目　　录

前　言

本书是收集网上有关软件配置管理资料，经过整理而来。阅读时，没有顺序之限制，可以跳阅。

第一部分　软件配置管理

第一章　软件配置管理概述

1　概述

没有任何一个行业像计算机工业发展的如此迅速。在今天的软件产业更是如此，技术和产品的更新日新月异令，所有技术人员和管理人员都感到明显的压力，这种压力集中体现在两个方面：提高产品质量，缩短面市时间。现在软件产品的开发，对市场投放速度的要求成倍增长；**Internet/Intranet**应用的发展改变着软件的开发，传递，和分发方式；不断提高的软件质量的要求，使越来越多的软件机构感到规范开发迫切性；多平台，多操作系统，多开发工具，多对象类型，多计算机语言等，使复杂的软件开发环境更加难以控制。

我国信息技术产业的蓬勃发展促使各种先进技术和产品在国内被广泛的应用，为国内的软件开发注入了活力。然而，值得注意的是，各

种先进的操作系统，开发工具等在带来效益的同时，也使得我们的开发环境日益复杂化而难以管理。而无组织的开发环境，会导致潜在问题的产生，这些问题一般难以发现，直至出现影响整个系统的致命错误，但此时往往又为时晚矣。

软件工程使软件开发从手工作坊上升到团队开发模式，其开发工作围绕着软件生命周期的分析设计、开发、测试、运行维护四个阶段进行。通过使用软件工程的方法及工具，可以避免开发过程中许多可能出现的错误，提高软件的可重用性，降低软件测试和维护中的工作量，从而大大提高软件产品的质量，缩短开发周期。

在团队开发的模式中，软件开发管理就显得更加重要，其管理的好坏将直接影响到软件产品的质量。如果缺乏对软件开发的统一管理，势必造成以下问题的出现：

- 由于开发经费及开发时间的限制，不可能一次开发就解决所有问题，许多问题有待维护阶段解决，因此带来的是软件产品的不断升级，而维护和升级所必需的文档往往非常混乱；
- 开发商开发过程缺乏规范化的管理，即使有源程序文档也由于说明不详细而不能对产品进行进一步的功能扩充，用户不得不再投入大量的经费去开发新产品，浪费大量的人力、物力和时间；
- 在软件的团队式开发中，人员流动在所难免，如管理不善，有些人员的流动将对开发产生致命的影响。特别是软件开发管理人员或核心成员的流失，有可能造成无法确定软件产品中各模块所处的状态及阶段，使软件产品的版本出现混乱，甚至可能泄漏公司的核心机密；
- 管理不善致使没经测试的软件加入到产品中，不但影响产品的质量，有时还会导致致命的错误，造成不可挽回的损失；

● 用户与开发商没有有效的沟通手段，用户投入了开发费用后，得到的是有关可执行程序以及一堆杂乱无章的文档，即使是较好的文档，对不熟悉开发过程的专业人员来说也无从下手，更谈不上日后的维护和升级，用户的利益无法保证；

● 软件生产达不到规模化，无法生产出软件企业内部的软件标准构件仓库，使应用软件产品总处于一种低水平、重复开发的状态，不但时间得不到保证，而且成本也无法降低，使产品没有市场竞争力。

这些问题在实际开发中表现为，项目组成员沟通困难，软件重用率低下，开发人员各自为政，代码冗余度高，文档不健全等；造成的结果是：数据丢失，开发周期漫长，产品可靠性差，质量低劣，软件维护困难，用户抱怨使用不便，项目风险增加等。

2 软件配置管理

缺乏软件开发管理，会导致种种问题的出现，这些问题使得最终开发出来的软件产品的质量难以保证，应用难以稳定。那么，怎样进行软件开发管理才能生产出高质量的软件产品呢？在ISO9000 质量管理和质量保证标准中，制定了《在软件开发、供应和维护中的使用指南》标准，该标准除对软件生命周期的各个阶段做了严格的规定外，还在其质量体系中规定了与阶段无关的支持活动，其中软件配置管理（Software Configuration Management， 简称SCM）被放在首位。

2.1 SCM 概念

早在七十年代初期加利福利亚大学的**Leon Presser**教授就撰写了一篇论文，提出控制变更和配置的概念，之后在**1975**年，他成立了一家名为**Soft Tool**的公司，开发了自己的配置管理工具：**CCC**，这也是最早的

配置管理工具之一。之后，随着软件开发规模的逐渐增大，越来越多的公司和团队意识到了软件配置管理的重要性，而相应的软件配置管理工具也如雨后春笋一般，纷纷涌现，比较有代表性的有：**Marc Rocking**的**SCCS(Source Code Control System)**和**Walter Itchy**的**RCS(Revision Control System)**，这两种工具对日后的配置管理工具的发展做出了重大的贡献，目前绝大多数广泛使用的配置管理工具基本上都是基于这两者的设计思想和体系架构。

什么是软件配置管理呢？软件配置管理有多种定义，在**1986**年出版的**Wayne Babyish** 《 **Software Configuration Management: Coordinating for Team Productivity**》一书中把软件配置管理描述为"对软件开发组所建立的软件的修改进行标识、组织和控制的艺术，其目标是减少错误，提高生产力"。这个定义比较简单，而在**1993**年出版的**Steve McConnell**的《**Code Complete**》一书中，从另一个角度对软件配置管理进行了定义："配置管理能够系统地处理变更，从而使得软件系统可以随时保持其完整性。配置管理又可称为'变更控制'，可以用来评估提出的变更请求，跟踪变更，并保存系统在不同时间的状态。

" 软件配置管理是一套规范、高效的软件开发基础结构。作为管理软件开发过程有效的方法，**SCM**早已被发达国家软件产业的发展和实践所证明。**SCM**可以系统地管理软件系统中的多重版本；全面记载系统开发的历史过程，包括为什么修改，谁作了修改，修改了什么；管理和追踪开发过程中危害软件质量以及影响开发周期的缺陷和变化。**SCM**对开发过程进行有效地管理和控制，完整、明确地记载开发过程中的历史变更，形成规范化的文档，不仅使日后的维护和升级得到保证，而且更重要的是，这还会保护宝贵的代码资源，积累软件财富，提高软件重用率，

加快投资回报。SCM是通往ISO9000和SEI CMM标准的一块基石。

在软件开发团队中，正确地采用、实施软件配置管理系统，必将提高生产力，增强对整个项目的控制，改善软件产品的质量，从容面对快速面市和产品质量的双重压力。软件配置管理系统的实施，一般来讲要考虑两个方面的因素：流程和工具。流程和工具是相辅相成的，流程起决定性作用，它确定了管理的规则和方法，工具用来将变更存储在一个中央存储库中，可以重现任一时期的历史版本，一个好的工具可以提高效率，是贯彻实施流程的必要手段。

因此，在一个开发团队中，实施配置管理流程比采用配置管理工具更重要，我们需要充分考虑，制定出适合自己企业的配置管理流程，该流程必须与公司的开发规范、质量系统等完全结合。

2.2 配置管理的功能

配置管理系统应该具备以下主要功能：

并行开发支持：因开发和维护的原因，要求能够实现开发人员同时在同一个软件模块上工作，同时对同一个代码部分作不同的修改，即使是跨地域分布的开发团队也能互不干扰，协同工作，而又不失去控制

修订版管理：跟踪每一个变更的创造者、时间和原因，从而加快问题和缺陷的确定

版本控制：能够简单、明确地重现软件系统的任何一个历史版本

产品发布管理：管理、计划软件的变更，与软件的发布计划、预先定制好的生命周期或相关的质量过程保持一致；项目经理能够随时清晰地了解项目的状态

建立管理：基于软件存储库的版本控制功能，实现建立（build）

过程自动化

过程控制：贯彻实施开发规范，包括访问权限控制、开发规则的实施等

变更请求管理：跟踪、管理开发过程中出现的缺陷（Defect）、功能增强请求（RFE）或任务（Task），加强沟通和协作，能够随时了解变更的状态

代码共享：提供良好的存储和访问机制，开发人员可以共享各自的开发资源

2.3 配置管理的实施

实施配置管理系统，一般的步骤和需要考虑的问题如下：

2.3.1 规划、调整网络开发环境

一个规划良好的开发环境，是实施配置管理系统的前提。在此阶段我们要对配置管理系统做出规划，主要考虑以下问题：

网络的带宽、拓扑结构

服务器的选择、命名规范

存储区的定位

开发人员及组的命名规约等

2.3.2 设计配置管理库

根据项目开发的要求，设计开发资源的存储模式，良好的存储模式有利于减轻管理上的负担，增强配置管理库的访问性能，同时便于控制访问权限，保护软件资产。

2.3.3 定义配置管理系统的角色

在此阶段，我们需要确定与配置管理相关的所有角色，包括他们的相应的活动。在开发过程中，一个开发人员可能兼任多种角色，但一项任务在同一时刻只能由一个角色来执行。

一般配置管理中的角色主要包括：

项目经理：项目经理在配置管理方面的职责是依靠配置管理员、系统管理员和系统体系结构设计人员的帮助，制定项目的组织结构和配置管理策略。这些工作包括： 定制开发子系统，定制访问控制，制定常用策略，制定集成里程碑，以及进行系统集成

配置管理员：配置管理员的职责是根据项目经理制定的开发组织结构和策略，实施、维护配置管理的环境。其主要职责如下：创建配置管理库，对存储库进行日常备份和恢复，维护配置管理环境，及管理配置管理相关的用户

软件开发人员：软件开发人员依据项目的开发和配置管理策略，创建、修改和测试开发工件

集成人员：对软件进行归并，形成相应的基线或发布版本

QA 人员：需要对软件配置管理有较深的认识，其主要工作是跟踪当前项目的状态，测试，报告错误，并验证其修复结果

2.3.4 制定配置管理流程

这是配置管理实施的一个重要阶段，其主要目的是根据项目开发的需要，制定相应的配置管理流程，以更好地支持开发，主要活动包括：

定制并行开发策略：合理的并行开发策略应该具有以下特点：协

调项目的复杂性和需求，统一创建分支类型和元数据，为开发过程中的变更集成制定有效的规范，适时反映开发过程中方法和需求的变化

发布版本管理：软件开发过程中的一个关键活动是提取工件的相关版本，以形成软件系统的阶段版本或发布版本，我们一般将其称为稳定基线。一个稳定基线代表新开发活动的开始，而一系列定制良好的活动之后又会产生一个新的稳定基线。有效地利用此项功能，在项目开发过程中可以至始至终管理、跟踪工件版本间的关联。

2.3.5 相关人员的培训

一般来讲，实施配置管理系统，相关人员需要接受一下培训：

管理员培训：针对配置管理员，主要学习配置管理工具管理相关内容

开发人员培训：针对开发人员，主要学习配置管理工具与开发相关的常用操作

管理流程培训：针对全体人员，目的是了解配置管理策略和流程，以及如何与开发管理、项目管理相结合

以上只是简单地介绍了配置管理系统实施的相关内容，软件配置管理作为软件开发过程的必要环节和软件开发管理的基础，支持和控制着整个软件生命周期。若要有效的实施软件配置管理，首先要通过一系列的培训，培养软件开发者的管理参与意识，同时更重要的是借助已有的经验教训，建立起真正适合自己团队的管理流程。

3 SCM 的意义

提到软件配置管理，作为从事软件的人来讲，想必都不陌生。要想

真正做到实施好配置管理，对于软件配置管理的意义及其重要性我想应该有必要的认识和理解。

软件配置管理，**software configuration management**，简称 **SCM**；

在软件配置管理中，有一个关键的一环就是变更管理，而变更管理的基础是配置项的确定与版本管理。要正确理解这些问题，我们不能仅仅

将**SCM**作为一个管理工具或者在项目洽谈与执行中一种合行规定的义务来履行。如果这样，在开展工作的过程中很容易使这种工作变成一种

官僚式的绊脚石。往往在我们开展项目时，很多合同对配置管理提出了明确的要求，需要认识的是，我们所需要进行配置管理的目的是为

软件开发过程中的不同的角色控制和跟踪管理自己的工作提供支持与帮助。

很多软件开发过程中遇到的问题都是因配置管理不善而造成的。而发生这些问题需要时间去确定，而且有可能很多可能是重复的问题。

有的是不必要的麻烦。比如说一个已花费较大精力和成本解决的高难度的软件错误突然再次出现，已经开发或完成测试的一个特性神秘的消失，一个已经通过完全测试的软件系统突然间无法运行。配置管理通过对同一项目中不同人员的所产生的工作产品来帮助我们减少和消除这些问题。

问题主要体现在：

-- 现在项目的开发大部分都是以叠迭式，渐进式的模型进行开发。在一个版本交付的同时，另一个版本可能还是进行测试，而进行同步开发的后续版本可能还在进行设计与开发阶段。在这个循环的过程中，如

果客户发现错误，那么不单只是针对客户的错误在现有的版本上进行修改完成就可以，同时要在后续的版本中体现。另外，如果在测试或开发的过程中发现了新的问题，那么对于以前正在使用的版本也需要考虑进行修改。在大系统开发的过程中，问题与修改问题的人，版本都会比较多，很容易出现混乱的情况。

--核心代码，或者公用构件和代码。在系统的开发过程中，当涉及到公用构件或代码的修改时，需要使与此相关的人都需要知道。如果

没有有效的代码管理与报告与协调机制，对于修改的代码如何使相关人员都提到通知就存在一个问题了。

--现在的软件项目，大多都是由一个团体协作完成的，那么，涉及到，对于最后某人对其所作的工作或输出很容易损害到其它相关人员的工作。如在一个应该系统的开发过程中，数据流程比较密集，如果其接口的变化，可能会引起很多相关地方的变化。

这些问题是由什么而引的呢，不言而明，在软件开发过程中的缺少规范的管理而导致出这些问题。需要我们花费很大的精力与时间来处理。

那么怎么来对这些问题来形成一个有效的解决方案呢，需要我们对以下的问题进行明确：

--在公司，目前的配置管理是什么，做了些什么？

--目前公司配置管理的状态是何状态？

--如何去控制配置的变更项？

--对于配置变更，怎么样通知相关的个人和组？

--公司的软件项目都有哪些类型的变更？

--如果在公司或同一项目组，其它人所做的变更会不会影响你所写

的软件部分？

第二章　SCM 的基本概念

配置管理的基本概念

随着国内软件业的崛起和成熟，软件配置管理越来越得到重视。可以说，软件业要想更好的发展，没有软件配置管理的支持是不可能的。手工作坊式的软件开发模式将会成为历史，如何把国外成熟的软件配置管理理论和经验消化吸收，进而应用到国内软件开发中就成为国内软件业迫在眉睫的任务了。软件配置管理是管理和技术相结合的一门学科。应该说，软件配置管理理论难以理解是其难以实践的原因。本文试从基本概念的角度来探讨这门对软件开发具有重要意义的领域。

什么是配置管理

在软件开发中，变更是不可避免的。从某种角度上讲，软件开发过程就是一个变更的过程。有些变更是有益的，是具有创造性的，但是，也有些变更是有害的，导致混乱的。正像**James Bach** 总结的那样：

我们为变更所困扰，因为代码中的一个极小的混乱可能带来产品的大的故障，但是，他也能够修复大的故障或启用奇妙的新能力。我们为变更所困扰，因为某个喜欢恶作剧的单个开发者可能破坏掉项目，但是，一些奇妙的思想也源自那些喜欢恶作剧的人员。

因此，如何管理这些变更是一个软件开发能否成功的关键。简言之，软件配置管理就是管理变更的过程，它贯穿着几乎软件的整个生命周期。成功的配置管理系统可以提高产品的质量、项目开发效率，而且最

大限度的减少对个别“英雄”式人员的依赖。

软件开发过程的输出信息可以分为三个主要的类型：(1)计算机程序（源代码、中间代码和可执行程序），（2）描述计算机程序的文档（针对技术开发者和用户），（3）数据（包含在程序内部或在程序的外部）。这些项包含了所有的在软件过程中产生的信息，总称为软件配置。该集合中每一个元素称为该软件产品软件配置中的一个配置项（CI，Configuration Item）。

尽管配置管理（Configuration Management ）这个概念被提出有几十年了，但是，业内还没有一个全面而权威的定义。Configuration 的意思是“使成形”，它来源于拉丁语的com-（表示“与”或者“一起”）和figurate (形成)。它还有一个意思是“组成部件或元素的相对排列”。因此，配置管理（Configuration Management ）指的是管理组成部件或者元素的相对排列。配置管理的概念来自于硬件领域，美国国防部最早使用了配置管理的概念。

我们知道一架飞机的构成非常复杂，比如机头、机身、机翼和机尾等。不同型号飞机的各个部分是不能随便组装的。因此，我们只有把相匹配的部件组装在一起，才能构成了一个功能完备的飞机整体。随着技术的提高，各个部件可能还要进行功能改善，我们还要使得不同版本的部件能够正确无误组合在一起。

准确地说：

配置管理是对产品进行标识、存储和控制，以维护其完整性、可追溯性以及正确性的学科。

从上面的描述，我们知道，配置管理的基本单位是配置项。软件配

置项可以是：

与合同、过程、计划和产品有关的文档和数据

源代码、目标代码和可执行代码

相关产品，包括软件工具、库内的可复用软件、外购软件及用户提供的软件

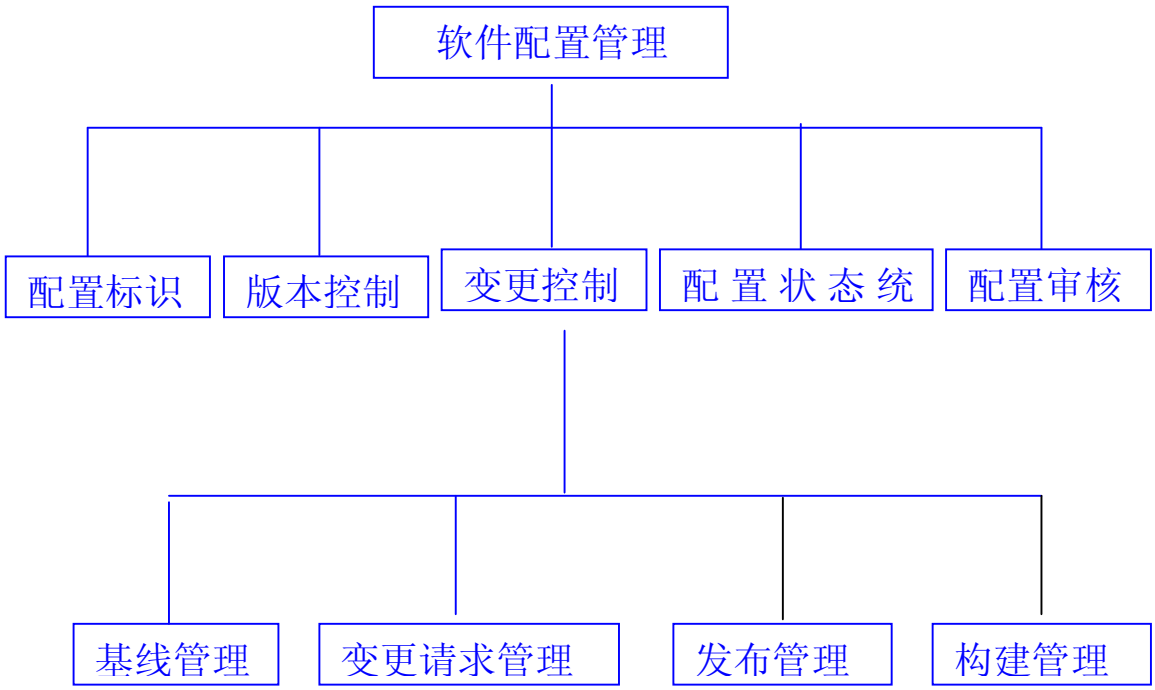
从“哲学”意义上讲，配置管理记录配置项的三个方面：

从哪里来？此项可归结为WWW 的问题，（Who）谁创建的？（When）什么时间创建的？（Why）为什么创建此配置项？

当前在哪里？此项纪录配置项当前的存储位置以及状态。

将到哪里去？通过配置控制来把配置项“组装”到正确的版本中去。

配置项可以是大粒度的，也可以是小粒度的。如果跟踪个别需求，那么不必要把整个需求规格说明文档定义为一个配置项，可以把每个需求定义为配置项；如果把软件开发工具也放入配置管理系统，那么把配置项定义为文件级就不合适了，只需要跟踪开发工具的版本，即把整个配置工具定义为一个配置项就足够了。简而言之，配置项可以是文件级粒度的，也可以使文件版本级粒度的。当然，粒度越小管理的成本越高，但是配置的精度也就越高。一个完整的SCM 系统要具有三个核心功能：配置标识、版本控制、变更控制、配置状态统计和配置审核。其中变更控制包括基线管理、变更请求管理、构建管理和发布管理。如下图所示。



下面，我们来具体理解这些概念。

配置标识

配置标识就是识别产品的结构、产品的构件及其类型，为其分配唯一的标识符，也就是说，每一个配置项要有一个唯一标识。一般说来，标识包括两个方面：一是文件名，二是版本，可用如下一个二元组来标识：<文件名，版本>。每个项目首先要确定一套命名规则，例如，采用“系统.子系统.模块.文件”的方式，**</videoConference /audio /compressing /main.c , 2.1>**就是一个唯

一.标识。

版本控制

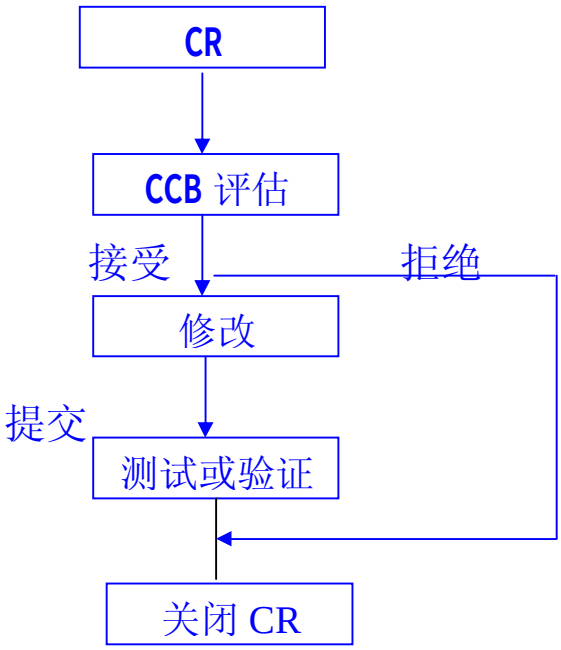
版本控制就是对在软件开发过程中所创建的配置对象的不同版本进行管理，保证任何时候都能取到正确的版本以及版本的组合。

当前，这方面典型的工具有如**VSS** 和**CVS**。

变更控制

在软件开发过程，要产生许多变更，比如，配置项、配置、基线、构建的版本、发布版本等。对于所有的变更，都要有一个控制机制，以保证所有变更都是可控的、可跟踪的、可重现的。对变更进行控制的机构称为变更控制委员会（**Change Control Board**，简称**CCB**）。变更控制委员会要定期召开会议，对近期所产生的变更请求进行分析、整理，并做出决定。而且要遵循一定的变更机制。

下面是一个典型的变更机制：



变更请求管理

变更请求管理就是对变更请求（**Change Request**，简称**CR**）进行分类、追踪和管理的过程来实现的。

变更的起源有两种：功能变更和缺陷修补（**Bug-Fix**）。功能变更是为了增加或者删除某些功能。缺陷修补则是对已存在的缺陷进行修

补。

对变更请求的有效管理可以提高产品管理的透明度，经理可以清楚的知道当前产品的进展情况，比如有多少个新产生的**CR**，已经解决了多少**CR** 等等，有利于经理做出正确的决策。

基线管理

基线是指经过正式评审和批准，可作为下一步工作的基准的一个配置。软件开发过程中，无论是需求分析、设计、测试都需要在完成时建立基线，以作为下一步工作的基础。通过基线管理可以使用户能够通过对 适当版本的选择来组成特定属性（配置）的软件系统，这种灵活的“组装”策略使得配置管理系统像搭积木似的使用已有的积木（版本）组装成各种各样、不同功能的模型。基线的变更需要一个严格的流程，需要提出申请，经过审批，然后才能进行。

构建管理

在做构建时，我们需要首先取出正确的配置，然后再做构建。我们可以利用基线，可以取出某个基线的所有配置项，也可以利用配置管理系统的构建功能直接在工作空间内做构建。构建管理需要配置管理工具的支持。

发布管理

软件产品的每个版本都是一组配置项（源代码、文档、数据）的集合。举个例子来说，我们要发布软件的**32.6** 版本，那么我们就要把源代码、文档、数据中所有应该包含到这个版本中的正确配置项检出。

所以如何管理每个版本中包含哪些配置项是非常重要的。

状态报告

状态报告要回答所谓**4W** 的问题：

What：发生了什么事？

Who：谁做的此事？

When：此事是什么时候发生的？

Why：为什么做此事？

状态报告要能够报告所有配置项以及变更请求的状态，通过量化的数据和报表反映项目开发进度的状态。

配置审核

配置审核要审查整个配置管理过程是否符合规范，配置项是否与需求一致，

记录正确，配置的组成是否具有一致性等等。比如，需求分析文档提交后，需要由一个由相关人组成的小组进行正式评审，只有通过了评审才能基线化。对于源代码也一样，一般说来，每行代码都要进行评审（**Review**），只有通过评审才能交由测试人员进行测试。

实施配置管理的好处

我们知道软件有三个要素：时间、预算和质量。一个成功的软件就是要在限定的时间内，不超过预算，交付符合质量要求的产品。真正实施配置管理后，我们会对产品的开发过程进行有效的控制，可以加快开发进度，降低开发成本，保证产品的质量。

产品经理可以得到什么好处呢？

准确掌握项目的开发进度。配置管理系统可以提供详尽的状态报告，例如当前系统有多少个**Bug**，所有**Bug** 的状态如何？已经解决了多少**Bug**？

了解项目组成员的工作负荷、工作效率以及工作质量。例如，我们可以知道当前分配给每个成员的工作量，每个成员已完成的工作

量，每个成员未通过正式评审的工作比例等等。

减少人员流动所带来的影响。每个成员的所有变更，包括文档、代码的增删都是可追踪的，而且对于变更的原因、描述也都有记录。这样，一旦成员离开，其它成员就可以在最短的时间里接手。

有效提高过程管理，配置管理产生的许多数据可作为管理者度量项目的依据。

开发人员和测试人员可以得到什么好处呢？

提交的代码被有效保存，开发人员再也不用花费精力去保存各个版本了。

提高团队的协作效率。开发人员之间以及开发人员和测试人员之间可以有效的沟通，大家都互相知道其它人的工作状态。

提高修复缺陷的效率。可以依据**Bug** 发现的版本，迅速重建环境，重现**Bug**，快速定位代码，找出根源。

职责清楚，任务明确。每一步的工作都是基于某一基线的，比如，设计文档是依据基线化了的需求分析文档，这样一旦出现问题，就可以找出问题出在什么地方。当然，实施配置管理的好处远不止这些。软件配置管理作为软件开发的基石，它提供了一个协作开发的环境，只有大家共同遵守配置管理规范，互相协作才能保证项目的成功。

结束语

配置管理本身无论从理论和实践都在不断丰富和发展。例如，配置管理应用于“知识库”的管理就产生了“内容管理”这一新的领域。配置管理提供的状态报告和数据统计也为软件度量提供了决策依据。配置管理为项目管理提供了各种监控项目进展的视角，为项目经理确切掌握

项目进程提供了保证。配置管理也为开发人员提供了一个协作的平台，在此平台上，大家能够更有效率的交流和协作。可以说，配置管理是软件开发的基石！配置管理近年来在中国得到了极大的认可，可以毫不夸张的说，没有配置管理，就谈不上软件开发，就谈不上软件质量，就谈不上软件业的发展。随着软件业规模的扩大，配置管理的实施不是要不要的问题，而是什么时间、如何实施的问题了。

第四章　过程支持

第一节　软件配置管理过程及其关键活动

随着软件产业的崛起，软件工程技术正吸引着越来越多关注的目光。特别是以 CMM 为代表的先进的软件工程理念在国内也正日益受到业界广泛的重视。

软件配置管理（Software Configuration Management，SCM）作为 CMM 2 级的一个关键域（Key Practice Area，KPA），在整个软件的开发活动中占有很重要的位置。正如 Pressman 所说的：“软件配置管理是贯穿于整个软件过程中的保护性活动，它被设计来（1）标识变化，（2）控制变化，（3）保证变化被适当的发现，以及（4）向其他可能有兴趣的人员报告变化。” 所以，我们必须为软件配置管理活动设计一个能够融合于现有的软件开发流程的管理过程，甚至直接以这个软件配置管理过程为框架，来再造组织的软件开发流程。

本章参照了 CMM 2 级中软件配置管理的相关的要求，充分考虑了

在 CASE 工具的辅助下实现自动的软件配置管理的可能性，描述了在一个比较理想的软件研发组织中软件配置管理的流程及其关键活动。

一. 角色职责

对于任何一个管理流程来说，保证该流程正常运转的前提条件就是要有明确的角色、职责和权限的定义。特别是在引入了软件配置管理的工具之后，比较理想的状态就是：组织内的所有人员按照不同的角色的要求、根据系统赋予的权限来执行相应的动作。因此，在本文所介绍的这个软件配置管理过程中主要涉及下列的角色和分工：

项目经理（Project Manager，PM）：

项目经理是整个软件研发活动的负责人，他根据软件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。其具体职责为以下几项：

制定和修改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开发里程碑；

接受并审阅配置控制委员会的报告。

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。其具体职责为以下几项：

定制开发子系统；

定制访问控制；

制定常用策略；

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。

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。其具体职责为以下几项：

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；

识别软件开发过程中存在的问题并拟就解决方案。

系统集成员（System Integration Officer，SIO）：

系统集成员负责生成和管理项目的内部和外部发布版本，其具体职责为以下几项：

集成修改；

构建系统；

完成对版本的日常维护；

建立外部发布版本。

开发人员（Developer，DEV）：

开发人员的职责就是根据组织内确定的软件配置管理计划和相关规定，按照软件配置管理工具的使用模型来完成开发任务。

二．过程描述

一个软件研发项目一般可以划分为三个阶段：计划阶段、开发阶段和维护阶段。然而从软件配置管理的角度来看，后两个阶段所涉及的活动是一致，所以就把它们合二为一，成为“项目开发和维护”阶段。

项目计划阶段：

一个项目设立之初 PM 首先需要制定整个项目的计划，它是项目研发工作的基础。在有了总体研发计划之后，软件配置管理的活动就可以展开了，因为如果不在项目开始之初制定软件配置管理计划，那么软件配置管理的许多关键活动就无法及时有效的进行，而它的直接后果就是造成了项目开发状况的混乱并注定软件配置管理活动成为一种“救火”的行为。所以及时制定一份软件配置管理计划在一定程度上是项目成功的重要保证。

在软件配置管理计划的制定过程中，它的主要流程应该是这样的：

CCB 根据项目的开发计划确定各个里程碑和开发策略；

CMO 根据 CCB 的规划，制定详细的配置管理计划，交 CCB 审核；

CCB 通过配置管理计划后交项目经理批准，发布实施。

项目开发维护阶段：

这一阶段时项目研发的主要阶段。在这一阶段中，软件配置管理活动主要分为三个层面：（1）主要由 CMO 完成的管理和维护工作；（2）由 SIO 和 DEV 具体执行软件配置管理策略；（3）变更流程。这三个层面是彼此之间既独立又互相联系的有机的整体。

在这个软件配置管理过程中，它的核心流程应该是这样的：（1）CCB 设定研发活动的初始基线；（2）CMO 根据软件配置管理规划设立配置库和工作空间，为执行软件配置管理做好准备；（3）开发人员按照统一的软件配置管理策略，根据获得的授权的资源进行项目的研发工作；（4）SIO 按照项目的进度集成组内开发人员的工作成果，并构建系统，推进版本的演进；（5）CCB 根据项目的进展情况，审核各种变更请求，并适时的划定新的基线，保证开发和维护工作有序的进行。
这个流程就是如此循环往复，直到项目的结束。

当然，在上述的核心过程之外，还涉及其他一些相关的活动和操作流程，下面按不同的角色分工予以列出：
各开发人员按照项目经理发布的开发策略或模型进行工作；
SIO 负责将各分项目的工作成果归并至集成分支，供测试或发布；
SIO 可向 CCB 提出设立基线的要求，经批准后由 CMO 执行；
CMO 定期向项目经理和 CCB 提交审计报告，并在 CCB 例会中报告项目在软件过程中可能存在的问题和改进方案；
在基线生效后，一切对基线和基线之前的开发成果的变更必须经 CCB 的批准；
CCB 定期举行例会，根据成员所掌握的情况、CMO 的报告和开发人员的

请求，对配置管理计划作出修改，并向项目经理负责。

综上所述，配置管理的工作流程如图 **1** 所示：

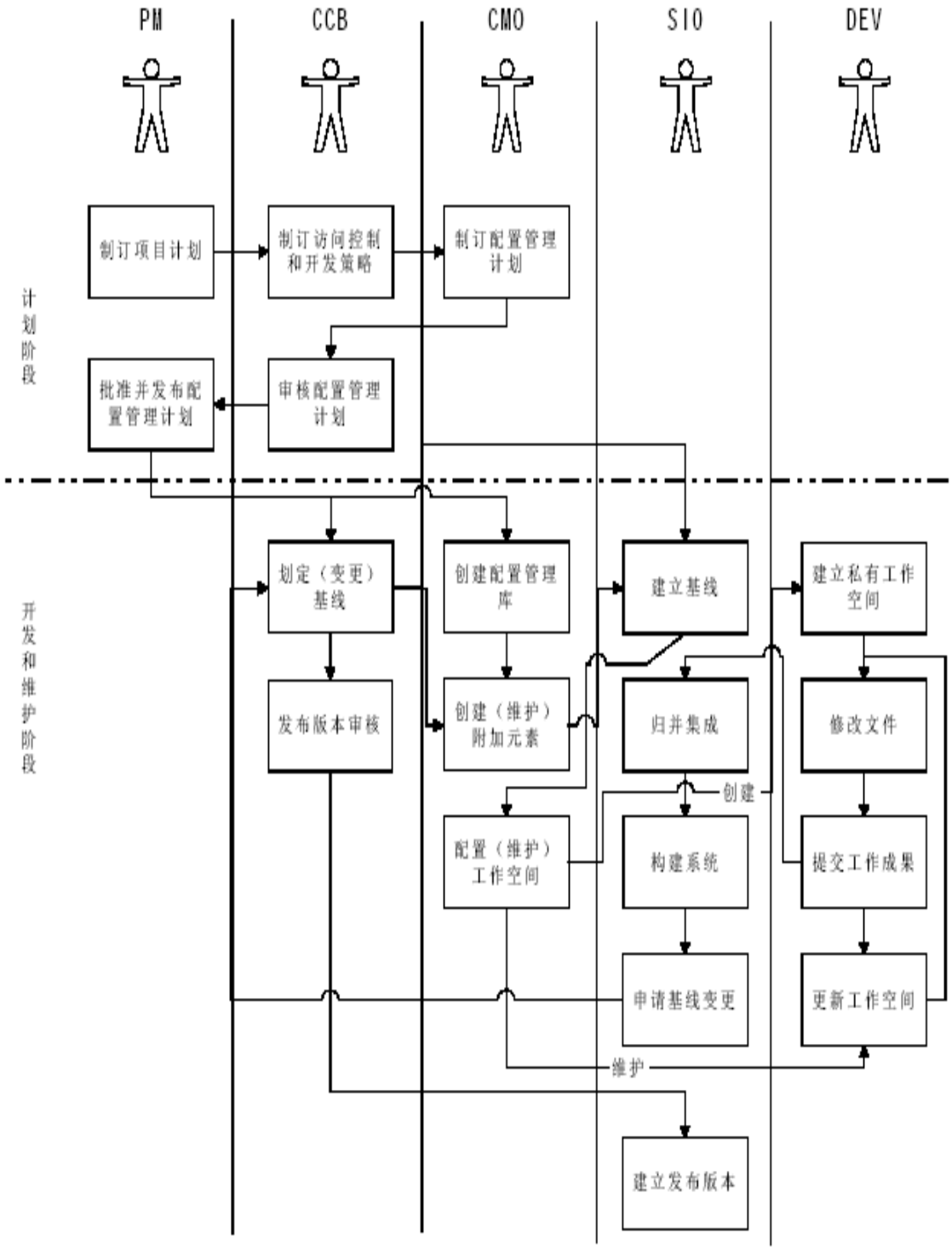


图 1. 软件配置管理基本流程

三．关键活动

1　配置项（Software Configuration Item，SCI）识别

Pressman 对于 SCI 给出了一个比较简单的定义：“软件过程的输出信息可以分为三个主要类别：（1）计算机程序（源代码和可执行程序），（2）描述计算机程序的文档（针对技术开发者和用户），以及（3）数据（包含在程序内部或外部）。这些项包含了所有在软件过程中产生的信息，总称为软件配置项。”

由此可见，配置项的识别是配置管理活动的基础，也是制定配置管理计划的重要内容。

软件配置项分类软件的开发过程是一个不断变化着的过程，为了在不严重阻碍合理变化的情况下来控制变化，软件配置管理引入了“基线（Base Line）”这一概念。IEEE 对基线的定义是这样的：“已经正式通过复审核批准的某规约或产品，它因此可作为进一步开发的基础，并且只能通过正式的变化控制过程改变。”

所以，根据这个定义，我们在软件的开发流程中把所有需加以控制的配置项分为基线配置项和非基线配置项两类，例如：基线配置项可能包括所有的设计文档和源程序等；非基线配置项可能包括项目的各类计划和报告等。

2　配置项的标识和控制

所有配置项都都应按照相关规定统一编号，按照相应的模板生成，并在文档中的规定章节（部分）记录对象的标识信息。在引入软件配置管理工具进行管理后，这些配置项都应以一定的目录结构保存在配置库

中。

所有配置项的操作权限应由 CMO 严格管理，基本原则是：基线配置项向软件开发人员开放读取得权限；非基线配置项向 PM、CCB 及相关人员开放。

3 工作空间管理

在引入了软件配置管理工具之后，所有开发人员都会被要求把工作成果存放到由软件配置管理工具所管理的配置库中去，或是直接工作在软件配置管理工具提供的环境之下。所以为了让每个开发人员和各个开发团队能更好的分工合作，同时又互不干扰，对工作空间的管理和维护也成为了软件配置管理的一个重要的活动。

一般来说，比较理想的情况是把整个配置库视为一个统一的工作空间，然后再根据需要把它划分为个人（私有）、团队（集成）和全组（公共）这三类工作空间（分支），从而更好的支持将来可能出现的并行开发的需求。

每个开发人员按照任务的要求，在不同的开发阶段，工作在不同的工作空间上，例如：对于私有开发空间而言，开发人员根据任务分工获得对相应配置项的操作许可之后，他即在自己的私有开发分支上工作，他的所有工作成果体现为在该配置项的私有分支上的版本的推进，除该开发人员外，其他人员均无权操作该私有空间中的元素；而集成分支对应的是开发团队的公共空间，该开发团队拥有对该集成分支的读写权限，而其他成员只有只读权限，它的管理工作由 SIO 负责；至于公共工作空间，则是用于统一存放各个开发团队的阶段性工作成果，它提供全组统一的标准版本，并作为整个组织的 Knowledge Base。

当然，由于选用的软件配置管理工具的不同，在对于工作空间的配置和维护的实现上有比较大的差异，但对于 CMO 来说，这些工作是他的重要职责，他必须根据各开发阶段的实际情况来配置工作空间并定制相应的版本选取规则，来保证开发活动的正常运作。在变更发生时，应及时做好基线的推进。

4 版本控制

版本控制是软件配置管理的核心功能。所有置于配置库中的元素都应自动予以版本的标识，并保证版本命名的唯一性。版本在生成过程中，自动依照设定的使用模型自动分支、演进。除了系统自动记录的版本信息以外，为了配合软件开发流程的各个阶段，我们还需要定义、收集一些元数据（Metadata）来记录版本的辅助信息和规范开发流程，并为今后对软件过程的度量做好准备。当然如果选用的工具支持的话，这些辅助数据将能直接统计出过程数据，从而方便我们软件过程改进（Software Process Improvement，SPI）活动的进行。

对于配置库中的各个基线控制项，应该根据其基线的位置和状态来设置相应的访问权限。一般来说，对于基线版本之前的各个版本都应处于被锁定的状态，如需要对它们进行变更，则应按照变更控制的流程来进行操作。

5 变更控制

在对 SCI 的描述中，我们引入了基线的概念。从 IEEE 对于基线的定义中我们可以发现，基线是和变更控制紧密相连的。也就是说在对各个 SCI 做出了识别，并且利用工具对它们进行了版本管理之后，如何保

证它们在复杂多变得开发过程中真正的处于受控的状态，并在任何情况下都能迅速的恢复到任一历史状态就成为了软件配置管理的另一重要任务。因此，变更控制就是通过结合人的规程和自动化工具，以提供一个变化控制的机制。

在本文的前面的部分中，已经把 **SCI** 分为基线配置项和非基线配置项两大类，所以这里所涉及的变更控制的对象主要指配置库中的各基线配置项。

变更管理的一般流程是：

A) （获得）提出变更请求；

B) 由 **CCB** 审核并决定是否批准；

C) （被接受）修改请求分配人员为，提取 **SCI**，进行修改；

D) 复审变化；

E) 提交修改后的 **SCI**；

F) 建立测试基线并测试；

G) 重建软件的适当版本；

H) 复审（审计）所有 **SCI** 的变化；

I) 发布新版本。

在这样的流程中，**CMO** 通过软件配置管理工具来进行访问控制和同步控制，而这两种控制则是建立在前文所描述的版本控制和分支策略的基础上的。

6 状态报告

配置状态报告就是根据配置项操作数据库中的记录来向管理者报告软件开发活动的进展情况。这样的报告应该是定期进行，并尽量通过

CASE 工具自动生成，用数据库中的客观数据来真实的反映各配置项的情况。

配置状态报告应根据报告应着重反映当前基线配置项的状态，以作为对开发进度报告的参照。同时也能从中根据开发人员对配置项的操作记录来对开发团队的工作关系作一定的分析。

配置状态报告应该包括下列主要内容：

A) 配置库结构和相关说明；

B) 开发起始基线的构成；

C) 当前基线位置及状态；

D) 各基线配置项集成分支的情况；

E) 各私有开发分支类型的分布情况；

F) 关键元素的版本演进记录；

G) 其它应予报告的事项。

7 配置审计

配置审计的主要作用是作为变更控制的补充手段，来确保某一变更需求已被切实实现。在某些情况下，它被作为正式的技术复审的一部分，但当软件配置管理是一个正式的活动时，该活动由 SQA 人员单独执行。总之，软件配置管理的对象是软件研发活动中的全部开发资产。所有这一切都应作为配置项纳入管理计划统一进行管理，从而能够保证及时的对所有软件开发资源进行维护和集成。因此，软件配置管理的主要任务也就归结为以下几条：（1）制定项目的配置计划；（2）对配置项进行标识；（3）对配置项进行版本控制；（4）对配置项进行变更控制；（5）定期进行配置审计；（6）向相关人员报告配置的状态。

在此，我想特别指出的是：由于软件配置管理覆盖了整个软件的开发过程，因此它是改进我们的软件过程、提高过程能力成熟度的理想的切入点。希望本文所描述的这个软件配置管理的角色分配和工作流程能在实践中不断地得到完善，从而使我们的软件开发活动能够更加有序、高效的进行！

第二节　如何构造软件企业的配置管理方案

1　引言

1.1　什么是配置管理

配置管理(Configuration Management)是通过技术或行政手段对软件产品及 其开发过程和生命周期进行控制、规范的一系列措施。配置管理的目标是记录软件产品的演化过程，确保软件开发者在软件生命周期中各个阶段都能得到精确的产品配置。

配置管理过程是对处于不断演化、完善过程中的软件产品的管理过程。其最 终目标是实现软件产品的完整性、一致性、可控性，使产品极大程度地与用户需求相吻合。它通过控制、记录、追踪对软件的修改和每个修改生成的软件组成部件来实现对软件产品的管理功能。

1.2　配置管理在软件开发过程和项目管理过程中的作用

随着软件系统的日益复杂化和用户需求、软件更新的频繁化，配置管理逐渐成为软件生命周期中的重要控制过程，在软件开发过程中扮演着越来越重要的角色。一个好的配置管理过程能覆盖软件开发和维护的各个方面，同时对软件开过程的宏观管理，即项目管理，也有重要的支持作用。良好的配置管理能使软件开发过程有更好的可预测性，使软件

系统具有可重复性，使用户和主管部门用软件质量和开发小组有更强的信心。 软件配置管理的最终目标是管理软件产品。由于软件产品是在用户不断变化的需求驱动下不断变化，为了保证对产品有效地进行控制和追踪，配置管理过程不能仅仅对静态的、成形的产品进行管理，而必须对动态的、成长的产品进行管理。 由此可见，配置管理同软件开发过程紧密相关。配置管理必须紧扣软件开发过程的各个环节：管理用户所提出的需求，监控其实施，确保用户需求最终落实到产品的各个版本中去，并在产品发行和用户支持等方面提供帮助，响应用户新的需求，推动新的开发周期。通过配置管理过程的控制，用户对软件产品的需求如同普通产品的订单一样，遵循一个严格的流程，经过一条受控的生产流水线，最后形成产品，发售给相应用户。从另一个角度看，在产品开发的不同阶段通常有不同的任务，由不同的角色担当，各个角色职责明确，泾渭分明，但同时又前后衔接，相互协调。好的配置管理过程有助于规范各个角色的行为，同时又为角色之间的任务传递提供无缝的接合，使整个开发团队象一个交响乐队一样和谐而又错杂地行进。正因为配置管理过程直接连接产品开发过程、开发人员和最终产品，这些都是项目主管人员所关注的重点，因此配置管理系统在软件项目管理中也起着重要。 配置管理过程演化出的控制、报告功能可帮助项目经理更好地了解项目的进度、开发人员的负荷、工作效率和产品质量状况、交付日期等信息。同时配置管理过程所规范的工作流程和明确的分工有利于管理者应付开发人员流动的困境，使新的成员可以快速实现任务交接，尽量减少因人员流动而造成的损失。

1.3 配置管理方案的构成

配置管理过程对软件开发有如此重要的影响，它的构造、实施过程也必定相当复杂。不借助工具，纯粹靠手工方式或只利用简单的工具来实现配置管理是很难做到满意程度的，而且其中的繁琐庞杂最终必定让管理者一愁莫展。因此，实现配置管理过程的通常做法是借助于专业化的配置管理工具，结合开发组织的实际情况制订出相应的配置管理规范，由开发人员在工作过程中依据规范，通过配置管理工具来实现。在这整个过程中，由配置管理工具负责那些非智能的、可自动化的管理过程，如身份角色验证、修改轨迹记录、版本控制等；由配置管理规范来控制那些 需要开发人员行方案确用智力去判断的因素，如需求合理性和优先级判定、任务分工、产品 的结构定义、版本发定等等。配置管理工具的采用和配置管理规范的制订 是紧密联系的，二者构成了一个软件开发机构的整体配置管理方案。这种方案是因组织的差异和配置管理工具的差异而变化的。构造一个配置管理方案涉及到软件开发组织和开发过程的各个方面，是一个复杂的工程应该当作一个项目来做。本文试图给出一个构造配置管理方案的基本策略和主要步骤。

2 组建配置管理方案构造小组

构造或完善一个软件开发组织的配置管理过程需要在构造初期花费较大的人力物力。这种工作一般是由一个临时组成的软件配置管理过程构造小组来完成。这个小组负责构造配置管理过程中的所有工作，包括了解本组织的现有开发、管理现状，选择配置管理工具，制订配置管理规范，安排试验项目的实施，沟通部门间关系，获得管理者支持和开发人员的认同。

配置管理过程构造小组的成员应该包括：

小组负责人

其对整个构造过程负责。主要职责是协调与其它部门或与上级主管的关系， 监督工作进程，协调小组内部关系。

技术支持专家

其负责在技术、设备方面为本组提供支持和服务，并负责本同其它部门就技 术问题进行联络，如了解相关项目情况、开发环境、开发人员状况等。

配置管理技术专家

其对配置管理过程的构造和配置管理工具十分熟悉。主要任务是指导配置管 理过程的构造，帮助制订配置管理规章，负责对开发人员进行配置管理工具的培 训。通常是配置管理工具提供商或专门的配置管理顾问机构的人员担当此任。

配置管理系统用户代表

他们是从将来要在实际的项目开发过程中使用该系统、遵照该过程的开发人 员中挑选出来的。他们负责从构造初期了解配置管理系统和规程，根据开发经验协 助制订、修改配置管理规程，并在试验项目中担任部分开发角色。这部分成员应包括软件开发项目经理、设计人员、编码、测试和构造、发布人员。 该项目小组成立后，将按后述步骤开展配置管理过程的构造工作。

3 对目标机构进行了解、评估

"知己知彼，百战不殆验"。配置管理过程的构造过程也是如此，必须对相互 作用的双方都有较透彻的了解才能达到预期的效果。因此首

先要做的事情是调查了 解，既要了解目标机构（即将要采用该配置管理过程的软件开发组织）的情况，又

要了解配置管理工具的情况。 目标机构的调查评估工作由配置管理技术专家领导，配置管理系统用户代表参与，提供基本信息，并由小组负责人协调，对相关部门人员进行深入调查获得较 全面的数据。 对目标机构的了解、评估应从这几个方面入手：人员、技术、工作流程、现 有项目和期望值。

3.1 人员评估

人员评估的目的是了解目标机构的员工对现有配置管理过程的评价和对采用新工具、制订新规范的态度，预测新的配置管理过程构造中的工作难点和可能遇到 的阻力。调查的方面包括： 该组织员工对引入新工具的反应，以前是否有过类似的偿试。 该组织负责人对新工具、新流程的支持程度。 开发人员的素质、教育程度、沟通能力。开发队伍的稳定性。 该组织的沟通渠道是否通畅。

3.2 技术评估

对目标机构技术方面的的调查、评估将直接导致对工具的选择。要了解的信息有：

目标机构有哪些可用的计算资源。

在什么软硬件平台上进行开发。

是否存在资源瓶颈，是什么。

现用什么开发工具，用户对该工具评价如何。

现用什么网络环境。

使用什么编程语言。

目标平台是否与开发平台一致。

代码更新程度如何，新编代码、重用代码和历史代码各占什么比例。

3.3 现有流程评估

对目标组织现有工作流程的评估直接影响新的配置管理流程和规章的制订。

调查的方面是：

现有流程的成熟性、适用性和执行情况。

现有流程是否能进一步提高自动化程度。

现用什么开发模型。

对分析、设计、编码、测试、产品管理等过程是否有严格的成文规范，如何

保证该规范的执行。

开发流程中的哪些质量控制信息被收集，如何使用。

3.4 项目评估

配置管理系统对正在开发的产品、正在进行的项目有直接的影响，因此对即将纳入管理的项目应有充分的了解。

了解的方面有：

项目的平均工期（人月）。

项目的组织方式，是主程序员制还是开发小组制，按深度结构还是按广度结构组织。

项目的产品规模（功能模块数、源码行数）。

项目开发支持状况，是否有专门的开发环境、开发工具和配置管理等方面的 支持人员。

3.5 期望值评估

对目标机构的开发、管理人员对新系统的期望值的了解有利于对症下药，解 决其当前紧要问题，提高对新系统的信心。

调查的方面包括：

对当前本组织的生产率和产品质量的满意程度，期望有怎样的提高。

对现有流程的评价，现有流程中哪个环节希望改进或加强。

期望增减哪些文档或规则。

期望等到什么样的通信交流方式，现有方式的优缺点是什么。

期望收集哪些新的开发度量数据或简化哪些数据。

4 配置管理工具及其提供商评估

通过对目标组织的评估，了解该组织的现状和需求后，就需要选择适合该组 织的配置管理工具。市场上现有的配置管理工具不下数十种，它们各有所长，在功 能，性能等方面有较大的差别，只有经过仔细地对产品及其提供商进行分析评估， 核对目标机构的需求，才能挑选出合适的工具，实现一个理想的配置管理过程。

这种评估可从三个方面进行：配置管理工具的评估、供应商评估和其它用户 使用经验的评估。

配置管理工具评估

对工具的评估应侧重于功能的适用性，而不应一味强调功能的全面性。产品评估应了解如下问题：

该产品的哪一方面功能可解决目标组织的当前问题满足该组织在配置管理上 的需求。

该产品在目标机构的峰值负荷下的运行效率将如何。

该产品对并发使用的支持情况如何

该产品与现有系统、工具、流程、环境的兼容性如何。

该产品的成熟性和稳定性如何。

该产品是否易学易用。

该产品的购买、安装、实施、维护费用是否可以接受。

供应商评估

供应商的实力和它所能提供的服务和支持对配置管理系统的实施至关重要。

因为配置管理工具不象其它的工具那样，只要安装完成后按照使用手册和在线帮助

就能使用，而是必须在系统之外有一系列的操作、管理规范，有一套完整的方案。

这些些必须在系统提供者或顾问机构的帮助下才能制订、实施。因此，系统提供商

对配置管理过程的实现有重要影响。对供应商的评估包括：

供应商在相应行业的从业时长。

该产品是否是该供应商的主导产品。

该供应商的年销售额。

供应商在五年之内的稳定情况。

该供应商是否有专业化的客户支持队伍。

是否提供安装、用户培训等服务。

供应商的声望、信誉如何。

供应商的支持人员在地理位置上是否与目标机构邻近。

另外，通过了解同一产品的其它用户对该产品的评价可以对该产品和供应商

有较为客观、综合的认识。这种评价可从所知的用户组、专业会议、配置管理工具

公告板等途径获得。

5 制订实施计划

经过对目标机构和选用工具的评估，工作小组可以制订出一份完整的工作计 划作为下一阶段的行动纲要，同时也是向上级主管汇报，取得支持的有力佐证。

工作计划由如下部分组成：

必要性和影响因素

结合目标机构的开发过程组织和配置管理现状，论证构造或完善配置管理过程的必要性；根据所选配置管理工具的功能特性和供应商的实施支持，阐明新的配 置管理系统可对目标机构的开发和管理工作带来改进和驱动。另外，对该配置管理 系统和相应的配置管理过程对现有的人员、工序和管理等方面可能带来的影响作出适当的预测，以便减小将来实施时可能遇到的阻力。 配置管理目标和配置管理过程的构造成功标准 对正待构建的新的配置管理过程制订出一个较为长远的目标，即要达到哪种 控制程度，加强哪些方面的管理，是否按照相关的国家或国际标准实施，达到何种 级别等等。 另外，对构造配置管理过程的工作本身，如前所述应当作一个项目来做，因 此也必须制订一个明确的完成标准。该标准应该在本小组内部统一并获得上级主管认可。

人员组织和分工

进一步明确工作小组的组织成员和成员关系，为每个成员分派相应的任务和

职责。这些任务应该具体、细化到可操作的程度，如张三负责与某部门接洽，了解

原有某一管理规范，李四负责试验环境的准备工作等等。

进度计划

罗列出构造过程中所要解决的问题，设置里程碑。

风险管理

预测构造过程中可能遇到的外在困难因素，如硬件短缺、平台差异、与相关

部门冲突、试验项目的特殊性等等。为这些风险因素设计出降低或规避风险的方法。

6 定义配置管理流程

配置管理流程是软件开发机构进行配置管理的依据，也是配置管理构造工作小组的最重要的工作成果。配置管理流程规定开发过程中需要做哪些配置管理方面 的工作，由谁做、如何做。前两个问题有较为通用的答案，在后文将会涉及，第三个问题则必须根据目标机构的具体情况解决。制订配置管理流程的方法是：通过对目标机构的调查、评估，定义现有的配 置管理流程，由配置管理技术专家对它进一步分析，结合常规的配置管理方法制订出新的流程。之后，依据选定的配置管理工具的功能，将新流程中可自动化的环节交由配置管理工具处理，其它环节由新制订的配置管理规范控制。除了制订配置管理规范外，该小组还应制订出适合目标机构的配置管理基本章程。该章程进应包括配置管

理部门的设立、该部门的责能（通常是负责监督配置管理规范的执行情况，对配置规范行完善，并担当日常的内部配置管理过程支持任务），定义配置管理过程与开发过程的协调关系，以及各开发阶段的开发人员构

成、在配置管理流程中的责任划分等等。一般说来，配置管理包括四个方面的活动：配置项标志，配置项控制（修改控制），配置状态报告和配置审核。配置管理规范的制订也应按这四个方面内容进行。每一个方面要考虑的问题是：

配置项标志制订文档或文件编号、标记体系。

定义文档和文件之间的联系。

确定受控的配置项的取舍，如软件源码、硬件描述文件、中间文件、目标文件、测试方案、系统数据等等。

确定产品版本、基线的标志体系。

确定库程序的标志和管理机制。

配置项控制确定产品的版本的演化策略，规定何时、何人创建新的基线，如何创建。

确定修改请求控制机构（**CCB—Change Control Board**） 的人员组成、职能、工作程序。

确定修改请求的处理流程和终止条件。

确定修改请求处理过程中各开发人员的职能。

确定修改请求和所生成的结果的对应机制。

确定文档的修改方式。

确定配置项的提取方式。

配置状态报告定义报告的内容、形式、提交方式。

确定产品的发行事宜，包括发行时间如何确定、发行说明的生成发布方式、发行方式等。

配置审核确定审核的执行人员、执行时机，审核的内容和方式。

确定发现问题后的处理方法。

7 试验项目的实施

这一阶段的任务是选取目标机构中的一个现有项目，按既定的配置管理流程去进行开发和配置管理工作。这种试验的目的是在一定风险范围内，通过实地运作来确定所选配置管理工具、所制订的配置管理规范是否能满足目标机构的需要。要做的工作有：

选定试验项目

该项目应该具有一定的复杂性，但又有较强的独立性，不会对目标机构的关键项目造成重要影响。

选定试验组成员

通常应包括构造小组的部分成员和该项目原有的成员。

定义试验成功的标准和试验时间表

应以配置管理流程和项目开发管理过程的协同程度和总体工作效率为依据。

人员培训

包括配置管理工具培训和配置管理规范培训。

配置管理工具的安装和项目环境的搭建

包括将历史代码导入到新系统中，将原有配置管理信息转换成新系统的形成，置于新系统控制之下；搭建项目所需的软硬环境等。

开发过程按新的配置管理流程进行试验项目的开发，及时收集项目

开发人员的反馈信息。

调整配置管理流程

根据项目的进行情况和开发人员的反馈信息，找出新配置管理流程的不足，及时调整改进。

8 全面实施

经过试验项目证实、校正后的配置管理流程就可以在目标机构的各个项目、各个相关工作环节中去应用、实施，最终使配置管理过程日常化、规范化。全面实施过程主要由配置管理部门根据新的配置管理流程来指导。配置管理过程构造小组的作用趋于淡化，主要起监督和支持作用。该小组在全面实施过程中逐步解散，小组中部分成员可转移到配置管理部门中去。

全面实施阶段的任务有：

组建或完善配置管理部门，并完成配置管理流程的移交。

由配置管理部门制订各个项目的配置管理实施计划。

进行全组织范围的配置管理系统和规则的培训。

帮助各个开发项目向新流程转移。

进行日常的监督、抽查、评估和规范的完善工作。

9 结束语

配置管理过程的建立是一个复杂而漫长的过程，因为它受软件开发机构的许多方面的影响，包括技术、设备、项目、制度、人员、文化等因素。就象其它任何新事物的出现一样，在一个机构内刚刚建立的配置管理过程必然会受到各方面的挑战和考验，因此需要有一个适应、融合的过程。另外，配置管理过程的建立也不是一件一劳永逸的事情，不同

机构、同一机构在不同的发展阶段或不同的项目中会有不同的配置管理细节，这些都需要配置管理部门在长期工作过程中对配置管理过程不断调节、充实、完善

第二节　SCM 的最佳实践

现在大家都已经认识到了有效的　软件配置管理工作对于提高团队开发效率、保障软件产品质量的重要意义，很多朋友也开始了在配置管理实施方面的一些研究，市场上我们也可以看到一些软件配置管理工具厂商针对具体配置管理工具提供的实施服务；但是，实施软件配置管理到底应该做哪些东西？团队的配置管理现状怎么评估？在哪些方面还可以进行改进？我们相信，这些问题可能正困扰着大多数研发主管和项目经理。

国外软件产业界在软件配置管理这个专题上已经进行了多年的理论和实践上的研究。在多年经验积累的基础上，产业界总结出来一系列“最佳实践”（Best Practices），我们可以使用这些“最佳实践”来作为评估一个组织软件配置管理能力的标尺，也可以作为我们实施软件配置管理的指南。这些“最佳实践”包括：

1、 标识需要进行存储的工件（Artifact）并保障安全存储；

2、 控制并且审计（Audit）对于工件的修改；

3、 设立并管理基线（Baseline）；

4、 记录并跟踪变更请求；

5、 维护稳定、一致的工作空间；

6、 支持对于工件和控件的并发修改；

7、 尽早集成、持续集成；

8、 保证软件构建的重现能力；

9、 以控件（Component）为单位实施版本控制；

10、 使用“活动”（Activity）来组织和整合版本集。

下文将介绍前 5 条最佳实践。

1、标识需要进行存储的工件（Artifact）并保障安全存储

在软件开发过程中，我们会得到各种各样的产出，比如各种文档、模型、源代码以及测试脚本等，我们把这些大家劳动的成果统称为工件（Artifact）。对于一个软件开发组织来说，这些工件就构成了组织的核心资产。对于如现金、有价证券之类的资产，我们都会准备一个保险箱，好好地保存；对于软件资产，我们也需要相似的措施。所以，软件配置管理工作的第一步就是建立一个安全、可靠的存储库（Repository），用于保存组织的核心软件资产。 这个库对于开发团队来说，就像是财务室里的保险箱。因此，容错能力和高可靠性是这个库最重要的属性。除此之外，随着组织的增长，置于库中的数据会越来越多，为保证运行效率，库的可扩展性也是非常重要的一个属性。

对于存储库来说，良好规划的备份和灾难恢复过程是必不可少的。令人惊讶的是，很多软件组织在这方面都没有给予必要的重视，因而也给组织的发展留下了严重的隐患，一旦灾难发生，后果不堪设想。

在建立好存储库以后，需要做的工作就是确定将哪些工件置于库中。根据实际需要，组织可能会决定只将正式文档、模型文件、源代码、发布版本等文件放入库中，而对于临时文档、编译时产生的中间文件等，则不将它们放入库中。我们把放入库中的文件称之为配置项（Configuration Item）。

2、控制并且审计（Audit）对于工件的修改

在标识相关的工件并将它们置于存储库中以后，我们需要建立对于这些工件的修改控制机制以及审计机制。

库里的工件不是谁想修改就可以修改的。控制机制必须保证只有拿到授权的人员才能对相关工件进行修改，而审计机制则保证修改的动作被完整地记录，也就是说，谁修改了这个工件，什么时候做的修改，为什么原因做出这个改动，以及修改了哪些地方（Who、When、Why、What）。

审计机制通常通过“检出/检入”（Check out/Check in）模式得到实现。在这种模式下，工件一旦入库，读写权限就变成只读（read only），如果要对该工件进行修改，则需要通过“检出”这个步骤；在修改结束以后，如果希望将修改的成果入库，则需要通过“检入”这个步骤。在经过一次“检出/检入”步骤以后，会形成该工件新的版本，因此也有人把上边的过程称之为“版本控制”（Version Control）。在版本控制过

程中，如果利用一些配置管理工具（或者版本控制工具）的支持，则可以自动地记录审计工作所需的四个“W”（Who、When、Why、What）。

3、设立并管理基线

通过审计机制我们可以保存一个工件完整的变更历史；但是一个项目通常是由成百上千个工件构成的，每个工件在变更过程中都会形成一系列的版本，如何确认系统在某个时刻分别由哪些工件的哪些版本构成？这就需要引入一个概念：配置（Configuration）。对于软件系统来说，在开发过程中某个时刻存储库中所有工件的一个“快照”（snapshot），就形成一个“配置”。对于一些重要时刻的系统配置，我们可以使用基线（Baseline）来进行标志。

IEEE 对于基线的定义是：已经通过正式复审和批准的某规约或产品，它因此可以作为进一步开发的基础，并且只能通过正式的变更控制过程进行改变

简单地说，基线就是项目储存库中每个工件版本在特定时期的一个“快照”。它提供一个正式标准，随后的工作基于这个标准进行，并且只有经过授权后才能变更这个标准。建立一个初始基线后，以后每次对它进行的变更都将记录为一个差值，直到建成下一个基线。

建立基线的主要原因是：重现能力、可追踪性和报告能力。

重现能力是指返回并重新生成软件系统给定发布版本的能力。可追踪性建立项目各种类型工件（需求、设计、实现、测试等）之间的横行依赖关系，其目的在于确保设计满足需求、代码实施设计以及使用正确

代码编译生成可执行文件。报告能力来源于一个基线内容同另一个基线内容的比较，基线比较有助于程序调试并生成发布说明（Release notes）。

建立基线有以下几个好处：

(1) 基线为开发工件提供了一个定点和快照。新项目可以从基线提供的定点之中建立。

(2) 当认为更新不稳定或不可信时，基线为团队提供一种取消变更的方法。

(3) 可以利用基线重新建立基于某个特定发布版本的配置，这样也可以重现被报告的错误。

在开发过程中，需要定期建立基线以确保团队开发人员的工作保持同步，通常，在项目生命周期中的里程碑处定期建立基线。

4、记录并跟踪变更请求

以上我们谈论的都是对于工件的变更活动的实施，下面我们要谈到的是软件配置管理的另一个方面：对于变更请求的管理。这是变更活动的源头。

著名的软件大师 Brooks 曾经谈到导致软件开发困难的一个原因就是软件的可变性。大家都知道，各种要素，如市场的变化、技术的进步、客户对于项目认识的深入等等，都可能导致软件开发过程中的变更请求的提出，而且承认这种变更请求的合理性也已经是工业界的共识。

但是，如果缺乏对于变更请求的有效的管理能力，纷至沓来的变更就会成为开发团队的噩梦。缺乏有效的变更请求管理会导致以下一些问题：

（1） 软件产品质量低下，对一些缺陷的修正被遗漏；

（2） 项目经理不了解开发人员的工作进展，缺乏对项目现状进行客观评估的能力；

（3） 开发人员不了解手头工作的优先级别，可能出现将紧急的事情放在一边、而工作在一般优先级任务上的情况。

变更请求管理的复杂程度与变更的具体类型有关。简单地说，变更请求管理会涉及到变更请求的提交、变更请求的复审、变更任务分配、变更结果的验证等一系列活动。通常，变更请求管理的流程是：由请求者提交变更请求，变更控制委员会（Change Control Board，CCB）召开CCB 复审会议对变更请求进行复审，以确定该请求是否为有效请求。如果是，则基于项目团队所确定的优先级、时间表、资源、变更难度、风险、严重性以及其他相关标准，判定对该变更的修改程序，并分配实施变更任务的人力资源和时间资源；变更任务实施人员负责实施该变更；实施结束以后提交验证人员，由验证人员负责对变更结果进行验证，如果变更成功则通知相关人员，否则由变更实施人员返工。

典型的变更请求管理如需求变更管理、缺陷追踪等。实施有效的变更请求管理有以下好处：

（1） 提高软件产品质量；

（2） 提高开发团队沟通效率；

（3） 帮助项目管理人员对产品状态进行客观的评估。

关于变更请求管理的详细过程以及相关准则 PMT 将在相关报告中阐述。

5、维护稳定、一致的工作空间

在我们把相关工件纳入集中的存储库、大家也都遵照“检出/检入”的工作模式对工件进行修改以后，下一步的工作就是要为每位开发人员设定“私有”的工作区，或者叫做工作空间。

工作空间通常以特定的基线为基础创建，要求能够做到为指定的任务方便地取出正确的工作版本建立私有工作空间；开发人员根据项目要求在自己的私有空间中对工件进行修改和测试活动，而与其他开发人员相对保持隔离，也就是说，自己的修改活动不会受到他人的影响，也不会影响到其他开发人员。但是，在保持隔离的同时，又应该提供相应的机制，当开发人员希望共享工作成果的时候，能够很方便地实现共享。

开发人员日常的开发工作都是在工作空间里进行的，因此，稳定性应该是工作空间首要的特性，只有高度稳定的工作空间才能保证开发人员的工作效率。

第三节 软件配置管理实施体会

随着软件产业的崛起，软件工程技术正吸引着越来越多关注的目光。作为软件工程的一个重要的领域，软件配置管理（**Software**

Configuration Management）也日益受到人们的重视。在这里，笔者并不打算对软件配置管理的细节进行讨论，几乎任何一本关于软件工程的教材中都有专门的章节对此进行介绍，而是想从一个实践者的角度来阐述关于软件配置管理的一些想法。

一. 软件配置管理的目的

对于任何一个软件组织（企业）来说，开发出满足用户需求的、高质量的软件产品是其追求的目标。而要实现这一目标的关键是建立起一个稳定、可控、可重用的软件流程（Software Process）。因为某一软件产品的成败可能维系于关键技术的突破和创新；但对于软件组织而言，要想永葆竞争优势并不断取得成功，那就必须不断地改进它的软件流程。要进行软件流程改进（Software Process Improvement）就需要有明确的、量化的对现状的分析和对未来的预期，这些数据来源于对软件过程的度量，而进行度量的前提和基础就是软件配置管理。

与一般制造业相类似，软件流程就像是一条流水线，在它的各个环节上都会有“零部件”产生，它们就是我们所熟悉的程序、相关文档以及数据。这些正是软件配置管理的对象——（软件）配置项。它们不仅是大量人力物力投入的结晶，更是开发经验的积累，是软件组织最宝贵的财富。软件配置管理贯穿于软件开发活动的始终，覆盖了开发活动的各个环节，它的重要作用之一就是要全面的管理保存各个配置项，监控各配置项的状态，并向项目经理及相关的人员报告，从而实现对软件过程的控制。

那么我们对这些配置项进行管理只是为了保存这些信息吗？众所周知，人员的高流动性和知识和技术的快速更新是软件业的重要特点。应对这样的特点我们只有努力地把开发人员个人的成功经验转化为团队的以及整个组织的经验。在这样的一个转化过程中，软件配置管理也起着极其重要的作用。因为对于一个大型的软件企业来说，它的配置库有如一个巨大的图书馆，随着产品版本的不断演进，越来越多的配置项会充斥其间，以至于没有任何一个人能了解其中的全部内容。当我们需要在开发组织内部迅速的共享以往的成果时，配置管理就能发挥作用了。它就像常见的图书编目法那样，帮助图书管理员（配置管理员）迅速的找出所需的资料（配置项），而不必彻底了解其中的确切内容。这样工作效率大为提高，很多常见的容易引起混乱的问题都能尽量得以避免。

所以，我们在从事软件配置管理工作时应以整个软件流程的改进为目标，为软件项目管理和软件工程的其它领域打好基础，以便于稳步推进整个软件组织的能力成熟度。

二． 工具的选择

古语有云：“工欲善其事，必先利其器。”软件配置管理是一项十分繁琐的工作，同时又和整个软件的开发活动紧密地联系在一起，所以在实际工作中更需要有得力的工具辅助。目前常用的配置管理工具主要有 **MS SourceSafe**、**Rational ClearCase** 等，这些工具各有所长，因而只有根据项目的预算和开发组织的些实际情况出发来选择，正所谓“好用就好”。在这里，笔者提出一些个人的看法供大家参考。

首先，配置管理工具应该提供完善的版本管理的功能。在该工具的所管理的配置库中，所有的配置项都应清晰、完整的得到保存，相应的操作纪录完备，使得开发组织中的任何人员都能迅速的了解任一配置项的演进过程，并快捷的找到所需的资源。

其次，配置管理工具应具备一定的工作空间的管理功能。正如前文指出的那样，一个软件企业往往有多个项目同时进行着开发，为了最大程度的利用组织的经验、共享成果，我们有必要在一个共同的配置库里提供多视角的观察手段，在逻辑上按照不同的角色分工来组织信息的选取规则和显示方式，从而能根据需要，在开发人员间灵活的进行分工合作。

由于我们把配置管理工作立足于软件过程的改进，那么我们所选用的工具最好能具有一定的过程控制的能力，能利用它按照企业本身的开发流程来灵活的建立相应的电子流，并在此过程中记录用于过程度量的相关数据，整合软件过程管理的各个环节，以便于客观的发现问题，高效的解决问题。

另外，我们选取得工具一定要操作简便，不能给开发人员增加过多的负担，因为过多的形式化的约束往往带来人们的反感，使得大家不约而同的选择规避的措施，其结果只能是事倍功半，甚至和我们的目标南辕北辙。

三. 实现的策略

笔者所在的软件组织从事的通信软件的研发，我们把配置管理作为推进软件过程改进的一个很重要的工作领域。我们明确定义了配置管理相关的角色、工作职责和工作流程，通过一段时间的努力，已经取得了明显的效果。

1. 配置库的设置

决定配置库的结构是配置管理活动的重要基础。一般常用的是两种组织形式：按配置项类型分类建库和按任务建库。

按配置项的类型分类建库的方式经常为一些咨询服务公司所推荐，它适用于通用的应用软件开发组织。这样的组织一般产品的继承性较强，工具比较统一，对并行开发有一定的需求。使用这样的库结构有利于对配置项的统一管理和控制，同时也能提高编译和发布的效率。但由于这样的库结构并不是面向和各个开发团队的开发任务的，所以可能会造成开发人员的工作目录结构过于复杂，带来一些不必要的麻烦。 而按任务建立相应的配置库则适用于专业软件的研发组织。在这样的组织内，使用的开发工具种类繁多，开发模式以线性发展为主，所以就没有必要把配置项严格的分类存储，人为增加目录的复杂性。因此，笔者认为特别是对于研发性的软件组织来说，还是采用这种设置策略比较灵活。

2. 分支的划分

在实际的开发活动中系统中，为了让每个开发人员和各个开发团队能更好的分工合作，同时又互不干扰，我们基本上为每个配置项从建立开始就划分成 3 个不同的分支，让它们分别对应 3 类工作空间。

- 私有分支

私有分支对应的是开发人员的私有开发空间。开发人员根据任务分工获得对相应配置项的操作许可之后，他即在自己的私有开发分支上工作，他的所有工作成果体现为在该配置项的私有分支上的版本的推进，除该开发人员外，其他人员均无权操作该私有空间中的元素。

- 集成分支

集成分支对应的是开发团队的公共空间。凡是要为同组人员共享的配置项都从该分支获得。即各开发人员必须将私有工作空间中的开发成果归并（Merge）到该分支后才能进入下一个开发活动。所有涉及多人协调的开发工作（如集成测试等）都必须工作在这一空间中。该开发团队拥有对该集成分支的读写权限，而其他成员只有只读权限。该分支的管理工作由系统集成员及相关指定人员负责。

- 公共（主干）分支

公共分支对应的是整个软件开发组织的公共空间。各个开发小组在现阶段的任务完成后，将可以发布的版本归并到该分支上，将来需要查

阅相关资料时，以该分支上的版本为准。该分支对组织内的全体软件人员开放只读权限。该分支的管理工作由系统集成员负责。

上面定义的 3 类工作空间（分支）由配置管理员统一管理，根据各开发阶段的实际情况定制相应的版本选取规则，来保证开发活动的正常运作。在变更发生时，应及时做好基线的推进。

3．变更控制

对于大型的软件开发项目，无控制的变更将迅速导致混乱，变更控制就是通过结合人的规程和自动化工具，以提供一个变化控制的的机制。本文所涉及的变更控制的对象主要指配置库中的各基线配置项。变更管理的一般流程是：

A) 由开发人员或系统集成员提出变更需求；

B) 由 SCCB（软件变更控制委员会）审核并决定是否批准；

C) 配置管理员根据 SCCB 的决定临时开放相应的权限，并备案；

D) 系统集成员执行相应的变更。

在这里，将要涉及的变更控制分为两类：一类是基线的变更控制，另一类是软件版本的变更控制。 l 基线的变更控制

基线的变更是指在一个软件版本的开发周期内对基线配置项的变更，主要包括基线的应用和更新等活动。

基线变更所涉及的操作主要包括基线标签的定义和标签的使用。基线标签属于严格受控的配置项，它的命名必须严格按照相关的命名规范来进行。基线在建立时，按照角色职责的分工，须经 **SCCB** 同意并以正式的将该基线的标识和作用范围通知系统集成员，由后者负责执行；基线一旦划定，由该基线控制的各配置项的历史版本均处于锁定或严格受控状态，任何对基线位置的变更请求都必须按变更控制流程，提交 **SCCB** 批准，然后由系统集成员执行。

- 软件版本的变更

软件版本的命名规范应事先制定，并按照开发计划予以发布使用。在软件版本的演进过程中既需要从以前的版本中继承，又需要相对的独立性。所以在对于一个子版本（例如某特定用户的定制版本）就需要对一系列配置项从统一的开发起始基线所确定的版本上建立新的分支，然后在此分支上开发新的版本。因此在这样的变更控制流程中，受控的对象还应包括特定的分支类型，以及工作视图的选取规则，同时配置管理员将在这一过程中担负更多的操作职责

第三章　基于 CMM 的 SCM

CMM（软件能力成熟度模型）与 SCM（软件配置管理）

在 COCOMO 模型中，对一个项目团队的软件开发能力作了如下定义：

软件项目的开发能力= （团队技能）（工具）（规模）（过程）

其中：

团队技能：开发团队的技能、经验，以及精神

工具：开发团队使用各种工具的自动化程度

规模：开发团队的人员规模，以及人为因素造成的复杂程度

过程：团队软件开发的方法、标记及成熟度

基于此模型，我们可以看出要提高团队的开发能力，可以从以下四个方面入手：

降低问题的复杂性，减少开发团队规模：

尽可能多采用第三方现成的组件，多采用软件复用规范标记法，基于模型的工件， 或由工具直接产生工件

改进过程的效率和成熟度：

吸纳新技术、新工具、新经验

提高团队技能的技术熟练程度：

培训、指导、顾问咨询

扩展工具的自动化程度：

双向工程分析、评估自动化

在这四个方面中，软件开发过程的改进最为重要，目前国际上有多种改善或评估软件开发过程的方法和工具，其中以卡耐基梅隆大学的软

件工程研究所（SEI）提出的软件能力成熟度模型（Capability Maturity Model，CMM）最具影响力。CMM 将软件过程成熟度划分为 5 个级别，从一级到五级分别为：混乱级、可重复级、定义级、管理级和优化级。CMM 级别越高，就表示软件开发过程越规范。每个成熟级别由一系列关键流程领域（Key Process Areas，KPA）组成，每个 KPA 确定一组相关活动。当这些相关活动一起开展时，它们完成一系列被认为对在该成熟级别建立流程能力有重要影响的目标。

在 CMM 二级中，有以下六个 KPA：需求管理、软件项目规划、软件项目跟踪与勘察、软件分包管理、软件质量保证、软件配置管理。其中最后一个 KPA 软件配置管理的目的就是在项目的整个软件生命周期内建立并维护软件项目产品的完整性，更好地管理开发。实际上，软件配置管理是大多数软件工程和管理流程的一个重要构成部分。

针对配置管理，在 CMM 中仅仅定义了一系列的流程，而没有对实施方法提出任何要求。因此，如果要达到 CMM2 的要求，我们需要自己建立配置管理流程，并选用相关的工具，来贯彻流程的实施。

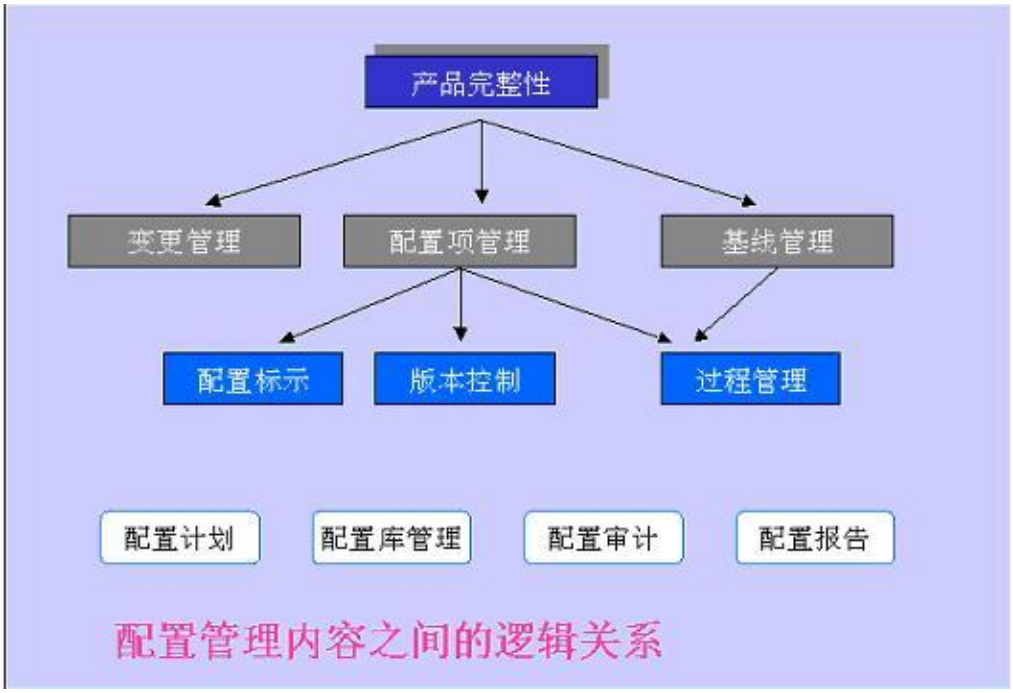
基于 CMM/CMMI 的配置管理

本节主要从 CMM 和 CMMI 的要求出发，介绍了标准主要涉及的配置管理内容，并对相应内容进行初步地说明，最后提供了一个配置管理在项目实施的指南和一个在组织中部署配置管理的模型。

1　配置管理内容的逻辑关系

在 CMM 和 CMMI 中，将配置管理的目的定义为“建立和维护产品的完整性”，这个目标没有提到对项目管理的支持，也就是说，它定义

的配置管理的目标比当前业界对配置管理的认识有些缩小。但是，仔细分析可以发现“建立和维护产品的完整性”是其他配置管理目标的基础。下面就从这个目标出发进行分析。逻辑关系见下图：



配置完整性（对标准的理解）

1. 产品完整性：就是项目提交的工作成果是“产品集合完整、子产品的正确”的

2. 产品集合完整：产品包含的子产品（配置项）是完整的

3. 子产品的正确：子产品（配置项）达到了需求要求，满足标准、规程的要求

逻辑关系分析

1. “基线管理”支持“产品集合完整”，明确产品的“子产品”（配置项）集合，并进行管理和控制

2. “配置项管理”，提供了了对子产品（配置项）的控制管理，支持“子

产品的正确”

3. “变更管理”，同时支持“产品集合完整、子产品的正确”，用于控制子产品（配置项）和产品（基线）的变更

4. “配置标示”，建立对配置项（子产品）的识别、命名，支持“配置项管理”

5. “版本控制”，控制配置项（子产品）生命历程，保留配置项（子产品）演进历史

6. “过程管理”，就是对配置项、基线的建立、变更的状态标示、过程控制，保证产品（或子产品）按照规定的流程进行了操作；例如“配置项”进入“基线”的过程包括：配置项标示、产品验证、进入配置、配置审计等

7. “配置计划”、“配置库管理”、“配置审计”、“配置报告”等是整个配置管理得支持系统。提供了配置管理“可视性”和监督管理

2 配置和配置项

在配置管理中，“配置”和“配置项”是重要的概念，“配置”是在技术文档中明确说明并最终组成软件产品的功能或物理属性。因此“配置”包括了即将受控的所有产品特性，其内容及相关文档，软件版本，变更文档，软件运行的支持数据，以及其他一切保证软件一致性的组成要素，相对与硬件类配置，软件产品的“配置”包括更多的内容并具有易变性。

受控软件经常被划分为各类配置项（Configuraion items, CIs），这类划分是进行软件配置管理的基础和前提，CIs 是逻辑上组成软件系统的各组成部分。比如一个软件产品包括几个程序模块，每个程序模块及其相关文档和支撑数据可能被命名为一个 CI。一个系统包括的 CIs 的数目是

一个与设计密切相关的问题。一个纯软件的 CI 通常也称之为软件配置项（CSCI）。

现在所有的配置管理工具均提供对配置项的管理工具，包括(Check in 和 Check out 机制的)版本管理和版本标号功能。由于版本和标号管理比较繁琐，一般推荐使用配置管理工具，减少事务性工作。

3 基线

在配置管理系统中，基线就是一个 CI 或一组 CIs 在其生命周期的不同时间点上通过正式评审而进入正式受控的一种状态，而这个过程被称为“基线化”。每一个基线都是其下一步开发的出发点和参考点。基线确定了元素（配置项）的一个版本，且只确定一个版本。一般情况下，基线一般在指定的里程碑处创建，并与项目中的里程碑保持同步

一般地，第一个基线包含了通过评审的软件需求，因此称之为“需求基线”，通过建立这样一个基线，受控的系统需求成为进一步软件开发的出发点，对需求的变更被正式初始化、评估。受控的需求还是对软件进行功能评审的基础。

每个基线都将接受配置管理的严格控制，对其的修改将严格按照变更控制要求的过程进行，在一个软件开发阶段结束时，上一个基线加上增加和修改的基线内容形成下一个基线，这就是“基线管理”的过程。

基线具有以下属性：

通过正式的评审过程建立

基线存在于基线库中，对基线的变更接受更高权限的控制

基线是进一步开发和修改的基准和出发点

进入基线前，不对变化进行管理或者较少管理

进入基线后，对变化进行有效管理，而且这个基线作为后继续工作的基础

不会变化的东西不要纳入基线

变化对其他没有影响的可以不纳入基线

建立基线的好处：

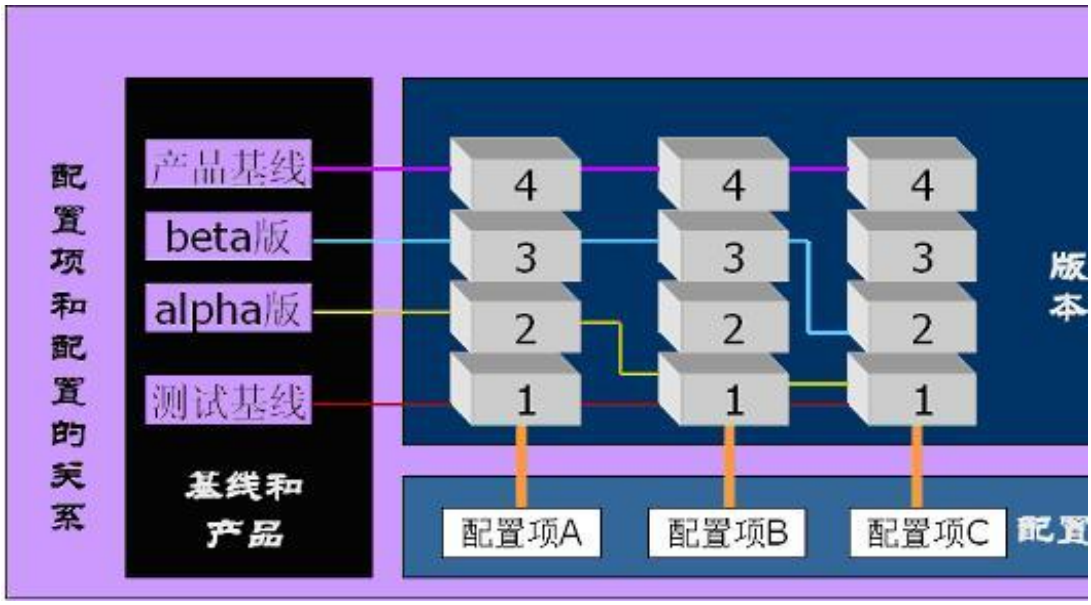
重现性：及时返回并重新生成软件系统给定发布版的能力，或者是在项目中的早些时候重新生成开发环境的能力。当认为更新不稳定或不可信时，基线为团队提供一种取消变更的方法。

可追踪性：建立项目工件之间的前后继承关系。目的是确保设计满足要求、代码实施设计以及用正确代码编译可执行文件。

版本隔离：基线为开发工件提供了一个定点和快照，新项目可以从基线提供的定点之中建立。作为一个单独分支，新项目将与随后对原始项目（在主要分支上）所进行的变更进行隔离。

4 基线、配置、配置项的关系

基线的组成，以及配置项和配置的关系如下图：



基线管理的步骤：

1、在开发前确定基线的“配置”

2、基线批准前，根据“配置”检查配置项是否齐备

3、对各个配置项，确认其版本的正确性

4、对每个配置项建立基线标志，

例如上图为：测试基线=（配置项 A=1，配置项 B=1，配置项 C=1）

alpha 版=（配置项 A=2，配置项 B=1，配置项 C=1）

beta 版=（配置项 A=3，配置项 B=3，配置项 C=2）

产品基线=（配置项 A=4，配置项 B=4，配置项 C=4）

5、基线变更管理

6、基线的各类报告和审计信息

针对某个具体得项目，可以根据基线的内容，建立一个基线信息

的跟踪表，例如如下：

基线 配置项	功能基线	需求基线	设计基线	编码基线	测试基线	Alpha基线	产品基线
X项目的目标							
X项目的 SRS							
A 系统设计							
B 系统设计							
A1 模块代码							
A2 模块代码							
B1 模块代码							
B2 模块代码							
B3 模块代码							
A 系统程序							
B 系统程序							
X的操作手册							

[注]：被色块覆盖的表示，此配置项属于对应列的基线

[注]：在色块的栏目填写对应配置项的版本号

5 变更管理

在有效标示了配置并进行了管理之后，如何保证它们在复杂多变得开发过程中真正的处于受控的状态，并在任何情况下都能迅速的恢复到任一历史状态就要依赖有下的变更管理。

缺乏有效的变更请求管理会导致的问题：

软件产品质量低下，对一些缺陷的修正被遗漏

项目经理不了解开发人员的工作进展，缺乏对项目现状进行客观评估的能力

开发人员不了解手头工作的优先级别，可能出现将紧急的事情放在一

边、而工作在一般优先级任务上的情况 可能错误使用和引用已经变更的产品，引起开发工作混乱

变更管理的流程： （获得）提出变更请求； 由 CCB 审核并决定是否批准； 为（被接受）修改请求分配人员，提取 SCI，进行修改； 提交修改后的 SCI，并测试（或者评审）； 重建软件的适当版本； 复审（审计）所有 SCI 的变化； 发布新版本。

为了更好的指导变更范围的影响分析，可以通过两种表格来帮助发现受到变更影响的内容，一种是《需求跟踪表》，一种是《配置项依赖关系表》，分别如下：

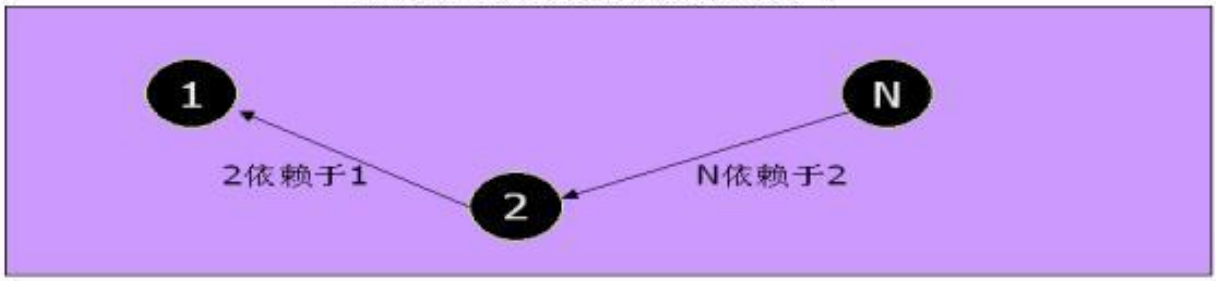
需求跟踪表

需求	设计	编码	程序
需求 1	系统设计 1 系统设计 2	模块 1 模块 2	程序 1
需求 2	系统设计 3 系统设计 4	模块 3	程序 2
需求 N	系统设计 1 系统设计 3	模块 1 模块 3	程序 2

配置项依赖关系表

依赖项 / 被依赖	配置项 1	配置项 2		配置项 N
配置项 1		×		
配置项 2				×
配置项 N				

配置项依赖关系有向图（对应上表）



6 配置库管理

在实际的开发活动中系统中，为了让每个开发人员和各个开发团队能更好的分工合作，同时又互不干扰，必须规划好工作空间的管理。主要的手段是设置配置库（即文件夹设置），和设置版本的分支，来实现对配置项权限管理。

设置版本分支

基本上要为每个配置项从建立开始就划分成 3 个不同的分支：私有分支、集成分支、公共（主干）分支。让它们分别对应 3 类工作空间。

私有分支： 私有分支对应的是开发人员的私有开发空间。开发人员根据任务分工获得对相应配置项的操作许可之后，他即在自己的私有开发分支上工作，他的所有工作成果体现为在该配置项的私有分支上的版本的推进，除该开发人员外，其他人员均无权操作该私有空间中的元素。
集成分支： 集成分支对应的是开发团队的公共空间。凡是要为同组人员共享的配置项都从该分支获得。即各开发人员必须将私有工作空间中的开发成果归并（Merge）到该分支后才能进入下一个开发活动。所有涉及多人协调的开发工作（如集成测试等）都必须工作在这一空间中。该开发团队拥有对该集成分支的读写权限，而其他成员只有只读权限。该分支的管理工作由系统集成员及相关指定人员负责。
公共（主干）分支： 公共分支对应的是整个软件开发组织的公共空间。各个开发小组在现阶段的任务完成后，将可以发布的版本归并到该分支上，将来需要查阅相关资料时，以该分支上的版本为准。该分支对组织内的全体软件人员开放只读权限。该分支的管理工作由系统集成员负责。

上面定义的 3 类工作空间（分支）由配置管理员统一管理，根据各开发阶段的实际情况定制相应的版本选取规则，来保证开发活动的正常运作。在变更发生时，应及时做好基线的推进。

配置库的设置

决定配置库的结构是配置管理活动的重要基础。一般常用的是两种组织形式：按配置项类型分类建库和按任务建库。

按配置项的类型分类建库的方式经常为一些咨询服务公司所推荐，它适用于通用的应用软件开发组织。这样的组织一般产品的继承性较强，工具比较统一，对并行开发有一定的需求。使用这样的库结构有利于对配置项的统一管理和控制，同时也能提高编译和发布的效率。但由于这样的库结构并不是面向和各个开发团队的开发任务的，所以可能会造成开发人员的工作目录结构过于复杂，带来一些不必要的麻烦。

而按任务建立相应的配置库则适用于专业软件的研发组织。在这样的组织内，使用的开发工具种类繁多，开发模式以线性发展为主，所以就没有必要把配置项严格的分类存储，人为增加目录的复杂性。因此，笔者认为特别是对于研发性的软件组织来说，还是采用这种设置策略比较灵活。

配置库的日常工作

配置库的日常工作是一些事务性的工作，主要保证配置库的安全性，包括：

对配置库的定期备份

清除无用的文件和版本

检测并改进配置库的性能等

7 配置报告

配置状态报告就是根据配置项操作的记录来向管理者报告软件开发活动的进展情况。这样的报告应该是定期进行，并尽量通过 CASE 工

具自动生成，用数据库中的客观数据来真实的反映各配置项的情况。

配置状态报告应着重反映当前基线配置项的状态，以作为对开发进度报告的参照。为了说明项目状态对变更的情况也应当进行报告。有时，对配置库的情况也进行说明，例如备份次数，磁盘占用空间等等。只要是关心的信息，均可作为状态报告的内容。这些信息进行有效记录，往往可以作为项目度量的重要数据来源。

8 配置审计

配置审计的主要作用是作为变更控制的补充手段，来确保某一变更需求已被切实实现。在某些情况下，它被作为正式的技术复审的一部分，但当软件配置管理是一个正式的活动时，该活动由 SQA 人员单独执行。 审计机制保证修改的动作被完整地记录，也就是说，记录了谁修改了这个工件，什么时候做的修改，为什么原因做出这个改动，以及修改了哪些地方。 在版本控制过程中，如果利用一些配置管理工具（或者版本控制工具）的支持，则可以自动地记录审计工作所需的四个“W”（Who、When、Why、What）。 同时配置审计工作应当可以说明如下信息。

配置审计应当说明的信息：
1. 变更要求被完成，并且对附加的修改已经执行了
2. 采用了正确的正式验证手段
3. 遵循了标准的要求
4. 变更的 4W 信息被完整记录，并和相关配置项关联

9 项目实施指南

一个软件研发项目一般可以划分为三个阶段：计划阶段、开发阶段和维护阶段。然而从软件配置管理的角度来看，后两个阶段所涉及的活动是一致，所以就把它们合二为一，成为“项目开发和维护”阶段。

一个项目设立之初项目经理首先需要制定整个项目的计划，它是项目研发工作的基础。在有了总体研发计划之后，软件配置管理的活动就可以展开了，因为如果不在项目开始之初制定软件配置管理计划，那么软件配置管理的许多关键活动就无法及时有效的进行，而它的直接后果就是造成了项目开发状况的混乱并注定软件配置管理活动成为一种“救火”的行为。所以及时制定一份软件配置管理计划在一定程度上是项目成功的重要保证。

在“开发阶段和维护阶段”，软件配置管理活动主要分为三个层面，这三个层面是彼此之间既独立又互相联系的有机的整体。

（1） 主要由配置人员完成的管理和维护工作；

（2） 由系统集成员和开发人员具体执行软件配置管理策略；

（3） 变更流程。

软件阶段	活 动	活动说明
计划阶段	制定软件计划	一个项目设立之初，项目经理首先需要制定整个项目的计划
	确定配置策略	配置管理委员会（CCB）根据项目的开发计划确定各个里程碑和开发策略

	制定配置计划	配置人员根据 CCB 的规划，制定详细的配置管理计划，交 CCB 审核
	批准配置计划	CCB 通过配置管理计划后交项目经理批准，发布实施
开发阶段和维护阶段	确定初始基线	CCB 设定研发活动的初始基线
	配置库管理	配置人员根据软件配置管理规划设立配置库和工作空间，为执行软件配置管理做好准备；并定期进行备份和清理工作
	授权开发	开发人员按照统一的软件配置管理策略，根据获得的授权的资源进行项目的研发工作
	集成	系统集成员按照项目的进度集成组内开发人员的工作成果，并构建系统，推进版本的演进
	管理基线	CCB 根据项目的进展情况，并适时的建立基线，批准基线变更，保证开发和维护工作有序的进行。
	产品发布	系统集成员进行产品集成，由 CCB 批准，进行发布
其他	配置会议	CCB 定期举行例会，根据成员所掌握的情况、配置人员的报告和开发人员的请求，对配置管理计划作出修改，并向项目经理负责。

	配置报告和审计	配置人员定期向项目经理和 CCB 提交审计报告，并在配置管理例会中报告项目在软件过程中可能存在的问题和改进方案
	变更管理	事件触发执行，由 CCB 批准、项目组执行，并执行审计

10 配置管理部署模型

基本过程

序号	阶段	活动	备注
1	获得相应管理权		
1.1		建立相应负责团队	
1.2		获得授权和资源	可召开启动会
2	评估配置管理现状		
2.1		绘制和评估当前过程的控制图	可采用 CMM 标准
2.2		了解员工对配置管理的态度	
2.3		了解组织的配置管理技术水平	
2.4		了解领导期望	

2.5		编制并评审评估报告	获得“现状信息”
3	配置工具选择		
3.1		编制、评审《评估评分表》	
3.2		评估配置工具和供货商	
3.3		收集其他用户的使用经验	
4	配置过程定义		制定配置管理过程草案
4.1		利用“现状信息”和收集的经验	
4.2		制定新的过程	
4.3		评审新过程，并建立新的过程基线	
5	试点		
5.1		选定试点项目	
5.2		确定试点负责小组	
5.3		定义试点成功标准和进度	
5.4		试点项目人员培训	
5.5		试点改进	同时对草案进行改进

5.6		试点总结/推广	完成正式过程的发布
6	全面实施		
6.1		组建相应部门和团队	
6.2		制定各个项目的实施计划	
6.3		配置管理知识、过程、工具的培训	
6.4		帮助各个项目向新过程迁移	
6.5		日常监督、抽查、沟通	
7	结束		总结、奖励

相应操作文件

对应过程：**2.2** 了解员工对配置管理的态度

建立一个 **CHECKLIST**，来进行调研，如下

序号	**调查内容**	**调查结果**
1	原来是否有过类似尝试，成功或者失败了	
2	是否有由于配置管理不好造成的痛苦经历	
3	对建立配置管理过程是否支持	

4	是否觉得配置管理过程会压抑创造性	
5	是否觉得配置管理过程太繁琐	
6	对配置管理是否有不合理的期望	
7	是否有些急功近利	
8	是否对实施配置管理工具感兴趣	
9	个人英雄主义和分工协作那个是主流	

对应过程：**2.3** 了解组织的配置管理技术水平

建立一个 **CHECKLIST**，来进行调研，如下

序号	调查内容	调查结果
1	是否已经有了配置管理过程，运作时间	
2	是否使用了配置管理工具，使用时间	
3	是否接受了配置管理的专门培训，培训时间	
4	对配置管理过程的认识程度	
5	对配置管理工具的使用程度	
6	企业员工的基本素质和学习能力	

对应过程：**3.2** 评估配置工具供应商

建立一个 **CHECKLIST**，来进行调研，如下

序号	调查内容	调查结果
1	工具可以解决当前问题，满足当前需求吗？	
2	产品的市场地位	
3	产品价格	
4	与现有环境的兼容程度	
5	运行能力（峰值情况、成熟性、稳定性）	
6	是否支持未来需求	
7	是否具备：工作空间管理	
8	是否具备：版本控制	
9	是否具备：配置报告	
10	是否具备：过程支持	
11	是否具备：安全和保护	
12	是否具备：工具集成	
13	是否具备：构造支持	
14	是否具备：图形界面	
15	是否具备：自定义支持	

16	是否具备：发行管理	
17	是否具备： WEB 支持	

对应过程：3.2 评估配置工具供应商

建立一个 CHECKLIST，来进行调研，如下

序号	调查内容	调查结果
1	配置管理服务从业时间	
2	成功案例数量和质量	
3	培训、技术支持队伍	
4	提供的培训和指导，以及其他服务	
5	近期关于配置服务的商誉、资产、销售额	
6	地理位置、服务及时性	

对应过程：4.2 制定新的过程

1. 配置管理过程至少应当包括的内容：配置标示、配置控制、报告、审计

2. 在考虑工具纳入配置过程中应当考虑下表内容

序号	考虑内容
1	从配置过程中分解出那些是事务性、那些是创造性的工作
2	考虑事务性工作的繁重程度和精度要求程度，理出一个“自动化优先级”

3	根据过程，确定工具可以运用的地方
4	根据“自动化优先级”选择那些工具功能进行自动化
5	考虑使用工具功能自动化的前提和结果
6	划分出“自动化”和“人工”的接口，并清晰描述
7	调整过程要素，适应工具，从而形成一个纳入了工具的配置管理过程
8	考虑这个过程的适用性和效益

对应过程：**6.1** 组建相应部门和团队

负责配置管理部署和实施的团队必须包括

序号	**团队成员**	**职责和要求**
1	组长	负责管理小组，并负责配置管理的部署和实施
2	技术人员	负责考虑将要和配置工具集成的各类工具之间的接口
3	配置专家	配置工具精通、配置管理理论知识熟悉
4	过程专家	负责过程建模和主要的过程分析工作
5	配置管理人员	负责评审新过程，并提供原来配置管理的经验
6	项目经理	负责评审新过程，并提供配置管理适应于项目的参考

对应过程：**6.2** 制定各个项目的实施计划

计划应当包括的内容：

序号	计划内容
1	目标和完成标准
2	投资和收益分析
3	阶段划分和进度安排
4	资源投入安排
5	人员分工和组织
6	风险管理

第四章　配置管理系统中的概念

摘要：现在，软件配置管理的环境及其工具越来越得到人们的重视。本文尝试就现存的 CM 体系中的以用户为主体的一些概念作详细说明。就如一个光谱，某些概念可能是另一些概念的延伸或总结。由于在整个软件工程家族中对于 CM 的功能性没有共通的术语,且许多 CM 系统在概念的应用上也是千差万别，因此要从 CM 系统中抽象出一些概念是难乎其难的事了。正因为这样，本文陈述的每一概念是其在某一具体的 CM 系统中的概念。有一部分的概念陈述是针对 CM 体系的用户极为重要的问题。没有哪一个 CM 系统能提供 CM 体系不同用户要求的所有功能。而且，每一 CM 系统解决的问题只是所有概念的一部分。为了完成本报告，对 CM 体系的功能以举例的形式作了简短的说明。

1. 简介

现在，软件配置管理的环境及其工具越来越得到人们的重视，这一点从 CM 体系中提供的概念谱中就显而易见。本文对这些概念进行了阐明。首先，在一典型的 CM 情形中，我们 对 CM 和 CM 体系做了更为广泛的定义。

A. 配置管理的定义

软件配置管理是一控制软件系统演变的学科。关于 CM 的经典讨论在条文[3]、[4]中进行了阐述。IEEE 标准 729-1983 就 CM 以下的内容进行了规范的定义。

在 **IEEE** 标准 **729-1983** 中，软件配置管理的定义包括：

标识——识别产品的结构、产品的构件及其类型，为其分配唯一的标识符，并以某种形式提供对它们的存取。

控制——通过建立产品基线，控制软件产品的发布和在整个软件生命周期中对软件产品的修改。例如，它将解决哪些修改会在该产品的最新版本中实现的问题。

状态统计——记录并报告构件和修改请求的状态，并收集关于产品构件的重要统计信息。例如，它将解决修改这个错误会影响多少个文件的问题。

审计和审查——确认产品的完整性并维护构件间的一致性，即确保产品是一个严格定义的构件集合。例如，它将解决目前发布的产品所用的文件的版本是否正确的问题。

生产——对产品的生产进行优化管理。它将解决最新发布的产品应由哪些版本的文件和工具来生成的问题。

过程管理——确保软件组织的规程、方针和软件周期得以正确贯彻执行。它将解决要交付给用户的产品是否经过测试和质量检查的问题。

小组协作——控制开发统一产品的多个开发人员之间的协作。例如，它将解决是否所有本地程序员所做的修改都已被加入到新版本的产品中的问题。

软件配置管理的解决方案涉及面很广，将影响软件开发环境、软件过程模型、配置管理系统的使用者、软件产品的质量和用户的组织机构。

配置管理解决方案将影响过程模型和模型的使用者，是因为它强行推行组织的方针政策和工作规程，并对工作过程进行跟踪。它从开发和维护的及时性方面影响产品的质量。例如，配置管理机制可以保证为每一个发布的版本提供内容清单，通过一致性维护提高产品的质量。配置管理解决方案通常在组织范围内推行，实际上配置管理系统是组织内部信息交换的中心，它影响组织内的每一个成员及组织的业务流程。

总之，一个配置管理解决方案的制定包括配置管理计划、过程的定义、与使用者的交流、自动化支持和做出管理决定等活动。

软件组织应该提出不同层次的配置管理视角，这些层次包括：公司

级、项目级、程序员级和应用级。公司级视角提供组织的全貌图和配置管理过程的描述；项目级视角是与项目相关的各项目组可以使用不同的配置管理方案；程序员级视角是专门为程序员提供的且具有某些特定的配置管理功能；应用级视角关心的是配置管理如何应用到具体的问题中去。

B．CM 系统的定义

至于怎样才算是构成 CM 系统的，对此还没有普遍接受的定义。例如：假如系统有版本控制功能，它是否就是一个 CM 系统呢？理想的 CM 系统是基于以上定义提供所有功能的系统。但是， 实际中的系统只能提供某种程度上实现的版本控制功能、配置识别功能、系统构建功能、系统建模功能，或某种程度上提供 CM 的意识就被软件工程大家族认为是 CM 系统了。应注意的是， 现有的 CM 体系提供只是一种功能的综和而不是一标准的体系。本报告提及 15 个 CM 系统，目前至少有 40 个系统可以为今所用。

这里，有必要将 CM 系统和 CM 工具两概念区分一下。CM 系统可看作是其支持环境的一部分且以这种形式被售出。譬如，在 RATIONAL[14]环境下 CM 功能成为该环境必不可少的一部分。CM 工具可看作是一独立的工具。譬如，版本控制系统（RCS）只是一个工具，因为它可被安装在一个现有环境中。由于这种区分在本文不是那么重要，术语 CM 系统就被用来表示这两概念。

C．CM 以用户为导向的典型情形

在讨论 CM 体系之前，我们描述了一个简单、典型的、以用户为导向的 CM 系统来作参考。在此情形下，包含了具有不同职责的人员：负

责软件小组的项目经理、负责 CM 规程和方针的配置经理、负责软件产品开发与维护的软件工程人员、负责验证产品正确性的测试人员、负责确保产品高质量的质量保证经理、使用产品的用户。

每一角色都有他们的目标和任务。对项目经理来讲，其目标是确保产品在一定的时间框架里得以开发。因此，经理监控开发过程并发现问题，解决出现的问题。这些又必须通过对软件系统的现状形成报告并予以分析以及对系统进行审核才能完成。

配置经理的目标是确保用来建立、更改及编码测试的规程和方针得以贯彻执行，同时使有关项目的信息容易获得。为了对编码更改形成控制，经理引入对正规请求更改的机制，评估更改的机制[通过更改控制机构（CCB），由它负责批准对软件系统的更改]，和批准更改的机制。经理负责为工程人员创建并宣导任务单，基本上创建项目的框架。同时，经理还收集软件系统中构件的相关数据，比如说用以判断系统中出现问题的构件的信息。

对于软件工程人员，他们的目标是有效地创造出产品。这就意味着工程人员在创建产品、编码测试及支持文档的产生中不必相互间干涉。与此同时，他们能有效地进行沟通与协作。他们利用工具以帮助创建性能一致的软件产品，通过相互通知要求的任务和完成的任务来进行沟通与协调。做出的更改通过将它们进行融合、分散和冲击而得知。产品中的所有元素的演变连同其更改的原因及实际更改的记录都予以保留。工程人员在创建、变更、测试及编码的汇合上有自己的工作范围。在某一点上，编码会形成一个基线，它使得进一步开发得以延续，为其它平行开发得以进行。

测试的目标是确保产品经过测试达到要求。这里包括产品某一特定

版本的测试和对某个产品的某种测试及其结果予以记录。将错误报告给相关人员并通过回归测试进行修补。

质量保证经理的目标是确保产品的高质量。这意味着特定的规程和方针应当完成并得到相关的批准。错误应得到纠正并应对变化的部分进行充分测试。客户投诉应予以跟踪。

不同的客户使用的产品版本也是不同的。客户总是遵循规则来做变更要求、错误显示及产品改进。

理想的 CM 系统在这种情形下应能够支持所有这些目标、角色和任务。这也意味着这些角色、任务和目标决定了一 CM 系统要求的功能。本文提出的一些概念就是为了解决这些问题。

D. 本章的结构

在简介中就 CM 和 CM 系统进行了定义，列出一典型的 CM 情形，这样一来也就暗示了 CM 体系的要求。第二节描述了 CM 系统中以用户为导向的一些问题。这些问题影响用户对 CM 系统的期望。第三节描述了 CM 概念谱。第四对 CM 体系的未来做了探讨，第五节是结论。附录是本文 CM 体系索引的概览。

2. CM 体系用户的有关问题

许多与 CM 有关的问题直接影响到 CM 系统的用户。现有的 CM 体系从不同的角度解决这些问题。尽管本文是为了就现有 CM 体系的特色进行探讨，但对这些问题的阐述仍然有必要因为它们影响到用户对一 CM 系统的期望。这些问题包括：

用户的角色问题：

不同 CM 体系用户对 CM 体系的功能的要求也就不同。

集成问题：

不同的集成问题影响到 CM 系统的功效。

启用 CM 的时机问题：

一项目组何时启用 CM 系统取决于该 CM 系统的能力。

控制水平问题；

一 CM 系统对产品及产品的管理的控制水平可以是不同的。

过程与产品问题：

一理想的 CM 系统提供 CM 的过程、产品及其附件。

自动化水平问题：

CM 功能的实现总是手工与自动程序的统一。

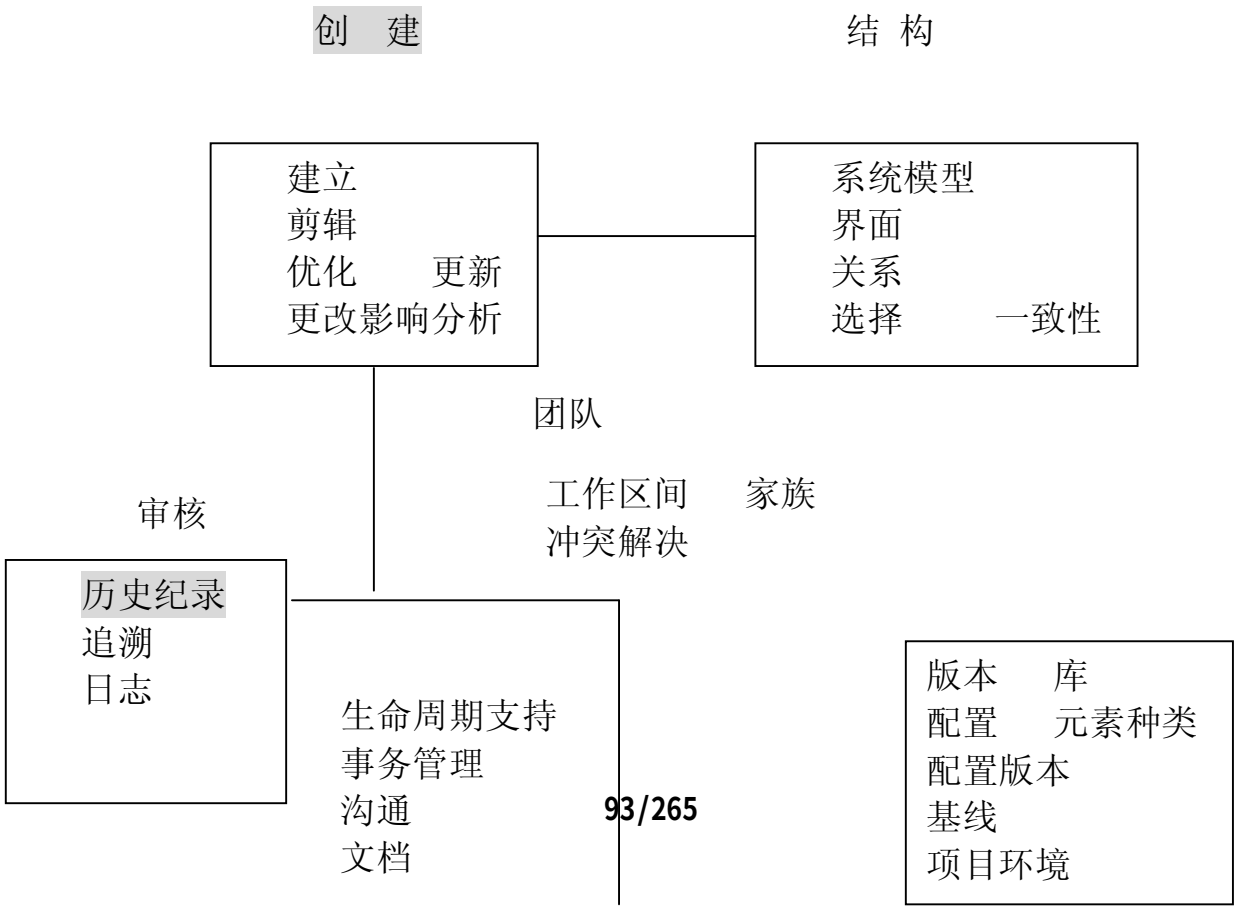
功能问题：

CM 体系具备实现 CM 众多功能的许多特点。

以下将对此做进一步说明。

A．用户的角色问题

正如 1。3 节中的情形表示的一样，CM 体系的用户是多种多样的。每一个用户都有特定的角色，对 CM 也有不同的观点，因此，对 CM 系统的要求也就不同。这种要求是很分明的同时又是互补的。图 1 是一功能组描述了项目经理、配置经理、软件工程人员、质量保证经理及客户对 CM 系统的期望。图 1 中的每一个方框代表的是一主要的功能区域。图 1 显示在方框外（审核、统计、构件、结构与创建）在任何 CM 系统中都可独立存在的功能区域，但当与团队和过程功能合并时，就得到一个完整的（或综合的）CM 系统了。



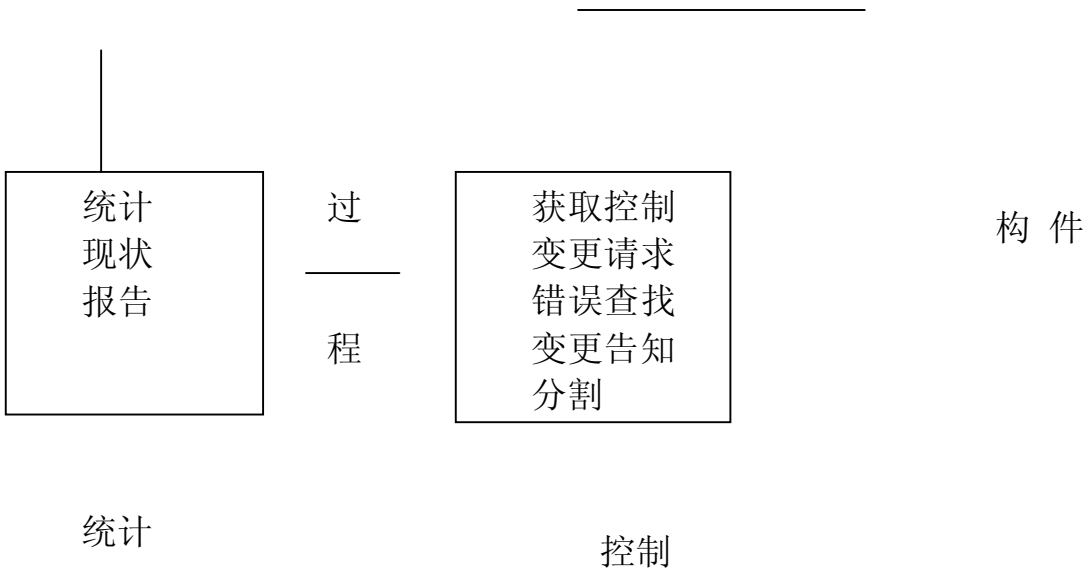


图 1：CM 功能要求

功能区域有：

- **构件：**标识、分类、存取构成产品的组件。
- **结构：**表示产品的架构。
- **创建：**支持产品的构建及其产品的附件。
- **审核：**对产品及其过程的审核予以保留。
- **统计：**采集与产品、过程相关的数据。
- **控制：**控制产品变更的方式及时间。
- **过程：**支持产品演变的管理。
- **团队协作：**促进项目组开发及产品维护。

以下将对这些功能区域的进一步探讨。

u 对于元素的要求，用户要：记录元素的版本及其差异，差异的原因；确定构成配置及配置版本的组件群；标识出产品的基

线及其外延产品，确定表示项目组件群及附件项目环境。而且，用户需要数据库来存取组件及 CM 信息，同时还有资源和对象编码、执行情况、图表、文档和基线。

u 对于机构的要求而言，用户要：通过表示产品组件库的系统模型来模拟产品的结构；标明组件、版本、配置的界面使之可以重用；确定及维护组件间的关系；选择兼容的组件使之形成有效的、一致的产品版本。

u 对构建的要求而言，用户要：容易创建产品的手段；能随时静态分析产品的现状；通过减少组件的堆积和节省区间来优化系统创建的机制；进行更改分析以预测因更改而导致的细小分化的手段；随时都能对产品的任何部分、在任何阶段容易得到更新。

u 对于审核的要求而言，用户要：所有更改的历史记录；所有与产品相关的组件与其演变的追溯性；完成任务的所有细节的日志。

u 对于统计的要求而言， 用户要：统计记录的机制，产品现状的检验，有关产品和过程的所有方面的报告能较易产生。

u 对于控制要求而言，用户要：为避免不必要的变更或变更冲突对系统中的组件的获取应予以控制，对于更改要求的表格及问题报告形成在线支持；错误查找的手段及何时对何人会产生什么影响；在不同但相关的产品版本之间以受控的方式进行更改告知；将产品进行分割的手段以限制更改影响。

u 对于过程要求而言，用户要：对生命周期模型及组织方针予以支持；确定要完成的任务及如何完成、何时完成的能力；将相干的事务的讯息在适当的人员之间进行沟通的能力；将产品的

经验文档化的手段。

u 对于团队协作的要求而言， 用户要：个人和小组的工作区间；在汇合时产生冲突的解决办法；对产品的创建及其维护予以支持的手段。

注意：图中过程方框与团队方框代表功能区域极为重要的部分。这是因为它们影响所有其它区域或受到所有其它区域的影响。对于用户来讲，理想的 CM 系统随团队协作和过程的完全融合应当能支持所有的功能区域。但目前还没有此类系统。

B．CM 系统的集成

任何 CM 系统在某种程度上都能与它的环境融合。CM 系统可与其它工具并存或完全融合。适合与不同环境方面融合的有：过程、工具组和数据库。过程集成是将 CM 系统的使用模式（指 CM 过程）同环境的使用模式（指软件生命周期过程）的结合；工具组的集成是将 CM 系统安装在环境中使之至少能环境中其它所有工具共存。譬如，在编辑过程中，每当用户发出“SAVE”命令时，用户就会要求 CM 功能能建立一新的版本。数据集成指的是 CM 数据库的逻辑定位——它是否能与现存环境的数据库能做某种方式的合并，或它的数据库是否独立存在的，或它能否利用其它数据库中的信息。所有此类集成都是普通的工具集成和技术的转换问题。但，由于 CM 将影响到环境中的绝大部分物件并贯穿每一物件生命周期的所有阶段，CM 系统的集成势必对环境中的很多工具有重要影响。大多数 CM 系统能与其它工具共存，有些环境把 CM 看成其必不可少的一部分。

C．何时启用 CM 系统

对于在开发和维护产品过程中，项目组何时启用 CM 系统是不定的。有些项目组选择在产品经历开发生命周期并准备发到用户地时开始启用。有的选择在项目一开始就将一切置于 CM 下。二者都有各自的一般费用。譬如，项目组可能基于变更要求的费用上来决定何时启用。如果有许多的手工程序（如：将变更申请表归档、寻求 CCB 的批准与确认可），项目组会选择在大部分开发完成之后将软件置于 CM 的控制之下。但如果变更要求程序能在线很快地得到处理，CM 将在软件生命周期的早期就被用上。理论上讲，CM 在产品的整个生命周期都能派上用场 —— 从创建、开发、产品发放、交付、使用到维护。在理想的情形下，CM 能在较少的花费下对此予以支持，由此 CM 才能在项目中尽可能早地予以应用。

然而，现有的 CM 系统只关注生命周期的某一特定的阶段，用户因此受到限制。

D．CM 的控制水平

很多的程序、方针和工具组合一块来支持 CM 的应用。它们在对用户的支持和产品的演变予以不同程度控制水平支持。譬如，它们会要求开发人员递交正式的书面的更改请求。配置经理则会建立一个工作区间给软件开发人员。配置经理可从受控库存中抽取所要的文档并将其置于该开发人员的工作区间里。当然，不同的程序、规定和工具事实上允许开发人员也可以通过电子邮件的方式将更改请求通知配置经理及 CCB 的其他成员。成员之间通过电子邮件予以迅速回应。一经批准，更改请求将被指派给开发人员，他可以直接从库存中抽取相关的文件并作出更改。所有这些无需手工介入。由于 CM 系统会自动记录所有的登入，更

改过程的正式记录就可创建。

前一种情形可被看成对行动具有积极的控制，后者较为松且被动。在前一种情形中由于手工耗费的原因，经常性的更改不予以主张，而后者恰恰相反。这种不同的控制水平在产品生命周期的特定阶段有其适用性。譬如，前者更适合维护而后者适合开发。不管 CM 系统这么用，它对于用户和产品的演变历史都有一定程度的控制。它将驱动用户的过程并将其加强。现有的 CM 系统提供或松或紧的控制水平但很少能灵活地允许用户选择控制的种类。

E．过程与产品的区分

CM 包括过程和产品。CM 过程表示执行 CM 是所需的一系列工作任务。从根本上讲，过程是一个计划，它定义要做什么、谁来做及如何做。支持这过程是管理的功能。过程模型将组织的方针、程序和软件开发生命周期模型通盘考虑。CM 的产品是工程管理任务的结果。CM 系统的功能需要为二者予以支持。现有的 CM 系统提供一些产品及过程的支持，但在同一 CM 系统中一般不能形成对二者完全支持。

F．CM 自动化水平

目前，CM 一般是手工和自动程序二者兼而有之。无需任何在线支持来实施 CM 也是可能的。但这样做的效率很低。我们的目标是将 CM 中非创造性的部分尽量多地自动化。譬如，书面更改表格和对此回应的监控一般只在组织方针里予以记录，而不能在线获取与加强。

G．CM 系统功能

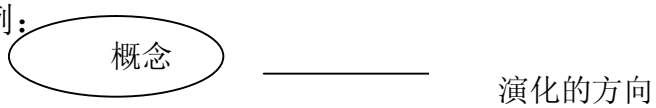
现有的 CM 系统提供的只是所有不同种类的用户的部分功能。但随

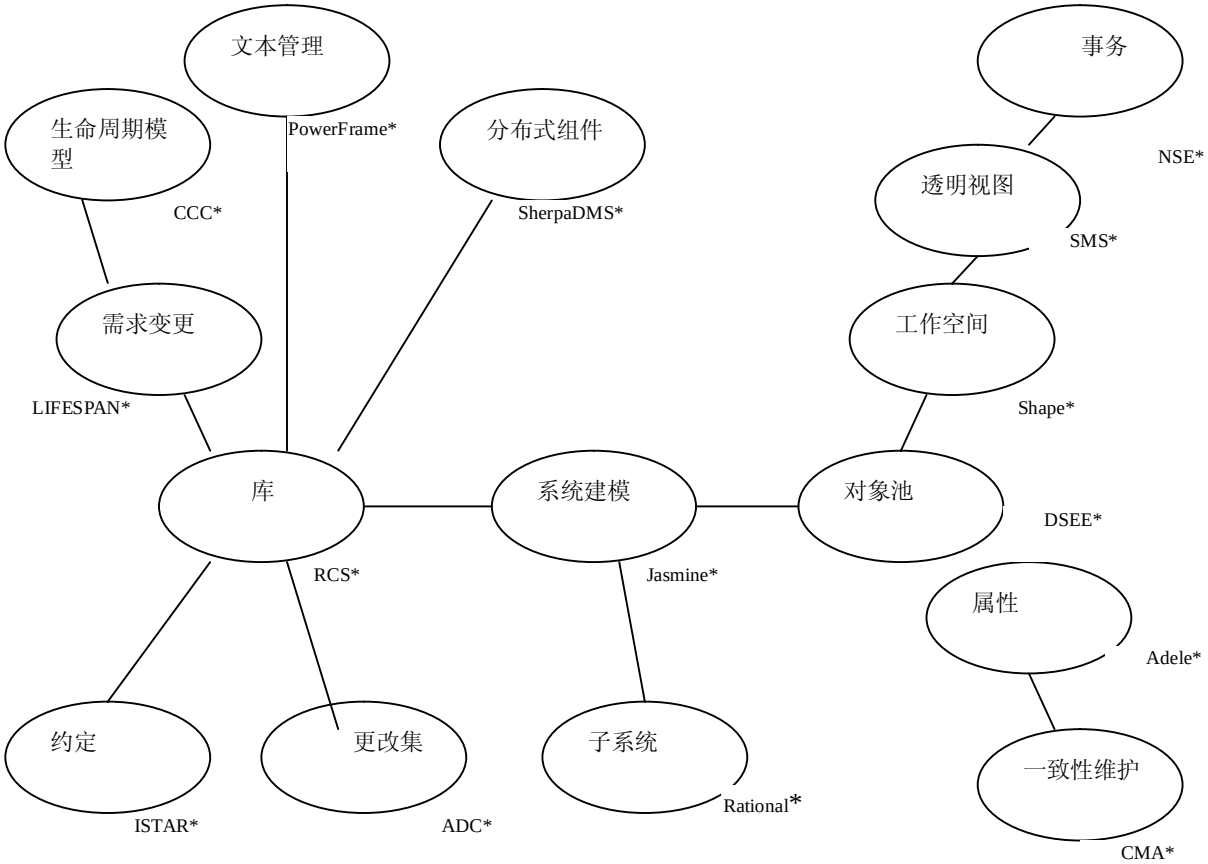
时间的推移和用户的需求和环境结构的能力得到更好理解后，这种情况将可能得以改进。以下部分描述的是现有 CM 系统概念范围。

3． 配置管理系统概念光谱图

以上部分解释了有关配置管理系统需求问题的范围，本部分将细述配置管理系统的具体功能，并对于支持前文所述某些功能的概念特别加以考察。这些概念将被组织成一幅光谱图来表示配置管理系统的演化过程。每个概念都将置于一个特定的配置管理系统中来描述。以下是要讨论的我们感兴趣的配置管理系统概念中的功能，包括：组件，过程，结构和特色架构的组合，团队概念。图 2 展示了整个概念光谱图和它们对应的，有代表性的配置管理系统实例，然后给出每个概念的一个简单描述并着重突出它的优势。在本部分的末尾，将对概念和概念光谱图的作用和局限性作一个分析总结。

图例：





*表示本节点所示概念的系统实例

图 2：配置管理概念光谱图

A. 注意事项

值得指出的是要讨论的概念和系统仅仅是现有系统的表示，而不是现有系统完整的评估和总结。对于每个概念，都用一个配置管理系统实例来讨论。但是需要注意的是，许多配置管理系统实际上提供了不止一个光谱图所示概念。既然谈到配置管理系统自然形成的功能时没有通用的术语，而这些概念是直接从特定配置管理系统中提取——所以每一个配置管理系统有自己的概念和定义。为了注意力集中，概念的描述都尽量简化。这样一来，就无法对所有概念的能力（不是它们的系统）面面俱到。但是，因为要提出概念光谱图和精简出一个配置管理系统概念集的缘故，简化是必要的。本文参考的每种配置管理系统在附录中都有一

个简评，它提供了每种系统配置管理能力一个更全面的清单。

B．组件的概念

组件的概念是与标明和访问软件产品的组件相关。它们包括下文所描述的库和分布式组件。

a.库

库的概念是配置管理系统的根本。*修订控制系统(Revision Control System（RCS）)[15]*提供了 ASCII 码文件库的概念。从效果上来说，库是集中控制的文件库并提供对库中所存储文件的版本控制。任何库中的文件都被视为在确定的配置管理之下。库中的文件是不会变的——它们不能被更改。任何更改被视为创建了一个新版本的文件。文件所有的配置管理信息和文件的内容都存贮在库中。所以，任何配置的管理和控制都与库中的文件相关联。当工作于一个文件时，用户将某个版本的文件导入工作目录，然后开始工作，处理完了，然后将文件导回库中。这样就生成了这个文件的新版本。所以用户不可能导出一个文件并同时在库中修改源文件。

从库的角度来看，导出的文件自动被锁定直到文件重新被导入，一个版本号自动与新版本文件相关联。这样一来，用户可以随时根据特定的版本号来导出任何文件（缺省的是最新的版本）。对最新版本的修改的结果是产生一个新的，顺序递增的版本；而对更老版本的修改的结果是产生一个分支版本。在版本编号策略和使用模式共同作用下，产生了文件版本历史树，用来表示祖先和后代版本。库中不但存储了文件的不同版本，更改的理由，而且存储谁在什么时候替换了某个版本的文件等文件历史信息。请注意，对于每个不同版本文件，不是所有的代码都存

储起来，而只是不同版本间实际的差异才存储起来：这称为*增量*。这种方法有利于节省空间和节省对最新文件版本的访问时间。另外，可以根据状态给文件加上标签，然后基于状态的值进行导出。它们同样也可以根据修订版本号，日期和作者进行导出操作。库总是和文件所在的目录相关联。总而言之，库捕捉配置管理信息并把不同版本的文件存储为不可修改的对象。

b.分布式组件

Sherpa 设计管理系统（**The Sherpa Design System**（**DMS**））**[7]**提供一个文件库，其中的文件分散分布在不同的硬件平台上。在逻辑上，库是集中控制的，但在物理上，库中的数据是分布的。**Sherpa** 设计管理系统自己知道数据的分散分布，并把这个因素考虑到配置管理系统中去，例如，在提供必要的文件格式转换时提供一定的容错能力。这样，对于用户来说，数据的分布是透明的——用户对库进行的任何工作感觉上和所有文件放在自己的本地工作站上一样。一组地理上分散分布的用户可以针对同样配置的文件一起工作。多个文件的副本可以在不同的工作站上存在。**Sherpa** 设计管理系统总是知道最新文件版本的位置。任何对从库中所导出文件的更改会导致所有分散的本地工作站上的副本更新，因为系统知道所有本地副本放置的位置。更新可以是一步一步交互式样地发生，也可以是批处理式地完成。有效的，分散分布的用户能够直接访问集中控制的库。对他们来说，配置管理能力看起来遍布整个异构网络。

C．过程的概念

处理与过程相关的功能的概念有以下几个：上下文管理，约定，变

更请求和生命周期模型。以下是详细描述。

a.环境管理

PowerFrame[13]是专为计算机辅助工程/设计领域而设计的系统。对于用户，它实际上是把文件系统和配置管理底层的细节屏蔽起来。用户只能够看见和他们特定工作领域相关的，一个电路设计的世界，而**PowerFrame** 管理工作中的上下文。项目的数据不是隐藏在目录里而是显式地用图形表示出来。贯穿整个工作过程的始终，**PowerFrame** 提供工作流管理，来引导团队的成员。例如：工具—运行可能涉及电路的生成，置电路有效，然后通过进行仿真来决定他们的性能特点。在这一串的动作中，**PowerFrame** 自动根据工具—运行提取相关的上下文，诸如数据集，命令文件和激活工具的选项等等。　等下一次，用户仅仅需要选择电路设计和工具功能就能开展工作。用户所看到的是：针对特定任务的合适的工具，特定的数据表示表格：如逻辑图和布局设计；与特定任务相关的数据；和特定工作领域相关的命令表。用户可以在不同的尺度上执行不同的动作：如上下文数据中一个简单的数据项或者到整个配置管理。用户不必去操心象版本控制或文件之间的关系等这些任务，因为在屏幕背后，系统知道如何从不同版本的电路设计提取数据，系统完成了这些任务。从效果上来看，配置管理系统针对特定工作领域捕捉用户工作的上下文，通过这样的方式减少了用户的工作，如记住如何到达某个具体工作状态，所有的数据项和它们的关系是什么等。

b.约定

ISTAR[9]环境根据正式的约定提供对部分软件开发过程的建模。所谓约定是指指定输入和输出条件下任务的执行。约定的产物被记录下来

作为配置项。一个约定把信息流，包括从任务的开始到完成，任务之间结果的传递和产品中组件的传递，进行建模。并且，约定之间也是可交换的。怎样来满足约定呢？约定的满足是根据一定的接受标准，把输出传递给过程模型中的特定的元素，如生命周期的不同阶段，或人等。约定的产物活动被随后记载下来。因为不同的约定产物（如通信）会被记载下来，所以约定中的工作过程是被监视的。从效果来看，约定表示一个工作团队在一个配置项下的正式计划和记录。

c.需求变更

在 **LIFESPAN** 中,软件需求变更表现在文档的需求变更和相关过程模型的变更。**LIFESPAN** 通过一系列的表单来实现需求变更的建模,在通过一系列状态,任务和角色来实现过程的变更。客户可以提交用来确认错误或请求为组件版本升级的在线软件性能报告。这就允许此报告能被反馈给那些可以诊断出此问题的原始设计和编程人员来研究。对于软件性能报告和改变冲突分析的反应,一个在线的设计变更被提交表决。确切的说这就详细到什么组件被改变和怎样改变的问题。**LIFESPAN** 分析了谁将会被此变化影响。然后那些人就会被自动的选出组成控制变更委员会。关于设计变更的报告将会通过电子邮件来通知他们，不管他们是否同意这些变更，都必须在一定的时间内对此做出表决。一旦设计变更被通过，一种可变更的代码的新开发版本就产生了。则设计变更就此开始使用而那种代码的变更就被锁定。在变更完成后，新的版本形成了，需要被提交给具有 **QA** 特权的人来检测并批准。经过批准后代码变更就需要一种确认状态，设计变更的状态也变为确认的，有关的用户就被通过电子邮件来通知一种新的版本可以使用了。用户收到软件状态报告表，

这就取消了原始的软件性能报告。因此，软件性能报告，设计变更和软件状态报告不仅为用户和维护人员提供了一种交流的方式，而且体现了这种特殊变更需求的历史变更；在过程中变更的状态报告；变更完成的最终审计结果；改变冲突分析的机制和确保相关人员按时完成任务。结果需求变更就促成了那种变更的过程。

d.生命周期模型

变更和配置控制提供了对一种特殊的生命周期模型的理解，此模型在某种程度上支持一个生命周期中的各阶段及人员之间的转变，而那些任务和数据管理能在那些阶段被执行。他通过把这些阶段分为对产品的开发，测试，鉴定和推出来实现。这种划分允许象软件工程师和测试员这样不同种类的使用者能够在同样的代码下同步的实行他们的工作。阶段和独立工作的划分及其间的转变通过贯穿于代表每一阶段的独立配置的代码来实现。就是说，产品作为一系列的基线被开发。每一基线存在四种配置：开发，测试，鉴定和生产。配置是组件的一个层次。每一基线包括一种特殊的方法。代码的开发就是开发配置，通过反复对配置进行的测试，然后确认配置，最后生产顾客使用的配置产品。为了顺利到达下一阶段，交互作用的草案必须要被不同的用户（例如项目经理和测试经理）批准这一转变。任何时候，对于一组件所通过的标准是由他所属的配置体现的。结果，生命周期模型经过不同的配置状态被实现。

D．结构和解释的概念

所要了解的概念是：选择一种结构的组件；获得一组件和其结构的变更；描述一产品的结构；存取这种结构；构造这种产品以及保持这种结构的各个部件的一致性称之为变化集，系统建模，子系统，对象池，

属性和一致性维修。见下述。

a.修改集合

ADC 把在数据库中的一个基本概念 — 各部件之间的版本的不同 — 抽象成一种不同的关系，这种关系对于用户是可以访问的。这样不同的关系伴随着与之相匹配的文件以及其它变化的细节组成了变化集。**ADC** 把变化构造成变化集中的配置，变化集可用来构造某种配置的定制外形。这种变化集有一个名字，这意味着它可用在操作中，用户制定一个公式来创建某个配置的特殊实例。这个公式指定一个被选中的变化集都适应的基本线。一个变化集可视为与以前的变化集是相联系的（即版本的历史的延续）或是相互独立的（即历史版本的可选部分被应用），特别是变化集。 因此，用户要么从最延版本中工作，要么在一种配置定制版本下工作。由于某些变化以及谁，何时引起的这些变化等细节，这个变化集可以捕获对在某个配置中所有文件的变化。用户指定这种变化的范围，**ADC** 自动的纪录这些变化的细节。例如，由于一个错误用户想使主要变化适合某种配置。用户指出一个变化集，对这些文件做出许多变化。在这个变化集中被捕获的：由于对在配置中所有文件做出的变化所有原代码都得改变（在这个配置中对每个文件来说是不同的）；所有有关文件的改变；以及谁，何时做出的改变。当用户浏览每个文件或变化集时可以看见很多信息。总之，变化集表示对某种产品和创建一个配置的各种版本方式的逻辑变化。此配置不必依赖于本配置的最新版本信息。

b.系统模型

系统建模用来描述软件产品—软件产品的结构、组件和如何组建

它。Jasmine 系统建模就是用户能变更的文本描述以及一些工具可以用这些描述来存取完成他们的任务。Jasmine 系统建模是由体现以下四类信息的集和函数来描述的:（1）组件产品的关系,（2）绑定的信息版本,（3）构造规则,（4）验证规则。关系描述为象子组件等级的产品模块的分解,产品的独立性（比如模块组建的顺序）和基于属性的组件组（比如各种资源和对象模块的分组）。通过关系描述的产品称之为模板且获得它的结构。通过这些函数操作和关系用户可以使用简单关系定义复杂关系。这就能使 Jasmine 工具来解决用户定义的查询，比如：通过改变一个特定的组件来影响那个组件。系统建模包括进一步了解该产品系列的历史。此系列产品描述了该产品的后续版本。某种产品的用户指定的版本构成了一个产品系列。和每一版本关联的是创建日期、作者等属性。构造规则记录了现有的组件是如何生成的和将来的组件是如何构造的。比如记录编译器、版本和所需的编译选项 。验证规则指定合和记录对产品的结构和组织的限制比如资源和绑定模块必须匹配（意思是所有的绑定模块都是由那些资源模块编译而成的）为了选择一种组件版本，用于系列的选择方式是避免使用体现查找模块路径的内容来评估的。被选定的结果模块易把图像的数据对象当作一种模块。象浏览器、模块查看器、调试器和模块间的分析器等工具能引用和处理系统建模。最终，系统建模是来自于实例产品的抽象，为了全面描述产品，系统建模用工具来维护产品的完整性。

c.子系统

Rational 环境提供了把一个很大的 Ada 产品分成多个小模块以及限制变更影响范围的功能。这些小模块被称为子系统，子系统包括接口说

明书和实现主体并指出配置项目，因此，他们可被看为一个整体并通过他们的名称被评价。在一个子系统内的组件不可被其他子系统内的组件所访问除非为了被输出而通过接口说明书将这些组件指明。**Rational** 环境检查实现主体完全匹配上接口说明书所需的运行时间。结果是，工作可以在实现主体上展开而独立于当用户想用时就可以被改变的接口说明书，到接口被改变时仅针对那个子系统中的组件会发生二次编译。这时使用了这个接口产品的任何模块都将进行二次编译。对一个接口说明书所作的更改可能需要整个产品进行二次编译。子系统对其组件进行了版本控制，子系统本身可以是一个特定版本。用户可以用过组合匹配系统的版本来形成该品的一个特殊产品。概括的说，子系统为用户指示了一种方法，它限制了变化和二次编译所带来的影响，并提供了检查一个产品的各组成部分的有效性的环境。

d.对象池

运用系统建模的概念，**DSEE** 已拥有一切必须的信息，此信息能够确认产生一个生成对象的特殊版本需要些什么。生成的对象被放置在用户们共享的对象池中。一旦用户暗示了对对象属性的需求 **DSEE** 就能够共享。被产生的对象池包括一个由转换工具生成的二进制代码和其它对象组成的集合。每一个被产生的对象和其所有的信息有联系,而这些信息是关于其包括原始版本的系统建模和与转换项一起使用的转换工具,被包括的用户注释的出处、日期、时间、人员和引出的位置.这个信息被认为是一个**BCT.**当**DSEE**构建一个系统时,会为系统中每一个组件计算需要得到的 **BCT** 数。**DSEE** 在对象池中查找来看生成的对象和所需的一个已存在的对象是否匹配。如果匹配，它就被用；如果不匹配，它被构建。

因此，任何时候一个用户需要一个特殊的生成对象时（或是一个一致的对象）。DSEE 能够从对象池中再使用从而消除了生成这个对象的需要。用户不需要知道生成现存对象要做的；DSEE 作了全部的检测。一旦池中的对象成为死亡的（基于一阶段无作用）DSEE 能够删除他们，从而释放空间。这就节省了大量的编译时所需的时间和空间，再使用的工作已经在进行。DSEE 也提供了各种不同的对象池，例如从源文件中得到的对象仍是对提供给特殊用户的库的检测。结果，CM 系统使再生成组件的需求最佳化且最大数量的分享生成对象。

e.属性

Adele 系统通过用一个有数据建模能力的关系数据库实体来普及库和系统的建模。产品在一个数据模型方面被描述，Adele 基于那个模型进行它的运算。产品的组件被描绘为拥有属性和关系的数据库对象。属性和每一对象及那个对象的特性相关。属性有一个名字和一个值。一个例子是名为 delta 的用于描绘对象是否存在于 ASCII 表中从而能够被理解的属性;它可以有一个为真或为假的值。有两种属性被区别：预先确定和用户确定。前者被 Adele 管理而后者被用户定义和声明。一个预先确定，特殊的属性是“类型“。这个命令属性是强制的且对每一对象都是不可变的。它在 Adele 中表现为主要的的 CM 实体（例如对象的组成，文献，修改和元素）。关系在对象间独立定义，例如对象 B 源自对象 A。用户能够按照对象的特性而不是按照一系列对象的特殊版本来描述一个配置。Adele 例示和构建一个配置用来选取规则和强制围绕属性和关系为中心。用户能够按照所需特性对一产品定义任何结构。从而用户能在一经由其特性的抽象的高水平描述一产品，其优于按照冗长的文件列

表的组成来描述。

f.一致性维护

CMA 提供了配置的释义和确认，其是基于一个对于产品的抽象描述也是基于有关形成配置的组件的使用用法成功或不成功的信息。数据建模便利的包括预先确定用户所描述的配置的属性和关系。基于那些属性和关系的语义，CMA 能够决定一配置（就是一系列组件的实例）是否是可用的。成为可用的，一配置必须是完全的，无歧义的，一致的和没有歪斜的版本。这意味着一个配置必须有全部组件所需的实例组成且不必包括多重的一个组件的实例。属性的等级描述了象约束，类型和版本这样的用户定义的特性。关系的等级表现了各种依赖性，例如，合理性，兼容性，构成，实例和可继承的独立性。每次一个新的配置被组建，CMA 就利用经由先前对形成配置的组件的使用在数据库中积累的信息。这样，CMA 预见配置是否可用。这种新的配置为了将来分析可用性而加入数据库。从而，用户能够依靠系统来识别任何不一致以及在构建和重复使用用配置时保护此不一致。

E．团队概念

描述工作在一个工程项目上的软件工程团队间的独立、合作、同步的术语是工作区，透明检查和协调。描述如下：

a.工作空间

工作空间为开发人员提供独立的工作空间。

在“形状”中的工作空间是被设计用来防止用户之间的相互干扰。它提供了在配置管理下的能在可调对象上持续的工作空间。工作空间是

通过版本状态模型来获得的。这就意味着属性“状态”是和构件的版本相连系的。依靠那种状态（例如状态“忙”或“冻结”），构件或者被认为是一个私有的工作区或者被认为是一个公有的库。“忙”构件是可调的并且不能被其他人所使用，象“冻结”就是一个对公共使用来说能获得的但不可调的例子。构件被提交给公共库的同时使得它们在被适当的用户证明后，对公共用途来说是可获得的。在效力上，工作区提供工作的独立性且建立在一个全局的、长期的为不可调对象的库和一个为可调对象且私有的短期的库之间的区别。

b.透明视图

透明视图提供从主配置库到工作区的访问机制，该机制具有防止非法存取的功能。

软件管理系统通过使工作区成为一个透明（清晰）的对象和提供在那个工作区的库的透明检查来增强了工作区的术语。这就意味着仅仅用户感兴趣的文件版本能在工作区中看到，所有其他的版本都不可见（尽管它们在物理上是存在的）。例如，任何对最新公有版本的变化都不需在工作区里显示出来，用户从公有变化中分离出来，并且工作区提供给用户一个特定库的外表。相关工作区版本计数的版本控制提供在工作区中。新版本是私有的并且在从工作区中释放出来之前是不可能被公共用户所见的。一个配置从公有库中检测出来提供给工作区。用户访问分配给自己的工作区。工作区里的组件有效地属于那个工作区而非一个用户。仅仅在那个工作区已登记的用户才能改变配置，且仅有那个工作区的构件能被访问。总的来说透明检查通过防止对一个配置的非授权访问而提供了一个检查机制。

c.协调控制

协调控制协调开发组成员对同一配置项的修改。

网络软件环境（NSE）[12]协调控制代表了一个工作协调单元。它反应了工程的结构并且支持工作的独立性、用户间的相互影响和合并变化。一个协调包含一个环境和一系列命令。环境提供了类似于工作区和透明检查的术语。它显示了用来存储资源和派生对象的目录结构。那些命令，例如“获得”、“退后”、“重新同步”和“解决”，在不同环境中提供相互活动。它们代表了用来协调和同步用户间活动的协议，也代表了实际变化的通信。用户独立地工作在他们自己的环境里，改变相同的或不同的配置。用户用配置的新版本来更新库。网络软件环境支持将变化合并到库里。但是它检查什么已存在于库里（可能被其他用户放置在那儿）而且不和正在进行的变化产生冲突。假如有冲突，网络软件环境提示用户注意合并问题，同时提供减少冲突的帮助。对于任何库的变化，用户能请求进入他们自己的工作区。总而言之，协调同步和协作团队们改变工程产品的相同或不同部份。

F．光谱摘要和分析

图 2 代表了一个不同配置管理系统的配置管理术语光谱。这些术语和它们的目标是：捕获不可调文件历史的库；在配置管理下数据分布的已分布构件；一个工作单元计划的合同；一套捕获配置变化和允许最新版本独立配置选项；增强一个组织软件进化过程变化的生命周期模型；完整地描述和记录结构和建立工程的系统建模；使得重使用的派生对象的对象池能优化产品构建；允许基于特性的配置选择属性而非一长串文件列表；支持持续的自动检查和配置组件之间非持续的预测；分离可调

配置的私有变化工作区；一个查看配置和防止非授权访问的可调配置的透明检查；一个协调配置变化的团队协调。这些术语代表了配置管理系统功能方面的先进性。

光谱拓朴的目的是显示一个术语的进化过程。例如，图 **2** 从左到右总的来说有不同过程的建模、捕获组件、描述产品的构件，优化产品工程。特定构件间相互关联的协调团队工作。光谱的“臂”显示了相关过程。例如，需求变化和生命周期模型（如本书描述的一样）是相关联的：生命周期模型小计了一个特定变化需求模型，同时变化需求操作了一个库。

有一些术语在光谱上没有显示。那些不能显示的术语如：构件的细微进化（例如从版本标识到配置标识到不同配置的不同版本）；系统建模过程（例如从命令文件的进化到创造文件到系统模型如版本对象）；“角色”的识别和不同类型的变化（例如增强反病毒功能，病毒出现提示）；目前的研究工作。

在本书上简化了从配置管理系统是提炼出来的术语。相对于已实施的系统来说是为了找到一些共同的术语。没有共同的词汇来表达术语。术语和它们的实施之间的区别并不总是清晰的。例如，工作区的实施在不同配置管理系统中变化，同时为用户提供不同的功能。此外，工作区的术语应该是所有实施的最低共同命名或相反？既然协调统计了工作区和检查的术语，那么工作区、透明检查、协调又真的是同一术语吗？或者它们真的如在光谱上显示的一样是三个术语吗？

另外一个在提炼术语时的难点是大多数配置管理系统都有过多的术语。那就是一个术语有许多目的（这些目的在配置管理系统中通常是

不统一的）。例如，**Rational** 子系统术语在光谱中被看作为限制变化范围而提供支持。然而，子系统比那个术语提供了更多的功能。它们能：提供一个名字范围边界，支持系统分区，代表一个基线，一个工作区，代表一个意思（为工作在不同的配置或一个团队的同一配置）。检查接口提供的细微变化或代表一个不可调的可执行的组件（在 **Rational** 术语中的一个“装载检查”）。因此，为了讨论子系统，在其一个特定方面的磨合是必要的。此外，过多的术语使得提炼基本的术语变得困难。同样，组合不同术语的不同部份，或一个特定术语的实施副影响都使得术语的提炼更加困难。例如，当考虑一个变化需求时，角色（象配置经理和测试经理）和生命周期术语（如开发和测试）对那个术语是至关重要的，或者它们是独立的？

无论如何，这些术语的光谱为开发提供了一个起始点，或者至少从已存在的配置管理系统中提炼出一个配置管理模型——一组基本的配置管理服务。此外，需要进一步的工作来决定：光谱的使用价值，是否还有其它的术语，怎样定义、命名和表达这些术语以及它们的多种语义，并且怎样将这些术语组合成一套有用的配置管理系统。

4.配置管理系统的未来

图 **2** 所示配置管理概念光谱图表示了商用配置管理系统的典型概念。我们预计，随着研究的继续，和不断从这些概念的结合使用中获取经验，光谱图上的许多分支将会互相连接。这意味着，每个配置管理系统最终都可能将提供一个基本的配置管理服务集，从而更好地适应用户需求。但是，即使不考虑是否每个配置管理系统设计者都试图实现这些共同的特征，还有政治和技术方面的因素都会影响未来的软件配置管理

系统。（政治层面的因素是指与市场和标准化相关，技术因素则是关乎实现某一特定机制的可行性。）

一个主要的政治因素是关于 **CASE**（计算机辅助软件工程）工具的发展。例如：**CASE** 工具经销商是否应该假设环境经销商会在他们的框架内提供配置管理支持，所以他们自己可以避免在他们的工具中实现配置管理。或者，是否应该由 **CASE** 工具开发商在他们的工具中提供配置管理支持。如果 **CASE** 经销商合并他们自己的配置管理支持，那么当用户安装不同的 **CASE** 工具时，用户将不得不自己解决如何集成不同的 **CASE** 工具的问题。同样，从经销商的视角看，他们会真正重复去做那些已经被整个环境框架尝试过的工作吗？

另一方面，如果 **CASE** 经销商不把配置管理合并到他们的工具中去，他们能依赖环境集成商提供合适的环境框架，去集成 **CASE** 工具并同时提供某种通用的配置管理能力吗？这些问题的答案都是未知的。我们都可以看到，任何一种情况都意味着，对于环境来说，配置管理系统需要一定的标准化。反之亦然。

许多技术、研究方面的问题都影响着配置管理系统的能力，冒出来了如下这些问题：

什么适当的技术可作为配置管理系统的基础？对象命名约定不变的面向的对象数据库技术是最合适的吗？在环境体系结构中软件配置管理是在哪一层？它是否应该作为环境框架中一部分，放在数据库的基础层，还是把配置管理看作一个过程，处于体系结构的较高层？配置管理的机制能否从所有的配置管理功能中分离出来？也就是说，是否有一个标准的配置管理本质部分，能够在任何环境中使用，并支持所有的配

置管理功能。存在一个统一的配置管理模型吗？是否可能提供分布式的配置管理支持？在地理上分散的软件开发组能否与本地配置管理和系统集成使用同样的配置管理系统。这是工业界的一个主要难题，尤其是对于国防合同承包商来说。配置管理支持跨软件开发吗？工程师是否能够在主机上开发产品，然后在保持对产品的配置管理控制的同时轻易地将它转移到目标机上去吗？规模是配置管理系统的一个限制性因素吗？配置管理对一百万线产品的支持和对一兆线产品的支持是一样的吗？有可能将配置管理过程中，包括劳力密集型的部分，所有方面都建模，并在配置管理系统中实现它吗？

对以上这些问题的回答都不是显而易见的。因为很可能要管理的过程有着不同的来源，从配置管理系统经销商，开发环境集成商，研究人员，工具继承商，软件过程建模论坛，还有计算机辅助设计，计算机辅助工程，计算机集成制造等不同的领域。

5. 结论

配置管理是对软件产品发展演化进行的管理。从配置管理系统的操作层面上看，配置管理是认证，控制，状态统计，审计，评估，制造，过程管理和团组合作。它是软件工程领域的一部分。它的工作对象是这个领域中产生的过程。这从概念光谱图可以明显地看出来，同样也可以从已有的配置管理系统的数量和它们的能力看出来。本文的光谱图表示的是许多不同的配置管理系统经已实现的概念的一个快照。每个存在的系统的重点都不同，在用户问题——包括角色，集成，控制，自动化层，过程等等，与产品支持，什么时候是开始使用配置管理的最佳时机，系统能提供哪些功能等之间进行竞争和权衡。希望提供这个光谱图能够有

助于对配置管理系统能力的理解，并且提供一个讨论配置管理支持工具的通用框架。

6. 附录：CM 体系总览

这个附录给出了此份论文中前面章节提到的不同 CM 体系能力的总体印象。既不是整个体系的评估也不是完整描述，目的只是让读者对下列 CM 体系能力范围有一个了解，这些是存在于今天的不同种类的 CM 体系的代表：Adele，ADC，CCC，CMA，DMS，DSEE，ISTAR，Jasmine，LifeSPAN，NSE，PowerFrame，Rational，RCS，“shape”，SMS。这些体系在下面描述。

A. Adele

Adele 是一个来自于格勒诺布尔大学的配置管理体系。它的基本特征是数据模型，届面检查，展示产品系列，配置建立和工作现场控制。Adele 是用来成为软件工程环境的核心。Adele 数据库是一个实体关系，一个为物件提供定义，如界面和它们的实现（instances of bodies），配置和家族。物件有特性：描述它们的特点，和 DEP 关系：描述它们的从属关系，Adele 用这些功能来帮助组成配置。使用者可以指定一种基于合意的特性的配置。特性可以用户定义或体系定义。用户可以指定规则基于特征价值，局限和优先。Adele 可以探测到不完整和不连续的配置描述。

B. Aide-De-Camp (ADC)

ADC，来源于 Software Maintenance and Development Systems, Inc.，由基本的 ADC 体系和一个看守系统组成。基本的 ADC 提供了一

个数据库以获取 CM 信息。用户在文件内定义特征和关系。数据库可以贮存资源和二进制码，它贮存易变的(“塑料”)和不变的（“安装的”）信息。ADC 的列表处理语言有效地允许用户在一个文件或一组文件上工作。ADC 冲突解决方案在登陆（check-in）和标记时执行。改变设置俘获改变了配置和允许用户指定任何版本的通过一个改变设置清单从而创建它们自己的版本树。报告可基于数据内容而产生。程序建立得到支持，结构的关系被自动得到。一些非—ADC 的 CM 信息可以输入至 ADC 数据库。监管系统直接支持配置，集成问题报告，改变需求，和了解用户，承担分派工作指令和建立当地的工作站（“干净房间”）的角色。这意味着当一个变化需求被送到 CM 经理并得到认可时，经理把工作分派给软件工程师。当工程师执行那项活动时，一个被复制的本地的路径和文件工作站建立了。一旦工程师完成那项工作，工作站自动删除，变化被加入数据库。

C. Change and Configuration Control (CCC)

Softool 的 CCC（称为 CCC/发展和维护）被作为一个监管系统或作为一个本地的产品出售。CCC 提供一个变化控制方法论，配置标识和状态会计，以及起源建筑。所有的这些被用来假定瀑布生命周期模型。CCC 下的部件在适当的认可之后，经过了不同阶段的生命周期。CCC 支持一些文件化的标准。五个等级的客户构成权限的层次列入数据库。他们是数据库管理员，CM 经理，项目经理，开发者，及测试经理。一些层次的通道控制了存在，例如密码控制，用户等级，指定数据或改变需求分配。CCC 数据库层次，代表产品的结构，由多层次数据结构组成，包括数据库，体系，配置，模块和文本。编码的平行版

本可用于通过实质拷贝实现同时发展。这些可以合并或选择，变化可跨配置运用。在合并中冲突可监测到。CCC 的变化需求，如*项目*，可以处理一个部件的小变化，或产品的下一次发布所需的所有变化。电子邮件事件通知与变化需求相关。紧急变化绕过大多数的变化控制是允许的。

D. Configuration Mnanagement Assistant (CMA)

来自于 TARTAN 实验室的 CMA 提供 mechanism（无方针）创建 CM 系统。Mechanism 使用的是实体关系特性数据库。特性和关系的等级详细说明了部件的特征，一个产品的分解和部件之间的相互依赖。特性的等级是分割，演示，和版本；关系的等级是逻辑从属，一致性，兼容性，部件，立即和可继承的从属性。CMA 用来录制和获取配置描述，部件的组成，录制和获取关于一个配置的部件之间已知的（不）一致性和从属性。它预告新形成的配置的完整性，不明确，和一致性。任何数据库的变化是通过对简单“交易”的承诺产生的。每种配置可以有它自己的通道控制 mechanism。配置之间的名字冲突通过使用间隔来避免。

E. Design Management System (DMS)

来自于 SHERPA 公司的 DMS 适用于电脑辅助设计/工程师市场和硬件的一部份，设计工程师环境。DMS 提供逻辑的集中仓库，内含清晰的分布的数据。文档可包含任何种类的信息，如 ASCII，图形，以及设计数据。文档的版本通过当地操作系统的版本控制来实现。所有信息（产品结构，发布程序，事件警告，用户定义特性和关系）被集中在一个核心的数据库。“发布”的意见通过促销水平（代表项目通过时

的台阶）获取。这些代表公司方针用于评审，确认或完毕信号。用户可以指定谁可以获得什么样的数据，数据群，谁应该被告知状态变化，完毕信号及促销需要什么样的确认和检查。DMS 通道控制是在用户等级和 promotion level of file 的基础上实现的，文档名可以加密，实质的团队可以定义（这些是地理上分散但分享同一数据库的用户）。可要求自动同步更新或分批更新。变化可以在小组成员间得到交流沟通。不管在网络的哪个地方，文档的最新版本都可以定位。DMS 用这个结构来执行检查。并可提供报告及预评审。变化需求（包括相关文件）确认后自动随附。

F. Domain Software Engineering Environment(DSEE)

DSEE 提供版本控制、系统建模、配置发放、分散系统建立、物件组、用以查找要做的事务及已经完成任务的任务单、将特殊事件通知用户的控制。版本控制置于一资源文档库中。一 DSEE 系统模型是对一产品或产品一部分的描述。它是一针对静态和结构特点的公开描述，包括资源文档、派生物件和从属工具、组件的阶层、创建规则、创建顺序、数据库及路径的确定、转换工具的选择和一些控制过程规则。

G. ISTAR

ISTAR 来自于 imperial software technology ltd. 是一个环境设计的特别用来支持项目管理的。软件项目个体之间的关系被模仿为合同。一个合同理论上是对期望产品的描述，并被构造成数据库。一个配置是在合同之间移转的单元，移转时被认为是“冻结的”。合同的移转暗示了一定的任务或阶段已完成。CM 为合同数据库内的项目而存在，并在合同之间可交付。为数据库内的部件提供继承者和不同的控

制。用户可以定义 CM 部件之间的关系，可以为问题报告分配部件。这是对系统建立的支持。

H. JASMINE

JASMINE 是应用于室内 CM 的 XEROX 信息系统分配上开发的大型程序设计系统。系统模型是其核心。它描述一个软件系统，这个软件系统使用在设置和功能上构建的代数模式。用户能用这个代数模式来定义复杂的询问和简单的译本。软件结构则被定义在模板中，翻译捆绑体由图象支持，后继的翻译记录在一个族中，这个族支持并行开发。专业译文被分类后组织成特殊历史记录（如：一个项目专业历史记录）正文内容和这些族均被提供给译本，同时定义它的语法结构和连贯性。

JASMINE 工具利用系统模型信息拷贝文件并存档，编译源代码，浏览并释放空间。

I. LIFESPAN

生命期来自 YARD 软件系统，严格支持变化控制。它适用于项目经理监控各种变化的情形，只有经过授权的用户才能使用它。生命期使用相关的数据库和询问语言，存储文字、二进制代码和图表，并为这些项目提供版本控制。

目标集 BELONG TO。。。负责批准对包进行改动的管理人员被指派此包。生命期使用制图办公模型，这些模型建立在硬件设计方法论的基础上。它识别状态量，偏移量，偏移触发器，命令行和用户权限。电子邮件提供自动识别功能。报告建立在库存项目基础上，并可以进行改动。在安全方面，配置项目设有密码和加密的文件名。它支持各种国际标准的问题的提出，跟踪和正式改动控制。

一般认为测试信息也是一种配置项，它依赖于其它项目。生命期监控改动的一致性标准过程。它决定什么系统使用回顾性模块，标识所有需要被纳入回顾系统的开发人员并发布必要的控制文件。

改动一经批准，如何授权它并分配源代码是一项管理策略。以上工作完成后，项目被从存储区调出，模拟，以开发项的身份重新提交。此过程重复进行。

J. Network Software Environment（NSE）

由 SUN 软件系统开发的 NSE 是管理操作系统目录结构并从源代码获取附加文件的一套应用体，附带一个数据库。NSE 为开发代码的项目组提供工作空间。此工作空间通过一个合并并升级处于子空间和父空间之间的文件的协议来支持递归转换。工作空间里的文件表示为一种结构体，它代表对这种结构体的多种版本，除了最后一种，其它的结构体都是不可变的。同一个工作空间的不同用户在此工作时都得经过检查并登记在文件中。合并交叉工作空间的冲突问题 NSE 提供了交互性支持。工作空间能够高效地获取目录结构，这种目录结构用于存储源代码并从产品，已建成的结构体和产品的逻辑结构中衍生构件。

K. PowerFrame

Powerframe 这个工具来自 EDA 系统，对计算机辅助设计工作提供配置管理。它用一种统一的图解接口把用户屏敝在操作系统和文件系统之外。操作时，用户拖出一个合适的工具菜单。POWERFRAME 自动检索所有相关数据，运行这个工具，在用户使用完毕时保存所有的改动。POWERFRAME 把在产品中数据的几种组织方式合并起来以便用户集中精力于那些仅适合于完成特定任务的数据，工程，一个展望，一个见解和

一个数据包。一项工程是一个数据的集合，这些数据构成协作体的主题（如，一个包含了所有电流设计线路文件版本的产品）。一项展望是一套工作，专业工程师任何时间都可以使用这个装置的文件版本。见解使用户集中精力于设计的特定方向（如，那些仅与逻辑显示和规划布局有并的信息将被显示），一个数据包是一个逻辑单元（如算术逻辑单元），这个逻辑单是正在设计的几个组件的抽象。它允许诸如由不同工具产生的细节数据被隐蔽并在需要时获得。在效果上，**POWERFRAME** 把此摘要的所有相关信息分类。一部份对象由某版本控制，其它的在检测中确定其版本。

L. Rational

RATIONAL 的环境体支持开发大型 **ADA** 产品的软件人员组。**RATIONAL** 的计算机管理设备依赖于其子系统。**ADA** 程序库与它们的计算机管理系统交互相关，一个子系统代表 **ADA** 产品的一部份。子系统可以独立于产品的其它部份仅由一个软件工程师开发或者由一个工作组协同完成。一个子系统有一个版本标识符，可以被释放回收。不同版本的子系统可以同时操作，其差异被合并，子系统之间也可以进行合并。通过活动桌面可以分辨出哪些子系统的哪些版本要进行合并。

RATIONAL 提供对 **ADA** 单元重编译最小化的机制。通过子系统 **ADA** 单元被放置在版本控制器中。用户可以根据设计需要开启，关闭版本控制器。

M. Revision Control System(RCS)

修订控制系统（**RCS**）是一套由 **W.Tichy** 开发的，库里的源文件提供版本控制器的工具。库对每个文件建立一棵版本树。树上的一个分支

代表文件里的一个变量。RCS 对版本和分支的操作自带一套计数方案，为了节省空间并且尽快获取最近的版本，我们只存储文件版本之间的差异。获取文件库的通常使用模式包括用户检索库文件的特定版次（通过锁定方式），修改文件。修改完毕后登记回原版本所在的出处。与此同时，RCS 会记录修改的细节，如作者，日期，时间和修改原因。如果需要，RCS 可以自动将一个特有的标识放入文件。RCS 能对比文件的不同版本，终止一项配置以及通过识别源代码行的差别合并各个分支。库文件标志（如配置标志或状态标志）可以用于标识文件之间的关连。

N. SHAPE

SHAPE 系统来自柏林大学，它借助版本状态，配置标识符为我们提供一个带有特定文件系统，版本控制器和工作间的库。它集成系统模型设备并从中获取二进制代码池。我们可以通过用户定义/系统定义的属性模式描述各项配置。串行和并行的配置版本均支持开发组件。工作间由版本的状态量激活。此版本还可以确定文件的不稳定性。工作间文件的状态值“忙”“已保存”“激活”以及公用办公数据库使用的状态值“已打印”“完成”和“终止”相互转换。

O. Software Management System

软件管理系统（SMS），提供版本控制，工作区管理，系统模拟，。。。改变库探测方式，对接口说明书进行加工，以及对基于属性的版本区进行加工。工作于与任务相关的版本时，工作区提供保护措施并支持对每个任何基底的认证和登陆。

一旦特殊事件发生，物件的变动就受到监控和跟踪。已获取的物件有一个连续状态量，（“合法”“受保护”“废止”“非法”）用来代表与已

构成系统的关系；此物件还带有一个程度状态量（“同意”“警告”“严重错误”），用来指明版本的一致性。

7．配置管理的商业模型

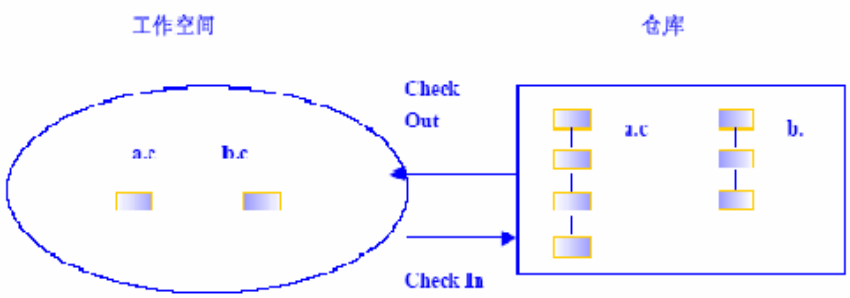
配置管理的实施包括两部分：工具和规范。

在软件开发过程自动化的今天，没有工具的支持而实施配置完整的配置管理是不能想象的。因此选择一个符合公司或项目的工具至关重要。在配置管理系统中，我们可归纳出四种模型。当前商业工具一般采用其中一种或几种模型。我们通过对商业模型的理解可以帮助我们了解某种工具是否适合我们公司或项目。

CICO 模型

CICO 模型主要关注的是单个文件的版本控制。图显示了一个支持CICO 模型的CM 系统的工作过程。用户利用库和文件系统来进行工作。文件被版本化并存储到库中，新版本的产生是由库工具控制的。然而， 文件在库中不是可以直接存取的，用户必须去检出（即Check Out）一个文件的版本到工作空间中以便读取它的内容。更改后的文件可以被检入库中（即Check in）,产生文件的一个新版本。

此模型的代表工具是SCCS 和CVS。



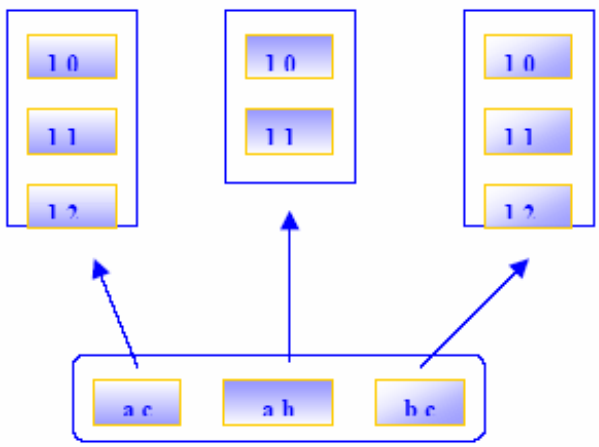
组织模型

组织模型由CICO 模型自然导出，建立于构件版本图的基础之上，同时依赖于存储库和工作空间的概念，可以通过对构件加锁进行并发控制。组织模型的重点是在CM 系统支撑下加强了对创建配置、对有关的历史信息的管理和使用他们作为工作环境的支持。组织模型中的配置由系统模型和版本选择规则组成。系统模型列出了组成系统的所有的构件。版本选择规则指出了组成配置的每一个构件选择版本。选择规

则用于系统模型，选择构件版本，即绑定一构件到某一版本。这个模型的操作方式是：开发员根据模型的构件定义整个系统，并在每一步骤中给每个构件选择合适的版本。版本操作的工作方式如图所示。CM 支持主要关心的是维护系统和其构件的版本历史，并选择符合一致性配

置的构件版本。只有在所选构件的版本与所选其它构件版本一致时才认为一个配置版本。

此模型的代表工具是CCC。

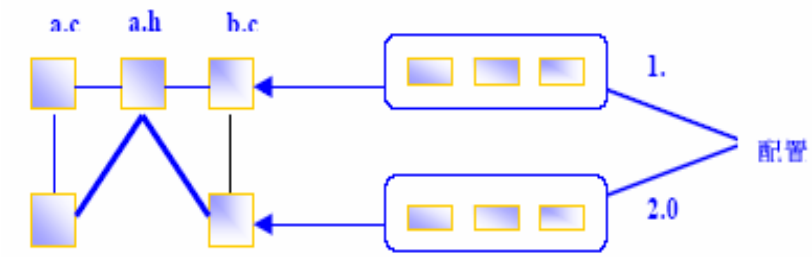


长事务模型

长事务模型主要支持包括一系列原子变更的全系统演变和由团队开发员对系统变更的协调。开发员主要操作配置而非单独的构件。事务提交的结果是新配置版本，一系列连续的变更结果生成一系列的配置版本，称为开发路径。

在长事务模型中，开发员主要的工作对象时配置，开发员首先选择系统配置版本，接下来把关注重点放在系统结构上。构件的版本由配置隐式决定。长事务由两个概念组成：工作空间和并发控制方案。工作空间来源于存储库或一个封闭工作空间中的一个固定配置。工作空间由工作配置和一系列已保存的配置组成。工作配置代表构件和系统结构能够被动态更改的配置。提供通过工作空间进行的透明库访问、将高效的库存储技术应用于工作空间和管理派生构件的版本。

此模型的代表系统是NSE。



变更集模型

主要集中于对系统配置的逻辑变更的支持。在这个模型中引入的变更集表示组成逻辑变更的对不同构件修改的集合，它是创建变更的活动完成后对逻辑变更的记录。支持这个模型的CM 系统用户可以直接操作

变更集。在变更集模型中，配置可描述为由基线和一组变更集组成。

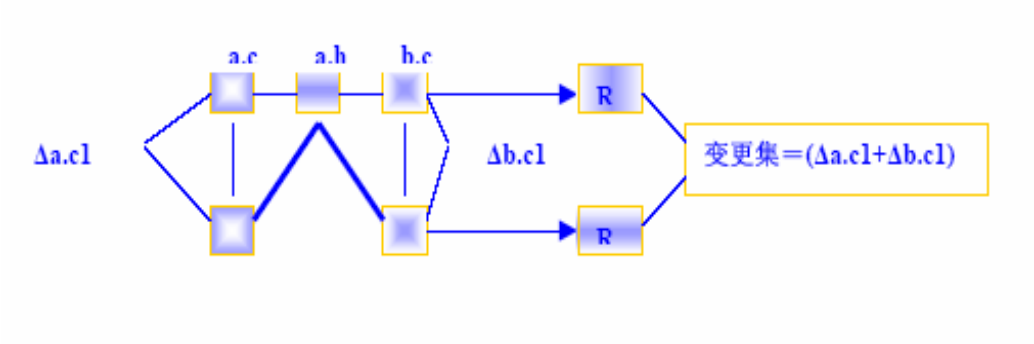
变更传播给其它配置可通过包含各自变更集来进行。开发员使用不同的集成策略将逻辑变更集包含到一个新的系统发行中。这样的好处非常明显，例如，我们现在维护10 个不同版本的产品，现在要对所有的版本修改一个缺陷（Bug）。如果相同的工具简单的重复10 次显然是不可接受的。而通过变更集把这个逻辑变更从一个版本自由的传到另外一个版本。

开发员可跟踪逻辑变更和确定这些变更是否属于特定配置。这种配置管理的方法，因为其将重点放于逻辑变更上，所以被称作面向变更的配置管理。它不同于现在的其他3 种CM 模型，因为其它3 种CM 模型使用的面向版本的方法把重点放在构件和配置版本上。

在单一构件的情况下，变更集是两个文件版本之间区别的集合，通常指的是增量内容。对配置来说，变更集就是两个配置版本之间区别的集合。这组区别就是两个配置版本间的修改构件增量集合，即变更构件集的增量。

面向变更的观点不同于面向版本的观点。这有两点不同，一是逻辑变更的显式表示允许对与单个构件和配置有关的变更集进行跟踪。二是引用单个变更集并有选择地将它们纳入配置管理中的这种能力提供了对系统演化管理的支持，这种演化是基于将逻辑变更传播到维护的系统配置进行的。

此模型的代表工具是UCM 和SABLIME



第五章　配置管理工具评估/选择过程

1　SCM tool 比较

VSS

SourceSafe是Microsoft公司推出的配置管理工具，是Visual Studio的套件之一。SourceSafe是国内最流行的配置管理工具，用户量绝对是第一位。

SourceSafe长得很象早先土气的文件管理器，的确难看。但是难看不碍事，SourceSafe的优点可以用8个字来概括“简单易用，一学就会”，这个优点是它老妈Microsoft遗传下来的，是天生的。

虽然SourceSafe并不是免费的，但是在国内人们以接近于零的成本得到它，网上到处可以下载啊。当然Microsoft也不在乎这个小不点的软件，它属于“买大件送小件”的角色。如果你合法地得到Visual Studio，你就得到了免费的SourceSafe。

SourceSafe的主要局限性：

只能在Windows下运行，不能在Unix, Linux下运行。SourceSafe不支持异构环境下的配置管理，对用户而言是个麻烦事。这不是技术问题，是微软公司产品战略决定的。

适合于局域网内的用户群，不适合于通过Internet连接的用户群，因为SourceSafe是通过“共享目录”方式存储文件的。

人无完人，物不尽美。有些卖配置管理工具的软件供应商经常贬低SoureSafe，讽刺它是Source not Safe。我不想为谁辩护，只是给出一个例证说明SourceSafe的效用。有一个软件事业部（约百名开发人员）的十余个项目全部采用SourceSafe来管理，只用一台PC机作配置管理服务器，运行一年都没有发生异常现象。

CVS

CVS 是 Concurrent Version System（并行版本系统）的缩写，它是著名的开放源代码的配置管理工具。

CVS的官方网站是http://www.cvshome.org/ 。官方提供的是CVS服务器和命令行程序，但是官方并不提供交互式的客户端软件。许多软件机构根据CVS官方提供的编程接口开发了各色各样的CVS客户端软件，最有名的当推Windows环境的CVS客户端软件--WinCVS。WinCVS是免费的，但是并不开放源代码。

与SourceSafe相比，CVS的主要优点是：

SourceSafe有的功能CVS全都有，CVS支持并发的版本管理，SourceSafe没有并发功能。CVS服务器的功能和性能都比SourceSafe高出一筹。

CVS服务器是用Java编写的，可以在任何操作系统和网络环境下运行。CVS深受Unix和Linux 的用户喜爱。Borland公司的JBuilder提供了CVS的插件，Java程序员可以在JBuilder集成环境中使用CVS进行版本控制。

CVS服务器有自己专用的数据库，文件存储并不采用SourceSafe的“共享目录”方式，所以不受限于局域网，信息安全性很好。

CVS的主要缺点在于客户端软件，真可谓五花八门、良莠不齐。Unix和Linux 的软件高手可以直接使用CVS命令行程序，而Windows用户通常使用WinCVS。安装和使用WinCVS显然比SourceSafe麻烦不少，这是令人比较遗憾的。

ClearCase

Rational公司的ClearCase是软件行业公认的功能最强大、价格最昂贵的配置管理软件。

ClearCase主要应用于复杂产品的并行开发、发布和维护，其功能划分为四个范畴：版本控制、工作空间管理（Workspace Management）、构造管理（Build Management）、过程控制（Process Control）。ClearCase通过TCP/IP来连接客户端和服务器。另外，ClearCase拥有的浮动License可以跨越UNIX和Windows NT平台被共享。

ClearCase的功能比CVS、SourceSafe强大得多，但是其用户量却远不如CVS、SourceSafe的多。主要原因是：

ClearCase价格昂贵，如果没有批量折扣的话，每个License大约5000美元。对于中国用户而言，这无疑是天价。

用户只有经过几天的培训后（费用同样很昂贵），才能正常使用ClearCase。如果不参加培训的话，用户基本上不可能无师自通。

几点补充：

1.ClearCase既不是最贵的也不是功能最强的配置管理软件

至少PVCS Dimensions（不是PVCS VM啊，那个巨烂）就比ClearCase

功能强大，且贵不少

2.VSS不是微软的产品，是微软收购的产品

VSS最初的名字叫**Source Safe,**是一家小公司的产品，**92**年曾经获了最佳小型管理工具奖，然后立即被微软收购

但是微软收购的只是**source safe**的**Windows**版本，在美国还有另外两家公司分别获得了继续开发和销售**source safe**的**Mac**版本和**Unix**版本的许可。

他们仍然在销售**mac**和**unix**版本的**source safe,**根据当时的协议，三家公司的软件始终是保持兼容的。

此外，在**ms**买进**vss**之后，基本上没有对**vss**进行任何的研发，**ms**内部自身也不用**vss**

3.cvs的服务器软件不是用**java**开发的

cvs的历史要比**java**的历史久远的多，严格意义上**cvs**并没有真正意义上的服务器

pserver/ntserver之类的类服务器模式实际上都只是完成用户鉴别权限的工作

4.ClearCase是依赖与文件共享的

最早的**ClearCase**并不是**rational**的产品，而是一家叫做**applo**的公司在**HP**平台上开发的一套配置管理系统**DTEMS(**好像是这个名字**)**。**ClearCase**的核心是基于**NFS**的一套称作**MFS**的文件系统。后来**applo**几次转手倒卖给了**rational.rational**把它移植到了**windows**平台上，但是这个底层的架构至今没有发生变化。也就是说**clearCase**和**vss**一样是基于网络文件共享的。

目前配置管理工具可以分为**3**个级别**--**

第一个级别为--简单的版本控制工具，是入门级的工具，例如:CVS, Visual Source Safe;

第二个级别为--项目级配置管理工具，适合管理中小型的项目，例如：PVCS, MKS;

第三个级别为--企业级配置管理工具，具有强大的过程管理功能，例如：CCC Harvest; ClearCase。

CC： 价高，狂大，安全性不好，功能不错，管理复杂

CVS：免费，功能不全

VSS：不安全，功能太少

JBCM：低价，功能很弱，性能不好，管理、使用简单

Firefly：价稍高，功能不错，安全性好，管理方便，上手快

PVCS Professional：便宜，功能弱，不安全

PVCS Dimension：价高，功能全，管理复杂

StarTeam：价高，功能不全，支持差

CCC/Harvest：价高，功能一般，安全性不错，管理巨复杂

2 如何选择配置管理工具

每一个软件项目，无论是工程类项目，还是产品类项目，都必须经历需求分析、系统设计、编码实现、集成测试、部署、交付、维护和支持的过程。在这个过程中，将生成各种各样不同的工件，包括文档、源程序、可执行代码、支持库。更可怕的是，频繁出现的变更是不可避免的，因此面向如此庞大且不断变动的信息集，如何使其有序、高效地存放、查找和利用就成为了一个突出的问题。

针对这一问题，最早的开发人员尝试过的解决办法是通过手工来实

现：

1)文档：每次修改时都另存为一个新的文件，然后通过文件名进行区分，例如"XXX 软件需求说明书 V1.0，XXX 软件需求说明书 V1.1，XXX 软件需求说明书 V2.0."，并且在文件中注明每次版本变化的内容；

2) 源代码：每次要修改时就将整个工程目录复制一份，将原来的文件夹进行改名，例如"XX 项目 V1.0、XX 项目 1.01、."，然后在新的目录中进行修改；

但是这种方法，不仅十分繁琐，容易出错，而且会带来大量的垃圾数据。如果是团队协同开发或者是项目规模较大时，还是会造成很大的混乱。很显然，这样简陋的方法是无法应对这一问题的。

后来，有人尝试从制造工业领域引入了"配置管理"这一概念，通过不懈的研究与实践，最终形成了一套管理办法和活动原则，这也就是软件配置管理。

通过软件配置管理，将对软件系统中的多重版本实施系统的管理；全面记载系统开发的历史过程，包括为什么修改，谁作了修改，修改了什么；管理和追踪开发过程中危害软件质量以及影响开发周期的缺陷和变化。并对开发过程进行有效地管理和控制，完整、明确地记载开发过程中的历史变更，形成规范化的文档，不仅使日后的维护和升级得到保证，而且更重要的是，这还会保护宝贵的代码资源，积累软件财富，提高软件重用率，加快投资回报。

正如前面所述，由于软件配置管理过程十分繁杂，管理对象错综复杂，如果是采用人工的办法不仅费时费力，还容易出错，产生大量的废品。因此，引入一些自动化工具是十分有裨益的，这也是做好配置管理的必要条件。

正是因为如此，市场上出现了大量的自动化配置管理工具，这些工具的实现原理与基本机制均十分接近，但由于其定位不同，因此各有特点，下面我们就对一些常见的配置管理工具做一简单的介绍。

元老：CCC、SCCS、RCS

上个世纪七十年代初期加利福利亚大学的 Leon Presser 教授撰写了一篇论文，提出控制变更和配置的概念，之后在 1975 年，他成立了一家名为 SoftTool 的公司，开发了自己的配置管理工具：CCC，这也是最早的配置管理工具之一。

在软件配置管理工具发展史上，继 CCC 之后，最具有里程碑式的是两个自由软件：Marc Rochkind 的 SCCS (Source Code Control System) 和 Walter Tichy 的 RCS (Revision Control System)，它们对配置管理工具的发展做出了重大的贡献，直到现在绝大多数配置管理工具基本上都源于它们的设计思想和体系架构。

中坚：Rational ClearCase

Rational 公司是全球最大的软件 CASE 工具提供商，现已被 IBM 收购。也许是受到其拳头产品、可视化建模第一工具 Rose 的影响，它开发的配置管理工具 ClearCase 也是深受用户的喜爱，是现在应用面最广的企业级、跨平台的配置管理工具之一。

ClearCase 提供了比较全面的配置管理支持，其中包括版本控制、工作空间管理、Build 管理等，而且开发人员无需针对其改变现有的环境、工具和工作方式。

其最大的缺点就在于其价格不菲，每个客户端用户许可证大约需要几千美金，所以在国内应用群体有限。

1） 版本控制

ClearCase 不仅可以对文件、目录、链接进行版本控制，同时还提供了先进的版本分支和归本功能用于支持并行开发。另外，它还支持广泛的文件类型。

2）工作空间管理

可以为开发人员提供私人存储区，同时可以实现成员之间的信息共享，从而为每一位开发人员提供一致、灵活、可重用的工作空间域。

3） Build 管理

对 ClearCase 控制的数据，既可以使用定制脚本，也可使用本机提供的 make 程序。

其最大的缺点就在于其价格不菲，每个客户端用户许可证大约需要几千美金，所以在国内应用群体有限。

新秀：Hansky Firefly

做为 H a n s k y 公司软件开发管理套件中重要一员的 Firefly，可以轻松管理、维护整个企业的软件资产，包括程序代码和相关文档。Firefly 是一个功能完善、运行速度极快的软件配置管理系统，可以支持不同的操作系统和多种集成开发环境，因此它能在整个企业中的不同团队，不同项目中得以应用。

Firefly 基于真正的客户机/服务器体系结构，不依赖于任何特殊的网络文件系统，可以平滑地运行在不同的 LAN、WAN 环境中。它的安装配置过程简单易用，Firefly 可以自动、安全地保存代码的每一次变化内容，避免代码被无意中覆盖、修改。项目管理人员使用 Firefly 可以有效地组织开发力量进行并行开发和管理项目中各阶段点的各种资源，使得产品发布易于管理；并可以快速地回溯到任一历史版本。系统管理员使用 Firefly 的内置工具可以方便的进行存储库的备份和恢复，而不依赖于

任何第三方工具。

开源奇葩：CVS

CVS 是 Concurrent Versions System 的缩写，它是开放源代码软件世界的一个伟大杰作，由于其简单易用、功能强大，跨平台，支持并发版本控制，而且免费，它在全球中小型软件企业中得到了广泛使用。

其最大的遗憾就是缺少相应的技术支持，许多问题的解决需要自己寻找资料，甚至是读源代码。

小工作组级：Merant PVCS

MERANT 公司的 PVCS 能够提供对软件配置管理的基本支持，通过使用其图形界面或类似 SCCS 的命令，能够基本满足小型项目开发的配置管理需求。PVCS 虽然功能上也基本能够满足需求，但是其性能表现一直较差，逐渐地被市场所冷落。

入门级：Microsoft Visual Source Safe

Visual Source Safe，即 VSS，是微软公司为 Visual Studio 配套开发的一个小型的配置管理工具，准确来说，它仅能够称得上是一个小型的版本控制软件。VSS 的优点在于其与 Visual Studio 实现了无缝集成，使用简单。提供了历史版本记录、修改控制、文件比较、日志等基本功能。

但其缺点也是十分明显的，只支持 Windows 平台，不支持并行开发，通过 Check out - Modify - Check in 的管理方式，一个时间只允许一个人修改代码，而且速度慢、伸缩性差，不支持异地开发。甚至于微软本身也不采用其做为配置管理工具，而是使用一个名为 SLM 的内部工具。

面对这些形形色色，各有千秋的配置管理工具，如何根据组织特点、开发团队需要，选择切合适用的工具呢？笔者就结合工作实践中的经验与大家做一些交流与探讨。

配置管理工具的选择所需考虑的因素大体包括以下几个因素：

功能是否符合实际需求？是否符合团队特点？性能是否满意？费用是否可以接受？售后服务如何？接下来，我们就这几方面逐一深入地探讨：

1）功能是否符合实际需求，是否符合团队特点

工具就是用来帮助您解决问题的，因此功能是否符合实际需求是最重要的判断因素。而大多数主流配置管理工具的基本功能都能够满足，因此主要需要判断以下几个因素：

并行开发支持

在团队协作开发过程中，有两种主要的模式：集体代码权和个体代码权。采用集体代码权模式进行开发时，一段代码可能同时会被多个开发人员同时修改；而采用个体代码权模式进行开发时，每一段代码都始终被一个开发人员独享，别人需要修改时也会通过该开发人员完成。

而配置管理软件针对这一情况，也采用了不同的策略：Copy-Modify-Merge(拷贝、修改、合并)的并行开发模式、Check out-Modify-Check in（签出、修改、签入）的独占开发模式。在并行开发模式下，开发人员可以并行开发、更改代码，Firefly 会自动检测到代码冲突，并自动合并，或提示开发人员手动解决。

表一　并行开发支持比较表

工具名称	说明
ClearCase	Copy-Modify-Merge 模式
Firefly	Copy-Modify-Merge 模式
CVS	Copy-Modify-Merge 模式

PVCS	Check out-Modify-Check in 模式
VSS	Check out-Modify-Check in 模式

异地开发支持

如果你的开发团队分布在不同的开发地点，就需要对工具的异地开发功能进行仔细的评估了。大多数工具都提供基于 Web 的界面，用户可以通过浏览器执行配置管理的相关操作，而且有些工具就通过这样的方法来实现对异地开发的支持。

这种实现方法有太多的局限性，例如网络（Internet）连接带宽的限制、防火墙以及安全问题等。真正意义上的异地开发支持，是指在不同的开发地点建立各自的存储库，通过工具提供同步功能自动或手动同步。这样做的好处是与网络无关，即便各个开发地点之间没有实时连通的网络，也可以通过 E-Mail 附件等其它方式将同步包发给对方，实现手动的同步。

表二　异地开发支持比较表

工具名称	说明
ClearCase	提供 MultiSite 模块，通过自动或手动同步位于不同开发地点的存储库的方式，支持异地开发
Firefly	提供 ServerSync 模块，通过自动或手动同步位于不同开发地点的存储库的方式，支持异地开发
CVS	无专门支持的模块
PVCS	无专门支持的模块
VSS	无专门支持的模块

值得说明的是，在不同开发点建立各自存储库的方式，主要适

用于两个或两个以上位于不同地点的开发团队协作开发的情况。如果仅是采用虚拟团队合作的方式，开发人员以个体的形式散落在不同地方，则更适合通过 Internet 直接操作远程的配置管理服务器。

跨平台开发支持

如果企业需要从事多个不同平台下的开发工作，就需要配置管理工具能够对跨平台开发提供支持，否则势必会给开发、测试、发布等各个环节带来不便，将使大量的时间被浪费于代码的手工上传、下载中。

表三　跨平台开发支持比较表

工具名称	说明
ClearCase	支持常见的平台
Firefly	软件本身基于 Java 开发，可在 Windows、Linux、Solaris、HP-UX、AIX 等常见平台上使用，平台之间的移植也非常方便
CVS	支持几乎所有的操作系统
PVCS	软件本身基于 Java 开发，能够支持常见的平台
VSS	仅支持 Windows 操作系统

与开发工具的集成性

配置管理工具与开发工具是编码过程中最常用到两种工具，因此它们之间的集成性直接影响到开发人员的便利性，如果无法良好集成，开发人员将不可避免地在配置管理工具与开发工具之间来回切换。

表四　与开发工具集成性比较表

工具名称	说明
ClearCase	直接与资源管理器集成，十分易用

Firefly	与常见开发工具无缝集成
CVS	对开发工具集成性较差
PVCS	仅支持 Windows 操作系统
VSS	与 Visual Studio 开发工具包无缝连接，其它开发工具集成性差

2）性能是否满意

配置管理工具软件的一些性能指标对于最终的选择也有着至关重要的影响。

运行性能

如果开发团队规模不大的情况下，配置管理工具软件的性能不会造成很大影响，但如果项目规模比较大，团队成员逐渐增多的情况下，其运行性能就会带来很大的影响。

表五　运行性能比较表

工具名称	说明
ClearCase	服务器采用多进程机制，使用自带多版本文件系统 MVFS，对性能有较大负面影响。做为一款企业级、全面的开发配置管理工具，适用于大型开发团队
Firefly	服务器采用了多线程的应用服务器，性能表现优秀，做为一款企业级、全面的开发配置管理，能适用于 50 人到上千人的团队
CVS	较高的运行性能，适用于各种级别的开发团队
PVCS	服务器采用文件系统共享方式，对 CPU、内存及网络要求较高，性能一般，仅适用于中小型项目团队，不适合

	于企业级应用
VSS	相对功能单一、简陋，适用于几个人的小型团队，在数据量不大的情况下，性能可以接受

易用性

表六　易用性比较表

工具名称	说明
ClearCase	安装、配置、使用相对较复杂，需要进行团队培训
Firefly	在提供全面配置管理功能的情况下，安装、配置、使用较为简单，包括安装、配置、培训在内的整个实施周期一般不会超过一个月。
CVS	安装、配置较复杂，但使用比较简单，只需对配置管理做简单培训即可
PVCS	使用比较简单，只需对配置管理做简单培训即可
VSS	安装、配置、使用均较简单，很容易上手使用

从用户界面、与开发工具的集成性角度来说，这几款主流的配置管理软件均有较好的设计，均有较好的易用性。

安全性

表七　安全性比较表

工具名称	说明
ClearCase	采用 C/S 模式，需要共享服务器上的存储目录以供客户端访问，这将带来一定安全隐患
Firefly	服务器上的存储目录不用共享，对客户端不透明，客户端不可直接访问存储目录，使系统更安全可靠

CVS	采用 C/S 模式，不需要共享服务器上的存储目录，安全性较好
PVCS	基于文件系统共享，而且需要以"可写"的权限共享存储目录，存在较大的安全隐患
VSS	基于文件系统共享实现对服务器的访问，需要共享存储目录，这将带来一定安全隐患

3）费用是否可以接受

Rational ClearCase、Hansky Firefly 两款均属于企业级配置管理工具软件,ClearCase 价格较贵，，相比之下 Hansky Firefly 是一款不错的选择。

而 PVCS 其价格大约是每客户端几百美元的水平，对于国内企业来说，性价比不太划算。VSS 是微软打包在 Visual Studio 开发工具包之中的，显然花费的精力不大，价格也比较便宜，可以做为个人、小项目团队版本控制之用。

而 CVS 则是一款完全免费的开源软件，性能较之企业级配置管理工具差距不大，也是一种不错的选择。

4） 售后服务如何

表八售后服务比较表

工具名称	说明
ClearCase	大型商用软件，已被 IBM 公司收购，但国内市场拓展有限，因此服务支持会受到限制。现在中国用户的支持是由位于澳大利亚悉尼的支持中心联系
Firefly	大型商用软件，已在中国成立分公司，全面拓展市场之中，在北京设有支持中心

CVS	做为开源软件，无官方支持，需要用户自己查找资料解决技术问题，现在也出现专门为 CVS 做技术支持的公司
PVCS	在中国市场开拓有限，国内没有支持中心
VSS	做为微软的非核心产品，技术支持有限。在其网站上有提供一些常见问题，只有对正式购买的用户提供一定的技术支持

售后服务与产品支持也是一个很重要的考察点，工具在使用过程中出现这样那样的问题是很平常的事，有些是因为使用不当，有些则是工具本身的缺陷。这些问题都会直接影响到开发团队的使用，因此随时能够找到专业技术人员解决这些问题就变成十分重要。

实例说明

最后，笔者介绍几个实际的案例，希望对大家选择软件配置管理工具软件有帮助。

案例一

某公司拥有 10 名专职开发人员以及一些兼职的开发人员，主要从事 Windows 和 Linux 平台下的软件开发，采用的工具包括 Visual Studio 系列、GCC 等。为了能够加强版本控制与配置管理工作，决定引入一些自动化配置管理工具。

经过慎重的选择，采用了两步走的方法：

1） 首先采用了 Visual Studio 软件包中的 VSS 做为配置管理工具；由于 VSS 安装、配置、操作都十分简单，上手容易，这样在执行配

置管理的过程中，工具的培训没有带来太大的阻力，大家可以集中精力理解配置管理。这样很快就在团队中形成了版本控制、配置管理的氛围与习惯。

2） 然后构建了 CVS 服务器，做为整个开发组织的配置管理工具；

CVS 能够有效地支援 Windows、Linux 两个平台上的应用开发，其性能优秀，而且免费，另外，它对于兼职人员的配置管理十分有效。采用 CVS 至今，效果明显，除了功能、使用上有些不方便之处外，没有出过任何大问题。

案例二

北京某公司拥用 230 名专职开发人员，长期从事金融业务的开发，随着业务的良性发展，在管理上也出现了一些不足：

1） 开发管理沟通滞后，开发人员孤立操作，变更和维护信息无法实时反馈；

2） 主管领导对所开发的 100 多种产品的项目开发进程不能及时了解，很多资源滞留在个人手中；

3） 随着产品的需求日益增加，无法快速标识和查找软件的历史版本；

4） 无法对处于不同开发平台上的项目进行统一管理和资源配置；

5） 无法实现异地开发团队的协调和沟通。

因此，该公司决定引入软件配置管理，在配置管理工具软件的选择上，考虑到其人员规模较大，项目较多，工作复杂，在针对可靠性、易用性、稳定性、安全性、技术支持能力以及软件的各功能进行了仔细的综合评估后，最后选择了国内技术支持较到位的 Hansky Firefly 软件配置工具软件。

在采用了 Hansky Firefly 之后，有效地解决了这些问题，还帮助其顺利地通过了 CMM 2 级认证，为企业的进一步发展打下了坚实的基础。

第六章　实用配置管理系统及工具

JBCM

青鸟软件配置管理系统 JBCM

（JadeBird Software Configuration Management）

JBCM 是北京大学软件工程国家工程研究中心（北京北大青鸟软件工程有限公司）在国家重点科技攻关项目——青鸟工程的成果，是在杨芙清院士的亲自指导下研制、开发的，是一种新一代的基于构件的软件配置管理工具。作为青鸟软件生产线的重要组成部分，JBCM 的研制采用了新一代的软件配置管理模型——大粒度、构造性的基于构件的配置管理模型，面向软件企业开发、管理相关资源的需求，符合 ISO9000-3 和 SEI CMM 软件质量保证体系的要求，支持基于构件的软件开发方法，可以有效地改善软件企业的开发过程。JBCM 是中国软件企业通往 ISO9000-3 和 SEI CMM 标准的最佳实施工具。JBCM 系统结合了领先的构件技术和软件开发过程中质量管理的精髓，将给软件开发带来更全面的管理和更高的效率，保证软件质量，最大限度地保护企业的知识财富。

要使一个企业的软件开发管理水平达到较高的层次，单单靠拥有 CASE 工具是不够的。因为每一套 CASE 工具都是一种软件工程管理的思想方法的具体支撑平台，只有深刻理解蕴含在这些 CASE 工具中的软件工程管理的思想方法，并按照这些思想方法进行日常的软件开发工作，CASE

工具才能起到它应有作用，使团队的软件工程达到一个很高的水平。因此，本公司不但注重 CASE 工具的研制开发工作，而且更重视对用户的专业咨询服务工作和软件工程管理思想的推广工作。

JBCM 产品系列

JBCM 系统采用 Client/Server 结构，建立在 Windows 平台上，面向软件企业不同层次的开发管理需求，分为两个产品序列：

Ø项目组级系统（JBCM/Team）

Ø企业级系统（JBCM/Enterprise）

JBCM/T 是面向项目小组级开发、管理需求的配置管理系统。JBCM/T 能够满足 ISO9000-3 和 CMM 2 级中的配置管理要求。通过 JBCM/T 可以有效地进行版本管理、配置管理、人员与权限控制、审计和统计等工作，并提供对团队中并行开发的支持功能。

JBCM/E 是面向软件企业级别开发、管理需求的配置管理系统。JBCM/E 可以全面支持企业实现 CMM 2 与 ISO9000-3 质量管理中的配置管理要求，并基本满足 CMM 3 级中对软件过程管理的需求。JBCM/E 可以完整地存储、管理大型企业内部资源及其重要关系，改善软件企业的开发过程。通过分布、多层次的结构，可在 JBCM/T 系统基础之上建立起覆盖企业范围的全面、完善、统一的资源管理系统。

（青鸟软件配置管理系统 JBCM 作为资源管理的基础，与青鸟软件变化管理系统 JBCCM、青鸟软件过程管理系统 JBPM 共同构成了青鸟软件项目管理支撑工具体系。）

JBCM 软件配置管理全面解决方案

1 JBCM 的功能特点：

JBCM 系统是基于构件的软件配置管理思想,采用先进的软件构件模型，并与现代工程管理方法融为一体的一套软件配置管理工具。它具有以下特点：

提升管理粒度，适应现代大规模、分布式、多层次的软件系统开发与维护

支持基于构件复用的软件工业化生产技术，工程化开发方法

具有完善、灵活的版本控制能力，通过可视化的版本树管理功能，对软件系统的不同演化方向提供管理支持，适应企业对需求变化的管理要求

提供强有力的人员管理与权限控制机制，确保资源安全

高效支持团队并行开发，可支持中长期同步的协作工作模式和紧密结合的短期同步协作模式(构件/文件合并功能)

具备审计与统计功能，可检索、管理系统和资源的日志和各种信息

2、实施与服务

虽然大部分 SCM 系统功能能解决大部分开发过程中出现的软件资源管理问题，如果客户购买到产品的不能正确的使用，理解不了系统中蕴含的管理思想，那么，SCM 就发挥不出它应有的作用，这样，用户就等于白白浪费了一笔资金。

因此，为了让用户的投资得到应有的回报，我们不仅仅是推广产品，而且一直把服务放在最重要的位置。基于这种原因，我们在售出产品的同时，附带一套专门针对配置管理实施，可以帮助客户理解配置管理思想，

熟练使用工具，制定计划，并制订详细实施过程的专业咨询服务。

我们的配置管理专业服务，目的在于指导并帮助用户根据先进的配置管理思想方法，去管理他们的开发过程，改进现有的管理水平。我们所提供的不只是 JBCM 这套配置管理工具的使用说明，而是青鸟软件工程有限公司基于对国内软件开发管理现状的调查研究，融入了国内外成功的软件开发管理经验、以及很多早期用户和自身的使用经验而总结出的基于 JBCM 的管理方法和过程。

根据用户的不同要求，我们可以电话服务或派出专业配置管理咨询人员，同软件开发组织指定的管理员一起，针对软件企业现有状况，如管理组织结构、角色责任、工作流程等进行交互，评估分析并确定其配置管理需求，然后据此需求，为软件开发组织制订可行的配置管理目标，编制符合组织现实状况的配置管理计划。

为了企业能够更好的全面实施配置管理，通常先组织实施一个实验项目，然后通过这个项目的实施，总结经验教训，然后根据这些经验全面实施配置管理，在实验过程中，我们可以提供以下帮助：

协助企业选择实验项目；
协助企业确定试验组成员；
协助企业定义试验成功的标准和时间安排；
对相关人员进行培训；
安装配置管理工具；
协助企业建立配置管理过程；
和企业试验组成员共同总结经验教训。

对于还没有实施配置管理的软件开发团队，通过我们提供的软件配置管理专业化的咨询服务，可以逐步建立起自己的配置管理体制，并逐步改善；对于已经实施配置管理的团队，也可以通过我们的服务来改进、完善自己的管理体制。

实施配置管理的效果

软件组织如果认真地实施配置管理，将会有效地保证软件产品的完整性、一致性和可控性，使其产品最大程度的与用户的需求吻合；

加快产品开发速度，降低维护成本，更好地保证软件质量；

可为软件组织节省大量的人力和物力，有助于规范工作流程，从资源方面更好地掌握项目的进度，了解开发人员的负荷、工作效率和产品质量状况、交付日期等信息；

管理者能够掌握企业的整体软件资源，减轻人员流动对开发过程的影响；

新老成员可以更加快速地实现任务交接，减少因人员流动带来的损失。

具体说来，可以为软件组织带来以下好处：

构件版本得到很好的控制

所有软件开发过程中相关的历史信息和有用版本的全部内容（包括：文档、代码，以及相对应的文件）都被保留，不会出现版本丢失；

并行开发井然有序，甚至不同的人对同一个文件进行同时更新，也不会出现丢失信息；

建立应用时构件取用方便、快捷、准确，不会出现错乱、混杂的现象；

所有代码都经过受控的途径进入产品，产品中不会出现未经确认的代码；

保证发布的产品中应用程序与配套文档一致；
构造系统时能够迅速得到精确的配置项。

配置支持使项目更易于管理
基线的建立，使得软件开发具有更好的阶段可控性；
配置管理的状态统计功能，使得软件资源状态更具有可见性。
可方便的管理同时进行的多个项目，以及相关的资源；

所有开发人员对资源的访问都得到有效的管理
保证开发和管理人员受控地获取所需要的软件资源；
支持分层的人员权限结构对资源的访问；
控制不同人员对资源访问的信息权限、可读权限和修改权限；
每个人的工作过程及相应资料（包括：源代码、文档，重要步骤的理解等），都会被详细的纪录并保存下来，有效的避免了由于开发人员的流动对工作造成的影响。

有效的审计和统计
只有批准实施的变更才会进入受控软件生产过程；
收集和统计各种信息，并实现对信息的查询和分析，便于项目管理人员跟踪、分析和决策。

ClearCase

1 Rational ClearCase 介绍

功能简介：

- 提供版本控制、工作区管理、Build管理及流程管理

• 提供分布式、跨区域的并行开发模式

• 可以和Microsoft Developer Studio,Powerbuilder,Oracle Developer2000集成

• 提供离线模式，让用户可以在家工作，然后合并到开发流程中

• 提供深入的build内核

• 对执行文件和目录进行自动图形化合并，文件间的差异明显显示

• 完整控制程序源代码、二进制码、执行码、测试项目、文档以及用户自定义的对象

• 支持多平台，适合各种开发环境

管理复杂的软件开发过程

开发软件不是一件容易的事，首先面临的是管理多种产品、版本等问题，更为复杂的是由两组或多组人员共同开发相同的程序，再加上多样化的开发程序，使得整个开发过程很难进行有效的管理。而Rational ClearCase就是一个软件开发管理工具来解决以前无法追踪整个开发过程的问题，它结合了完整的软件结构管理 (SCM -- Software Configuration Management)，包括版本控制(Version Control)，工作区管理(Workspace Management)，Build 管理和流程控制(Process Control)。它可以使开发团队能加速开发过程，而且确保得到正确的Release和可靠的build 版本，并建立有效的开发过程，不需要改变原有的开发环境和工作模式。

推进并行开发

在竞争的市场中，并行开发在软件开发中是一项实际的需要，然而很多机构因缺少合适的工具来执行有效的并行开发，结果导致问题未及时修

改、集成困难等问题。**Rational ClearCase** 提供分布式的并行开发模式，让多个开发人员能有效地设计、编写程序，测试及修改程序代码。

强有力的版本控制

Rational ClearCase能让你确认正确的版本，解决版本不一致问题。它追踪每一个文件和目录的改变，维护所有的程序源代码、二进制码、执行码、测试项目、文档以及用户自定义的对象，让开发人员能快速找出所需的内容，重新**build**并恢复到原先的版本。

透明的工作区管理

ClearCase views提供正确的文件版本来进行开发工作，以免除复杂的开发环境。开发人员可以选择所需**view**，**Dynamic view**提供网络使用者快速取得最新的程序代码和文档资料。**Snapshot views**提供**local build** ，而且支持离线作业。开发人员能容易地将自己的工作与整体计划同步，且无论选择哪以种**view**，**Rational ClearCase**都能完全地集成整个开发环境，提高开发效率。

有效的build管理

自动建立**system build**清单，快速正确地产生任何一种的版本，与一般常用的**nmake**及**GNU make**兼容。

有弹性的流程管理

一组丰富的管理工具能帮助开发团队制定开发政策、设定开发角色和工作流程，确立升级模式、流程状态等，自动监视存取状况并防止非法修改，对流程自动化与任务管理。**Rational ClearCase**同时也包含了**ClearGuide**。

可以和**Rational ClearCase**集成的程序设计工具

- Rational 的全线产品
- Microsoft 的Visual Studio
- ORACLE Developer/2000
- Powerbuilder

2 ClearCase 的功能和特点

ClearCase及配置管理

随着软件团队人员的增加，软件版本不断变化，时间的紧缺，多种平台的复杂环境，使得 ClearCase所拥有的特殊组件已成为当今软件开发人员（工程人员和管理者）所必须的工具。分布式操作使得基于Client/Server的运算结构跨越于网上客户机和服务器，ClearCase的先进功能直接解决了原来开发团队所面临的难以处理的问题。

软件开发所面临的问题包括：对当前多种产品的开发和维护，保证产品版本的精确，重建先前发布的产品，加强开发政策的统一和对特殊版本需求的处理。通过解决这些问题，ClearCase用资源重用的方法帮助开发团队使他们所有的软件建立得更加可靠。 Rational公司的ClearCase是软件配置领域的先导，它主要基于Windows和UNIX的开发环境。它提供了全面的配置管理——包括版本控制、工作空间管理、建立管理和过程控制，而且无须软件开发者改变他们现有的环境、工具和工作方式。

ClearCase的四种功能

ClearCase主要应用于复杂的产品发放、分布式团队合作、并行的开发和维护任务，包括支持当今流行软件开发环境Client/Server

网络结构。在激烈的市场竞争中，ClearCase的特点直接响应了软件团队的需求，如：软件生产、发布、维护等。

ClearCase在某些方式上和其它的软件配置管理系统有所不同，从本质上，ClearCase是无可比拟的，因为它包含了一套完整的软件配置管理工具而且结构透明、界面可亲。虽然ClearCase是一个可集成使用的环境，但实际上我们仍可以把ClearCase的所有特性划分为四个具体功能范畴。

Version Control

ClearCase自动追踪每一个文件和目录的变更情况，通过分支和归并功能支持并行开发。在软件开发环境中，ClearCase可以对每一种对象类型（包括源代码、二进制文件、目录内容、可执行文件、文档、测试包、编译器、库文件等）实现版本控制。因而，ClearCase提供的能力远远超出资源控制，并且可以帮助团队，在开发软件时为他们所处理的每一种信息类型建立一个安全可靠的版本历史记录。

Workspace Management

ClearCase给每一位开发者提供了一致性、灵活性和工作空间域（有时也称为"Sandboxes"）可重用的功能。ClearCase采用一种称为View的创新技术，它可以选择所指定任务的每一个文件或目录的适当版本，并呈现它们。View可以让开发者在资源代码共享和私有代码独立的不断变更中达到平衡，从而使他们工作更有效。

Build Management

ClearCase自动产生软件系统构造文档信息清单，而且可以完

全、可靠的重建任何构造环境。ClearCase也可以通过共享二进制文件和并发执行多个建立脚本的方式支持有效的软件构造。

Process Control

ClearCase有一个灵活、强大的功能，可以明确项目设计的流程。自动的常规日志可以监控软件被谁修改、修改了什么内容以及执行政策，如：可以通过对全体人员的不同授权来阻止某些修改的发生，无论任何时刻某一事件发生应立刻通知团队成员，对开发的进程建立一个永久记录并不断维护它。

优势

ClearCase帮助所有规模的开发组织进行更加有效的开发和维护、加强竞争力、增加收益、降低成本。独特的ClearCase带来的特殊利益：

增加团队效率——通过对并行开发的支持来实现，包括图形比较和归并、标签、版本目录 结构。

增加个人效率 ——通过自动的工作空间管理来实现，如：直接的版本访问、消除了在拷贝文件上的时间的浪费。

简单的维护和提高对客户的支持——通过快速准确的重建先前的版本来实现。

快速准确的产品发布 ——通过保证构造的准确性和对软件的每一个元件进行版本控制来实现。

减少错误发生 ——通过事件发生以后对每一个元件的变更进行追踪来实现。

硬件资源的优化 ——通过分布式构造、减少文件拷贝、可用对象的共享等功能来实现。

提高项目协调和编制 ——通过文件注释和开发周期阶段变更的自动关联来实现。

提高产品质量 ——通过灵活的进程控制，和图形接口定制，使得软件开发在实际中保持 一致。

更加有效的团队扩展——通过减少系统管理和维护的负担来实现。

支持分布式结构使得团队成长——通过**Client/Server**结构进行多点复制和及时的对象版本的更新来实现。

使用配置管理工具而降低风险——由于它不干扰软件程序员的工作，所以可以使用常用的工具和文件系统接口。

增加了软件的安全性和保护性 ——通过使用分布式的存储结构，所有的软件资源会随时更新、在硬盘或网络出现错误时那些被**ClearCase**存储的版本信息会立刻恢复。

减少培训和实现成本 ——**ClearCase**通过采用透明结构以及和标准开发工具进行集成来实现。

强有力的开发和维护 ——通过和其它工具（如：缺陷追踪）、系统、结构进行集成。

支持不同种类的开发 ——通过兼容不同平台的软件配置管理系统，如：**Windows NT**、**UNIX**、和一些**Client**端的软件，如：**Windows 95**、**Windows NT**、**Windows 3.1**和**Windows for Workgroups**。

连接UNIX 和Windows的桥梁

ClearCase全面支持软件配置管理，给那些经常跨越复杂环境（如：UNIX、Windows系统）进行复杂项目开发的团队带来巨大的效益。当UNIX和Windows的软件工程人员提出了平台的特性后，ClearCase for UNIX和ClearCase for Windows有高度互用性。

对于它所支持的平台，ClearCase通过TCP/IP来连接客户端和服务器。另外，ClearCase拥有的浮动License可以跨越UNIX和Windows NT平台被共享。ClearCase for Windows NT对ClearCase UNIX VOBs的访问与浏览可以通过一个NFS for Windows NT的产品来实现。

3 ClearCase 的组件

Rational,高质量自动化软件的先锋，向全世界范围的组织提供软件开发基础结构的产品。Rational分散了Windows和UNIX开发团队的解答形式并能使他们自动运行，提高质量和增加生产力。

Rational的软件开发基础结构产品-ClearCase，ClearCaseMultisite，ClearCaseAttache，ClearGuide和ClearDDTS-为软件开发团队提供必要支持。这些产品帮助团队有效管理软件配置，修改需求，开发进程，并测试复杂情况及实时压力。

一、ClearCaseMultisite

ClearCaseMultisite是ClearCase的系列产品选项之一，他支持地理-分布的项目团队的并行软件开发和软件重用。Multisite可以创建和更新被复制的ClearCase VOB，允许分散项目团队使用熟悉的ClearCase命令进行访问，开发和软件的集成。

Multisite这个产品扩展了ClearCase完整的软件配置管理功能，为开发者提供透明支持，为管理提供灵活性和安全性，为系统管理员提供熟悉的工具。ClearCaseMultisite的卓越特性和可靠性为有效的分布式开发作出承诺。另外，ClearCaseMultisite的对等体系结构为任何规模的软件团队提供了可调性和灵活性。

为分布式平行开发复制VOB

ClearCaseMultisite通过克隆有的VOB的内容，为多个地点创建完整的VOB功能。multitool mkreplica命令可以为指定地点创建新的VOB，并可以被复制无限次。复制VOB可以从从本地到每个现场并用于每天的开发。

Multisite组件为异地并行开发实现了ClearCase分支和合并模型的功能。在被复制的VOB中，每个站点都可以为其中的每个元素建立分支,正如作为维护团队可以在未被复制的VOB中建立自己的"bug_fix"分支。

VOB对象元素支持有序的并行开发，这使得项目集成更加容易，还可防止复制信息的变更冲突。不同站点可以读取所有站点的所有分支的修改信息，但只能更改（写入）主分支。另外，任何站点都可以作为集成站点，使用ClearCase自动归并工具可以在不同位置对更改进行归并。

自动同步更新不同站点VOB的更改

在不同站点的分支上进行修改操作会在复制VOB时暂时造成分歧。周期性的更新（同步）使得每个被复制的VOB在监控状态下被更新。在保证精确的情况下，ClearCaseMultisite自动更新复制VOB中的原文

件和meta-数据（事件历史记录，超级联接，属性，和访问控制）。Multisite仅将变化量传送到每个复制品中，消除了"全部-拷贝"复制模型带来的无效性和高成本。

需要时，项目领导和管理员可以计划Multisite VOB的更新，从多种更新结构中选择，包含Multisite建立和存储转发系统，标准文件传送设备或磁带。在更新期间开发工作在VOB中可以正常地继续，而没有必要"锁定"或使VOB "脱机"。

支持连续开发模式

ClearCase Multsite也支持分布在多个站点的团队进行连续开发模式。每一个分支上的指令允许某一站点为另一个站点提供特殊开发控制以及细致的更改共享软件的控制。

ClearCase无缝集成

ClearCase Multisite和ClearCase的紧密集成加快了合作速度并且简化了管理。对于项目队列成员，工作在被复制的VOB中就好象工作在自己的VOB中——无需改变现有的工具和工作规则。

对被复制的VOB需要最小的可持续维护，并为管理员提供详细的同步报表。通过使用ClearCase的熟练语法， multitool的基于字符接口可提供访问Multisite的指令，包括创建，更新，及复制、管理VOB等。

系统需求

ClearCase Multisite作为可选产品安装在ClearCase主机上，需要6-11MB磁盘空间（受硬件操作平台的影响）。每个用户在访问复制

的VOB时需要一个ClearCase Multisite License和一个ClearCase License

二、ClearCase Attache

为客户端使用Windows的项目团队带来强大的ClearCase功能

ClearCaseAttache为以Microsoft Windows 95, Windows NT, Windows 3.1或Windows for Workgroups 3.11作为他们的桌面开发环境的软件开发者提供了扩展ClearCase软件的强大的功能。ClearCase Attache是Windows客户端的软件，它可以与在UNIX和Windows NT服务器上的ClearCase完好配合共同工作。

ClearCase Attache可管理本地版本控制元素的工作空间，可以和ClearCase建立的视图进行关联；并且通过图形/字符接口提供直接访问ClearCase服务器命令。ClearCase Attache界面包含工具栏，下拉菜单，及滚动命令窗口。

本地工作空间的图形界面和ClearCase丰富的命令操作允许开发者在Windows客户系统中执行所有的开发活动，并且可以通过先进的ClearCase工具系列对于基本团队的开发提供更高的支持。

管理本地工作空间中版本控制数据的功能

通过ClearCase Attache，开发管理者可以管理多个地点版本控制数据的工作空间。本地的工作空间提供适当的原始开发版本；对日常的开发任务进行私人存储；隔离其他工作空间的活动等。本地的工作空间是私人的目录树，定位在Windows的客户端或可访问的文件服务器。

任何工作空间都符合一个ClearCase视图，寄存在ClearCase

UNIX或Windows NT的主机上。通过配置文件－－一套为特殊任务选择合适版本的用户配置规则，视图可"过滤" 存储于VOB和ClearCase主机中软件元素（文件，目录，二进制等等）。开发者之后可以检出版本进行编辑（或"get"他们用于本地建立或浏览），拷贝版本到工作空间作为通常文件。当修改已经完成，文件通过视图被检入进VOB。

ClearCase Attache通过ClearCase版本控制系统中的高级特性提供基于Windows的开发者：包括跟踪所有的软件开发对象，永久使用，安全数据储藏所，及对并行开发的功能支持。ClearCase Attache也包括Microsoft Visual C++与 Visual Basic的集成，提供对大众开发环境中版本控制的直接访问。

提供访问到cleartool命令

除了普通工具条按钮和下拉菜单，ClearCase Attache提供一个基于字符接口到每一个ClearCase命令（和cleartool命令设置中的形式是一致的）。这些命令为Windows用户的并行开发、版本历史和报告提供了全面的支持。

通过在UNIX或Windows NT主机上提供的ClearCase工具的直接访问，ClearCase Attache也实现了ClearCase强大的，灵活的过程控制。所有ClearCase结构－触发器，属性，超级连接，权限，加锁等等都可以在ClearCaseAttache中进行。允许管理员横跨Windows,Windows NT,UNIX平台定义增强功能。

与Microsoft Visual C++和Visual Basic进行集成

ClearCase Attache包含与Microsoft Visual C++和Visual Basic的集成，允许从大众的开发环境中提供对版本控制功能的直接访

问。该集成支持微软Source Control Code（SCC）接口设置，一个在IDE工具和原代码控制工具之间交流的API。ClearCase Attache函数被映射到SCC接口，从IDE中提供直接访问，使用标准（微软）SCC对话框。

在Visual C++中，用户可以访问公共SCM操作，包括：增加新文件到源控制；检出/入文件及装载的ClearCaseAttache历史和属性的显示。近似地，在Visual Basic IDE中的用户可以增加Visual Basic项目到ClearCase Attache中；增加新文件；检入/出；及装载ClearCase Attache历史和属性的显示。

可选的建立工具

ClearCase Attache用户建立软件象以前一样，使用同样的工具及makefiles。可选的建立器通过ClearCase在Windows NT中的omake 、Borland Make、Intersolv Configuration Builder(Polymake)、Microsoft NMAKE提供一致的makefile功能。

通过ClearCase Multisite支持分布的开发

ClearCase Multisite组件提供对地理分布的开发团队使用ClearCase和ClearCase Attache的支持。ClearCase Multisite可以跨广域网和本地站点复制并更新的ClearCase VOB。该组件允许ClearCase Attache用户访问，修改，复制在分布项目上的软件元素。

系统需求及License

ClearCase Attache需要386/486 Pentium PC、Windows 95、Windows NT、Windows 3.1、Windows for Workgroups 3.11、8MB内存10MB以上硬盘。磁盘空间的需求取决于本地工作空间的数量和大小。

ClearCase Attache也需要运行在UNIX workstation(DEC、HP、IBM、SGI、Sun)上的ClearCase R.2.1或更高的版本；运行在Windows NT个人电脑上的ClearCase R.2.0。在www.Rational.com可获得更多当前的ClearCase Attache的系统需求。

在实际中，站点上的每一个正式用户都需要ClearCase Attache的License。每一个ClearCase Attache的站点至少需要一个ClearCase系统管理员的License（UNIX或Windows NT）。近似地，任何使用ClearCase Multisite 的ClearCase Attache站点最少需要一个ClearCase Multisite License（UNIX或Windows NT）。

三、ClearCase 其它组件

ClearGuide

ClearGuide是Rational's新的软件过程管理（SPM）产品，它组合了项目管理，工作流，和过程模块的关键特性。ClearGuide超越了传统的工程变更管理系统（包括：项目计划，定义和过程执行和有关软件生命周期的所有任务的管理）的能力。使用ClearGuide，软件开发团队可以从强大的时间线，可预见性的软件项目和定义能力中收益。通过一个灵活的过程框架，重复并提高他们的软件开发过程。他们也可以通过常规任务自动化提高软件开发生命周期中的活动精确度和效率。ClearGuide需要ClearCase3.x版本。

ClearDDTS

ClearDDTS是分布式的变更管理系统，它可以帮助开发者和质量保证组织测量产品质量和管理变更需求。

ClearDDTS广泛的缺陷管理能力记录并追踪所有信息（关于缺陷报告，提供项目查询，报告，图表，为缺陷提交提供分布支持。ClearDDTS'强大的变更管理工具可以和ClearCase集成，并且存储信息（关于被检入或检出的文件）在预检的基础上确保完整的追踪。

4 ClearCase 结构与设置

一、客户/服务器结构

ClearCase是运行在分布式Client/Server结构中的"组件"产品，。 ClearCase函数和开发数据的程序可以被分配到整个本地网络。这使得ClearCase的工作范围－－从工作站上被加到网络中以便更多的开发者可共享，ClearCase的数据存储和数据处理资源的能力大大提高。

数据仓库组成如下：

永久性，共享数据存储库是一种VOB的集合。多种VOB也可以存放在同一主机中（要有充足的磁盘空间和处理资源的能力）。

开发者使用单独的（或共享的）工作区域称为视图－－任何人都有一个小的私人库区域。视图的存储区域一般位于独立的工作站或PC上。主控服务器可作为为共享视图或为那些将被重建或发布应用程序建立视图。

增加灵活性，可以跨两个或更多的主机，为单独的 VOB或视图进行数据存储。

开发者使用ClearCase客户端程序访问这些数据（例如，clearmake建立工具），以及标准的操作系统工具及第三方应用程序。ClearCase服务器的程序可间接访问在VOB和视图中的数据。客户端和服

务器通过使用远程调用过程（RPC）互相进行进程通讯。这使得开发者不必涉及数据存储的物理定位而进行ClearCase网络通讯；ClearCase服务器使数据完全有效。

二、图形用户界面

ClearCase包含传统的命令行界面和Motif及Windows点击图形用户界面（GUIs）包括任务设置GUI组件。UNIX和Windows NT的ClearCase的GUIs提供下拉和弹出菜单，工具条，context-sensitive帮助显示来简化公共用户级的命令。另外，界面包含文件浏览器，视图，VOB，版本树，超级联接，可选择的数据，及更多的可以简化在ClearCase中的公共数据对象的查询和选择。

GUI也提供直观比较和归并功能，用高亮度颜色来描述插入，删除，修改。GUI可以通过扩展脚本语言被定制，使用户能创建自己的按钮，工具条，和多水平菜单。定制的组织政策和脚本能在GUI中被访问，而且外部命令也能与GUI进行集成。

另外，ClearCase具有图形事件和属性显示的功能。事件显示可提供相关ClearCase控制元素的历史记录信息，可以被定制成当前全部元素的历史。关于当前元素或设置版本的属性信息，使用制表键显示命令信息，标签，属性，超级联接，触发器，安全性，加锁。

三、ClearCase for Windows NT

ClearCase for Windows NT包含附加的GUI功能可以增强NT 4.0用户界面的功能。ClearCase扩展的context-sensitive菜单的使用提供给用户快速访问公共ClearCase操作和工具。

四、Windows 资源管理器的集成

ClearCase for Windows NT包含和Windows资源管理器的集成，使得公共的ClearCase操作对于用户简单有效。此集成允许用户打开视图，mount VOB，检出/入元素，激活版本树浏览器，检查元素历史和属性，寻找检出元素，比较新老版本，及激活ClearCase详细应用，在线帮助也包括在内。

五、ClearCase Details 工具

ClearCase Details工具显示与ClearCase相关连的文件和目录的信息，比如检出状态，用户视图选择的元素版本，及用户选择的版本的配置设置。ClearCase Details工具允许用户去修改显示的属性，访问到其他目录，去调用更多的ClearCase命令和工具。

六、视图描述工具

ClearCase视图描述工具打包了被开发团队共享的ClearCase配置信息。视图描述包含以下信息：

选择版本属性到团队工作的配置设置。

识别团队基线的检查标签列表。

团队计划工作的VOB。

团队正在使用的系统管理的VOB。

一队并行工作的开发者可以在ClearCase视图描述上奠定他们的视图。在这种自动格式大部分工作需要设置和保持团队共享的ClearCase配置。

视图描述浏览器允许项目管理者创建和修改ClearCase视图描述。视图创建程序提示用户通过需求来创建视图，并且也可以让用户基

于存在的视图描述中选择视图。

七、归并管理器

归并管理器是管理归并元素过程的图形工具。他自动为归并、开始归并，及跟踪归并收集信息。他同时可以结合使用ClearCase Diff归并工具来比较版本并完成归并操作。

八、与 Visual C++和 Visual Basic 的集成

在Windows NT中，ClearCase支持Microsoft公共源代码控制（SCC）接口配置，支持在Visual C++，Visual Basic工具和源代码控制工具之间关联的-API。ClearCase函数被映射到SCC接口，提供从Visual C++和Visual Basic的IDE到ClearCase的直接访问，使用标准（微软）SCC对话框。

在Visual C++中，用户能访问公共ClearCase操作，包括：增加新文件到源控制；检出/入文件和激活ClearCase历史和属性的显示。

类似地，从Visaul Basic IDE中用户可以开始视图；mount VOBs；增加Visual Basic项目到ClearCase；增加新文件；检入/出；激活ClearCase历史和属性的显示。

九、系统管理员

ClearCase包含一套工具，命令，和GUI应用以便建立、扩展及管理VOB，视图，和跨越站点的策略。系统管理员能管理物理磁盘存储，网络间的系统转换，确信VOB保密性，管理用户的License，限制对软件元素的访问。状况和错误记录信息被送入记录浏览器。必要的系统管理信息和命令在VOB属性框架中，其他context-sensitive菜单，和在ClearVobAdmintool(UNIX)中被设置。管理员使用他们现有的备份工具

备份ClearCase VOB。

十、视图和 VOB 的储存注册

在每天的工作中，一般地，ClearCase用户会涉及配置VOB和视图使用名称（"tags"）。例如，项目团队可以在mount为"/vobs/gui"（UNIX）或"\vobs\gui"（Windows NT）的VOB中使用共享的"bug_fix"视图来访问项目。系统管理员通过ClearCase储存注册管理这些相应的视图和VOB库区域中完整的名称和物理定位（路径名称）。储存注册是广域网资源，定位在指定的服务器主机中，他映射一般使用的视图和VOB名称到属性存储区。系统管理员能定义多个网络区，在客户端使用不同?quot;完整"路径名来访问相同的储存目录。它可以登记结构以便支持（比如）不同主机空间的多个子网。

5 ClearCase 四大功能详述

版本控制

ClearCase的核心功能是版本控制，它是对在软件开发进程中一个文件或一个目录发展过程进行追踪的手段。ClearCase对所有文件系统对象（包括文件、目录和链接）增强了版本控制系统功能。可定版本的文件包括源代码、可执行文件、位图文件、需求文档、设计说明、测试计划、和一些ASCII和非ASCII文件。目录的版本记录了整个组织基础资源的发展状况，包括源文件的建立、重新命名、重新构造和删除操作等。 这种版本控制系统提供了先进的版本分支和归并功能用于支持并行开发。

1 控制任何文件的版本

ClearCase可以对每一个软件组件或元件的版本进行维护和控

制。ClearCase也可以维护一个非文本文件、目录和工具的版本。正如：它可以管理库文件、编译器、需求文档、 测试包和数据库而不仅仅是源代码。

ClearCase的元件类型可以管理版本内容。用户可以定义自己的元件类型，也可以使用ClearCase中的预定义类型：文本文件、压缩文本文件、文件、压缩文件和二进制增量文件。

ClearCase可以利用增量算法将文本文件存储在一个特殊结构的文件容器中。ClearCase采用标准的压缩技术和增量算法存储一个压缩文本文件。（这比以往的存储形式节省了50%－70%的存储空间。）

这种元件类型文件和压缩文件可以被用于控制任何操作系统文件——比如，可执行程序、程序资源库、结构数据库和结构文档文件。二进制增量文件类型可以随时被用于二进制文件格式。

2 在版本树中组织元件发展的过程

在ClearCase中，元件版本的组织体现在版本树结构中。一个版本书的结构可以按目录结构定制， 还可以包含多层分支和子分支。

在一个典型的开发环境中，很多元件的版本树结构最初仅包含一个分支，即， 元件的版本排列在同一条线型队列中。随着时间的发展，当用户做一些错误修复、代码的组织、一些实验性修改或指定平台的开发时，它们可以给一些相关元件定义子分支，从而脱离主干进行开发。ClearCase可以支持多级的分支操作，还可以给版本或分支命名。

3 对目录和子目录进行版本控制

ClearCase可以对目录和子目录进行版本控制，允许开发者对他们数据的组织发展过程进行追踪。目录版本对一些改变进行控制，如：

建立一个新文件、修改文件名、 建立新的子目录或在目录间移动文件等。

ClearCase也支持对目录自动进行比较和归并的操作。

4 存储数据在一个可访问的版本对象类中（VOBS）

ClearCase把所有版本控制的数据存放在一个永久、安全的存储区中，这个存储区被称为版本对象类（Version Object Bases），项目团队（或管理者）可以决定它们所需要的VOBs的数量，可以决定什么样的目录或文件需要被维护。VOBs不仅是一个可连接的文件系统而且也是网上的资源——主机可以连接任何数量的VOBs.

ClearCase VOBs的组成模式跟UNIX、Windows NT的文件系统和分布式的数据库系统非常类似。ClearCase采用Raima数据管理机制区维护VOB数据库。当在ClearCase中连接和访问时，VOB象一个标准的软件作为目录树的形式出现在客户面前，包含标准的文件对象：目录、文件、符号链接和硬链接。但事实上，文件系统已经有广泛的版本控制组件：它包含目录元素、目录元素版本、文件元素、文件元素版本、VOB动态链接和VOB硬链接。开发者也可以查看和这些文件系统对象相关的数据。这些数据包括事件记录，建立审核以及用户定义的项如：版本标签和属性。

5 使用常见的检出/编辑/检入范例

ClearCase的命令可以控制元素的变化，确保存储区有序的繁衍并使数据损坏的程度达到最小。ClearCase采用一种检出/编辑后检入的范例，类似于传统的版本控制工具如：RCS和SCCS。ClearCase除了可以进行检出、检入以及非检出操作外，它还可以通过命令设置另外的操

作，如：删除版本、建立/删除分枝、可按时间顺序排列或结构排列顺序列出版本历史、比较版本间的差异，并且可以归并并行开发的版本。

当开始对于一个指定的文件进行工作时，该文件具有只读属性——这意味着它不能被编辑或删除。而检出操作可以对该文件的最近版本形成一个可编辑的拷贝。它无须将文件拷贝到另一区域工作。检出的注释可以被提供。当编辑完成后，该文件被检入，于是在版本树中形成一个新的版本并且将可编辑的拷贝删除。为了检验文件的变化，在检入过程中可以填入注释信息。文件一旦被检入，即刻回复到只读状态成为共享数据，可被所有成员使用。

ClearCase支持两种检出，保留以及非保留。保留检出可以保证版本历史形成的正确范围，并且同时只允许一个人做保留检出的操作。非保留检出无须保证建立一个成功的版本，如果多个用户同时对同一元素执行非保留检出，也企图进行检入操作，那么第一个检入操作被允许，而其他用户必须通过归并操作合并它们的结果。

6 丰富的注释信息和版本数据的报表

ClearCase存储了和文件系统对象相关又截然不同的信息类。这些信息实际上并不包含在对象中，它是一些额外数据。这些数据可以由**ClearCase**产生，也可以由用户自己定义。在**VOB**数据库中存储了所有的数据。

ClearCase产生的这种数据信息提供了可靠的、面向文件系统的版本注释信息。比如：这些数据可以验证在某一时刻，元素**A**建立了一个新的版本。用户定义的数据可以用来表达额外的功能——比如：该文件的版本曾被用于构造应用系统的**4.31**版。

ClearCase的操作（如：检出、检入、和版本归并）可以建立时间记录，记录数据包含这些操作信息。这些记录被存储在VOB数据库中，主要描述了该操作的属性"谁做的、做什么、什么时候、在哪个地方及为什么"，比如：敲入命令的人员的ID号，操作的种类，操作的时间，主机名称及用户填入的描述。可以通过"lshistory"的命令显示存储在VOB中的事件记录，并且可以通过历史信息浏览器提供的图形接口观察VOB中的事件记录。

用户可以针对多种目的定义数据，包含分支的名称、版本标签、元素任一版本的注释信息。

ClearCase数据的另一种应用是形成注释的文本文件。注释命令可以通过行显示的形式列出任何一个版本文本文件的内容，这使得我们可以更容易的看到什么时候在不同的地方做了添加或删除的操作。

ClearCase也可以针对文件系统对象建立客户报表。而报表的种类可以由用户自己定制输出格式。

7 通过分支功能支持并行开发

ClearCase支持并行（同时）开发，每一个元素都可以沿着不同的分枝同时发展，即新的版本加到独立的分支上。ClearCase可以很容易的产生分支，也可以很容易的将不同分支进行合并。这样一来，即便某一部分的工作被冻结或加锁，开发者仍然可以继续自己的工作（如：在软件集成期）。在这种情况，开发者可以在分支上工作，我们知道，ClearCase的自动化操作和图形归并工具可以让我们很容易的重新集成新的工作。

并行开发是非常重要的，因为：

（1）它允许不同的项目在同一时间使用同一资源树。

（2）它将目前不可和其他人员共享的修改成果进行隔离。

（3）它将绝对不可和其他人员共享的修改成果进行隔离（如：已发布版本中的错误修复）。

（4）它使得在软件集成期间开发工作无需停止，程序员可以先在分枝上开发，以后再集成。

为了支持并行开发，ClearCase允许进行分支建立，追踪分支的使用，文件比较，自动归并功能。

8 自动的比较和版本间的归并

并行开发的特点是对同一元素的不同版本进行定期比较，也需要对版本间内容进行归并。在ClearCase中，对于元素或文本文件进行比较和归并的操作有两种：基于字符型和图形界面型。其中，diff命令执行多文件比较，不执行归并。而归并命令可以处理32个"成员"，并把它们生成一个独立的文件。 ClearCase可以自动辨认归并选项并实现归并。ClearCase也可以对需要归并的项目元素进行定位。如果所有的"成员"（归并元素）是同一元素的版本，系统会自动确定基础"成员"，通常是最低版本。此外，ClearCase会记录基础版本和某一归并元素版本间的差异。如果，所有的"成员"间差异互不相同，ClearCase会自动建立归并版本。如果两个或多个归并"成员"文件内容部分不同，归并功能会提示开发者选择归并内容。ClearCase也可以实现反向归并－－从主分支向子分支归并。

ClearCase的加归并功能可以在归并其它分支时选择指定的版本（那些在分支上自始至终进行变化的版本）。负归并操作可以删除部

分版本差异，从而形成一个新的版本，该版本除了那些被删除的变更外包含所有的改变。

工作空间管理

快速、有效的工作空间建立对于提高个人和团队的效益是非常重要的。通过视图（VIEW）的使用，ClearCase提供了一套独立的工作空间管理设施，可以实现动态评估、选择指定用户版本和透明的访问多种配置的功能。

1 版本间的透明访问

ClearCase提供了对版本进行透明访问的功能。通过VOB机制（包含文件或目录的多个版本），ClearCase可以让开发者和应用者以一种标准文件目录树的形式访问VOB。这个特性被成为透明－－ClearCase的版本控制系统因而变得可视化。

透明是一个非常重要的特性，它允许ClearCase在使用系统软件、商业应用和内部工具时进行平滑的工作。比如：象grep，more，ls，cc这种标准UNIX程序，在操作ClearCase版本控制数据时与操作一般的文件系统对象的方式一样。

通过ClearCase的多版本文件系统可以（MVFS）在虚拟文件系统上实现透明操作。MVFS可中断标准的I/O调用，并且ClearCase的版本选择结构可以细化到从一个元素到另一个元素版本的目标调用。

对于Windows NT，ClearCase的MVFS一般缺省作为"M："驱动盘出现，活动视图作为"M："盘的根目录出现。正常情况下，ClearCase可以为每一个活动视图分配更多的虚拟盘（从"Z："以后工作－）。把VOB设置成每一个虚拟盘的子目录。这样就可以让开发者使用自己的工具透

明的访问被ClearCase控制的数据，甚至是UNIX VOBs和视图。

2 通过规则视图选择并显示版本

ClearCase的视图提供了强大的、独立的工作空间管理（也称作"环境管理"或"沙盒管理"）。通过使用动态评估、用户指定版本选择规则，视图可以让开发者对任何元素的任何版本进行透明的、文件级的访问。ClearCase的视图具有灵活性、可调性、有效性并可随时自动更新。

通过开发者对ClearCase控制的数据和程序的版本进行选择，视图可以对完整的文件系统配置进行动态管理。它也可以访问主机上的其它数据和程序。

ClearCase支持规范的开发环境，它可以维护公有和私人两种数据存储类型。所有的ClearCase用户可以共享或公开在VOB中存储的数据，它们包括一些常规访问的计划信息。存储在视图中的私有数据一般包含属于开发者个人的文件，如：通过标准工具被检出的文件元素版本，在视图中由ClearCase建立的原始对象，和由视图用户在VOB目录中建立的文件和目录。视图在"虚拟工作空间"存储了这两种数据，开发者每天对其执行检入、检出、编辑原文件操作、建立软件和修复系统等操作。

在视图中选择的版本可以称为视图配置。视图配置是动态的并可以在任何时候被开发者修改。视图配置在配置规格说明的一系列规章被定义。一般的，视图的配置在通配符和助记符的术语表中被定义，而不是通过指定具体的版本名称。 每个开发者都可以拥有多个视图，并且可以在任何视图中设置过程。此外，不同视图可以看同一路径名下的同一元素的不同版本。比如：一个视图可以浏览某一元素最近的版本；

另一视图也可浏览该元素的某一版本，它可能曾经用于构造某一具体的发放版本；可能还有其它视图浏览该元素用于修复错误的版本。

此外，那些不受**ClearCase**版本控制的所有的文件和目录（标准文件、本地的脚本和程序，等。），也都可以通过视图进行浏览。从而使得**ClearCase**成为开发者的好友，当他们使用视图浏览数据文件、修改框架脚本、编译程序时，通过使用扩展视图的路径名或扩展版本的路径名，开发者可以提高透明度。扩展视图路径名可以覆盖当前视图并且可以访问当前出现在其它视图中的元素的版本扩展版本路径名是一种独立的视图，它可以通过版本树的位置或版本标签定制一个特殊的版本，而不管该版本究竟出现在哪个视图中。

3 从没有安装ClearCase的主机平台进行视图访问

在局域网中**ClearCase**所控制的数据对于未安装**ClearCase**的机器也可使用。比如：一个**ClearCaseUNIX**主机可以通过一种特殊的视图输出**VOB**；而网上的其他主机可以通过**NFS**机制连接它。这样它就让开发者在未安装**ClearCase**的主机平台上使用自己的工具对视图进行读写访问，编译并建立自己的应用。未安装**ClearCase**的主机必须重新注册或使用安装**ClearCase**的**UNIX**主机上的**X-Windows**系统做检入、检出操作。

过程控制

软件开发的策略和过程由于行业和开发队伍的不同而有很大差异，但是有一点是肯定的：即提高软件质量，缩短产品投放市场时间。**ClearCase**为团队通信、质量保证、变更管理都提供了非常有效的过程控制和策略控制机制。这些过程和策略控制机制充分支持质量标

准的实施与保证，如：**SEI Capability Maturity Model** 和**ISO 9000**。

ClearCase具有以下过程控制的优势

1 集成了一些灵活、定制的工具

ClearCase提供了过程和策略控制机制以提高软件质量，缩短产品投放市场时间，以及调控整个软件开发过程。**ClearCase**所具有的监测和控制开发过程的工具无需指定预定义方法学、政策、以及过程。它本身的灵活性、强有力性，为管理者实现现有策略的自动化和巩固以及创建其它新的过程管理系统成为可能。**ClearCase**中所包含的灵活机动的工具可以让开发人员实现：

■ 监控开发过程；

■ 组织、交叉查询开发中涉及到的所有数据，如：源代码、记录、设计初衷、技术手册等；

■ 在个人和团队之间实现自动化的通讯；

■ 自动处理冗长、有错误倾向的步骤。

这些工具都是基于元数据操作的，所以过程管理所涉及到的数据结 构和程序都是独立于元素变量内容的。总而言之，主要的过程管理特征就是：以元数据抓取状态信息，策略增强工具、**?quot;**通知"特性。

2 利用元数据抓取状态信息

ClearCase元数据（在**VOB**中与对象相关联的数据）抓取特定对象的状态信息。在过程控制中共有三种类型的**ClearCase**元数据可用：

■ 属性。一个属性是一对值： 名字**=**取值。开发者可对大多

数对象赋予属性。属性可取多种类型的值，整型、字符串、日期等。取值被限制在特定的范围内，或限定于特定的枚举值。例如，**Codequality**属性可有**A**、**B**、**C**、**D**或**F**五个值。其强有力的查询工具允许用户查找，如一个叫**John**的用户在上个月创建的包含**Codequality=A**的所有版本文件。而增强机制则自动为对象分配了属性。

■ 超级链接。所谓超级链接是一个连接着两个对象的逻辑"箭头"。例如，一个超级链接可以连接设计文档和资源代码模块。超级链接可追溯到所有的元素变量、特定的版本（需求追踪也同样需要）、或者对象中的某一部分。它可跨越**VOB**并重命名、移动一个对象或这个对象所在**VOB**。利用超级链接浏览器，用户还可以显示、创建、访问、维护**CLEARCASE**超级链接的网络。

■ 历史事件。**ClearCase**自动记录下来重要的状态信息，当对象发生变更的时候，它会收集"谁、何时、为什么"、用户注释、以及其它的重要数据。系统也会保留创建、释放项目时的类似信息。

3 定制的策略增强工具

ClearCase的策略增强工具支持管理者建立并加强一个好的软件开发策略。**ClearCase**的工具包括：

■ 事件预触发。事件预触发机制监视每一特定**ClearCase**操作（如：检入 **check-in**）或操作类（如：改变**VOB**的任一命令）的使用。在操作执行之前，触发开始，经历以下特定步骤：程序、批处理文件、脚本、其它内置动作之一。同时，一个触发还可要求在执行某个操作命令之前对它进行检查，并据此判断是继　葱小11.故侨∠　僮鳖**?/p>**

■ 锁。针对于一个对象，锁禁止对象发生变更。锁可被划分

的很细（例如，只锁住指定的元素变量），也可以笼统而论（例如，锁住整个VOB）。一个典型的应用就是：在软件集成阶段，锁住所有主干元素。而且，每个锁可定义"锁定例外表"，允许特殊用户修改对象。

■ 访问控制。对所有元素采用类似UNIX的保护机制。这种保护机制控制读、写、以及基于传统标准上的对象执行：单个用户的、开发团队的、或其它。同时，它还对文件系统之下的物理存储施加保护，有效的制止那些试图逃避ClearCase或破坏原始操作系统存储的小动作。

■ 自动创建分支。当所有的变更动作是在分支上以同种模式进行时，最易于维护工作。而ClearCase恰恰增强了这一点，当元素检出的时候，ClearCase会为它自动创建分支，并指定一个名字。

4 “通知”特性能自动生成报表、交流信息

■ 事件后触发。事件后触发机制好象一个监视器，它在特定操作完成后运行。实际上，这一触发会在某个命令执行后、或给某个对象赋予属性后，把这些动作通知给用户。为了便于脚本和程序实施触发动作，ClearCase自动设置了一些环境变量。以一个"检入"的事件后触发为例，它会告之质量保证部门有一个用户已修改过某一特定的文件，并且，还会包括在 "检入"时那个用户输入的注释。

■ 查询功能。ClearCase中有一个 find（查询）命令，使得开发者迅速的获知当前项目的状态。实际上，find（查询）命令就是在一个或多个VOB数据库上实施查询操作。例如，查找不具有 Passed=QA 属性且属于Release 2.0的所有版本文件。

■ 动态配置规格。配置规格的方法是根据标签、属性、超级

链接、以及历史事件选择版本文件。和find（查询）命令一样，这些方法同样具备查询功能。如，配置规格可以选择“具有Passed=QA属性的最新版本，或者是由用户drp创建的最新版本。”

建立管理

使用ClearCase，构造软件的处理过程可以和传统的方法兼容。对于ClearCase控制的数据可以使用自制脚本或本机的make程序，但ClearCase的向上兼容建立工具clearmake和omake为构造提供了重要的特性：自动完成任务、保证重建的可靠性、存储时间和支持并行的分布式结构的建立。

1　支持UNIX和Windows型的makefile的建立

ClearCase包括两种独立的建立程序，clearmake和omake。这两种程序合并了ClearCase的主要建立特点，包括配置lookup，二进制文件共享，和配置记录。Clearmake程序主要适用于使用UNIX型的makefile包含（gnumake）的用户。Omake主要适用于那些需要和Windows上的建立程序（包括：Borland Make、Microsoft NAMAKE、Intersolv Configuration Builder、和OpusMake）兼容的用户。

2　自动检测所关联的原文件，包括所关联的头文件

clearmake和omake通过使用当前原文件（向一些被检入、检出文件）的配置，可以在视图中灵活的建立整个或部分软件系统。Clearmake和omake在makefile时无须描述所关联的头文件（或任何所关联的原文件）。

在ClearCase开发环境中，原始对象扮演着决定性的角色。源对象是由clearmake和omake 建立的文件对象或目录对象。典型的源对

象应该包括由文档系统产生的对象模块，可执行程序，库文档，规格文档，内容表。源对象组件包括：作为目标被建立的文件名；独立的源对象ID；数据容器指针（存储建立脚本所产生的数据的文件）；配置记录指针（信息清单）；和参考计算（指示源对象当前出现的视图号）。

3 自动的追踪建立，产生永久性的资料清单

在执行建立脚本期间，clearmake和omake在ClearCase的多版本文件系统下执行一个建立追踪。这MVFS记录了在连接的VOB中每一个被读或执行的文件的版本；它也可以注释哪些文件被建立（或被覆盖）。在执行建立脚本之后，clearmake和omake将追踪的数据写入配置记录中，存储到VOB数据库中。VOB数据库指针将配置记录分配到每一个建立过程的源对象中。

配置记录就是源对象信息清单，包含它的内容和建立时的有用信息：

■ 存储在VOB中，在重建时使用的文件元素的版本——包含ClearCase控制下的源文件和工具（比如：编译器）。

■ 在建立过程中使用的每一个私人视图文件。

■ 在makefile过程中使用的非ClearCase文件。

■ 建立脚本的文本及所有的可扩展宏。

■ 操作系统版本和CPU类型。

■ 执行建立过程的用户；执行建立脚本的主机；由clearmake设置的视图和建立过程开始的日期和时间。

ClearCase的配置文件可以让源对象进行比较——不依靠对象

数据，而依靠它们的建立配置信息。**Diffcr**（**compare config rec**）命令可以输出不同配置文件间的差异，包括：

■ 源文件的差异，非源文件修改的时间戳。

■ 建立过程中执行脚本的差异，包含**makefile**中不同的宏值。

■ 那些不影响**clearmake**或**omake**建立的非必要差异，包括建立时间/日期，主机名，视图名。

4 开发者间共享二进制代码，时间和存储空间的存储

基于**makefile**一个很重要的方面，避免不必要的建立过程。**Clearmake**和**omake**的建立策略是非常优秀的，专为并行开发方案做的特殊设计。 **Clearmake**和**omake**可以通过配置文件检测现场情况，检测哪些源对象可以在多个视图中被共享。这个工具还可以进行磁盘存储和建立时间存储。**Clearmake**和**omake**提供了三种可供选择的建立方式：

■ 重用视图中现存的目标－－**clearmake**和**omake**使用一种技术，它比比较时间戳更熟练。配置信息可进行源版本对照，建立脚本对照，建立选项对照。

■ 执行传送建立脚本－－**make**、**clearmake**和**omake**以同样的方式执行目标建立。但是**clearmake**和**omake**可以对建立过程进行追踪，并将追踪信息分配到每一个重建过程相关的文件中。文件和它的配置信息组成了源对象。

■ 从某一视图中**wink-in**源对象－－**clearmake**和**omake**可以了解到早先在其它视图建立的同一目标的多种实例。在验证后，正确的源版本，建立选项和建立脚本被用于建立其它的实例，**clearmake**和

omake将对视图执行一个wink-in操作。一个源对象现在可以被其它视图所共享。

5 跨越不同机型进行并行分布式建立

Clearmake支持分布式建立（使用其它主机上的执行脚本）和并行建立（执行一致的建立脚本）。比如：clearmake可以进行三方建立，所有的进程都在一个多处理器的计算机服务器上执行。在局域网中，它可以跨越所有工作站进行分布建立。

Clearmake也支持跨多种开发环境的建立。

6 自动的跨多种主机（UNIX）的平衡加载，分布建立

clearmake有一个尖端平衡加载技术，可以优化分布式建立的执行。用户指定功能等于分布式建立服务器的主机，并且设置变量，包括：时间、机器装载和控制每一台机器建立的用户id 。clearmake可以跨越这些主机自动平衡装载进行分布建立。

VSS

1. 前言

如今随着软件项目规模的日益增大以及项目复杂性的不断加剧，软件配置管理（SCM）的重要性已越来越受到大家的认可。许多优秀的软件配置管理工具也应运而生，使得我们能够轻松有效地管理我们的软件项目，作为这其中的一员，Microsoft Visual SourceSafe具有简单易用、方便高效、与Windows操作系统及微软开发工具高度集成等优点。

我相信有相当一部分人曾经使用或者正在使用VSS，对于它的一些基本用法在这里不再赘述，本文将主要探讨Visual SourceSafe 6.0中的一

些高级特性，希望给大家在实际的工作过程中带来一些帮助。

我们知道随着软件规模的日益增大，软件开发周期会变得越来越长，投入的人力也会越来越多。在整个过程中，将不可避免地会产生并行开发的情况，那么在VSS中是如何支持并行开发的呢？在并行开发的过程中要注意些什么呢？这就是本文希望与大家分享的。

本文要求读者对VSS有一定的熟练程度。

2. 一些有用的设置

在讲解并行开发之前，让我们先来看看VSS中一些非常有用的系统设置，他们可以用来加强管理或者简化你的工作。首先是一些与你的项目安全管理策略有关的设置，比如"是否允许multiple checkouts"、"是否激活项目安全机制(project security)"等。还有一些是方便工作的，比"设置影子目录(shadow folders)"、"重用上次注释"、"双击直接编辑文件"、"是否采用图形化归并"等。这些你都可以在"Tools"-〉"Options"里进行设置。

3. 什么情况下会出现并行开发

不知道有没有喜欢听评书的朋友？我想一定和我一样最讨厌听到：欲知详情如何，且听下回分解。不过我们今天要讲的是另外一句话：花开两朵，各表一枝。我记得当时虽然这边打得热火朝天，大呼过瘾，但是心里总是惦记着那边怎么样啦！真恨不得有两个电台，一边是鲁智深大闹野猪林，一边是林教头风雪山神庙。那么在实际的项目开发过程中，什么时候会出现花开两朵的情况呢？

1、 你想修改项目早期版本中的某个bug。

2、 其它的小组成员占有了你希望处理的文件。你可以等他将该文件check in，或者你可以采取multiple checkouts，你还可以选择在一个分支上工作。

3、 你所做的工作涉及到许多文件，你选择一个分支流就可以避免经常打乱别人的工作。同时你可以将所有的工作测试完后再将它们集成到你的项目中。

4、 项目规定你不能够在主线上工作。相反地，项目被分成许多小的部件，每个部件是一个分支，只有该部件完成后，才会将它合并到主线中。

那我们是不是需要两个电台呢（建立两个项目）？可喜的是，VSS中提供了共享、分支等操作来解决并行开发的问题。

4. 文件共享(share files)的概念

在VSS中你可以在不同的项目之间共享文件。我们知道，在VSS的数据库中有且仅有文件的唯一拷贝--主控拷贝(master copy)，因此共享文件也就是在不同的项目里建立了指向该主控拷贝的链接。如果你在其中一个项目中改变此文件，那么其他项目中所有的共享也随之改变。此外，如果选择共享某个项目中的所有文件，我们也可以称之为共享该项目。

如何实现文件共享

1、 在VSS Explorer中，选择你希望在其中共享文件的项目。

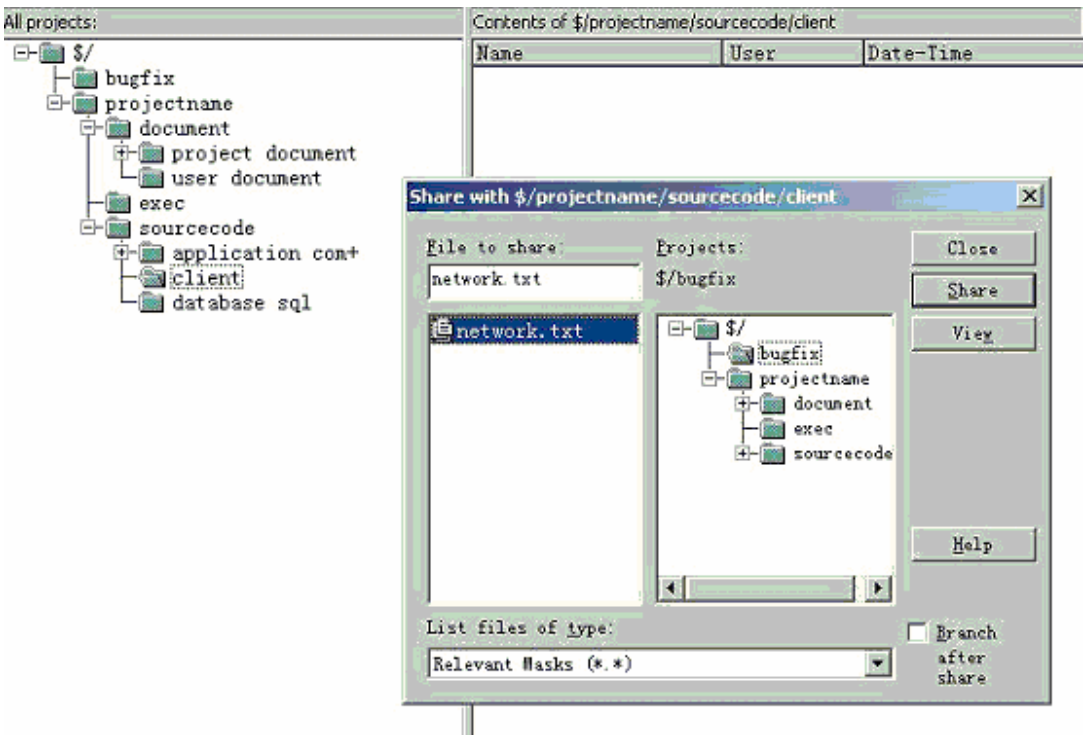
2、 选择SourceSafe菜单，单击Share，或者单击右键菜单中的Share显示Share对话框。

3、 通过选择**Project**和**File to Share**下拉框来选择你准备共享的文件。

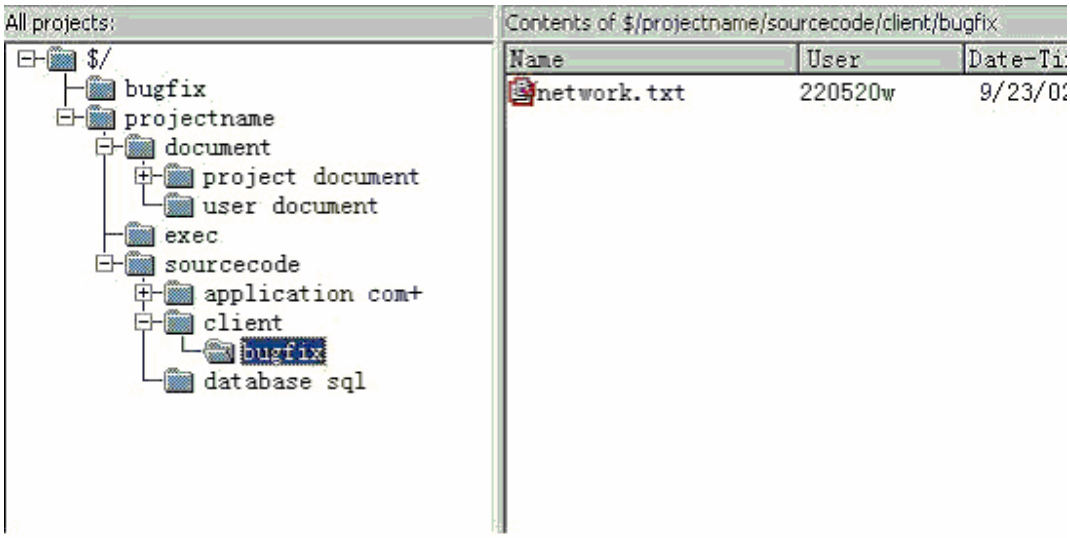
4、 点击**Share.**

5、 点击**Close.**

如果你选择的是某个项目，系统还会提示你输入新的共享项目名称。操作完成后，你会发现在当前项目中增加了刚才选择的文件。如附图一所示，我在项目**$/projectname/sourcecode/client**中共享了项目**bugfix**，操作完成后在**client**下增加了目录**bugfix**，而且我将**$/bugfix**项目中的**network.txt**文件**check out**出来后，项目**$/projectname/sourcecode/client/bugfix**中的**network.txt**文件也自动被**check out**。（见附图二）



附图一：文件共享



附图二：文件共享后

文件共享机制确实为我们带来了不少便利之处，它可以减少数据库中文件的数量，能够实现文件的重用。但是，文件共享机制的"非各自独立性"也限制了它扮演更重要的角色，真所谓"成也萧何，败也萧何"！

5. 分支操作(branching)

要真正地支持并行开发，就不得不用到分支操作。与共享操作不同的是，分支操作实际上是将文件放在不同的项目中来实现完全的独立性，此时在一个项目中的文件修改不会影响到其他的项目中的文件。此外，共享操作后形成的位于不同项目里的两个文件拥有一个共同的祖先，也就是他们的历史(history)记录是从同一点分离出来的。

如何实现分支操作

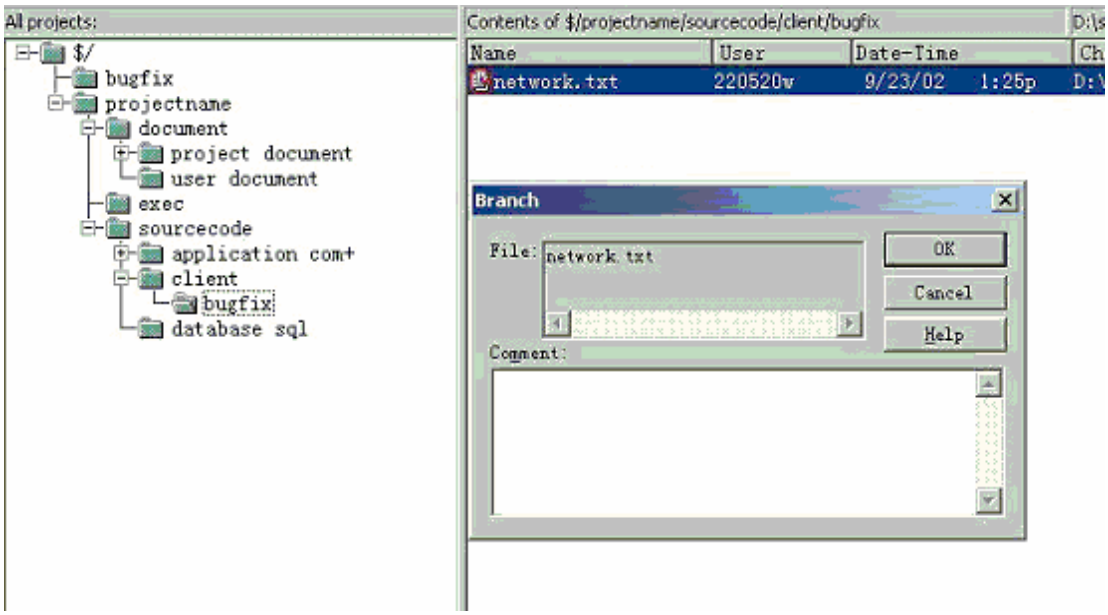
注意，在进行分支操作前，你必须确认已经共享了该文件。当然，你也可以将共享、分支操作放在一起完成。

1、 在VSS Explorer中选择目标文件。

2、 选择SourceSafe菜单，单击Branch显示Branch对话框。

3、 如果需要，你可以在Comment框中加入注释。

4、 单击OK。



附图三：分支操作

操作完成后，你将会发现项目$/projectname/sourcecode/client/bugfix中的文件network.txt由check out状态变成了uncheck out状态，而且改变项目$/bugfix中的network.txt文件对它也没有任何影响。

如何在具体的应用中使用这些特性

那些书上的大侠们学好本领后，都要到江湖上去闯荡一番。那我们了解这些基本的操作后，也一定希望能够在实际的项目中小试牛刀，我们就以早期版本中的bug修改为例。

我们假定我们项目的2.0版本刚刚完成，项目开发小组继续朝着3.0版本前进，同时试用项目维护人员需要一个临时的2.1版本来修改试用过程中发现的bugs。具体的步骤如下：

1、 将当前项目$/projectname/sourcecode/client加上标签(Label)--Version 2.0。

2、 继续在该项目上进行修改，形成新的版本。（如附图四）

3、 这个时候在版本2.0的试用过程中发现错误，你需要一个临时的版本来修改错误同时又不影响版本3.0的开发。

4、 选择Tools菜单，单击Show History显示Project History Options对话框。

5、 选中Include Labels复选框。

6、 单击OK显示History of Project对话框。

7、 选择加有标签"Version 2.0"的版本。

8、 单击Share显示Share From对话框。

9、 选择将要产生的项目的父项目，我们选择$/。

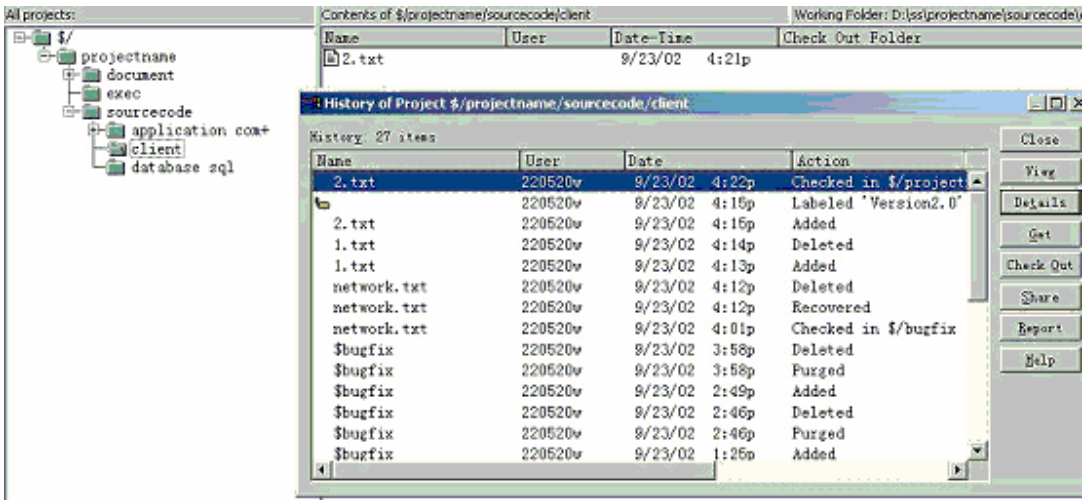
10、单击OK显示Share对话框。

11、将此项目命名为bugfixAfterV2.0，单击Close退出History of Project对话框。

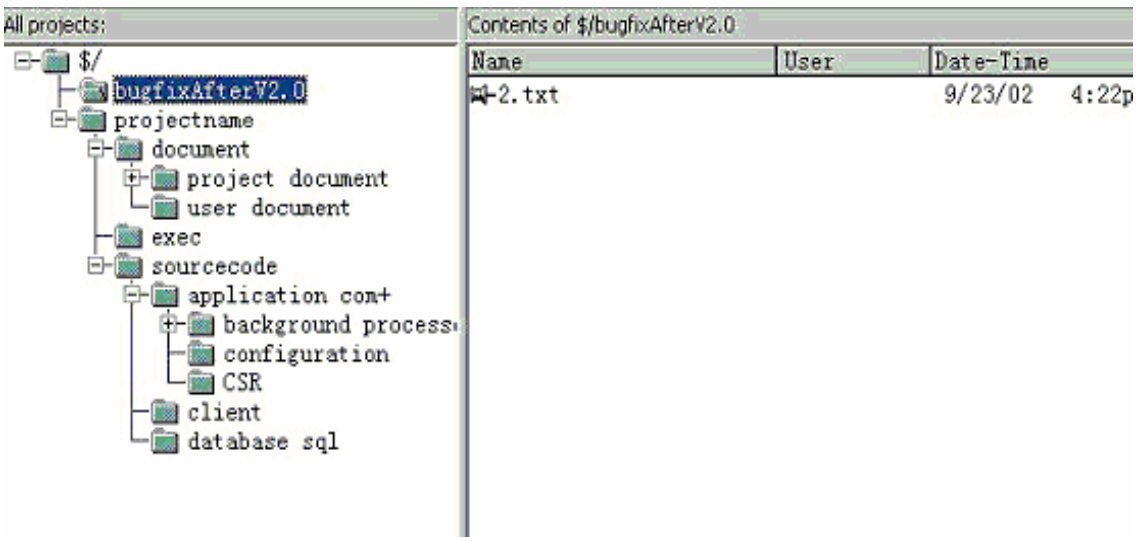
12、操作完成后会发现在VSS中增加了一个项目$/bugfixAfterV2.0（注意此时文件2.txt前面的图标形状，如附图五），试着check out文件2.txt，系统会提示你所有的文件已被"钉住

"(pinned)，操作不成功。是的，你还需要下一步操作。

13、选定那些你确实需要修改的文件，然后进行分支操作。这样你就可以任意修改这些文件，而且你会发现图标也恢复到原来的样子。



附图四：分支操作前的项目历史记录



附图五：分支操作完成后

6. 文件归并(merge files)

正所谓，分久必合，合久必分。分支操作以后，你肯定需要重新将这些文件合并到一起，比如上面那个例子，你肯定希望在版本2.0中被修改了的错误不要在版本3.0中再出现，这个时候你就需要用到归并操作。

所谓文件归并就是将由一个文件产生的多个不同拷贝重新形成一个唯一的新的文件版本，一般都是将分支上的改变反映到主线上，所以称之为归并操作（当然还存在其他两种情况，使用multiple checkouts，某些情况下get一个文件的时候）。在文件归并过程中VSS并不能去决定文件差异的取舍，它只是将这些文件间的异同提交给你，由你自己来确定最后的文件内容。当然比较的基准就是我们前面提到的它们共同的祖先。

有两种方法可以进行归并操作，一种是默认的visual merge，另一种是manual merge。一般推荐使用前一种方法。

文件归并过程中涉及到两个概念，一个是投送者(contributor)，另一个是目标(object)。投送者就是你在分支上的那些文件，目标就是你在主线上的那些文件，归并操作完成后，只有目标的内容改变，而投送者的内容保持不变。

如何实现归并操作

1、 在VSS Explorer中选择目标文件或者项目。

2、 选择SourceSafe菜单，单击Merge Branches显示Merge To对话框。

3、 在Projects框中选择投送者所在的项目名称，所以说该对话框的名称应该为Merge From，而不是Merge To。

4、 单击Merge显示Comment对话框。

5、 输入注释，点击OK。

6、 如果两者之间的差异很明显（比如一个文件比另外一个文件多出一行），系统会自动帮你决定目标文件的内容。反之，系统显示Visual Merge对话框，你可以自己决定目标文件的内容。（3）

操作完成后，查看一下目标文件是不是变成了你希望的内容。

好啦，至此所有的工作已经完成。你可以放松一下来杯浓茶，不敢喝咖啡，怕被Sun公司控告侵权。但是中国人有个习惯，不管什么东西都喜欢分出个高低，排个名次。我记得小的时候对隋唐演义中的好汉排名津津乐道，什么第一名李元霸，第二名宇文成都等等。那好，我们也来啰嗦一下，评评VSS的功过是非。（4）

7. 评说 VSS

在支持并行开发方面，VSS提供了大量优秀简洁的特性，但是在以下几个方面稍显不足：

1、 在进行分支操作的时候，灵活性不是很大，只能是单一地选择某个项目中的所有文件，不能进行一些自定义的设定，比如我只希望选取所有的文本文件。

2、 由于VSS中主线和分支是采用不同的项目来区分，所以很容易被混淆而且增加项目的数量，不如采取版本号来区分直观。

3、 在进行归并操作时，投送者与目标的概念很模糊，而且没

有显示地提出这两个概念。

4、 由于VSS中没有版本树的概念，所以分支操作后没有一个直观形象的版本演化的感官认识，就像Rational ClearCase中那样，这样使得不能把分支和主线很好地联系在一起。

8. 总结

本文主要讲解一些VSS使用中的高级特性，使得你能够用来应付一些比较复杂的情况。但是，VSS毕竟只是一种工具，项目配置管理的成败主要取决于项目的配置管理策略。

9. 说明

1) 本文所有的例子都是基于Visual SourceSafe 6.0英文版，其他的版本可相应对照。

2) 这两个概念是从Rational ClearCase借用过来的。

3) 关于Visual Merge对话框的详细信息，可以参考VSS的帮助文档。

4) 当然本文不准备在整体上来评价 VSS，主要是从支持并行开发这个方面来说说 VSS 的不足之处，只要与 Rational ClearCase 进行对照。

CVS

一 CVSNT 的安装及配置

CVS是Concurrent Versions System（并发版本系统）的缩写，基于Unix体系中成熟的SCCS（Source Code Control System）和RCS（Revision Control System）开发，是一个开放源码的项目，目前已

是版本控制系统的主流软件。一个很常见的使用CVS的场合，就是开放源码项目。由于开放源码项目的开发者的分布性，对于版本管理的要求更加严格，而目前大部分的开放源码项目几乎都是采用CVS来管理源代码，CVS的标准性和强大可见一斑。

CVS采用客户机/服务器体系，代码以及各种版本存储在中心服务器内，每一个个体开发者开发时都首先从服务器上获得一份自己的拷贝，在此基础上进行开发，以避免直接影响服务器上的数据。开发者可以随时把自己的新代码提交给服务器，并通过更新获得代码的最新状态，保持与其他开发者的一致。

CVS对于网络是透明的，开发者可以使用客户端软件（几乎所有的平台上都有相应的客户端软件）在任何时候，任何地点通过网络来获取最新的代码。

对于Eclipse的开发者而言，Eclipse本身内置了CVS支持，不需要使用其他客户端软件，只要建立一个CVS服务器，就可以使用这一强大的版本控制系统了。

CVS起源于Unix/Linux平台，关于Unix/Linux平台下的安装使用的介绍文章很多，这里就不再重复，读者如果需要在Unix/Linux平台下建立CVS服务器，可以参考本文附录的相关资源。

在Windows平台上也有CVS的一个实现——CVSNT，CVSNT的安装有一定困难，这里我们做一个简单介绍。 CVSNT的安装

首先到CVSNT的主页http://www.cvsnt.org下载最新版本，目前是CVSNT 1.11.1.3 (Build 57f)。

CVSNT的安装有一些注意事项，请读者尽量按照下面所说的步骤来

进行安装，描述主要针对Windows 2000。如果读者在安装过程中还有问题，可以参考本文附录的资源中关于CVSNT的安装技巧的文章或邮件列表。

CVSNT可以安装在Windows NT4 服务器或工作站SP6，Windows 2000服务器或专业版，Windows XP专业版上。

以管理员账号登陆，首先修改环境变量。直接执行安装程序，很有可能在最后会出现无法创建路径变量的错误，为此我们首先修改环境变量，设定路径。假设我们要把CVSNT安装到D:\app\cvsnt目录下（与CVSNT相关的内容最好安装到NTFS分区上，也尽量不要使用含有空格的目录名或者文件名，虽然CVSNT已经尽量支持包含空格的目录名和文件名，但仍有可能出现问题），那么打开控制面板，系统属性，高级，环境变量，系统变量中的Path，添加上D:\app\cvsnt并保存设置。

接下来可以执行安装程序，修改安装目录，一步步完成安装。

从开始菜单的程序组中启动CVSNT配置程序Configure Server。这时应该看到服务器还没有运行（CVSNT作为系统服务运行），如果已经运行了，先把它停下来。

选择第二个选项卡Repositories，首先勾上Repositories prefix（数据库路径前缀）的选项。CVSNT中只有一个数据库路径前缀，在这同一个前缀下，可以有多个数据库。相应的，所有的数据库都位于数据库路径前缀对应的目录之下。这里我们假设数据库都存储在E:\work\cvsrepo下，点击省略号按钮来选择E:\work\cvsrepo作为数据库路径前缀。

点击下面的Add按钮添加数据库根，可以有多个。比如我们将/work作为我们的工作项目的存储根。注意添加时系统自动把已设定的

E:\work\cvsrepo作为了路径前缀。

选择第三个选项卡Advanced，勾上全部选项，包括Use local users instead of domain。设置临时目录，假设为E:\work\cvstemp。注意要保证临时目录的安全设置（右键点击目录属性，共享，权限）给所有帐号以完全控制权限，包括SYSTEM帐号。并且，绝对不能把临时目录设在诸如C:\WINNT\TEMP或者C:\Documents and Settings下的任何地方，因为这些地方对于用户的访问是有限制的。

点击应用以保存设置，这一点相当重要。

现在可以回到第一个选项卡，点击Start按钮，服务应该正常启动运行了。如果有问题，可以打开一个命令行窗口，输入path命令来检查路径是否已经设置正确，也许需要重新启动来使设置生效。

打开一个命令行窗口，输入如下命令，用你的实际计算机名和用户名替代尖括号内的内容，注意对于NT Server，不能用localhost作为计算机名，必须使用实际计算机名：

```
set cvsroot=:ntserver:<计算机名>:/work
```

这一命令通过设定cvsroot这一环境变量，设定/work为目前的cvs数据库根。这里使用ntserver模式，这一模式比较适合服务器就在本地的情形。它要求局域网或者域内所有机器的用户帐号相同，客户端使用Windows NT，Windows 2000或者Windows XP。pserver模式是缺省的，除非关掉2401端口，下面我们的Eclipse就是使用pserver模式。

```
cvs passwd -a <你的NT用户名>
```

这一命令设定CVS中的用户名和密码，输入后将提示你输入密码。

注意如果需要CVS 服务器同时以ntserver和pserver模式运行，那么密码最好不要和系统中用户的真实密码相同以保证安全。

这里的用户必须是服务器上的真实用户，不过可以给真实用户设定一个不同的使用名alias。使用命令：

cvs passwd -a -r <你的NT用户名**> <CVS**帐号别名**>**

必须注意，这些名字里最好不要使用任何空格。如果必须的话，可以用双引号括起来。

到此为止，CVS服务器已经初步设置完成，可以使用了。缺省情况下，服务器将作为NT服务自动运行。读者既可以使用命令行的CVS命令，也可以使用各种CVS客户端来连接CVS服务器，执行CVS操作。不过，下面我们主要介绍在Eclipse中通过内置的CVS支持来使用CVS系统。

二　WinCVS 的配置与使用方法

1、WinCVS 简介：

WinCVS是CVS的一个客户端软件，它运行在Windows上，用来在Windows上登录CVS服务器，然后进行一些CVS相关的操作与管理。由于当前很多的企业内部都采用Linux/Unix做服务器，而用Windows做客户端，所以，WinCVS与CVS服务器配合使用将组成最强有力的版本控制与管理的系统之一。

2、WinCVS 的下载与安装；

最新的WinCVS可以从

http://sourceforge.net/project/showfiles.php?group_id=10072地址下载到，也可以在http://sourceforge.net/project 上下载到最新的或其它版本的WinCVS。

下载到相应的版本后根据向导进行安装，已经要使用CVS的用户，安装这个WinCVS应该没什么问题吧！

3、配置 WinCVS：

a、一般选项的设置，选择**Admin->Preferences…**，出现如下界面：

第一、**Authentication**：用来配置**cvs**服务器的认证方式，可以从下拉框中选择其它的认证方式，不过一般只要选择默认的**pserver**方式就可以，要注意的是必须与**cvs**服务器配置时所指定的认证方式一致；

第二、**Path**：用来配置**cvs**在服务器上的主目录路径，也就是服务器上用进行**cvs**初始化的目录，如：**/home/cvsroot**；

第三、**Host Address**：用来配置**cvs**服务器所在服务器的地址，可以是**IP**地址，也可以是**DNS**名，如：**10.104.1.204**；

第四、**User name**：用来配置要使用些**WinCVS**来登录**CVS**服务器的用户名，如：**cvsyxwu**，用户的登录必须由管理员把其添加**cvs**用户组中；

第五、**CVSROOT**：此项一般都不需要用户进行修改，用户在输入上边的几个选项时，系统将自动根据用户的输入生成此项的相应内容。

b、全局选项的设置，在上一个界面上选择“**Globals**”：

此项的配置主要是要注意这几选项：

第一，**Checkout read-only**不要选上，否则，**checkout**出来的源代码将不允许用户进行

修改，并且此选项默认是选中的；

第二，**Prune (remove) empty directories**也不要选上，否则，会自动删除空目录；

第三，对一般配置没有特殊要求的，把**Dirty files support**、**Supply control when adding**

与**TCP/IP compression**选项选中；

4、登录服务器：

选择**Admin->login**，将出现如下对话框要求用户输入登录口令

输入口令后，选择“**OK**”按钮，如果**CVS**服务器与**WinCVS**的配置都没出错的话，将在**CVS**的状态栏中提示：

cvs -z9 -d :pserver:cvsyxwu@10.104.1.204:/home/cvsroot login

Logging in to :pserver:cvsyxwu@10.104.1.204:2401/home/cvsroot

******* CVS exited normally with code 0 *******

code 0表示正确的登录；而如果出错的话，将是**code 1**，那么要根据错误的提示进行相应的修改。

5、从 CVS 服务器上 check out 相应的模块：

第一，在**workspace**中的**Modules**选中要存放**checkout**模块的目录；

第二，选择**Create->Checkout Modeles**，将出现如下对话框：

其中，**Module name and path on the server**就是要存放**checkout**内容的目录，由用户输入；而**Local folder to checkout to**就是第一中用户所选择的目录。

6、修改之后把文件提交到 CVS 服务器

a)、只有一个用户对文件进行修改的情况

用自己喜欢的编辑器对**checkout**出来的文件进行修改，修改之后的文件在没有提交之前会是红色的，如下图**example.h**文件：

选中红色的文件**example.h**后右击选择“**Commit Selection**”选项，如果没有其它用户也对其进行修改并已经提交到**CVS**服务器上，一切正常的话将把**example.h**文件提交到**CVS**服务器并把图标恢复成原来的颜色。

b)、两个或两个以上的用户对同一文件的不同部分进行修改的情况

这种情况就是如用户A与用户B都checkout了文件example.h，内容如下：

```
int callby (int count)
{
   printf("ExcelStor!\n");
}
void main(int argv,char *argc)
{
//added by my cvs
   printf("I am Cather\n");
}
int mainexample()
{
   printf("OK\n");
}
```

然后用户A修改成如下，并提交到CVS服务器（一般将正常提交）：

```
int callby (int count)
{//add
   printf("ExcelStor!\n");
}
void main(int argv,char *argc)
{
```

```
//added by my cvs
   printf("I am Cather\n");
}
int mainexample()
{//add
   printf("OK\n");
}
```

接着用户B修改成如下：

```
int callby (int count)
{//add
   printf("ExcelStor!\n");
}
void main(int argv,char *argc)
{
//modified
   printf("I am Cather\n");
}
int mainexample()
{//add
   printf("OK\n");
}
```

当用户B选择“Commit Selection”时将提示：

```
cvs server: Up-to-date check failed for `example.h'
cvs [server aborted]: correct above errors first!
```

此时表明已经有用户对同一个文件example.h进行修改并提交到CVS服务器，这时

要先选择“Update Selection”对本地example.h与CVS服务器上的example.h文件进行

同步与合并，不用选中出现的任何选项，直接选择“OK”，这时将显示如下：

```
cvs -z9 update example.h (in directory C:\my cvs\STW\src\)
RCS file: /home/cvsroot/STW/src/example.h,v
retrieving revision 1.5
retrieving revision 1.6
Merging differences between 1.5 and 1.6 into example.h
M example.h

***** CVS exited normally with code 0 *****
```

表明用户B与用户A的修改已经合并成功，同时文件example.h的图标也将变成红色，合并后的文件是存放在用户B的本地机上，为了更新到CVS服务器还必须选中example.h并右击选择“Commit Selection”才能把用户A与用户B的修改合并后的结果提交到CVS服务器上。注：M表示此文件已经被修改过。

c)、两个或两个以上的用户对同一个文件的相同部分进行修改的情况

这种情况就是如用户A与用户B都下载了文件example.h，内容如下：

```
void main(int argv,char *argc)
```

```
{
   printf("I am Cather\n");
}
```

然后用户A把文件修改成如下，并提交到CVS服务器（一般将正常提交）：

```
void main(int argv,char *argc)
{
   printf("I am Cather\n");
   printf("I am Pat\n");
}
```

接着用户B又把文件修改成如下：

```
void main(int argv,char *argc)
{
   printf("I am Cather\n");
   printf("I love you Cather\n");
}
```

如果用户B这时选择“Commit Selection”准备把修改结果提交到CVS服务器，此

时将显示如下的错误提示：

```
  cvs -z9 commit -m "update in 11:20" example.h (in directory
C:\my cvs\STW\src\)
cvs server: Up-to-date check failed for `example.h'
cvs [server aborted]: correct above errors first!
```

```
***** CVS exited normally with code 1 *****
```

表明用户B的修改与其它用户的修改冲突，这时要先选择“Update Selection”，将显示如下提示：

```
 cvs -z9 update example.h (in directory C:\my cvs\STW\src\)
RCS file: /home/cvsroot/STW/src/example.h,v
retrieving revision 1.9
retrieving revision 1.10
Merging differences between 1.9 and 1.10 into example.h
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in example.h
C example.h

***** CVS exited normally with code 0 *****
```

example.h前面的C表示与其它用户的修改有冲突，并且文件的图标会加显示一个“C”，如下所示：

双击example.h将显示example.h的内容，如下：

```
  void main(int argv,char *argc)
{
   printf("I am Yanxi\n");
   printf("I am Cather\n");
<<<<<<< example.h
   printf("I love you Yanxi,too!\n"); //这部分为你的修改
=======
```

```
        printf("I love you Cather!\n"); //这部分为其它用户的修
```
改

```
    >>>>>>> 1.10
    }
```

这时你应该与用户A进行协商以决定最终要怎样修改。比如，可以修改成：

```
    void main(int argv,char *argc)
    {
          printf("I am Yanxi\n");
        printf("I am Cather\n");
        printf("I love you Yanxi,too!\n"); //这部分为你的修改
        printf("I love you Cather!\n"); //这部分为其它用户的修
```
改

```
    }
```

然后选择“Commit Selection”进行提交，将显示如下的提示信息：

```
    cvs -z9 commit -m "update in 11:20" example.h (in directory
C:\my cvs\STW\src\)
    Checking in example.h;
    /home/cvsroot/STW/src/example.h,v  <--  example.h
    new revision: 1.11; previous revision: 1.10
    done

    ***** CVS exited normally with code 0 *****
```

表明用户A与用户的修改已经合并成功。

这样，向CVS服务器提交文件所会遇到的问题也基本上就是这些，用户要根据所遇到的实际问题进行修改。

7、向 CVS 服务器添加新文件

在本地添加文件后，要提交到服务端。先选中文件，然后点击“添加按钮”，添加文件后，再在右键菜单中选择提交命令“**Commit Selection**”即可。

如图，选中文件**example.h.bak**，因为**example.h.bak**当前不是CVS的文件，此时“添加按钮”将由不可选状态变成可选状态，所以**Status**中显示为“**NonCvs file**”，选择“添加按钮”之后**example.h.bak**图标将变成红色并增加了一个**A**字母，如下：

然后选中**example.h.bak**，右击，选择“**Commit Selection**”把文件**example.h.bak**提交到CVS服务器上而成为CVS的一个文件。

8、结束语

读了本段，你基本上已经能为自己或公司配置一个实用的CVS服务器与WinCVS客户端，配合使用CVS进行系统开发或其它文档的版本管理与控制

第二部分　相关资源

第一章　一些相关概念

1　基线的理解

“基线”是一个很常见的术语，在配置管理和项目管理里面都能看到，而且还有很多衍生的术语，例如基线提升、基线化、基线审计，等等。

我个人以前对微软的那套开发流程（就是**product cycle model**）以及**PSP**、**TSP**了解比较多一些，这些流程里面对“基线”的概念提的不多。但 接触**RUP**、**MSF**以及项目管理以后，看到到处都有**baseline**，就觉得迷惑了。

经过我自己的理解，以及和几个同事的讨论，现在我觉得我们通常看到的“基线”这个术语有两个意思：

1）代表多个源代码文件的一组版本。

比如有三个文件，**aaa.c**、**bbb.c**和**ccc.h**。可以对这三个文件做一个基线，取**aaa.c**的版本**1.1**，取**bbb.c**的版本**1.3**，取**ccc.h**的版本**1.0**。**(1.1,1 .3,1.0)**就是一个基线。换句话说，通常在**vss**和**cvs**里面做**label**，就是在做基线。

这种基线对“构建审计”特别有用：在做**build**的时候，可以先对所有源文件做一个**label**，取名为**"Build2394"**，然后再编译、集成。这样，以 后如果要找到和**build 2394**对应的原文件，只需要到**vss**或者**cvs**里面把所有文件对应**label Build2394**的版本取回来就可以了。

2）代表文档的一个稳定状态。

比如有一个项目设计文档，当设计基本完成，开发即将开始的时候，需要把这个文档固定下来，内容不能再频繁改变，否则开发人员就无所适 从了，可能导致每个人所参照的文档并不是同一个文档。用一句上海这里的生活用语来说，就叫做要把这个文档“敲定”。

一个文档如果经过讨论被通过了，被固定了，就可以说这个文档被“基线化”了，然后所有人就可以在这个“基线”的基础上工作。

当然，文档不可能一成不变，所以当对文档的修改仍然会不断进行，但这种修改并不会随时随地的添加到被“基线化”了的文档中去。因为既然是“基线”，就不能随便动。

但是到了一定时候，修改积累到一定程度，就需要把很多修改合并到原来的文档中去了，并生成一个新版本的文档作为团队中所有的人的参考 标准，并把老的版本淘汰掉。这就叫做“基线提升”。

合同基线

当你和客户讨论后，“敲定”的合同

4）发行基线

你会对你要发行的代码，文档版本进行**label,** 比如 **Release2.2,**

这样，你可以随时取出此版本作**build**，进行测试，发布。

5）产品基线

当发布时，你会对产品中所有的配置项进行**label,**包括可执行命令，文档手册，库文件。。。

基线是与里程碑相对应的，所以每个里程碑处建立一条基线；

但变更同样属于配置管理的范畴，所以配置项变更后也应建立基线。

但是如果配置项的变更很微小，就没有必要再建一条基线了。

2 配置状态报告及申计

配置状态报告：

提供配置项的状态信息，以反映项目的进展情况，同时可以从中根据配置项的操作记录对开发团队的工作关系作一定的分析。

-报告应定期进行

-使用**CASE**工具生成，保证客观性

主要内容：

变更请求列表

基线库状态

Release信息

备份信息

SCM工具状态

其它应予以报告的事项

配置审计：

验证配置项信息与配置标识（需求、标准、流程…）的一致性

功能配置审计**(FCA,Functional Configuration Audit)**

物理配置审计**(PCA,Physical Configuration Audit)**

1.配置审计**-What**

FCA:验证配置项的功能特性与需求（原始需求、变更请求… ）的一致性

PCA:验证配置项的物理特性与期望（命名标准、变更流程… ）的一致性

2.配置审计**-Why**

预防提交错误的产品

捕获未完成的任务（原始需求、变更请求）

识别不同配置项之间的对应关系

确认配置项（们）已经被基线化

确认记录和文档维持可跟踪性

3.配置审计**-When**

软件交付或**release**时

每个阶段结束时

对于维护性项目，周期性地进行

4.配置审计**-Who**

非本项目组成员

其它项目中的配置控制者

内部审计者

SCM小组

5.配置审计**-How**

审计流程：

识别配置审计的时间**[PM]**

指派审计者**[QA/Audit Group]**

定义审计范围**[PM&Auditors]**

准备配置审计**Checklist[Auditor]**

通过评审（**Review**）、文档记录进行审计**[Auditor]**

识别不符合项**[Auditor]**

关闭不符合项**[PM]**

验证**[Auditor]**

3 配置管理标识规范

1 定义

配置项是受配置管理控制和管理的基本单位。配置标识是软件生命周期中划分选择各类配置项、定义配置项的种类、为它们分配标识符的过程。配置项标识的重要内容就是对配置项进行标识和命名。

2 必要性

配置标识是软件配置管理的基础性工作，是管理配置的前提。很难想象缺乏配置项标识的基线管理，如何去区分和定义基线；也很难想象缺乏配置项标识的变更控制，如何去标明版本的变化规迹。

3 原则

配置项标识可以根据项目的实际情况灵活掌握，但有一些基本的原则是需要遵从的。

- 标识唯一：这是为了避免混淆
- 与同类配置项不同的信息，应纳入标识：这是为了便于区分、查找
- 同类配置项的标识方法统一
- 容易记忆：对于经常使用的配置项，标识不宜过长

4 标识范例

4.1文档

对于常见的**word**、**excel**、**visio**等文档而言，标识也就是文档的文件名。文档的标识可能承载好几项信息，如项目编号、文档名、撰写时间等等。标识里要包括哪些信息，是和文档的性质有关的。比如每周产生的一次周报，标识里加上撰写时间，就一目了然了，这样便于查找和

整理。

不同的信息之间，可以用下划线“_”分割，也可以用括号“（）”分割，如果前后两项信息分别是字母、汉字；或汉字、数字等情况，不必加分割符号也是可以的。

以下是常用文档的标识规范，供参考：

? 次要的文档

命名方式：[文档名]

如：配置库使用规范

? 比较正规的文档

命名方式：[项目编号＋文档名]

如：**GC21007E**技术规划书

? 有版本变化的

命名方式：[文档名＋版本号]

如：**GC21007E**需求规格说明书**V1.2**

? 主要区别在于时间的

命名方式：[文档名＋撰写时间]

如：周例会会议纪要**20030901**

? 主要区别在于人员的

命名方式：[文档名＋人员名称]

如：工作周报_赵明

? 主要区别在于系统模块的

命名方式：[文档名＋模块名]

如：需求实现跟踪表_申报征收；

? 为了能排列顺序，需要加上数字的

命名方式：[序号＋文档名]

如，某一测试流程，不是写在一个文档里，而流程的执行是有时间顺序的：

02银税联网的电子税票日常对帐、03现金票证汇总

? 其它，加上能和同类文档区别的文字

命名方式：[文档名＋简单文字]

如：联调环境确认_暂行；联调及修改记录_模版；GC21007E项目WBS计划_提交稿

4.2版本号

版本号是比较抽象的配置项。

? 方式一：以版本发布日期作为版本号

这种版本号的标识方法比较简单，虽然无法看出系统一共发布的版本数量，以及各版本之间的不同性质，但可以直观的看出版本的发布日期。也比较容易记忆。

如果项目组成员需要对版本的情况进行统计或者沟通，不需要依靠任何记录就可以定位出某个版本发布的日期。

适用于：小型项目、版本频繁发布的项目。

举例：2003-9-7发布了两个版本，分别是20030907A，20030907B

应用的项目：ctais2.0。

? 方式二：采用“主版本号．从版本号．维护版本号．补丁版本号”的形式

这是大部分系统版本号的基础标识形式，每个版本号是阿拉伯数字，主版本号以1为起始数字，其他版本号以0为起始数字。可以进行裁

减，类似“主版本号．从版本号”的形式也是可以的。以这种方式来标识版本之后，当前版本的状态，以及版本发布的轨迹，都可以看得比较清楚。

至于何谓主版本和从版本，每个项目组可以有自己的约定。比如功能的大幅度修改、正式发布给客户、上线等里程碑事件，都是应该反应在版本号的变化上的。

适用于：比较正式的软件项目

举例：第一个版本为 **1.0.0.0**，上线使用的版本为**5．11.18**

? 可选方式三：给版本加上前缀以区分

方式三、四并不是一种新的软件版本标识方法，它们通常增加在方式二的基础之上，以更好的标识版本。

我们通常在软件前面看到**alpha**、**beta**、**demo**、**release**等前缀，这些标识有它们公认的含义，主要以区别版本的性质和用途。以下是参考资料。

Alpha版(内部测试版)：一般只在软件开发公司内部运行，不对外公开。主要是开发者自己对产品进行测试，检查产品是否存在缺陷、错误，验证产品功能与说明书、用户手册是否一致。

Beta版(外部测试版)：软件开发公司为对外宣传，将非正式产品免费发送给具有典型性的用户，让用户测试该软件的不足之处及存在问题，以便在正式发行前进一步改进和完善 。一般可通过**Internet**免费下载，也可以向软件公司索取。

Demo版(演示版)：主要是演示正式软件的部分功能，用户可以从中得知软件的基本操作，为正式产品的发售扩大影响。如果是游戏的话，则只有一两个关卡可以玩。该版本也可以从**Internet**上免费下载。

Release版(发行版)：不是正式版，带有时间限制，也是为扩大影响所做的宣传策略之一。比如**Windows Me**的发行版就限制了只能使用几个月，可从**Internet**上免费下载或由公司免费奉送。

Full Version版(完全版)：也就是正式版，是最终正式发售的版本。

举例：**beta2.0**，**release1.02**

适用于：较正式的软件、商用软件、游戏软件等

? 可选方式四：使用外部版本、内部版本两套机制

当我们点击**word**帮助菜单的“关于”时，在抬头部分看到的是“**word2002 （10．2627．2675）**”。这里的**2002**、**10．2627．2675**分别是外部、内部版本号。因为让客户记住繁琐的内部版本号是困难的，另一方面，出于宣传等商业原因，很多商业软件不但有外部版本号，还把功能有大幅度改善的版本以不同的名称命名。比如**windows98- windows Me- windows2000- windowsXP**； **RealPlayer8.0- RealOne Player V2.0**等。

适用于：商业软件

4.3 其他配置项

以下配置项没有对应实际的物理文件，但通常都由配置人员分配（特别是在小型项目中）。另一方面它们和项目组每个成员相关、经常被使用。对它们的标识需要容易记忆、规则统一，但由于涉及安全性，不能过于简单和容易猜测。

? 配置机、服务器的访问密码

例：用户名 **guest**；密码 **ctais123**（若有多台服务器，其用户名和密码最好都一致，容易记忆）

? **VSS**数据库、**butterfly**等配置工具的用户名和口令

例：用户名 **zhouyan**；初始密码 **zhouyan**（所有人员的命名规则都相同，**VSS**的和**butterfly**的最好一致）

? 数据库的**SID**、用户名和口令

例：**IP**为**10.1.120.11**的服务器，在上面搭建的数据库**SID**为 **11sid**；（由于有多台服务器设有数据库，所以**SID**最好和服务器的**IP**相关）

给测试人员使用的库，用户名 **test**；密码 **test**；

给开发人员调试使用的库，用户名 **dev**；密码 **dev** （用户名和密码表明了其功能和作用）

5辅助标识

配置工具有一些辅助方法来标识配置项，这里介绍**VSS**和**CVS**的方法。

5.1 VSS

VSS提供以下三种方式为配置项标识

? **version**

version通常是纯数字型的，它由**VSS**自动管理。每当用户对一个配置项进行了修改并提交变更（**check-in**）、分支等操作后，**VSS**都会将配置项的**version**自动加**1**。

Version的最大好处是能一目了然的看出配置项的变更规迹。在计算配置项的变更次数、在比较配置项两个版本之间的差异时，发挥了较大的作用。

? **user/date/time Stamp**

user/date/time Stamp也是由**VSS**自动管理的。和**version**类似，每当用户对文件做了更改、分支、删除等操作时，**VSS**都会自动记录操作

的时间和用户名。

严格的说user/date/time Stamp并不是标识，只是在配置项发生变化时，对修改人员、时间信息的记录。它在跟踪配置项变更、跟踪责任人方面，起到了很重要的作用。对于一个多人的团队，某一配置项可能由多人修改。如果没有user/date/time Stamp，当我们发现一个公共的源代码被更改有误之后，如何去查找修改人呢？如果没有user/date/time Stamp，如何去取得某一配置项在20日产生的版本呢？

? label

和以上两种方式不同，label是用户自发给某个配置项、或某些配置项的当前version创建的标识信息。因为同一配置项可能经历过多次变化，有对应的version，label能对应某个具体的version是最确切的，否则就很难通过label来查找和定位配置项的某个状态了。

Label之后， 配置项的version号自动加1，并和配置项做当次label的上一个version相对应。也就是说，我们可以选中某个label，对它进行view、get的操作，取得的内容是所选label的上一个version的内容。

如果一个或一组配置项达到了某种里程碑状态，配置人员想标明这种状态，使用label是个不错的主意。它可以在配置项的某个版本上随意增加一些我们想保留的信息。label是最长可达31个字符的字符串，比如“beta2.1.5”，“08.25评审通过”，“客户环境第一版”等等。

5.2 CVS

CVS有以下两种方式来标识配置项：

? revisions

和VSS的version一样，revision也是CVS自动分配的，通常操作人员不需要过于关心它变化的细节，只需要知道一个配置项的每个

revision都是唯一，并且对应配置项的某个版本就可以了。之所以配置项的版本称作revision，是为了和软件项目的版本release区别。

配置项的初始revision是1.1，通常情况下，CVS是以1.1?1.2?1.3这样的趋势去变化配置项的revision的。

增加一个新文件的时候，第二个数字将为“1”， 第一个数字等于该目录中所有文件版本号第一个值的最大值。例如，当前目录包含版本号为1.7, 3.1和4.12的文件，则一个新文件的版本号应为4.1。

? tag

和VSS的label有一些类似，tag也是用户给CVS里的某个或某些配置项创建的标识。但CVS对tag的名称控制的比较严格，一个tag必须以字母开头，后续多个字母，数字，减号和下划线，不能包括点号和空格。比如“release1-3-2”，“approvdBy Sqa”等。

Tag的作用和VSS的label一致，也是给配置项标明一些我们想保留的信息的。

6 总结

从以上的标识规范可以看出，配置项标识是一件比较细致的工作，也是配置管理的基础。每个项目的具体情况不同，可以根据以上的规范进行适当裁减与修改。但遵循一定的规范是相当必要的，只有这样，才能方便配置项的查找与规类，才能较清晰的看出配置项的状态，给基线控制、变更控制工作的开展创造基础。

4 配置管理经验

在整个项目过程中如何进行配置管理工作：

*1、由**PL**指派该项目的配置管理者（**CC**）。*

由**PL**指定比较重要的设计人员或有经验的项目组成员作爲該项目的配置管理者，配置管理者全面負責該項目的配置管理活動。

***2**、由配置管理者作成配置管理计划。*

配置管理者與**PL**充分溝通，了解項目的各種情況（项目的流程，客户的需求），识别配置项和为存放配置项提供的配置库等情況。配置管理者根據了解的情況，編寫該項目的配置管理计划。

***3**、由**PL**组织评审配置管理计划。*

根據作成的配置管理计划，由**PL**组织评审配置管理计划。對於评审中發現的錯誤，配置管理者應迅速改正。

***4**、基线化配置管理计划*

對於通過评审的配置管理计划进行基线化。以后对项目的配置管理计划所发生的变更处理，都应根据**CI**的变更控制处理流程来执行

***5**、搭建配置管理环境*

配置管理者根據配置管理计划中的規定，搭建配置管理环境。

***6**、获取配置项*

配置管理者根據配置管理计划中的規定，获取配置项，並放入配置管理庫中。

***7**、培訓項目組成員對配置庫的使用*

配置管理者培訓項目組成員對配置庫的使用，以使項目組成員熟練使用配置庫。

8、開始配置管理活動

（1）定期對配置庫進行备份，儅配置庫損壞時，可通過备份數據及時恢復。

（2）对通过评审的项目的配置项进行基线化。以后对此配置项所发生的变更处理，都应根据CI的变更控制处理流程来执行

简述配置管理工作流程示意图：

配置管理工作流程目的是概述描述开展SCM活动的整个过程活动，为参与SCM的人员提供指导。描述了为达到CMM2级（可重复级）所必需的SCM过程。

SCM的目的是在项目软件生命周期中建立和维护软件项目产品的完整性。SCM关注以下四个关键过程。

- 配置标识
- 配置控制
- 配置状态记录（CSA）
- 配置审核

流程图以这四项活动为重点进行了说明（见图1）：

（1） SCM实施准备

为项目实施SCM获取适当和充分的资源（人员和工具）、资金和高层管理者对实施SCM的支持。此外，准备工作还包括为项目建立SCM组织机构，包括SCM经理、软件配置控制委员会（SCCB）和SCM组或配置管理员。

SCCB*应代表项目经理和所有受软件基线变更影响的组的利益，其负责*：1）授权建立软件基线和对配置项/单元的标识；

2）评审和授权对软件基线的变更；

3）授权从软件基线库生成产品。

SCM组协调或实施：

1） 建立和管理项目软件基线库；

2） 制定、维护和分发SCM计划、标准和程序；

3） 标识受控于SCM的软件中间产品；

4） 管理对软件基线库的访问；

5） 变更软件基线；

6） 从软件基线库生成产品；

7） 记录SCM活动；

8） 制定和分发SCM报告。

（2）编制软件配置管理计划（SCMP）

SCMP应在项目总体策划的初期制定与软件项目策划同步进行。SCMP描述预期要在项目生命周期中开展的SCM活动，并阐述支持这些SCM活动的环境和工具。批准的书面的SCMP用来指导项目整个配置管理活动的开展。

该计划覆盖SCM执行的活动、进度、所需资源（包括人员、工具和计算机设备）、人员职责分工和项目成员所应遵循的SCM程序。

（3）补充（制定）SCM作业程序

SCM作业程序是指导SCM人员实际操作的文件。企业应具备基本的SCM作业程序比如配置标识、配置控制等，但针对具体项目需要可以补充相应的作业程序。

（4）提供SCM培训

按照需要对SCM组成员进行有关实施SCM活动的目的、作业程序和方

法方面的培训，比如SCM概念和SCM工具；对软件开发人员和软件相关组人员进行实施SCM活动的培训。

培训可能包括：

针对SCM组的：

- SCM标准、作业程序和方法
- SCM工具

针对软件开发组和软件相关组的：

- 在组内实施SCM活动应遵循的标准、作业程序和方法
- SCM组的任务、职责和权限

（5）执行配置标识

配置标识一词用于定义产品生命周期某点上软件集合的文档。配置标识是选择、标识和命名软件配置项（SCI）的过程。

SCI是作为单一项开发和管理，能独立于其它同类项启动和运行，来执行某一满足最终用户需求功能的软件的集合。例子有：代码模块、构件脚本和文档。

标识通常由简短的名称和版本号组成。唯一地标识使SCMG能准确跟踪和控制SBM项目地软件变更，和维护软件中间产品地多个版本。

（6）执行配置控制

配置控制主要包括配置管理库运作和变更申请处理..。配置管理库运作即以有组织的方式组织配置项、基线的登入和检出，从受控库发行基线和软件产品。 对配置项的变更是由提交对项目的SCM系统的变更请求开始的。变更

请求应具有一种机制，使与项目有关的任何人员（如，开发人员，

质量工程师，经理，用户）可提出问题或建议。

（7）配置状态记录(CSA)

CSA文档记录并向**SBM**软件项目经理汇报影响**CI**的措施。通过使用工具能让经理们能获取他们所需要的信息。**CSA**主要记录“批准的配置”（基线）和对基线变更的实施状态，为经理们提供反馈信息来确定**SCCB**的决策是否按自由化照指示实施。

（8）软件配置审核

SCM审核一般分为两类：功能配置审核（**FCA**）和物理配置审核（**PCA**）。这两种审核共同来保证项目开发出为进入下一阶段所必需的产品。

功能配置审核的目的是保证在需求和策划阶段规定的用户需求得到文档化，并转化到产品生命周期的设计和构造阶段。

物理配置审核的目的是检查产品已经生成，并在建立基线和进入下一生命周期阶段前进行了配置控制。

5 经验谈：

●目前的配置管理还处于一种版本控制的管理和可追溯性,我目前所做的配置管理是帮助我现在的公司建立一套完整、规范的配置管理体系，让软件的开发过程是可控的、受控的，每一配置项是可标识的和可追溯的，我想请教你，

做了五年的配置管理，你的主要经验是什么？还有配置管理与开发人员的关系问题你是如何处理的？再就是公司应该赋予ＳＣＭ人员什么样 的权利和职责，才能保证ＳＣＭ的管理活动的有效，而不流于形式。谢谢你，如有可能

▲做了这么久的**SCM**，我的主要收获是要做好公司的配置管理工作，

需制订一套适合公司开发项目的配置管理规范文档，并培训项目人员熟悉了解 配置管理流程，知道自己在里面的角色．这样使他们能在项目的每个阶段配合SCM活动的实施，确保进入配置库的工件是受控的、可用的且可追溯的。

SCM计划是通过项目组评审的，那么开发人员就应按照SCM计划在每个里程碑点上提交审批过的文档或代码类文件；若没按时提交，SCM人员可依 项目计划去追查项目人员．基于你的第三个问题，我觉得要保证SCM的活动不是流于形式，首先是你的SCM计划是参照配置管理规范做出来且经过项目组评审认可的。那么 ，在实施SCM活动时，SCM人员就有权利在这个里程碑点上去追查项目组应提交的文档或代码类。

为了不让项目组人员觉得增加了SCM反而是增加项目组开发过程的负担，你就得在配置管理规范里定义清楚，哪个阶段作为里程碑点这很重要。 我们现在一般分为三个重要里程碑点:需求分析\实现\测试.

在实现过程中,就是完全进行这个项目的详细设计及代码开发过程。 所以,在实现阶段开始时就要求项目经理逐渐细化项目计划,分出各个内部测试版的提交时间作为小里程碑点,这样就便于SCM在小里程碑点上去追查测试人员的测试用例及版本发布.进行各小里程碑点上的版本入库管理、标识。而实施严格的变更控制,1,一般在于当需求发生变更,就得由项目经理 根据变更的级别决定是否召开SCCB会议.当确定变更后由项目经理提交,SCM人员记录此次发生的变更;2. 当发布一个Beta版给客户后,所有的变更都要求进行控制,项目人员需要提交变更申请表,项目经理(PM)根据变更 申请表确定是否召开SCCB会议,小的变更由PM决定,当提交变更给SCM时,就需提交与上个版本的文件异常列表及变更描述.SCM人员在检查变更符合时,提交给测试人员,测试人员测

试需求满足时,再向SCM汇报,SCM将此版本纳入基线标识.并将变更记录纳入文档.在每个里程碑点之后提交配置变更记录报告到高层领导及项目经理。

●目前我们在公司建立的配置管理系统是从产品规划开始，我们有一个委员会根据市场信息＼行业背景＼调研报告负责提出新产品定义和产品计 划，并提供给研发中心进行策划，在开发过程我们设定了概要需求基线，详细需求基线，概要设计基线，详细设计基线，产品基线，产品开发阶段我们分了产品定义，产品计划，开发策划，需求，设计，编码与单元测试，客户化 应用，测试，产品发版，开发总结，维护等，我们编写了＜配置管理程序＞，＜配置项管理规范＞和＜配置项和版本控制规范＞，不知道这么做，分得这么细，是不是让开发人员觉得很烦，有什么可以精简的，可以使配置管理既 切实可行，又适合开发人员的开发习惯.

▲依你所说你们公司SCM的规划，在我看来，可以分为重点四大类. 系统文档，数据库，编码，客户文档，这样就会清晰许多. 在系统文档内又 可分出：计划、需求、概要设计、详细设计、测试等这些类别了。做基线标识时就分别在各类别的目录上建立标签即可。

从你说的你们编写的SCM文档来看，我觉得分得还不够明确。或者分为〈配置管理控制程序〉〈配置管理规范〉〈配置计划编制程序〉是不是 会好些呢？

〈配置管理控制程序〉来描述整个配置管理的工作程序之类，而不细化它的活动及流程。比如ＳＣＭ计划的前提是什么？如何度量ＳＣＭ工作 ？等等。

〈配置管理规范〉来描述整个配置管理活动规范与标准，象标识、

版本管理、变更控制的执行标识和规范在里面安排。

〈配置管理计划编制程序〉来描述适合于不同项目与产品的一个模板。包括目的、范围、角色、ＳＣＭ活动（SCCB的确定、配置库的构架、配 置项的建立与入库）、环境说明、SCM审核、SCM报告、培训这些。

其实象这些文档，需要要项目人员熟悉的就是

1.他们在里面的角色是什么

（从他们知道自己在ＳＣＭ里面的角色，来了解自己要配合ＳＣＭ什么活动的执行）

2.提交文档与变更的流程。

（从他们清楚2点，来了解自己在哪个时候有责任提交完成（或变更）代码文档类。）

其它内容仅是用于公司多个SCM的情况下统一、规范ＳＣＭ工作流程与ＳＣＭ活动。

" ★若没按时提交，SCM人员可依项目计划去追查项目人员。

基于你的第三个问题，我觉得要保证SCM的活动不是流于形式，首先是你的SCM计划是参照配置管理规范做出来且经过项目组评审认可的。那么 ，在实施SCM活动时，SCM人员就有权利在这个里程碑点上去追查项目组应提交的文档或代码类。"

我可告诉你，你是没权去要提交的配置文档的。你按计划，本来你的计划是随项目计划的变更而变更的，因为你的计划是项目计划的附属计划 ，文档没提交可能有多种原因，这在SPTO、SQA报告中可以反映出来，他们会管些东西。你管你的配置标识、建立和维护配置库、配置项的变更及变更的控制就可以了。至于对配置库的审核那也不是你的工作，SQA会来检查你的。

▲SCM计划是项目计划的附属计划这点说的没错,但如果说项目计划在那个时间点上没提交产出物却没提交,且项目计划时间没变,那么请问是否SCM 有权利去追问呢?? 当然文档没提交可能有多种原因,那么延迟提交的原因是否PM应该向相关组及时反应才对呢?若没反应,项目计划与实际应存在差异了. 不过确实这些原因应该在SPTO与SQA报告能反应出来,不过遗 憾的是我们公司这方面还做的尚未成熟(我想许多公司若SCM没做好,其它方面应该也处于尚未成熟阶段吧),所以当项目计划时间与实际提交时间出现偏差时,我一般就会去追问.当然若整个CMM过程都做顺了,公司 过CMM5级是没问题了.国内在做CMM过级及与实践真正能达到相吻合的还是较少吧.至于配置库的审核,由SQA来检查没错,不过将配置审核作为配置人员为保证有关人员完成他们的工作，并满足外面要求的一个手段列出我想也不 为过吧?

●你所说的配置管理人员应该是属于一个项目立项后在组内设的配置管理员,他只是根据配置管理计对该项目进行配置管理,那么他确实没有被赋 予那样的权利，他只是配合公司最高配置管理员做好配置工作。作为公司最高配置管理员，我觉得公司赋与他的权利应该是负责编制配置管理计划，并组织实施，负责建立配置库，参与配置库的审核，配合SQA人员检查配置计划 的完成情况，向公司最高层定期报告配置状态，并对配置计划的完成情况进行考核，考核的结果纳入公司对开发人员的业绩评定，这样才能确保开发计划的顺利完成。

▲我赞成你的观点，我觉得应该赋予配置人员更多的权利，才能保证配置管理工作的有效进行。而且在公司的管理处于这种初级阶段时，配置管理就显得更重要了。

配置管理员角色

配置管理员在软件过程中的地位是举足轻重的，主要操作如下：

-- 创建配置库，并且至少创建配置库的所有第一级目录；

--为每个项目成员分配操作权限，一般地，项目成员拥有**add,check in ,check out,download**等；

--根据“基线计划”创建与维护基线，冻结配置项，控制变更；

--定期清除配置库里的垃圾文件；

--定期备份配置库。

配置管理员所具备的素质：

1、职业道德是第一位的，这是由于配置管理人员负责管理软件公司最为重要的资产。

2、软件配置管理的专业知识，最好要精通一种配置管理工具，没有工具是不可能实施软件配置管理的，否则那只能是效率极其低下的纸上谈兵。

3、项目管理的知识，对于软件开发流程非常熟悉。一般而言，最好要经历几个软件项目的开发管理过程，或者担任过项目经理，对软件开发的全过程有比较清晰的了解；有软件开发经验可以增强说服力，降低实施的难度，

并且能够切身以开发人员的身份去体会配置管理，才能改进配置管理过程。

4、有一定的大局观，有一定的**IT**背景知识**,**对系统（操作系统、网络、数据库等方面）比较熟悉。

除了个人素质上的要求，在性格上也有一些共性的东西。

1、沟通技巧：在部署和实施配置管理的时候，肯定会遇到一些抵触，对于程序员而言，使用配置管理之前，没有什么约束，但是在

实施后，会有一些约束，认为这并不是自己的工作。如果在使用中出现了问题，就需要配置

管理人员进行沟通，并且能够解决问题。

2、稳重、细心、有耐心。配置管理工作需要和开发人员、测试人员、项目经理打交道，但是他们对于遵循配置管理流程和工具不会非常的热心，因此需要配置管理人员能够稳重、有耐心。

第二章　相关问题

1　软件配置管理

问：*基线审核的内容是什么呢？基线审核是审核计划的基线（在配置计划中标识了将要在何时建立哪条基线），还是实际开发中当前的基线配置项提交情况？*

答：一般按设计开发阶段来策划，按阶段建立基线，并说明需要有什么工作产品，有那些工作产品的证据，是否达到策划的质量要求，如何 配置，如：功能、性能、版本、配置项等内容。

当然，审核的内容就是按这些要求

问：*在写变更控制流程的时候，有个问题很困惑。一般来说当配置项符合什么条件时，对其的修改才需要进入变更控制流程*

答：不是配置项的修改导致进行变更控制流程，而是引起配置项修改的变更项才是变更控制流程的发起者，例如：测试过程中发现的缺陷，客户提出的新需求等.

变更存在机制是关键，至于需要什么样的有几种机制完全是灵活的，针对不同的对象或不同的资源可以有成千上万个选择，即

便是有了流程，同样可以对变更流程做变更，同样要进入变更流程的控制之中。不要认为只有一种，或者约定就不变了

问：*配置库怎么设计才算合理*

答：配置库可分为开发库，受控库和产品库

开发库是存储开发人员正在进行中的文档与代码

受控库是存储整个开发过程各个里程碑点上的文档与代码，数据库类，它的设计可分为四大类：

系统文档：可分为多小块，根据项目组而定

例如：需求，计划，概要设计，详细设计，测试用例

数据库

代码

用户文档：名阶段产生的用户手册

产品库主要是存放发布后的版本，及匹配的文档

问：*怎样组织结构设计才能满足实际的配置工作？*

答：

1. 项目计划地开始编制SCM计划
2. 项目组审批SCM计划，确定SCCB
3. 按SCM计划实施SCM活动，进行标识，版本管理，变更控制
4. 随项目计划的更新修订SCM计划
5. 每里程碑后提交配置状态报告
6. 配合高层领导及SQA审计SCM活动工作

问：*配置管理计划在整个开发周期中的作用有多大，如何更好地做这个计划？*

答：**SCM**计划的作用是决定**SCM**活动的按时执行，**SQA**在这方面也可以监督到**SCM**是否有按时入产口基线

要做好这个计划，必须要结合项目计划，时刻随项目计划的更新而更新**SCM**活动计划

问：*配置管理怎么做才算到位？*

答：在项目审批**SCM**计划的基础上实施活活动，让项目成员了解在**SCM**的角色与任务，在计划的**SCM**活动中声明各里程碑要提交的产物，达成共识.

2 CVS 工具常见问题

问：*在服务器上有 2 个用户，administrator 和 user1，我给 user1 设置了 1 个工作目录 user1wk。这个工作目录被设置了权限，只有 user1 可以进行 crw。但不知道为什么，每次我在这个目录里 add 文件的时候服务器返回的信息提示：USER administrator cannot change c:\workroot\user1wk，当时我是用 user1 的帐号登陆的，请问能帮我分析一下问题么？*

答：只有用 checkout 模块时的用户才可以对 checkout 模块进行操作。

writer 和 **reader** 文件是存放用户权限的，如果有读写权限的就要在 **writer** 中添加用户名，只有读的权限的在 **reader** 文件中添加，如果同时存在两个文件中的，默认为只读权限

第三章　配置管理流程

1　概要

1.1　内容

规范配置管理活动，确保配置项正确地唯一标识并易于存取，保证基准配置项的更改受控，明确基线状态，在贯穿整个软件生命周期中建立和维护项目产品的完整性和可追溯性。

1.2　适用范围

对于不同类别的软件项目，配置管理的流程不同，可在本流程的基础上进行裁减。

1.3　术语和缩略语

1.3.1　软件配置管理（Software Configuration Management，SCM）

软件配置管理是对软件修改进行标识、组织和控制的技术，用来协调和控制整个过程。是通过技术或行政手段对软件产品及其开发过程和生命周期进行控制、规范的一系列措施。配置管理的目标是记录软件产品的演化过程，确保软件开发者在软件生命周期中各个阶段都能得到精确的不同版本的产品配置。

1.3.2　配置（Configuration）

配置是在技术文档中明确说明并最终组成软件产品的功能或物理属性。因此配置包括了即将受控的所有产品特性，其内容及相关文档、软件版本、变更文档、软件运行的支持数据，以及其他一切保证软件一

致性的组成要素，相对与硬件类配置，软件产品的配置包括更多的内容并具有易变性。

1.3.3 配置项（Configuration Item，CI）

凡是纳入配置管理范畴的工作成果统称为配置项（Configuration Item, CI），配置项逻辑上组成软件系统的各组成部分，一般是可以单独进行设计、实施和测试的。一个纯软件的 CIs 通常也称之为软件配置项（Computer Software Configuration Items，CSCIs）。

配置项主要有两大类：

1) 属于产品组成部分的工作成果，例如需求文档、设计文档、源代码、测试用例等；

2) 项目管理和机构支撑过程产生的文档。这些文档虽然不是产品的组成部分，但是值得保存。

每个配置项的主要属性有：名称、标识符、文件状态、版本、作者、日期等。所有配置项都被保存在配置库里，确保不会混淆、丢失。配置项及其历史记录反映了软件的演化过程。

1.3.4 基线（Baseline）

在配置管理系统中，基线就是一个 CI 或一组 CIs 在其生命周期的不同时间点上通过正式评审而进入正式受控的一种状态，些配置项构成了一个相对稳定的逻辑实体，而这个过程被称为“基线化”。每一个基线都是其下一步开发的出发点和参考点。基线确定了元素（配置项）的一个版本，且只确定一个版本。一般情况下，基线一般在指定的里程碑（Milestone）处创建，并与项目中的里程碑保持同步。每个基线都将接受配置管理的严格控制，基线中的配置项被“冻结”了，不能再被任何

人随意修改，对其的修改将严格按照变更控制要求的过程进行，在一个软件开发阶段结束时，上一个基线加上增加和修改的基线内容形成下一个基线。

基线的主要属性有：名称、标识符、版本、日期等。通常将交付给客户的基线称为一个“Release”，为内部开发用的基线则称为一个“Build”。

建立基线的好处：

1) 重现性：及时返回并重新生成软件系统给定发布版的能力，或者是在项目中的早些时候重新生成开发环境的能力。当认为更新不稳定或不可信时，基线为团队提供一种取消变更的方法。

2) 可追踪性：建立项目工件之间的前后继承关系。目的是确保设计满足要求、代码实施设计以及用正确代码编译可执行文件。

3) 版本隔离：基线为开发工件提供了一个定点和快照，新项目可以从基线提供的定点之中建立。作为一个单独分支，新项目将与随后对原始项目（在主要分支上）所进行的变更进行隔离。

2 相关人权责

2.1 项目经理（Project Manager，PM）

责任：

1) 与 CCB 协商确定项目起始基线和开发里程碑；

2) 接受配置管理计划，并按相关规定贯彻执行；

3) 接受配置控制委员会的报告。

权利：

1) 提出配置管理计划的修改要求；

2) 提出管理管理的建议和要求。

2.2 配置控制委员会（Configuration Control Board，CCB）

责任：

1) 制定和修改项目的配置管理策略；

权利：

1) 批准、发布配置管理计划；

2) 建立、更改基线的设置，审核变更申请；

3) 根据配置管理员的报告决定相应的对策。

2.3 配置管理员（Configuration Management Officer，CMO）

责任：

1) 编制配置管理计划；

2) 执行配置项管理方案；

3) 执行版本控制和变更控制方案；

4) 编制配置状态报告；

权利：

向 CCB 汇报有关配置管理流程中的不符合情况。

2.4 程序库管理员（Program Librarian，PL）

责任：

1) 配置库的建立和权限分配；

2) 配置管理工具的日常管理与维护；

3) 配置库的日常操作和维护；

权利：

1) 各配置项的管理与维护；

2) 对开发人员进行相关的培训。

2.5 开发人员（Developer）

责任：

1） 根据确定的配置管理计划和相关规定，提交配置项和基线；

2） 负责软件集成和版本生成。

权利：

按照软件配置管理工具的使用模型来完成开发任务。

2.6 测试人员（Tester）

责任：

根据配置管理计划和相关规定，提交测试配置项和测试基线；

权利：

负责软件变更的测试验证。

2.7 软件质量保证员（Software Quality Assurance，SQA）

责任：

负责配置审核并提交报告。

权利：

对配置审核中发现的不符合项，要求相关责任人进行纠正。

3 实施细则

3.1 CCB 的成立

3.1.1 项目在设计发布后，由项目经理负责组织成立 CCB。

3.1.2 CCB 成员组成

CCB 成员人数一般为奇数，人数在 3～7 人范围内。CCB 成员一般包括：

1) 项目经理 PM；
2) 配置管理员 CMO；
3) SQA；
4) 测试人员 Tester；
5) 顾客代表；
6) 主要开发人员等。

3.1.3 CCB 的决策机制

寻求 CCB 成员的一致意见。若不能达成一致，可采取由顾客代表做出决策；或采取少数服从多数的原则，由 CCB 成员投票确定，投票超过半数即为通过。

3.2 确定配置策略

3.2.1 配置策略确定的时机

CCB 成立后，由 CCB 组织会议根据项目的开发计划确定各个里程碑和开发策略，CMO 负责整理确定的项目基线和配置项列表，并在编制《配置管理计划》时列明，按约定的时机收集配置项和建立初始基线。

3.2.2 配置项的范围

1) 技术文档（Documents）：项目开发计划、需求分析报告、软件设计书、质量保证计划、概要设计书、详细设计书、测试文档、技术报告、用户手册、总结报告等；

2) 程序（Program）：阶段产品、计算机程序、源程序、释放产品等；

3) 工具（Tools）：自动设计工具、开发工具、测试工具、维护工具等；

4) 交互文档（Communications）：与客户或项目组内交互产生文档，如会谈记录、E-mail、会议纪要、MSN 记录等。

3.3 制定配置管理计划

3.3.1 《配置管理计划》的编制

通常情况下，由 CMO 在设计发注后，开始编制《配置管理计划》；如有特殊需要，根据合同或项目要求，由 CMO 在某一项目或项目的某一阶段开始前制定《配置管理计划》。

3.3.2 《配置管理计划》的内容

《配置管理计划》应包括以下方面的内容：

1) 该项目对配置管理的要求；

2) 实施配置管理的责任人、组织及其职责；

3) 需要开展的配置管理活动及其进度安排；

4) 采用的方法和工具等。

3.3.3 《配置管理计划》的由 CCB 负责审批。

3.4 配置项标识规则

3.4.1 配置项标识要求

1) 合同有明确标识和追踪要求时，由开发人员按合同要求进行标识，以保证满足合同追踪要求。

2) 在开发过程中项目组人员提交的配置项，由项目组人员按照本节相关部分标识规则进行标识。

3) 项目组人员将要标识或已标识的配置项提交给 CMO 纳入配置库统一管理，并填写《配置状态报告》。

3.4.2 配置项标识方式

3.4.2.1 标识项

配置项标识属性包括：名称、编号、文件状态、版本、作者、日期等。本文标识规则对名称、编号、文件状态和版本进行了描述和规定。

3.4.2.2 名称

文件名称的标识按文档模板中统一名称为准。

a) 编号

文档编号格式为 CC_XXX_***_$$$_###，其中 CC 表示公司，XXX 是项目的三位英文字母缩写表示，***_$$$表示文档类别，###表示文档顺序号。同时对应每个内容都有固定的一个索引文件 CC_XXX_**_$$$_index，目的是为了为本类别下的文件建立一个概要说明列表，保证快速对文档进行识别和检索。

3.4.2.3 文件状态

文件状态分为“草稿”、“正式发布”和“修改中”三种。

修改处于“草稿”状态的配置项不算是“变更”，无需 CCB 的批准，修改者按照版本控制规则执行即可。

当配置项的状态成为“正式发布”，或者被“冻结”后，此时任何人都不能随意修改，必须依据配置变更控制的规则执行。

3.4.2.4 文档版本控制

对于计划性文档、技术文档和用户文档，其版本按修改的先后顺序确定。新生成的文档第一次发行为第一版，修改后第二次发行为第二版，以此类推。

3.4.2.5 发行版本控制

最终完成的软件版本用三位符号表示：“s.x.y”。各符号位的含义如下：

1) “y”为第二次版本号，表示纠正错误时的版本升级，用一位数字表示：“1~9”，对上一次产品或项目中的缺陷做修正，第二次版本号增加；

2) “x”为第一次版本号，表示增加功能时的版本升级，用一位数字表示：“0~9”。与上一产品或项目相比，功能进行了小量的增加或修正时，第一次版本号增加，第二次版本号为零，第二版本号为零时可以省略不写；

3) “s”为主版本号。对产品作重大调整，或与已发行的上一产品相比，在功能与性能上有较大改善时主版本号增加；产品或项目概念全新，第一次完成，版本号为 1.0。

3.4.2.6 基线版本标识

内部基线，如计划基线、设计基线等，在版本号前加 Build，如 Build

1.0；

发行产品基线在版本号前加 Release，如 Release 2.0。

3.5 配置库管理

3.5.1 配置库（Repository）的分类

配置库分为两类：

1) 文档库（Document Library）：由 CMO 负责管理，主要使用 eSM 系统管理除程序以外的文档资料（包括图片等）；

2) 程序库（Program Library）：由 PL 负责管理，主要使用 CVS 版本工具对程序代码进行管理。

3.5.2 配置库的建立

3.5.2.1 CCB 成立之后，PL 即可着手组织建立配置库。所有项目应建立配置库，以便管理各配置项。

3.5.2.2 文档库空间由 eSM 系统创建，PL 仅创建基线文档库，仅 PL 可以对其操作。

3.5.2.3 程序库主要通过设置版本的分支，来实现对配置项权限管理，基本上要为每个配置项从建立开始就划分成 3 个不同的分支（如图 1）：

图 1 配置库空间分配和版本迁移策略

1) 私有分支（Private Branch）：私有分支对应的是开发人员的私有开发空间。开发人员根据任务分工获得对相应配置项的操作许可之后，他即在自己的私有开发分支上工作，他的所有工作成果体现为在该配置项的私有分支上的版本的推进，除该开

发人员外，其他人员均无权操作该私有空间中的元素。

2) 集成分支（Integration Branch）：集成分支对应的是开发团队的公共空间。凡是要为同组人员共享的配置项都从该分支获得。即各开发人员必须将私有工作空间中的开发成果归并（Merge）到该分支后才能进入下一个开发活动。所有涉及多人协调的开发工作（如集成测试等）都必须工作在这一空间中。该开发团队拥有对该集成分支的读写权限，而其他成员只有只读权限。该分支的管理工作由 PL 及相关指定人员负责。

3) 公共分支（Common Branch）：公共分支对应的是整个软件开发组织的公共空间。各个开发小组在现阶段的任务完成后，将可以发布的版本归并到该分支上，将来需要查阅相关资料时，以该分支上的版本为准。该分支对组织内的全体软件人员开放只读权限。该分支的管理工作由 PL 负责。

3.5.2.4 上述定义的 **3** 类分支以及文档库由 **CMO** 统一管理，根据各开发阶段的实际情况定制相应的版本选取规则，来保证开发活动的正常运作。在变更发生时，应及时做好基线的推进。

3.5.3 分配权限

PL 为每个项目成员分配配置库操作权限。一般地，项目成员拥有 Add、 Checkin/Checkout、Download 等权限，但是不能拥有“删除”权限。PL 的权限最高。

3.5.4 配置库的操作与管理

3.5.4.1 开发人员根据获得的授权的资源进行项目的研发工作，操作配置库，例如 **Add**、**Checkin/Checkout**、**Download** 等。

3.5.4.2 PL 根据配置管理计划创建与维护基线，“冻结”配置项，控制变更。

3.5.4.3 配置库的检出

当发生变更且变更评审通过后，或者发现 Bug 且 Bug 评审通过后，由 PL 将 Common Branch 中 CIs 检出至开发人员的 Private Branch 上，供开发人员进行变更。

3.5.4.4 配置库的检入

当变更实施结束或 Bug 修正和测试结束，并通过配置审核后，由 PL 将变更后的 CIs 检入到 Common Branch。

3.5.4.5 PL 定期清除配置库里的垃圾文件。

3.5.4.6 PL 定期备份配置库。

3.6 配置项和基线管理

3.6.1 CMO 根据配置管理计划，对配置项和基线进行分阶段管理。

3.6.2 项目启动

配置项包括需求说明、订单及其评审结果等；项目发布后应封锁该子项目，建立发布基线。

3.6.3 需求分析

系统调研后开发人员进行系统分析，并整理需求分析报告。需求分析报告通过评审并需取得客户的确定。在需求分析报告取得客户的确认后，封锁该子项目，建立需求基线。如需升版，则必须通过评审并得到客户的确认。

3.6.4 项目计划

需求分析完成后即可制定项目的开发计划，包括项目总体进度说明、进度跟踪、计划修改、配置管理计划、质量保证计划、测试计划等。项目开发计划评审通过后，封锁该子项目，建立项目计划基线。

3.6.5 系统设计

系统设计可分为概要设计、详细设计和数据库设计等部分。针对需求分析报告进行系统设计，配置时应说明系统设计的版本与需求分析报告版本的对应关系。设计书评审通过后，建立设计基线。

3.6.6 编码

编码按功能模块分子项目，即每个模块计作一个配置项。代码基线分别在单元测试结束后建立 Alpha 版，Alpha 测试后建立 Beta 版，在集成测试时建立 Merge 后版。

3.6.7 测试

各测试阶段应提供测试计划、测试用例、测试结果和测试分析报告，项目启动后应提供项目测试计划书，项目验收结束后应提交项目测试总结报告等。配置时应说明测试的版本与编码版本的对应关系。各阶段测试（如单元测试、集成测试）完成后建立测试基线。

3.6.8 交付与验收

在交付前配置审核完成后建立产品基线，产品基线包含程序以及有关文档配置项，包括交付施工文档、工具等。

3.6.9 项目总结

项目总结应经过部门内部评审，包括项目质量报告、测试报告等。

3.6.10 相关资料与培训

相关资料与培训也应作为配置项纳入配置管理，此部分包括：

1) 相关法律、法规；

2) 必须遵照或项目组约定的技术规范；

3) 与客户或项目组内部重要的交互信息记录，如 Question Sheet、会议记录、会谈记录、e-mail 和 MSN 记录；

4) 必要的业务或技术培训等。

3.7 配置变更控制

3.7.1 变更的分类

软件及其相关文档的变更按照变更的影响范围进行分类：

1) A 级：变更会影响系统级需求、外部接口、产品价格或者交付期；这类变更必须经过 CCB 审核并有客户批准和确认。

2) B 级：变更会影响配置项间的功能接口、组件级成本或者项目 Schedule;这类变更必须由 CCB 或上级管理部门的批准和认可。

3) C 级：变更会影响配置项内部功能的设计和分配；这类变更可以由配置项的管理人员负责批准。

图 2 变更控制流程

3.7.2 变更请求的提出

3.7.2.1 由发起者（客户、最终用户或开发部门）确定变更，填写《变更请求/评审单》，描述变更原因和变更内容，并提交给 CMO。

3.7.2.2 CMO 对填写的申请表是否清晰、明确和完整性进行审查，若 CMO 发现变更不明确或不完整，应返回申请表给发起者。CMO 对通过审查的变更申请分配变更 ID，以便跟踪和记录变更信息。

3.7.3 变更评估

3.7.3.1 CMO 将《变更请求/评审单》发送给项目经理（或者其他授权人员），由项目经理负责对变更进行评估。

3.7.3.2 变更控制的一个重要环节就是变更评估，变更评估要分析每个变更对系统功能、接口、成本、进度以及约定需求的影响，同时还要分析对软件安全性、可靠性、可维护性、可移植性和性能的影响。

3.7.3.3 变更评估产生的文档应描述若实施变更必须变更的配置项、文档和资源；变更评估文档在完成变更评估后发送给 CMO。

3.7.3.4 CMO 收到评估后的《变更请求/评审单》后，更新变更记录，并安排 CCB 会议日程。

3.7.4 变更审核

3.7.4.1 CCB 对提交的变更申请进行审核，并根据变更评估确定变更的影响级别；CCB 审核可能的结果有三种：接受变更；拒绝变更；延期变更。CCB 也可能需要更多的变更分析的信息。

3.7.4.2 CCB 批准的变更，由 CMO 将变更项目发送到指定的开发人员（Assigner）进行下一步的实施变更工作；对于拒绝的变更，由 CMO 将 CCB 拒绝变更的原因发送给发起者，并保存《变更请求/评审单》，

更新变更记录，关闭变更活动；需要进一步分析的，由 CMO 将变更项目随同 CCB 的 Question Sheet 发送给评估分析人员；对于延期的变更，由 CMO 对变更的相关文档进行归档，以便在适当时机提交 CCB 审核。

3.7.4.3 CMO 负责整理 CCB 会议记录，填写《变更请求/评审单》中相应审核项；更新变更记录，如果是接受变更，还需将要变更的 CIs 状态改为“修改中”；将变更文档分发给相关人员。

3.7.5 变更实施

3.7.5.1 变更被批准后，PL 负责将要变更的 CIs 以及相关文档迁移至变更负责人的 Private Branch 上，由变更负责人开始实施变更，并详细记录变更的内容；项目管理部门要对变更的实施进行跟踪。

3.7.5.2 对于代码变更，必须修改设计、代码、测试以及变更正确性的验证。而且与变更相关的文档必须修订，以反映变更。当变更以及测试完成后，进行 Merge。

3.7.5.3 对于开发计划、配置管理计划发生变更的，项目组人员要按照变更过的开发计划、配置管理计划提交配置项。

3.7.6 变更确认

3.7.6.1 变更后的程序 Merge 后必需经过测试组进行回归测试，以确保变更没有引入新的 Bug。不会引起程序变更的文档（如计划文档）的变更不需经过测试。

3.7.6.2 通过 Merge 后测试后，SQA 需对变更进行审核，审核的范围一般涉及以下方面：

1) 测试记录；

2) 变更请求；

3) 配置项的检入及检出；

5) 文件的命名；

6) 版本的编号。

3.7.6.3 SQA 审核后，开发人员才能生成新的版本，由 PL 更新到基线库中。

3.7.6.4 PL 应重新标识所有被影响的配置项及版本。

3.7.6.5 A 级和 B 级的变更项也可能直到下次系统发行版本时才生成。

3.7.6.6 生成新版本后，CMO 负责收集所有变更信息归档，修改变更 CIs 状态为“正式发布”，关闭变更，并将变更报告发送给发起者。

3.8 配置状态报告

3.8.1 配置状态报告的目的

记录和报告整个软件生命周期演化状态。

3.8.2 配置状态报告记录的内容

配置状态报告记录的内容包括：

2) 软件和文档的标识；

3) 目前状态；

4) 基线演化状态；

5) 变更状态；

6) 版本交付信息等。

3.8.3 配置状态报告的生成

配置管理报告自第一个基线创建时建立，由配置管理系统生成，及时反映当前配置状态。

3.9 配置审核

3.9.1 配置审核的类别

配置审核分为：

1) 功能配置审核（Functional Configuration Audit，FCA）：审核软件功能是否与需求一致，并符合基线文档要求；通常要审查测试方法、流程、报告和设计文档等。

2) 物理配置审核（Physical Configuration Audit，PCA）：审核要交付的组成项是否存在，是否包含所有必需的项目，如正确版本的源代码、资源、文档、安装说明等等。

3.9.2 配置审核执行的时机

通常选择以下几种情况由 SQA 负责实施配置审核：

1) 软件产品交付或是软件产品正式发行前；

2) 软件开发的阶段工作结束后；

3) 在产品维护工作中，定期地进行。

3.9.3 不符合项的处理

对配置审核中发现的不符合现象，SQA 进行记录，并填写《不符合项报告》，交由责任部门限期进行纠正，SQA 负责纠正措施的验证。所有的不符合项报告均关闭后，才能发布新版本。

3.10 发行管理

3.10.1 通过配置审核后，由项目经理负责生产新版本，并由 **PL** 检入产品库中，并按照 **5.4** 节配置标识规则进行版本标识。

3.10.2 交付管理

这里“交付”是指从配置库中提取配置项，交付给客户或项目外的人员。交付出去的配置项必须有据可查，避免发生混乱。流程如下：

1) “索取人”向 CCB 提出交付申请。

2) CCB 审批该申请。如果该申请不合法（合理），则拒绝交付配置项。如果同意交付，CCB 应给出详细的交付清单。

3) PL 依据 CCB 的批示，从配置库中提取配置项交付给“索取人”。

4) “索取人”验收后签字。

第三章 解析本土化软件配置管理

软件工程犹如一座大迷宫，道路曲折，但是却隐藏有无数的珍宝。每个探险迷宫的人在找到捷径和珠宝后，都会有对于迷宫的独特感悟。为了更好的指引有心的探险人，编辑部将从迷宫各个不同的入口，请来那些已经发掘过迷宫的先行者们，和他们一道感悟这座迷宫。

如果您已经找到了迷宫的珠宝或者即将向迷宫探险，都可以和我们联系。

版本控制，是软件开发中一项必不可少的管理手段，也是软件配置管理（Software Configuration Management，SCM）的一个部分。而软件

配置管理，在软件开发过程中占据着非常重要的地位，并且是 CMM 2 级的一个关键域。

2004 年 3 月 3 日，本刊有机会请到了六位业内软件配置管理的专家，组织了一次专门的研讨会。通过讨论，我们也许可以为中国的软件配置管理作一个有益的点评。

迅速发展的软件配置管理

配置管理的概念源于美国空军，为了规范设备的设计与制造，美国空军 1962 年制定并发布了第一个配置管理的标准“AFSCM375-1，CM During the Development & Acquisition Phases”。

而软件配置管理概念的提出则在 20 世纪 60 年代末 70 年代初。当时加利福利亚大学圣巴巴拉分校的 Leon Presser 教授在承担美国海军的航空发动机研制合同期间，撰写了一篇名为“Change and Configuration Control”的论文，提出控制变更和配置的概念，这篇论文同时也是他在管理该项目（这个过程进行过近一千四百万次修改）的一个经验总结。

Leon Presser 在 1975 年成立了一家名为 SoftTool 的公司，开发了配置管理工具：Change and Configuration Control（CCC），这是最早的配置管理工具之一。

随着软件工程的发展，软件配置管理越来越成熟，从最初的仅仅实现版本控制，发展到现在的提供工作空间管理、并行开发支持、过程管理、权限控制、变更管理等一系列全面的管理能力，已经形成了一个完整的理论体系。同时在软件配置管理的工具方面，也出现了大批的产品，如：最著名的 ClearCase；开源产品 CVS；入门级工具 Microsoft VSS；新秀 Hansky Firefly。

在国外已经有 30 多年历史的软件配置管理，但在国内的发展却是在 21 世纪这几年的事。但是通过专家们的介绍，我们感受到，国内的软件配置管理已经取得了迅速发展，并得到了软件公司的普遍认可。

刘晓：1997 年，开始在国内从事配置管理。那个时候如果和客户讲解配置管理，先要讲配置管理是干什么的，能对企业起到什么作用。到了 2002 年，就可以直接介绍软件配置管理的策略和使用。现在很多公司都能够自主的向工具厂商咨询，确实是已经意识到配置管理的重要性了。

思维加速是一家致力于业务架构平台研发的软件公司，有 20 多人的专业开发队伍。对于这样一种规模的软件公司，在国内应该是最为普遍的。也许它实施配置管理的过程就是国内企业实施配置管理的一个典范。让我们看看思维加速公司的总工宋兴烈是如何描述的。

宋兴烈：思维加速一直在从事较大规模的复杂软件开发，需要对代码、文档进行版本控制，早期采用了最简单的 VSS（Visual Source Safe）进行版本控制。随着开发的深入，版本控制已经远远不能满足需要了，分支管理、权限管理的需求就出来了。但是 VSS 却不能满足这些要求，因此我们对 VSS 进行了二次开发。现在在 VSS 上能够基本实现分支管理、权限控制、版本发布构建管理了。但随着软件规模的增长，我们正准备用 CVS 等开源软件作为我们产品新版本的版本控制和管理工具。

这时，思维加速对于配置管理的理解上也有了一个较大的变化。配置管理已经不仅仅是对软件代码、文档的版本管理，而应该是需求变更、缺陷管理等的一个整合体。对于需求变更、缺陷管理，最简单的传统处理方式是采用文档记录的形式，比如用 Word 文档进行控制。对于一个逐渐成熟的开发团队而言，这样的传统方法已经完全不能适用。对于商

业性的成熟配置管理工具，由于相对昂贵的价格，也没有采用。因此思维加速的办法是：自己进行开发，建立一个适用的管理系统。

配置管理的三大误区

国内软件公司实施配置管理，已经取得了很多进步，也提高了软件的质量。但是对于软件配置管理，有很多公司对它的理解比较模糊，或者在真正的配置管理实施过程中存在着误区。从专家们的讨论中，我们了解到国内的软件配置管理主要有三个方面的误区。

误区一：版本控制=软件配置管理

也许很多人不承认自己对于软件配置管理的理解局限在版本控制上，但在具体实施配置管理的过程中，就只见版本控制，而不见真正的配置管理。其实版本控制只是配置管理最基本的层次和功能。当然只有进行了版本控制，其他的功能才可能会逐渐提升。就是一个基本的版本控制，在部分软件公司中也并不是一个非常正规和完善的过程。

这种问题，归根到底在于软件公司对软件开发流程的管理在意识上不够重视。国内软件企业的开发管理不是很规范，即使在大的软件公司里面，项目组对于开发管理的关注也是有限的。另外一个原因是由于开发管理中资源的不足，比如:资金的缺乏（导致不能购买功能齐全、价格昂贵的商业产品）、人力资源（不能招聘专业的配置管理人员），因此不能在公司内部实施体系化的配置管理。

误区二:编码水平最差=配置管理员

配置管理人员是配置管理具体实施的人。可以说公司制定了配置管理的流程和规章只是配置管理实施的基础，而真正配置管理能否实施，能否有效，关键在于从事配置管理的人员。但国内的一个误区是：在选

择配置管理人员的时候，是寻找开发团队中编码水平最差的人。比如张三写代码不行，测试也不行，那就只好去从事配置管理工作了。

谷炼对此深有体会。谷炼有在日本 Rational 和国内在配置管理领域工作的经历。相比国内低水平的配置管理人员，国外公司一般都由有丰富编程经验的人担任软件配置管理人员，有的时候配置管理部分的工作职责直接由开发经理担任。配置管理人员的级别也相当高，被认为是项目经理的左右手，拿的是双薪。

其实一个 SCM 人员的责任相当重大，一个团队所有的代码、文档都由其负责，国内好像是处于一个相当尴尬的境地，认为一个什么都不懂的人担任，才能保证这些代码文档的安全。

当然国内也不泛重视配置管理的公司。据传说华为就非常重视软件配置管理，除了设置 CTO、CEO，好像还设置了一个 CMO（Configuration Management Officer,或者叫配置经理）。

误区三：采用配置管理工具=有效的配置管理

配置管理工具在软件配置管理中起着不可替代的作用。没有工具的支持，实施一个完整合格的配置管理是不可想象的。也许正是因为工具的重要，造成了很多软件公司对于工具的迷信，以为只要部署了配置管理工具，尤其是一个专业的商业工具，就自以为建立了配置管理体系。

使用好的工具并不能代表就能实施好配置管理。因为工具就是工具，工具不能代替管理。否则为什么总是说配置“管理”而不单单说配置“工具”呢？一个成功的配置管理工具实施，需要两个方面的条件:一是规范的软件开发流程;二是合格的配置管理参与人员，这里的配置管理参与人员包括了配置管理员、开发人员、项目经理等。

对此，邓小年认为：“无论怎么样，没有流程和规范地使用工具，

那么再强的工具也没有灵魂。比如简单的一个 check in 操作，不同的人用起来可不一样。有人修改后，进行 build，然后 check in；有人修改后，进行 build，并简单的测试再 check in,也有人修改后马上 check in，……可看出不同的人使用工具的同一操作有不同的后果。”

人--为何如此重要？

配置管理员在配置管理中是一个比较奇妙的角色，对于一个实施了配置管理、建立了配置管理工作平台的团队来说，他是非常重要的，整个开发团队的工作成果都在他的掌管之下。如果他管理和维护的配置管理系统出现了问题，轻则影响团队其他成员的工作效率，重则可能出现丢失工作成果、发布错误版本等严重的后果。

配置管理员应该做什么

一般而言，配置管理人员在软件公司中应该具有下面的几项主要职责：

1、 提交配置管理计划；

2、 软件配置管理工具的日常管理与维护，各配置项的管理与维护；

3、 执行版本控制和变更控制方案；

4、 完成配置审计并提交报告；

5、 对开发人员进行相关的配置管理培训；

6、 识别软件开发过程中存在的问题并拟出解决方案。

你是合格的配置管理员吗

一个高水平的配置管理人员，对开发团队在整体上有非常重要的作用。如果在一个企业中实施了配置管理工具如 ClearCase，但没有专业的

配置管理人员管理，就像一个拖拉机安装了一个奔驰的马达，还是跑不快。早期在国内企业中，找一个合适的配置管理人员很困难，最后由系统管理人员来担任。并且使用不同的配置管理工具，对配置管理人员的要求就不一样，如 VSS 对配置管理人员的技术水平要求就较低。

按照配置管理的职责要求，一个合格的配置管理人员需要具备哪些素质呢？

1、职业道德是第一位的，这是由于配置管理人员负责管理软件公司最为重要的资产。

2、软件配置管理的专业知识，最好要精通一种配置管理工具，没有工具是不可能实施软件配置管理的，否则那只能是效率极其低下的纸上谈兵。

3、项目管理的知识，对于软件开发流程非常熟悉。一般而言，最好要经历几个软件项目的开发管理过程，或者担任过项目经理，对软件开发的全过程有比较清晰的了解；有软件开发经验可以增强说服力，降低实施的难度，并且能够切身以开发人员的身份去体会配置管理，才能改进配置管理过程。

4、有一定的大局观，有一定的 IT 背景知识,对系统（操作系统、网络、数据库等方面）比较熟悉。

除了个人素质上的要求，在性格上也有一些共性的东西。

1、沟通技巧：在部署和实施配置管理的时候，肯定会遇到一些抵触，对于程序员而言，使用配置管理之前，没有什么约束，但是在实施后，会有一些约束，认为这并不是自己的工作。如果在使用中出现了问题，就需要配置管理人员进行沟通，并且能够解决问题。

2、稳重、细心、有耐心。配置管理工作需要和开发人员、测试人

员、项目经理打交道，但是他们对于遵循配置管理流程和工具不会非常的热心，因此需要配置管理人员能够稳重、有耐心。

3、能够吵架。有的时候，如果沟通不行，就需要采取强迫的手段来保证具体配置工作的要求得到执行。记得在网上见到这样的一句话：搞配置管理原来很好玩，就是要--凶～！

配置管理员的困惑

用友软件工程公司的耿延煜现在担任一个项目的配置管理员，她对于软件配置管理人员的看法更有代表性。在用友软件工程公司，采取的是一个项目设置一个配置管理人员，主要工作是项目产品的版本管理，并配合项目经理对项目中的文档、代码进行检查。但是这个配置管理人员并不是专职的，在承担配置管理职责之外，还会承担一些项目的开发、测试工作。作为一个兼职的 SCM 人员，耿延煜认为，有两个问题需要注意：

一是如何在工作任务紧张的时候保证配置管理工作？

作为一个配置管理人员，并不是仅仅从事配置管理工作，很多时候，会接受项目经理指派的开发工作，这个时候如何处理配置管理工作和开发工作的权重就非常重要，尤其是在一个项目处于紧要关头的时候，开发进度紧，很多公司就忽视了配置管理，但是往往这个时候，配置管理才是最为重要的，并且这个时候出了问题，对于项目的影响会更大。因此在很多情况下，必须付出时间从事配置管理工作，如加班。出现了问题，配置管理人员必须立即进行修复。

二是定位模糊。很多 SCM 人员对自己的定位都比较模糊，没有将自身置于一个项目管理者的角色。感觉自己只是项目组的一个无关紧要

的角色。国内软件开发中，向来就重开发人员，轻视测试人员，配置管理人员就更得不到重视了。然而，配置人员应该是一个项目经理的Backup，应该向项目经理发展。

配置管理员的最佳实践

对于配置管理人员的部门设置，邓小年认为，一般国内大中型软件公司在配置管理部门可以设置如下的三个职位：

1、配置管理经理：负责公司全面的配置管理方面的工作；

2、创建发布工程师：主要负责创建和发布，部署产品；

3、工具管理工程师：主要负责开发、维护配置管理工具，对工具的使用进行培训。

考虑到我国的现实情况，在一个软件公司中的每个项目专门设置一个 SCM 人员还不现实。从上面可以看出，配置管理员的最佳实践和推广方式可以采用是“兼职＋专职”的形式来进行。具体而言，可以这样安排：

1、软件公司在公司级必须有一个整体的配置管理解决方案和策略，对于各个具体开发的项目也有一个适合项目需要的配置管理策略。

2、公司级的 SCM 策略上，设置专职的配置管理人员，一般由水平较高的人员担任，符合上面提到的配置管理员的素质要求。

3、项目级的 SCM 策略上，设置兼职的配置管理人员，一般可以由开发人员或者质量人员来兼任。

4、专职 SCM 人员和兼职 SCM 人员之间的沟通协调。并且对于 SCM 工具，如 ClearCase，一般在前期部署的时候，任务比较紧张，在实施以后，操作就比较简单，只需要一个兼职人员就可以了。通过专职 SCM

人员和兼职 SCM 人员之间不断地反复沟通，才能将一个 SCM 过程具体实施好。

实施--从老板的角度思考

配置管理应该如何实施？软件配置管理活动是一项支持性、保障性的工作，它本身并不直接为企业产生直接赢利的工作成果，它每一项活动都需要消耗企业的资源，还需要购置配置管理工具，这些都会导致企业生产成本的增加。因此对一个软件公司来说，实施配置管理就需要从老板的角度进行思考。

什么是成功的配置管理

一个没有实施配置管理的软件项目，常常出现的问题有：版本混乱、文档不统一、工件遗缺等。这些问题归根到底是软件质量的问题。因此对于什么是成功的软件配置管理，一个最简单的方法是比较配置管理实施活动前后，软件产品的质量是不是得到了提高、开发团队是不是能够工作在一个有助于提高整体工作效率的配置管理平台上。

具体到配置管理中的每项活动，是否成功的标准是：这项活动是不是真正有助于我们实施配置管理的活动？它对于提高我们产品的质量有多大的帮助？能否帮助开发团队更高效地工作？

配置管理流程再造

软件配置管理的实施首先需要有一个规范的软件开发流程，因此对于一个软件公司而言，要实施配置管理，首先是需要对自身的软件开发流程进行再造。再造水平的好坏决定了配置管理是否仅仅是版本控制，是否能够有效的实施配置管理。流程不好，也许就仅仅到达版本控制的

水平，不能达到变更控制、过程管理等“配置管理”的较高水平上。

选择合适的配置管理工具

有了配置管理流程，为了保证配置管理的效果、范围和质量，就需要应用配置管理工具。对于如何选择一个合适的配置管理工具，在这次的研讨会上也是我们讨论的热点。我们认为，公司在选择 SCM 工具的时候，可以遵循下面四个原则。

首先是工具的价格。一般国内中小型软件企业都没有足够的资金来实施一些商业工具。因此就只能退而求其次了，采用一些变通的方法来实施。比如：对于变更管理，采用签发变更单解决。有些公司采用一些简单的 MIS 来管理，数据保存到数据库，能够做到查询、保存等目的。

二是在功能上，满足需要就可以，太过复杂的产品使用也麻烦，对配置管理人员的要求也高，价格也昂贵。工具只要符合需要就行，不必追求功能齐全。

三是性能。由于配置管理的都是开发中最为重要的资源，一旦崩溃，损失就会很大。因此配置管理工具的运行需要非常稳定，不出故障，出了故障应该有补救措施。

四是实施的过程是否顺利，售后服务和技术支持是否完善。

正确实施配置管理工具

在实施配置管理工具的时候，工具必须和开发管理流程良好的结合起来，这样才能保证工具的正确实施。而这点也是目前软件公司在使用各种工具时一个最大的难点。

谷炼:一个规范的公司，实施软件配置管理的时候，只是将他书面的东西电子化就可以了。而一个不规范的公司，第一步就是先要对流程就

行规划化，然后才是电子化的过程。

刘晓：工具和管理流程之间的关系是辨正的，其实我们更看重的是公司是否形成了一套有效的管理规范，是否有一个配置管理的流程和思想。工具只是起到一个辅助作用，当规范建立起来以后，采用工具就能够事半功倍。没有规范，使用什么工具都是没有用处的。厂商一般都是采用工具结合现有的咨询服务。就像：给你一支好笔，不一定写出好的字，如果你字写不好的话。但是如果你能够写一手好字，什么笔都没有关系。工具只是起到一个辅助的作用，关键是开发流程。作配置管理，厂商的作用主要是将企业的流程进行规划化，然后配置管理工具将开发过程进行质量的提升。最后能够为用户培养一些配置管理方面的专家，也才是最大的价值。同时当一个不规范的软件企业中采用配置管理工具的时候，一个强制性的工具是否可以有效的实施下去呢？这对于企业流程、企业的员工都是一个非常关键的考验。工具一般是强制性的，能否和软件公司员工的工作过程相符合，他对软件工程的了解能否满足这种需要呢？

谷炼:在实施配置管理工具时，有一个软件工程部署曲线，在工具实施后的一段时间内，软件开发的流程会有一个震动，质量会有大幅下滑，这个时期是非常关键并且重要的。

宋兴烈:我们就非常担心这一点，有些许震动不要紧，但是震动过程必须可控。

梁峰：一般在公司中实施配置管理工具时，可以采取先试点后推广的过程。先是一两个项目或者一两个开发模块，通过实施配置管理工具的过程，为公司规范配置过程，培养配置管理人才；然后再推广到整个公司的开发过程中。

配置管理的未来

软件危机一直都存在着，软件工程也一直都在不断发展。对于软件配置管理，未来究竟会如何发展呢？

刘晓：质量问题是软件开发的根本问题，不仅仅和配置管理有关，还和设计、测试、代码等有关。如果能够有一个协同开发平台，从需求，到代码、测试用例都有一个对应，这样的一个整体过程将会是发展的趋势。以前，很多工具之间都是零散的，之间难以进行系统的集成化。Hansky 的做法是有一个协同的开发平台，将各个工具整合在一起，这是一个方向。

宋兴烈：我希望整个配置管理能够为软件开发全过程形成一个开发的知识库。从需求分析、讨论；任务的制定、分工；详细设计；软件测试；缺陷管理等等都有一个详细的管理过程。在配置管理中，如何做到将版本管理、测试、构建、发布等过程完整的结合在一起，做到自动化和程序化。同时我希望有这样一个协同平台，能够将所有的开发过程集合起来。

谷炼：质量关乎每个环节，一个环节的好坏并不能提高质量，各个分开的环节也难以保证质量。只有一个整体的过程才能提高质量。软件开发有一个 IDE，对于软件开发过程的管理也应该有一个 IDE，这应该是必然的趋势。Rational 公司，已经有这样一个整体的解决方案。以后也会以一个单独的产品出现，包括需求、建模、自动化测试、配置管理等都整合在一起。

梁峰：在贝尔，配置管理和变更管理从来就是一个整体，没有严格的将它们区分，是一个整合的管理过程。

刘晓：配置管理的发展趋势是一个整合的过程。配置管理工具将会集成变更管理工具、需求管理工具、软件建模工具等，形成一个统一的IDE，在使用上也会越来越简单（主要体现在用户界面和功能上）。当软件公司在开发软件的时候，一个完整的软件开发生命流程管理工作将会在一个统一的软件平台上进行，这样将会带来软件开发管理水平的提高，最终能够提高软件的质量。

而上海的邓小年对于配置管理的发展，则有着自己的见解。

1、配置管理应该更自动化

变更管理应该支持从项目开始到维护整个生命周期不同的变更类型；代码的版本管理到发布应该一气呵成；版本的配置关系可以简单表现出来。这些方面虽然配置管理工具涉及到，但没有一个工具能够完全的自动化。很多时候，我们需要额外的工作来维护配置管理系统本身。可以说：项目总的复杂度不变，实施配置管理后，降低了软件开发的复杂度，却增加了管理的复杂度。因此配置管理的自动化非常重要

2、配置管理应该基于流程

没有流程来执行配置管理是不切实际的，而现有的配置管理工具大多支持流程的管理，但是流程不一定适合于每个公司。如 ClearCase 的 UCM 流程，虽然很好，但是很难定制，我们只能让公司项目环境来适应这个工具，而不是让工具来适应现有的开发环境，这样的做法并不少最好的。配置管理工具不仅要支持流程，并且能够定制流程。

3、策略和标准化更加容易使用

实施配置管理的时候，需要定义许多的标准，比如版本名称规则、文档命名规则、什么时候 check in、什么时候创建分支等。目前的一些

配置工具，都没有提供这方面的专门定义，需要配置管理员另外写脚本或者程序来限制执行。这些地方都是需要改进的。

结束语

配置管理本身无论从理论和实践都在不断丰富和发展。例如，配置管理应用于“知识库”的管理就产生了“内容管理”这一新的领域。配置管理提供的状态报告和数据统计也为软件度量提供了决策依据。配置管理为项目管理提供了各种监控项目进展的视角，为项目经理确切掌握项目进程提供了保证。配置管理也为开发人员提供了一个协作的平台，在此平台上，大家能够更有效率的交流和协作。可以说，配置管理是软件开发的基石！

配置管理近年来在中国得到了极大的认可，可以毫不夸张的说，没有配置管理，就谈不上软件开发，就谈不上软件质量，就谈不上软件业的发展。随着软件业规模的扩大，配置管理的实施不是要不要的问题，而是什么时间、如何实施的问题了。