IBM Rational

软件配置管理
IBM Rational 技术白皮书

版本 1.1

Rational. software

IBM

目录

Rational. software

IBM

1. 什么是软件配置管理?

随着软件团队人员的增加，软件版本不断变化，开发时间的紧迫以及多平台开发环境的采用，使得软件开发面临越来越多的问题，其中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等等，解决这些问题的唯一途径是加强管理，而软件开发管理的核心是软件配置管理。

1.1 软件配置管理定义

配置的概念最早应用于制造系统，其目的是有效标识复杂系统的各个组成部分，如在制造行业很早就有材料清单（BOM: Bill of Materials）的概念，例如，计算机系统的 CPU、磁盘以及外设配置及其相应的规格、型号等等。随着计算机软件的发展，它已由最初的“程序设计阶段”经历了“程序系统阶段”进而演变为当前的“软件工程阶段”，软件的复杂性日益增大。此时，如果仍然把软件看成一个单一的整体，就无法解决所面临的问题，因此软件产品同样需要类似材料清单的概念，于是配置的概念被逐渐引入软件领域，人们越来越重视软件配置的管理工作。

那么，什么是软件配置管理？软件配置管理，简称 SCM（Software Configuration Management），简单而言就是管理软件的变化,它应用于整个软件工程过程，通常由相应的工具、过程和方法学组成。

IEEE“软件配置管理计划标准”（IEEE 828-1998）关于 SCM 的论述如下:

“软件配置管理由适用于所有软件开发项目的最佳工程实践组成，无论是采用分阶段开发，还是采用快速原型进行开发，甚至包括对现有软件产品进行维护。SCM 通过以下手段来提高软件的可靠性和质量:

- 在整个软件的生命周期中提供标识和控制文档、源代码、接口定义和数据库等工件的机制;

- 提供满足需求、符合标准、适合项目管理及其它组织策略的软件开发和维护的方法学;

- 为管理和产品发布提供支持信息，如基线的状态，变更控制、测试、发布、审计等等。”

1.2 实施软件配置管理的好处

实施有效的软件配置管理可以解决以下软件开发中的常见问题:

- 开发人员未经授权修改代码或文档;
- 人员流动造成企业的软件核心技术泄密;
- 找不到某个文件的历史版本;
- 无法重现历史版本;

- 无法重新编译某个历史版本，使维护工作十分困难;
- “合版本”时，开发冻结，造成进度延误;
- 软件系统复杂，编译速度慢，造成进度延误;
- 因一些特性无法按期完成而影响整个项目的进度或导致整个项目失败;
- 已修复的 Bug 在新版本中出现;
- 配置管理制度难于实施;
- 分处异地的开发团队难于协同，可能会造成重复工作，并导致系统集成困难;

2. 软件配置管理功能

2.1 版本控制

版本控制是所有配置管理系统的核心功能。配置管理系统的其它功能大都建立在版本控制功能之上。版本控制的对象是软件开发过程中涉及的所有文件系统对象，包括文件、目录和链接。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 文件等等。目录的版本记录了目录的变化历史，包括新文件的建立、新的子目录的创建、已有文件或子目录的重新命名、已有文件或子目录的删除等等。

版本控制的目的在于对软件开发进程中文件或目录的发展过程提供有效的追踪手段，保证在需要时可回到旧的版本，避免文件的丢失、修改的丢失和相互覆盖，通过对版本库的访问控制避免未经授权的访问和修改，达到有效保护企业软件资产和知识产权的目的。另外版本控制是实现团队并行开发、提高开发效率的基础。

文件或目录的版本演化的历史可以形象地表示为图形化的版本树（Version Tree)。版本树由版本依次连接形成，版本树的每个节点代表一个版本，根节点是初始版本，叶节点代表最新的版本。最简单的版本树只有一个分支，也就是版本树的主干；复杂的版本树（如并行开发下的版本树）除了主干外，还可以包含很多的分支，分支可以进一步包含子分支。

一棵版本树无论多么复杂，都只能表示单个文件或目录的演化历史，但典型的软件系统往往包含多个文件和目录，每个文件和目录都有自己的版本树，多个文件的版本需要相互匹配才可以协同工作，共同构成软件系统的一个版本或发布。因此为管理软件系统的发布，单有版本树是不够的，还需要引入基线（Baseline）的概念。基线由每个工件的某一个版本构成，是开发和进一步演化的基础。如果将全部版本树看作一个森林，基线则是该森林的一个横截面（该横截面往往不是水平的，而是上下起伏的折面）。

有了版本控制和历史版本，版本之间的比较就变得非常有意义，配置管理工具应能有效支持版本的图形化的比较。

2.2 工作空间管理

仅仅有版本控制功能还是不够的，在某一时刻，开发人员的开发工作往往只工作在文件的某一选定的版本上，因此配置管理系统需要提供一种便捷的访问正确文件的正确版本的机制，这正是配置管理系统的工作空间管理功能。所谓工作空间，就是为了完成特定的开发任务（如开发新功能、进行软件测试、或修复 BUG，等等），从版本库中选择一组正确的文件/目录的正确版本拷贝到开发人员的开发环境。举例说明：为修复一个旧版本，如 REL1，中的 BUG，开发人员首先需要在自己的开发环境中完全重现 REL1 所对应的源文件和目录结构，也就是说，需要建立一个对应于 REL1 的工作空间。

存在两类工作空间，一类是开发人员的私有空间，在私有空间中，开发人员可以相对独立地编写和测试自己的代码，而不受团队中其他开发人员工作的影响，即使其他人也在修改同样的文件；另一类工作空间是团队共享的集成空间，该空间用于集成所有开发人员的开发成果。

工作空间管理包括工作空间的创建、维护与更新、删除等，工作空间应具备以下特点：

- 稳定性：工作空间的稳定性指的就是私有空间的相对独立性，在私有空间中，开发人员可以相对独立地编写和测试自己的代码，而不受团队中其他开发人员工作的影响。
- 一致性：工作空间的一致性指的是当开发人员对自己的私有空间进行更新时，得到的应该是一个可编译的、经过一定测试的一致的版本集。
- 透明性：工作空间的透明性指的是工作空间与开发人员本地开发环境的无缝集成，将配置管理系统对开发环境的负面影响降到最小。

缺少有效的工作空间管理会造成由于文件版本不匹配而出错和降低开发效率，更长的集成时间等问题。

2.3 并行开发支持

如图 1 所示，在传统的串行开发模式下，同一软件的多个发布（Release）是顺序地被开发出来的，也就是说 Release 1 全部开发结束后才开始 Release 2 的开发；同样，等 Release 2 全部开发结束后 Release 3 的开发工作才会启动。但这种串行开发模式在当今的市场环境下越来越行不通，因为所有的软件产品都面临越来越大的迅速上市的压力，唯一有效的解决方案就是引入并行开发机制。在并行开发模式下，同一软件的多个版本会同时进行开发，如 Release 1 开发尚未完成，往往就会同时启动 Release 2 的开发，甚至会同时启动 Release 3 的开发，从而有效缩短软件的上市周期。为实现有效的并行开发模式，需要一种机制将 Release 1 中后期开发的功能合并到 Release 2 中，这正是配置管理工具需要完成的。

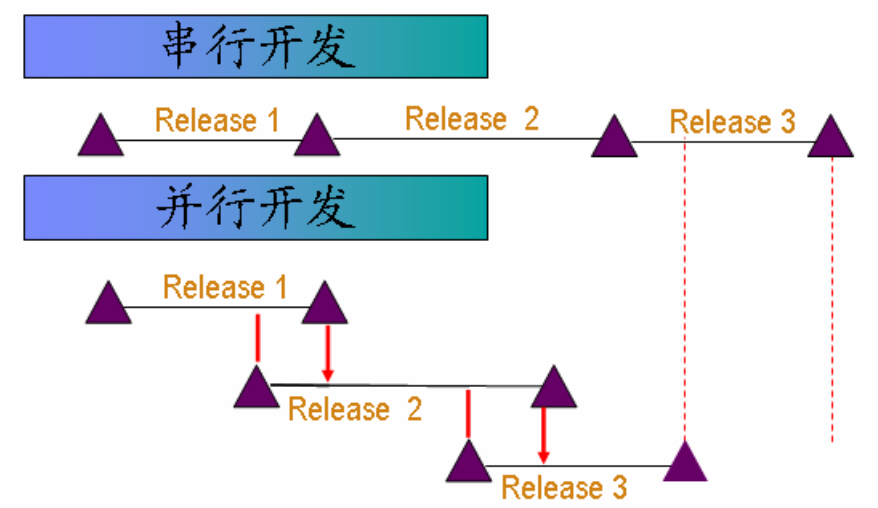

图 1: 串行开发与并行开发的比较

为实现并行开发，配置管理系统需要提供灵活的分支机制和工作空间管理。创建分支的过程实际上就是一个建立副本的过程，针对每个发布分别建立相应的分支，分支之间具备相对的独立性，这样不同的发布就可以在各自的分支上并行进行开发，在适当的时候，分支之间可以进行合并，从而实现将 Release 1 中后期开发的功能合并到 Release 2 中。

对同一软件的多个发布的并行开发其实只是诸多并行开发模式和分支策略之一，其他常见的并行开发模式有:

- 并行进行同一产品的维护和开发:
- 并行开发同一产品的多个定制版本:
- 并行开发同一产品的多个特性:
- 团队协作并行开发一组相同的文件/目录:

更多的并行开发策略可参见第四部分: 软件配置管理策略。

2.4 过程控制

不同的开发组织往往具备不同的组织结构、不同的开发环境、不同的开发策略，因此，配置管理系统应该能够支持灵活的配置管理策略和配置管理流程，并实现过程自动化以提高配置管理效率，这就是过程控制需要实现的功能。

大家都知道，在商用关系型数据库中有一种触发器（Trigger），通过触发器机制，可以定义在记录被插入/删除之前或之后自动执行一个预定义的脚本来实现特定的逻辑功能；配置管理系统应能够提供类似的触发器机制。通过可定制的触发器，可以定义在执行特定的配置管理操作（如检出、检入）之前或之后自动执行特定的任务（可以是批处理、可执行文件、或其他的配置管理操作），从而自动化实现预定义的配置管理策略。

过程控制还可以通过对配置项加锁/解锁来实现，如在版本库备份阶段对版本库进行加锁以禁止在备份过程中的版本库的修改。

通过过程控制还可以设置额外的安全访问机制以加强配置管理系统的安全策略。

IBM Rational 更是提供了一个基于最佳实现经验的流程——统一变更管理 UCM（Unified Change Mangement），UCM 的经验来自上百个客户的数千个项目，用户可以在短时间内（不需要额外的定制，或仅仅通过简单的定制）共享到其他客户在软件配置管理上的经验，一方面可以充分发挥软件配置管理带来的好处，另外节省了大量时间及成本。

2.5 构建和发布管理

简单而言，构建和发布管理的目的有三个：确保软件构建是可重现的、高效的、和可维护的。在小型的软件开发项目中达到以上三个目标并不困难，但随着开发团队、软件规模、软件复杂性的增长，以上的三个目标已经成为一种挑战。

典型的构建和发布管理包括以下几个步骤：

- 步骤一：确定参与构建的全部资源（如源代码、库文件、配置文件等）的正确版本；
- 步骤二：基于步骤一中选定的正确版本创建一个干净的仅仅用于构建目的专用工作空间，这里“干净”意味着该工作空间中没有多余的文件/目录，不存在旧的中间文件，所有文件应该是只读的，不允许进行检出和修改；
- 步骤三：执行构建过程，并对构建过程进行审计。审计信息包括但不限于：谁执行的构建？什么时候执行的构建？构建生成的可执行文件或库包含哪些内容？执行构建的机器是什么？机器上运行的操作系统版本是什么？执行构建使用的是什么编译器？使用了编译器的哪些选项？等等。构建审计是确保构建重现性和可维护性的有力保证。
- 步骤四：对构建和审计过程中产生的导出文件（Derived objects）进行版本控制。这一点很重要，正是这些导出文件构成了软件发布的重要组成部分。
- 步骤五：为业已受控的导出文件建立基线。

- 步骤六：生成软件发布介质。

构建的高效依赖于对导出文件在多次构建中的共享，通过避免反复生成同样的导出对象来大大缩短构建的时间。

2.6 异地开发支持

经济全球化趋势、越来越多的企业兼并以及软件规模和复杂性的不断增加等诸多因素使得地理上分布的多个软件开发团队进行协作开发的开发模式日益普遍。这些团队可能分布在同一个城市的不同的办公地点，也可能分布在一个国家的不同城市，甚至分布在全球不同国家的不同城市。配置管理系统应该能够有效支持以上地理上分布的团队之间的远程协作。

典型的异地开发模式具有以下特点：

- 支持跨多个地点的分布式开发：异地开发团队应该能共享开发成果，在甲地的开发工作可以在乙地进行集成；甲地可以在乙地的开发基础上进行。
- 复制和同步配置数据：无论网络带宽如何发展，相对本地网络及磁盘存取速度，远程网络的带宽总是相对有限的，因此如何高效、可靠地在异地之间复制和同步配置数据是异地开发和远程协作的关键。
- 提供配置数据的本地存取：出于效率上的考虑，开发人员的日常开发工作应该是基于本地存取的方式，而并应该依赖与远程网络连接。

需要说明的是，异地开发支持不等同于各地通过远程网络连接到单一的版本库的开发模式，由于广域网带宽的限制，单一的 版本库模式被证明是低效的和不实用的；典型的异地开发模式在每个地点都有版本库的副本，日常开发针对本地版本库进行，版本库之间具有复制和同步机制。

IBM Rational ClearCase MultiSite 是业界公认的唯一的适用于地理位置分散的项目团队的高性能软件配置管理解决方案。

2.7 变更请求管理

变更请求管理是软件配置管理的一个重要组成部分，变更请求管理记录、跟踪和报告针对软件系统的任何变更，其核心是一个适合软件开发组织的变更处理流程，典型的变更处理流程涉及如何提交变更请求，如何对变更请求进行复审以便决定是否实施，由谁实施，如何实施，如何确定变更请求准确实施完成等方面。强大而且灵活的变更管理系统是以更快的速度开发更高质量的软件产品的有力保证，也是软件项目管理的利器。

变更处理流程应该是灵活的和便于定制的，不同的软件开发组织需要不同的变更处理流程，即使在同一软件开发组织内部，针对不同的变更类型，或不同的软件开发项目，往往也需要不同的流程。

变更请求管理系统应具备强大的统计、查询和报告功能，及时准确报告软件的变更现状，开发团队的工作进展和负荷，软件的质量水平以及变更的发展趋势。

典型的变更请求有：新的功能需求、对已有功能的优化和改进、针对发现的缺陷的修复等。

IBM Rational ClearQuest 提供业界最强大而且高度灵活的缺陷及变更跟踪系统。

3. 软件配置管理最佳实践

3.1 在安全的存储库中进行工件的标识和存储

配置管理的首要任务就是要求在安全的存储库中对工件进行正确的标识和存储，从而进行有效的版本控制。需要标识和存储的工件包括：项目计划、需求文档、设计模型，源代码、库文件、可执行文件、Web 内容、测试计划、测试用例、测试脚本，等等。对以上工件进行存储是显而易见的；对工件进行正确标识的目的就是确保在需要时能够简单、快速地找到它们的正确版本。事实一再表明，在没有有效实施配置管理的项目中，寻找正确文件的正确版本是一件非常费时、困难、极易出错的工作，其后果是显而易见的：要么项目失败，要么项目进度严重滞后、要么项目质量得不到保证。

简单地在存储库中对工件进行标适和存储是不够的，鉴于存储库在整个配置管理系统中所处的核心位置，存储库应具备以下的能力和特点：

- 容错能力/可靠性：存储库应具备强大的容错能力和高可靠性，以避免单点失败/故障给企业最有价值的软件资产带来灾难性的损失和破坏；
- 分布处理/可扩展能力：随着时间的推移，存储库中不断积累越来越多的历史版本，与此同时，组织/开发团队的规模也有可能在增长，新的项目不断加入到存储库中，这些都势必对存储库的分布处理能力和可扩展能力提出很高的要求；
- 远程开发/异地同步能力：经济全球化、互联网的发展以及越来越多的企业兼并使得很多在地理上分布的开发团队要求协同开发，这就要求存储库具备远程开发/异地同步能力
- 备份/恢复能力：存储库应支持有效的备份/恢复功能，以确保企业软件资产的安全。
- 访问控制能力：访问控制能力为存储库增加了另外一层安全保护，以防止对存储库未经授权的访问，有效保护企业的知识产权。

3.2 控制并审计对工件的变更

在标识和存储工件的基础上，配置管理系统应该对工件的变更提供有效的控制手段和审计能力。控制对工件的变更指的是能够设定谁能够对哪些工件进行修改；对工件的变更进行审计指的是能够记录与工件修改相关的一些信息，如：

- 谁进行的修改?
- 修改了什么?
- 什么时候进行的修改?
- 为什么要进行修改?
- 当前发布包含哪些新功能?
- 当前发布对已有功能进行了哪些增强?
- 当前发布修复了哪些 BUG?

控制是一种主动的变更控制策略，控制过严有可能降低开发效率，没有控制则会存在安全风险；相比较而言，审计是一种被动的变更控制策略，不影响开发效率，如果有效结合组织的其他策略如问责制度，审计功能同样可以有效降低安全风险。开发组织应根据自身的实际情况，正确、协调使用这两种策略，使得既保证开发效率，又能够有效消除安全风险。

3.3 将工件组织为版本化的构件

一旦系统包含成百上千的文件和目录，为简化管理，就有必要对文件和目录进行分组，使之转化为数量上较少的、粒度上较大的组织单元，业界对此有不同的习惯称谓，如，模块、子系统、包、构件，等等。在此，我们采纳“构件”这一叫法。所谓版本化的构件就是一组相关的文件和目录，这些文件和目录作为一个单一的单元进行版本控制、基线管理、编译/建立，共享和重用。

将工件组织为版本化的构件有以下好处:

- 降低复杂性: 构件通过提高抽象层次来有效降低复杂性，使得问题更加易于管理。举例而言，如一个软件系统包含 6 个构件，这 6 个构件共由 5000 个文件组成，为重建该软件系统的一个发布，一种方法是选择每个构件的一条基线，共 6 条基线；另一种方法是为 5000 个文件各选择一个版本，共 5000 个版本。显然，基于版本化构件的方法更加简单高效，更加不易出错。
- 标识构件的质量水平比标识单个文件的质量水平更加容易，更有意义: 一条构件基线标识其包含的每个文件/目录的单一版本，对单个文件的单一版本进行测试意义不大，而构件可以作为一个单元进行测试，根据测试结果可以明确标识被测试构件基线所代表的质量水平。
- 有利于共享和重用: 有了构件基线，明确构件基线所代表的质量水平，该构件就可以在不同的项目中进行共享和重用，相反，缺少构件基线，不知道构件基线的质量水平，构件的共享和重用就没有实际意义。
- 将逻辑设计构件映射成物理实现的版本化的构件有助于保证软件架构的完整性: 其带来的好处是，更高质量的代码，更清晰的构件接口。

3.4 在项目里程碑创建基线

基线记录了一个构件中所有工件的一个单一版本，至少要在项目的里程碑创建相应的基线，这也是 ISO-9000、CMM 的要求。正确的基线管理可以确保设计与需求的一致性、代码与设计的一致性、使用正确的代码进行发布等。适时创建基线有以下好处:

- 可重现性: 可重现性指的是有能力准确回到一个先前的软件版本。回到先前的版本有助于重现历史版本中的 BUG 以协助排错，并在历史版本的基础上开发对应的补丁; 在最新版本中发现问题时，回到历史版本可以帮助判断新发现的问题到底是在历史版本中业已存在（只是当时未发现）还是仅仅在新版本中引入; 同时维护多个版本、针对不同版本进行本地化、扩展的也要求准确回到历史版本。
- 可追踪性: 可追踪性保持项目需求、项目计划、测试用例等与源代码之间的一致性，为实现可追踪性，要求不仅对源代码进行配置管理，对项目需求、项目计划、测试用例等也要进行配置管理，并对所有工件统一进行基线管理。
- 配置状态报告: 有了适时创建的基线，就可以查询、报告、比较基线的内容，尤其是基线的比较能力对于排错、为新的发布生成准确的发布说明文档（Release Notes）等非常有帮助。

除了在项目里程碑（如使用迭代化开发模式，则在每次迭代结束前）创建基线外，实际上基线的创建要频繁得多。在典型的情况下，在每次迭代的后期或软件发布的前期，甚至每天都会创建新的基线。

3.5 记录并跟踪变更请求

软件的开发过程实际上就是一个不断地由变更请求驱动的过程，良好的配置管理系统应能够有效记录并跟踪变更请求。变更请求有多种形式并且来自不同的地方，如来自内部及外部的错误报告; 来自市场及工程部门的功能增强请求; 需求、设计、及文档变更请求，等等。

我们不仅需要一个合理的流程来对变更请求进行记录和跟踪，可能的话，还应该对实现变更请求而造成的相关工件的变化结果进行跟踪，也就是说同时统一管理变更的 WHY 和 WHAT 两个方面，既为什么要变更以及变更了什么。目前能达到该层面的配置管理系统并不多见，而 IBM Rational 的统一变更流程 UCM（Unified Change Management）则是一个杰出的代表。

3.6 使用活动来组织和集成

几乎所有的配置管理系统都提供基于文件的版本控制能力，而开发人员实现一个逻辑功能往往会涉及多个文件，每个文件都存在多个版本，因此人工决定哪个文件的哪个版本可以协调工作以完成特定的功能是一项相当烦琐、低效并极易出错的工作，经常会导致编译失败，从而降低开发效率。

使用活动来组织和集成则是一项伟大的创新。其基本思想就是通过抽象层次的提升，将配置管理的中心从文件的版本控制转移到对开发活动的管理，而由工具自动实现开发活动与文件的版本变化之间的关系。所谓的活动就是各种各样的实际开发任务，或者变更请求，如增加新功能、已有功能的增强、软件错误的修复，等等。配置管理工具自动为每个活动维护一个“变更集”，即该开发活动改变了哪些文件，形成了哪些新版本。变更集作为一个单一的单元参与组织和集成，从而有效保证了版本之间的一致性。

使用活动来组织和集成有以下好处：

- 更接近自然的工作方式：基于文件的版本控制是面向机器、面向代码的；而基于活动来组织和集成是一种更接近自然的工作方式，该方式有利于开发团队的沟通，无论是客户支持、项目经理、开发人员还是测试人员，都可以基于开发活动进行有效的沟通与交流。
- 对手工工作进行了自动化：由于工具自动为每个开发活动维护一个一致的变更集，基于活动可以对其变更集进行统一的检出（Check out）、检入（Check in）、集成、编译和建立。
- 更少的编译和集成错误：由于活动的变更集对应一个一致的逻辑改变，因此更加易于编译和继承，不易出错。
- 提供了对错误和项目管理的逻辑链接：对错误的管理乃至整个软件项目的管理本质上就是基于开发活动的，项目经理可以基于活动来分配开发任务，开发人员进行开发活动并自动更新开发活动的状态、项目经理可以轻松查询开发活动的状态，并据此掌控项目进度和作出及时的调整。

IBM Rational 的统一变更流程 UCM 就是以活动为中心的业界最佳的配置管理解决方案。

3.7 维护稳定和一致的工作空间

维护稳定和一致的工作空间是实现并行开发、提高开发效率的必要前提。存在两类工作空间，一类是开发人员的私有空间，在私有空间中，开发人员可以相对独立地编写和测试自己的代码，而不受团队中其他开发人员工作的影响，即使其他人也在修改同样的文件；另一类工作空间是团队共享的集成空间，该空间用于集成所有开发人员的开发成果。

所谓工作空间的稳定性指的就是私有空间的相对独立性，在私有空间中，开发人员可以相对独立地编写和测试自己的代码，而不受团队中其他开发人员工作的影响。每个开发人员都有自己的私有工作空间，不同开发人员的私有工作空间是相互独立、彼此隔离的。

所谓工作空间的一致性指的是当开发人员对自己的私有空间进行更新时，得到的应该是一个可编译的、经过一定测试的一致的版本集。

3.8 支持对构件的并行开发

传统的串行开发模式在同一时间只允许一个人对同样的文件进行修改，其他需要修改同样文件的人只能等到前面的人修改完成后再开始自己的修改，这样的好处是不会出现修改上的冲突，但在当今的市场环境下，这种串行开发模式显然是行不通的，因为它既不现实、也缺乏效率，取而代之的是并行开发模式。

配置管理范畴下的并行开发模式建立在独立的、相互隔离的私有空间和配置项分支的基础之上。在 3.7 中我们已经讨论了私有工作空间和集成工作空间的概念，这里我们对配置项分支作一些说明。我们可以将串行开发模式下的同一个文件的全部历史版本依次连接起来，由此形成只有一个分支（或者叫主干）的版本树，为实现并行开发，需要在主干的相应节点（一个节点代表一个版本）上建立分支，创建分支的过程实际上就是一个建立副本的过程，这样每个开发人员都可以建立属于自己的分支，从而拥有自己私有的副本，开发人员可以在自己的私有工作空间基于自己私有的副本进行开发而不相互影响，这就是并行开发的实质。当然仅仅有分支的能力还是不够的，当开发人员在自己的私有空间完成阶段行开发后，需要一种合并的机制来集成全部开发人员各自的开发成果，这正是在集成工作空间中完成的。因此，并行开发实际上需要在隔离和集成之间寻找一种有效的平衡。优秀的配置管理工具应具备强大的分支和自动化合并的能力，以有效支持并行开发，提高开发效率。

3.9 尽早和经常进行集成

有效的并行开发需要隔离，但过度的隔离会带来后期集成的噩梦，集成是并行开发模式下必不可少的重要环节。因此，在并行开发模式下，几乎所有的项目都要面临集成和隔离的矛盾，在集成和隔离之间寻找适合团队、适合项目的有效平衡是成功的关键。开发人员应该总是可以控制自己的私有工作空间；管理人员和集成人员必须在不牺牲个人开发效率的前提下用一种可靠、可行的方式进行集成以使开发人员同项目的进度保持一致。

尽早和经常进行集成可以使得每个人都基于最新的代码展开工作，在迭代化开发模式下这点尤其重要。集成得越早，越有可能尽早发现问题，这样修复问题的代价就越小；相反，较晚或不经常进行集成会不利于迭代，进而引发更多风险。

3.10 确保软件构建的再现性

有时出于排错的需要，或需要重现相同的构建（Build），我们需要知道软件是如何被构建的，构建中包含哪些内容，这就要求配置管理系统提供构建审计功能。构建审计功能应能自动记录以下内容：谁执行的构建？什么时候执行的构建？构建生成的可执行文件或库包含哪些内容？执行构建的机器是什么？机器上运行的操作系统版本是什么？执行构建使用的是什么编译器？使用了编译器的哪些选项？等等。有时候仅仅改变编译器的优化选项开关就可能引入新的错误，有了构建审计功能，就有可能进行不同构建的比较，从而有利于排错。缺乏软件构建的再现能力就很难进行软件系统的维护，对在客户现场运行的系统更难提供有效的技术支持。

IBM Rational 的 ClearCase 就具备强大的、独特的构建审计和管理能力。

4. 软件配置管理策略

4.1 对于同一产品的维护和开发

对于典型的软件产品，往往是针对当前发行版本乃至更旧的版本的维护、支持工作与新版本的开发同时进行：对已有版本的维护支持主要是修复其中发现的缺陷、开发相应的补丁程序，以保证用户的正常使用，从而提高产品的用户满意度，保持产品的市场占有率；新版本的开发主要是开发新的功能特性、或实现对新的平台、新的技术的支持。由此可见，对于同一产品的维护和开发，其相同之处是，二者都是针对现有产品的修改，都属于软件开发的范畴，都具有软件开发生命周期的共同特点，如编码、测试、集成、发布等；但二者也有明显的区别，主要表现在开发的目的不同、开发内容不同、周期长短不同、发布形式不同。

如何有效协调同一软件产品的维护和开发工作，使之有序、高效，这正是软件配置管理应该解决的问题。该问题的解决依赖于配置管理工具对并行开发、分支、自动化合并、项目管理等功能的支持。IBM Rational ClearCase 对此提供了很好的支持。

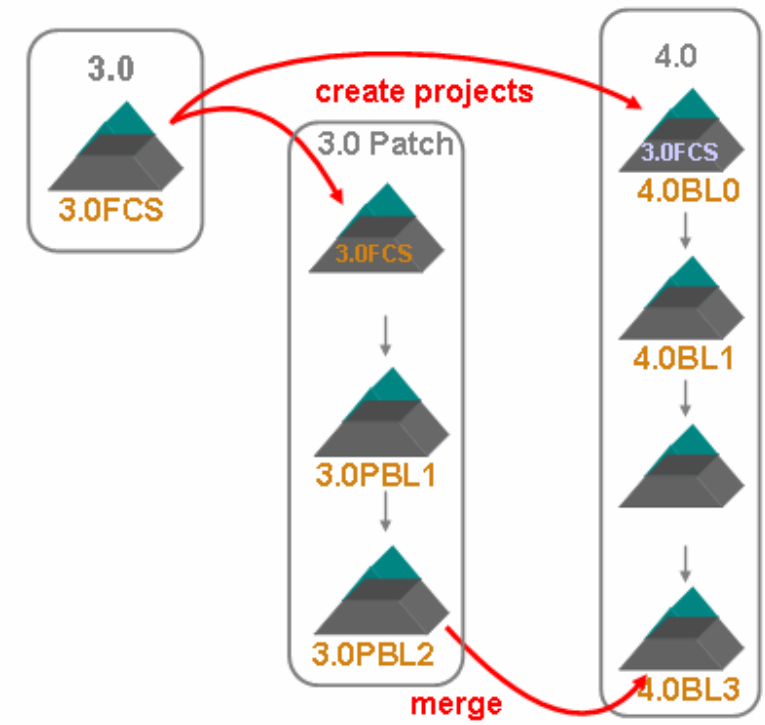

图 2：对于同一产品的维护和开发

以图 2 为例进行说明，某软件的当前发行版本为 3.0FCS，为同时开展针对 3.0FCS 的维护工作和下一版本 4.0 的开发工作，分别以 3.0FCS 为基线创建两个分支：分支 3.0 Patch 和分支 4.0，两个分支可以作为两个不同的项目进行并行开发，二者的开发工作相互独立、互不影响，这一点很重要，是并行开发高效、有序的保证，两个项目在各自的开发过程中根据需要可以创建各自的基线，如维护工作基线 3.0PBL1、3.0PBL2，开发工作基线 4.0BL0、4.0BL1、4.0BL2、4.0BL3 等。3.0 Patch 开发完成后即可作为针对当前发行版本 3.0FCS 的补丁程序进行发布。

鉴于如上所述的 3.0 Patch 和 4.0 开发的独立性，4.0 中并不包含 3.0 Patch 的内容，这就是说，4.0 版中仍然包含 3.0FCS 中的缺陷，为了达到对同一缺陷“一次修复，永不重现”的目标，需要一种机制将 3.0 Patch 的内容合并（merge）到 4.0 中，这就需要配置管理工具提供智能的、自动化的合并功能。

通过以上分支、并行开发、合并的策略，4.0 版中不仅包含新功能，同样包含对前一版本 3.0FCS 中缺陷的修复，这正是我们所期望的。

4.2 对于不同特性的并行开发

软件产品往往包含诸多特性，不同的特性之间往往又有较强的独立性，在实际的开发过程中，不同的特性往往由不同的开发人员或开发小组分工完成。如果不进行有效的分支管理甚至不建立分支，势必存在以下问题:

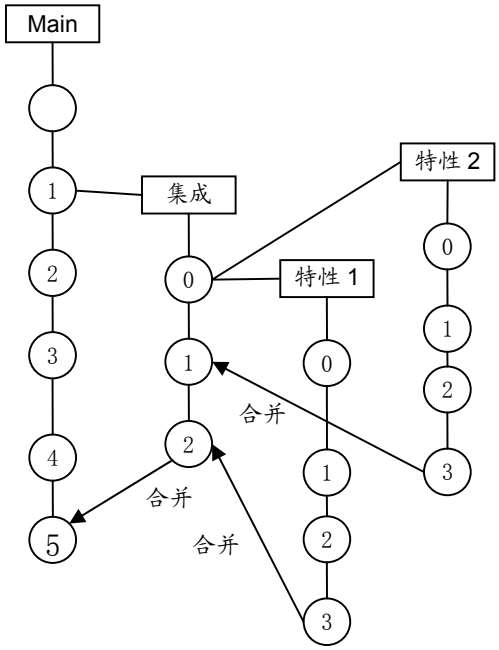

图 3: 对于不同特性的并行开发

- 集成或“合版本”时，正常的开发被迫停止，即使是单一特性的集成往往也会造成其他特性的开发无法进行，其后果必然是开发效率的降低、项目整体进度的延误、项目预算的超支等;
- 由于个别特性无法按期完成而影响整个项目的进度。举个极端的例子，项目 99%的特性业已完成，由于 1%的未完成的特性造成整个项目无法按期发布，最终用户可见和可使用的软件特性实际为 0。理想的情况是，项目完成多少特性就应可以发布多少特性，仍以上例而言，项目可以按期发布 99%的特性，最终用户可见和可使用的软件特性实际为 99%。

为达到上述的理想情况，可采取如下针对不同特性的并行开发策略。

以图 3 为例进行说明，首先在主分支（main 分支）上建立一个集成分支，然后在集成分支上再根据不同特性进行分支，每个特性拥有自己独立的分支，如分支“特性 1”对应软件特性 1，所有关于特性 1 的开发都在该分支上进行，不同特性的开发由于在各自的特性分支上进行，因此相互独立、互不影响。开发完成的特性通过智能的、自动化的合并功能集成到集成分支上，显然，该集成过程不影响其他特性的正常开发，全部业已完成的特性进一步合并到主分支进行测试和发布，从而实现“完成多少，发布多少”的管理目标。

4.3 同一产品不同版本之间的并行开发

在$2.3 中已经谈到同一产品不同版本之间并行开发的必要性，以图 4 为例说明该并行开发配置管理策略的实现机制，该机制与$4.1 中对于同一产品的维护和开发存在一定的相似性。

假设 3.0 版的开发尚未结束，当前基线为 3.0BETA，开发工作在分支 3.0 上进行，此时决定启动 3.1 版的开发，实现 3.0 和 3.1 两个版本的并行开发。首先应以基线 3.0BETA 为基准，从分支 3.0 创建新的分支 3.1，两个分支上的开发可以作为两个项目进行管理，在其上的开发相对独立、互不影响。3.0 分支上的开发继续完成 3.0 中尚未完成的开发任务，可以完全不理会 3.1 开发分支的存在；3.1 的开发则在 3.0BETA 的基础上进行 3.1 新功能的开发，3.1 中依赖于 3.0 尚未完成功能的部分可以延缓开发，优先开发在 3.0BETA 的基础可以展开的新功能的开发工作。

当 3.0 的开发完成，即到达基线 3.0FCS 时，可以进行 3.0FCS 的发布，与此同时，将 3.0 分支自 3.0BETA 至 3.0FCS 的变化合并到分支 3.1 中，使得 3.1 包含 3.0 的全部功能，然后继续进行 3.1 的后续开发。

两个以上更多版本的并行开发依次例推，依次进行低版本向高版本的单向合并，不需要进行高版本向低版本的合并。

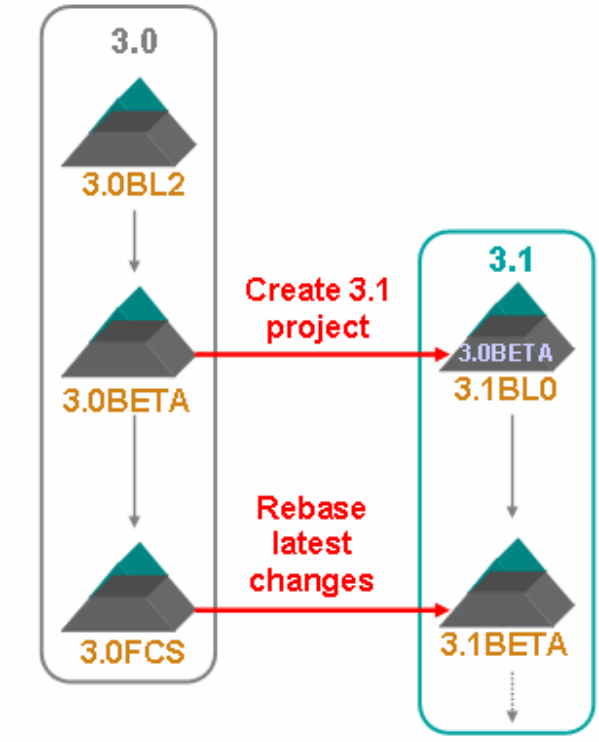

图 4: 同一产品不同版本之间的并行开发

4.4 基于同一个产品的多个定制版本开发

基于同一产品开发多个定制版本是一个非常典型的开发模式，可以是为不同的用户建立不同的定制版本，也可以是针对不同的平台，如 Windows，Linux，UNIX，建立不同的定制版本，还可以针对不同的技术/相关产品建立定制版本，如使用 DB2 和使用 Oracle 的版本。如何既保证产品的一致性，又维护特定定制版本的特殊性，提高开发效率，降低产品维护/升级成本正是配置管理策略应研究的问题。

解决此类开发问题的核心是确定产品的基准版本，或者叫标准版本，也就是各个定制版本的公共部分。

假设产品的标准版本已经建立，如图 5 所示，标准版本拥有自己的分支，以该分支为主分支，可以为各个定制版本建立各自独立的分支，也可以建立针对标准版本的补丁分支。针对标准功能的后续开发在主分支上进行，针对标准版本的缺陷修复在补丁分支上进行，并适时合并到主分支上，定制版本的分支上只进行定制部分的开发，并适时进行从主分支到定制版本的合并，使得标准功能能够覆盖到全部的定制版本中。

当然，如果定制版本之间具有明显的相似性或继承性，也可以从一个定制版本的分支上再建立另外一个定制版本的分支，如可以在针对 Windows 2000 的定制版本的分支上建立针对 Windows XP 的定制分支，而不是直接从标准版本建立分支，从而最大限度实现重用。

需要说明的是，在以上开发策略中，从标准分支向定制分支的单向合并是有意义的，反之，从定制分支向标准分支的合并则是无意义的和不必要的。

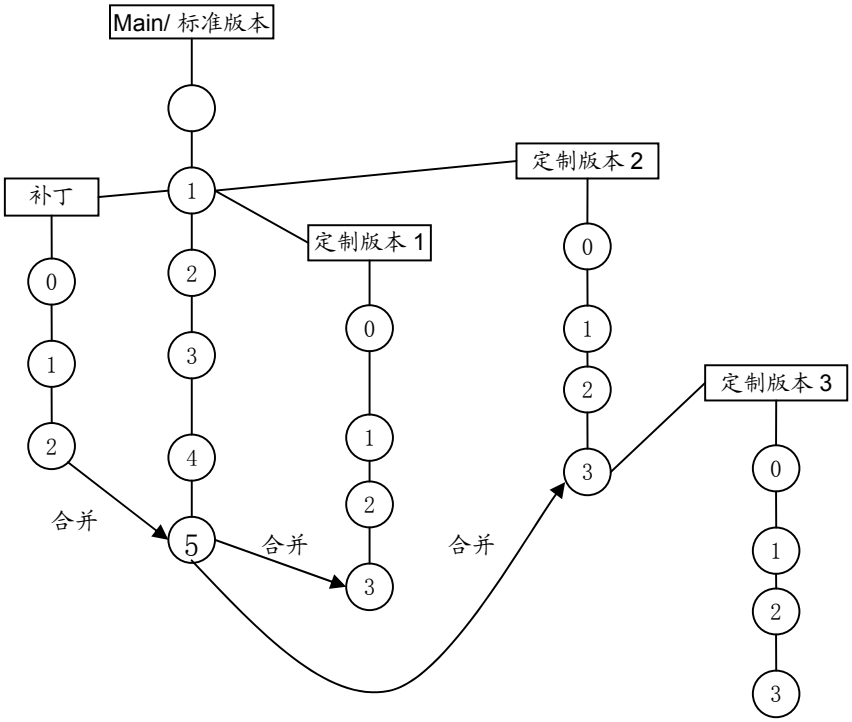

图 5: 基于同一个产品的多个定制版本开发

5. 软件配置管理工具

5.1 配置管理工具 ClearCase

IBM Rational ClearCase 提供业界最强大的软件配置管理解决方案，在全球拥有超过 240，000 用户。IDC（国际数据集团）在其报告“2001-2005 软件配置管理工具预测和分析”中将 ClearCase 评为“连续三年最畅销的软件配置管理解决方案”。ClearCase 的突出特点包括:

- 业界领先的统一变更流程
- 强大的并行开发功能
- 灵活的工作空间管理
- 成熟稳定的配置管理平台
- 强大的扩展能力，可从单个工作组扩展到跨国公司的多个远程团队
- 多地开发支持(MultiSite)

5.2 缺陷及变更管理工具 ClearQuest

IBM Rational ClearQuest 提供业界最强大而且高度灵活的缺陷及变更跟踪系统，在全球拥有超过 160，000 用户。ClearQuest 的突出特点包括:

- 便于定制缺陷和变更请求的信息域、流程、用户界面、查询、图表和报告等;
- 提供自动电子邮件通知;
- 高可扩展性，可支持任意团队规模、成员位置或使用任意平台的项目;
- 自动将变更传送给所有平台（Windows，UNIX 和 Web）的客户端界面上;
- 利用强大、无错的复制和同步机制，使地理位置分散的团队能即使访问缺陷和变更数据;