版本控制规范

# 简介

## 目的

版本控制规范用于确定软件配置项的命名与版本号管理的规则，以确保清楚地、唯一地标识软件的各个组成部分及其状态，并建立这些部分之间的一致性关系。

## 范围

版本控制的范围包括：

* 源代码：用计算机编程语言编写的源代码文件
* 文档：需求文档、架构设计文档、数据库设计文档等描述软件功能和结构的技术文档；项目计划等项目管理文档以及各种测试文档和用户文档
* 产品包：将源代码进行编译得到的可运行的软件系统

# 产品标识

在每个软件产品立项时建立该软件产品的标识，以唯一地代表一个软件产品或项目，产品标识也称为项目标识。



## 产品名称

新产品立项时，为产品赋予产品名称；当已有产品升级时，则沿用前一版本产品的名称。

产品名称包括：

* 产品中文名称：如：订单管理系统，仓库管理系统等等
* 产品英文名称：如：Order Management System，Warehouse Management System
* 产品英文简称：如：OMS，WMS

产品名称用于相关文档的编写和产品的发布。

产品名称不是某一产品的唯一标识，必须与版本号一起用才能标识特定产品。

## 版本号

版本号用来标识开发、测试、交付阶段的不同状态的产品，版本号格式为：

<主版本号>**.**<次版本号>**.**<小版本号>-[Build号]

* 主版本号：立项时设置，在整个项目开发过程中不改变
* 次版本号：立项时设置，在整个项目开发过程中不改变
* 小版本号：立项时设置，在整个项目开发过程中不改变
* Release号：又叫Build号，内部测试开始之前设置，初始值为0，此后每产生一次小的修改，Release号+1

版本号的一般形式如：1.0.7-101，2.0.0-900

# 版本规范

## 版本号设置规则

### 主版本号

1. 设置时间：产品立项时设置
2. 设置规则：

* 新产品立项，主版本号为1
* 产品构架发生改变，主版本号+1
* 产品主要组件（比如订单处理框架）进行重大修改，主版本号+1
* 产品对外接口协议发生更改，主版本号+1

### 次版本号

1. 设置时间：产品立项时设置
2. 设置规则：

* 新产品立项，次版本号为0
* 为处理产品Bug或改进现有功能/性能，对现有功能模块做大的修改，但不增加新的功能模块，副版本号+1
* 为增加产品功能，在原版本产品上增加新的功能模块，而产品的主体构件未做重大修改，并且产品的主体构件之间的接口协议也未做修改，副版本号+1
* 为适应不同用户需求，对产品进行更改，而产品的主体构件未做重大修改，并且产品的主体构件之间的接口协议也未做修改，副版本号+1
* 当主版本号变更时，副版本号同时置0

### 小版本号

* 新产品立项，小版本号为0
* 修复Bug或改进现有功能，但不对现有功能模块做大的修改，不增加新的功能模块，小版本号+1
* 当次版本号变更时，小版本号同时置0

### Build号

1. 设置时间：产品开发结束，内部测试开始之前
2. 设置规则：

* Release号初始值为0
* 测试过程中，每进行一次修改，Release号+1

## 版本管理



### trunk

任何时候trunk里包含的都是最新的开发代码。 这里的代码将会工作到下一个主要发布版本。

trunk应该只被用来开发将会成为你的下一个重要版本的代码。 不要给trunk加上版本号和发布名称。 仅需要保证trunk在任何时候都处于“开发模式”。

### branches

有几种不同类型的分支。在branches的目录里，可以为更多具体的目标创建路径，像即将发行版本。Brahches可以包含了trunk在不同发展阶段的副本。

#### Release Branches

当trunk达到准备发布的阶段时（或者你想冻结新特色的添加时），应该创建一个release branches。 Release branches只是当前trunk的一个副本。

这个branches可以被单独的签出，也可以启动branches和基于此版本的项目。还可以使用此分支在测试期间修复Bug。 这种方式能够保证trunk继续开发，而不会被发布某个具体的版本所干扰。 因此当准备发布一个新版本时，不会影响trunk增加新的功能。

#### Bug fix branches

分支也可以用于处理trunk或release branches里发现的严重的Bug。这些Bug很复杂，不能在一次提交时就修复他们。因此为了集中精力修正此错误，应该为此问题创建一个新的分支。这样就不会影响trunk 和 release branches的继续进行，并且也不会因为发现新的Bug 和测试而干扰此Bug 的修复。

#### Experimental branches

有时想将某个新技术引进项目。但是不想影响到整个项目。比如想把web应用从spring3x改为spring4x。要花多少时间？在这期间trunk停止使用？直到把所有到spring的转换做完。

可能Spring4x对程序变动较大，应该创建一个实验分支。 这样就可以在分支里进行更改，如果失败了，不影响当前应用，实验分支可以抛弃。 如果成功，可以很容易的将其合并到trunk。

### tags

tags用来备份代码，通常是readonly的，不被用来开发，只是用来标记代码的状态。

#### 1.3.1 Release tags

Release Tags 标记版本发布点的代码。 Release Tag 永远是相应发布分支的副本。 Release Tag命名规则：版本号+“Release”后缀。

# SVN使用规范

## 先更新，再提交

SVN更新的原则是要随时更新，随时提交。当完成了一个小功能，能够通过编译并且自己测试之后，谨慎地提交。

如果在修改的期间别人也更改了svn的对应文件，那么commit就可能会失败。如果别人和自己更改的是同一个文件，那么update时会自动进行合并，如果修改的是同一行，那么合并时会产生冲突，这种情况就需要同之前的开发人员联系，两个人一起协商解决冲突，解决冲突之后，需要两人一起测试保证解决冲突之后，程序不会影响其他功能。

在更新时注意所更新文件的列表，如果提交过程中产生了更新，则也是需要重新编译并且完成自己的一些必要测试，再进行提交。这样既能了解别人修改了哪些文件，同时也能避免SVN合并错误导致代码有错。

## 多提交

每次提交的间歇尽可能地短，以几个小时的开发工作为宜。例如在更改UI界面的时候，可以每完成一个UI界面的修改或者设计，就提交一次。在开发功能模块的时候，可以每完成一个小细节功能的测试，就提交一次，在修改bug的时候，每修改掉一个bug并且确认修改了这个bug，也就提交一次。提倡多提交，也就能多为代码添加上保险。

## 不要提交不能通过编译的代码

代码在提交之前，首先要确保能够在本地编译。项目经理在准备项目工作区域的时候，需要确保开发小组成员在签出代码之后能够在统一的环境中进行编译。

## 每次提交必须书写明晰的标注

在一个项目组中使用SVN，如果提交空的标注或者不确切的标注将会让项目组中其他的成员感到很无奈，项目经理无法很清晰的掌握工作进度，无法清晰的把握此次提交的概要信息。在发现错误后也无法准确的定位引起错误的文件。所以，在提交工作时，要填写明晰的标注，能够概要的描述所提交文件的信息，让项目组其他成员在看到标注后不用详细看代码就能了解你所做的修改。

## 提交时注意不要提交本地自动生成的文件

例如eclipse中的.classpath文件，Windows生成的缩略图Thumbs.db，项目编译生成的临时文件.obj, .class等等。如果项目中没有进行这方面的配置来强行禁止提交这样的文件，请自觉不要提交这样的文件。提交了这样的文件后，别人在更新后就可能与本地的环境冲突从而影响大家的工作。

## 不要提交自己不明白的代码 代码在提交入SVN之后，你的代码将被项目成员所分享。如果提交了你不明白的代码，你看不懂，别人也看不懂，如果在以后出现了问题将会成为项目质量的隐患。因此在引入任何第三方代码之前，确保你对这个代码有一个很清晰的了解。

## 慎用锁定功能 在项目中要慎用锁定的功能，在你锁定了一个文件之后别人就无法继续修改提交该文件，虽然可以减少冲突的发生率，但是可能会影响项目组中其他人员的工作。平时只有在编辑那些无法合并的文件（例如图片文件，flash文件等）时，才适当的采用锁定操作。

# 目录结构

示例

