定，可能会继续更改，直到稳定为止。

更坏的情况是如果两个小组对一个相同文件都要更改要求时，这时就将更为复杂。

### 2. 变更冲突的解决

对上述的两个问题，我们可以采用常规的办法——建立多个配置库来加以解决。对于前一个问题我们对每个需要平行开发维护的版本都各建立一个配置库，这样，每个版本的维护都可以以一个独立的项目来进行。对于后两个问题，我们可以采用在开发小组内设立一个配置库的方式来进行，更改先在组内配置库中进行，检验证明无误才提交到主配置库中。

但这样由于有多个配置库，那么要保证每个配置库内容的一致性和完整性就需要新的工作量来完成，如果不保持每个配置库的一致性就将造成灾难性的后果。如何保证每个配置库的一致性呢？

我们需要加强变更控制力度，在小组内对变更也应进行强有力的控制措施，避免随意改动，并且在提交到主配置库中之前还应该对变更带来的影响进行分析，将影响降到最低。

我们还得加强配置审核工作，力保更改的有效性和正确性。

但是，我们这样解决又会带来新的问题——安全问题，如果只有一个配置库，那每个项目相关人员都只有一个账户和密码，但如有多个配置库，则每个项目相关人员都有多个账户和密码，这样随着账户和密码的增多，当然会带来安全隐患。另外还会增加额外的工作量。

对这几个问题，其实我们还有更行之有效的办法，那就是工作视图。

## 26.5.3　工作视图

### 1. 分枝

如果一个配置项在多个版本上都需要进行修改，这时，就形成了不同的分枝。

如图 26-6 所示，一个小组对配置项的 1.0 进行修改，形成 1.1、1.2、1.33 个版本，另一个小组对配置项的 2.0 进行修改，形成了 2.1、2.2、两个版本。这样从 1.0、1.1 到 1.3 就形成了一个分枝，2.0、2.1 到 2.2 形成了另一个分枝。而 1.0、2.0、3.0、4.0 也可以看成一个分枝，这个分枝是主分枝。

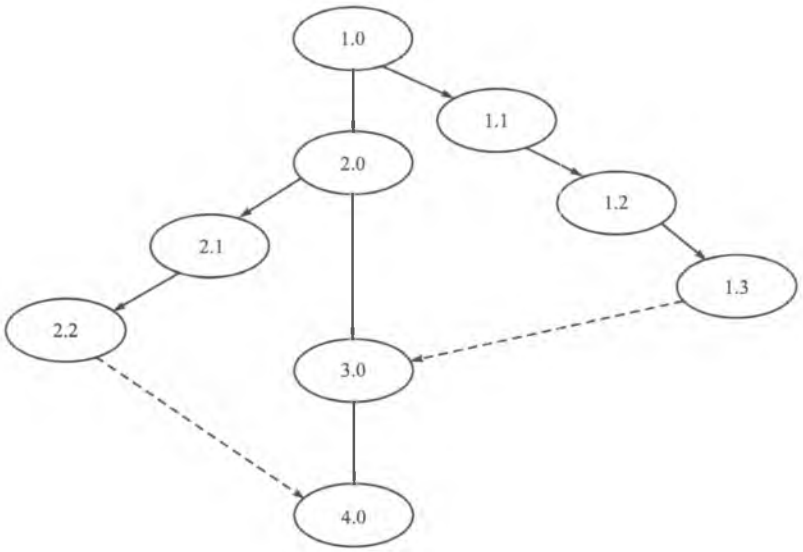


图 26-6　分枝图

### 2. 视图

项目中的每个人不可能对项目中的所有产品完全了解，并且对具体的产品也只能是对其部分版本了解，并且因为安全原因，每个人都应该只了解其应该了解的部分。于是项目中每个相关人员看到的项目信息就是对项目的一个视图，具体来说，视图就是对部分配置项分枝（可能是主分枝）的引用的一个组合。视图的使用者只是在视图的范围内对项目进行操作，对视图进行操作实际上就是对其引用的那些分枝的操作。

如表 26-2 所示为 3 个工作视图的示意，表中只列出视图中相关配置项的当前版本和修改属性。

表 26-2　工作视图的示意

| 配置项 | 视图 1 | 视图 2 | 视图 3 |
|---|---|---|---|
| CommonFunction.java | 3.0，只读 | 2.3，可写 | 2.0，只读 |
| Constraint.java | 2.6，可写 | 4.0，只读 | 4.0，只读 |
| JndiNames.java | 5.0，只读 | 5.0，只读 | 3.4，可写 |
| ServiceLocator.java | | 2.0，只读 | 2.0，只读 |
| StringMatcher.java | 1.2，可写 | 2.0，只读 | |
| Application.properties | 5.0 只读 | 3.1 可写 | 4.3，可写 |
| …… | …… | …… | …… |

视图有如下特点：

（1）每个配置项在同一个视图中最多引用一次。

（2）分枝在视图中可以是只读的也可以是可修改的，分别称为只读引用和可读引用。

（3）一个视图中引用的配置项有可能在别的配置库中，甚至是在 Internet 上的另一个服务器上。

（4）对视图中的引用进行检入，实际上是在引用的分枝上增加一个版本。对视图中的引用进行检出实际上是在引用的分枝上进行检出，先获取此分枝的最新版本然后再做检出标记。

### 3．分枝合并

分枝在使用一段时间后，可能已经稳定，这样需要将分枝合并到主分枝上（如图 26-6 中虚线所示，从 1.3 版到 3.0 版，从 2.2 版到 4.0 版），这个过程叫合并分枝，由于合并分枝后可能会对项目中其他相关人员造成重大的影响，因此合并分枝前应该有一个严格的评审过程，并且分枝的合并应该由专门的配置管理人员来进行操作。

### 4．视图的优点

视图的优点有如下几个。

（1）每个人都只关注自己应关注的部分，使自己能够专心致志地工作，提高工作效率。

（2）视图可以支持大规模的并行开发，一个配置项可以支持多个人同时检出。

（3）每个人都在自己的视图上进行操作，不用担心对别人造成什么影响。

（4）每个人的权限都得到有效的控制，每个人都只能操作自己能处理的部分，不用担心安全问题。

（5）分枝的合并由专门的配置管理人员来完成，减少操作失误的可能性。