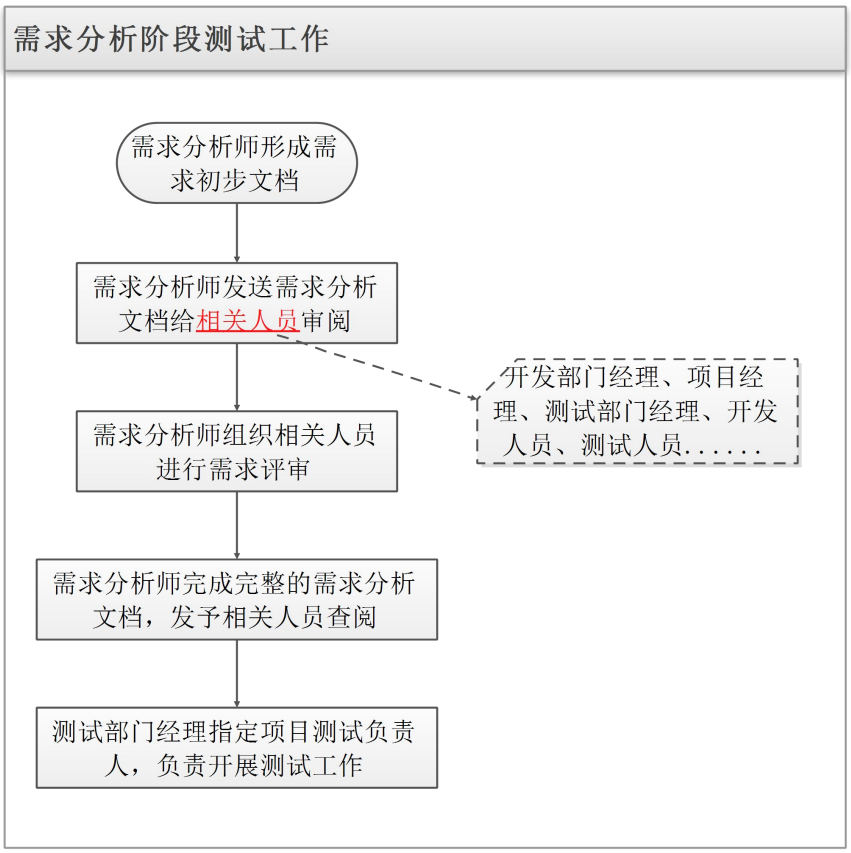
测试部门管理规范

编写人	曾一岚
邮　箱	elanzeng@hotmail.com

目 录

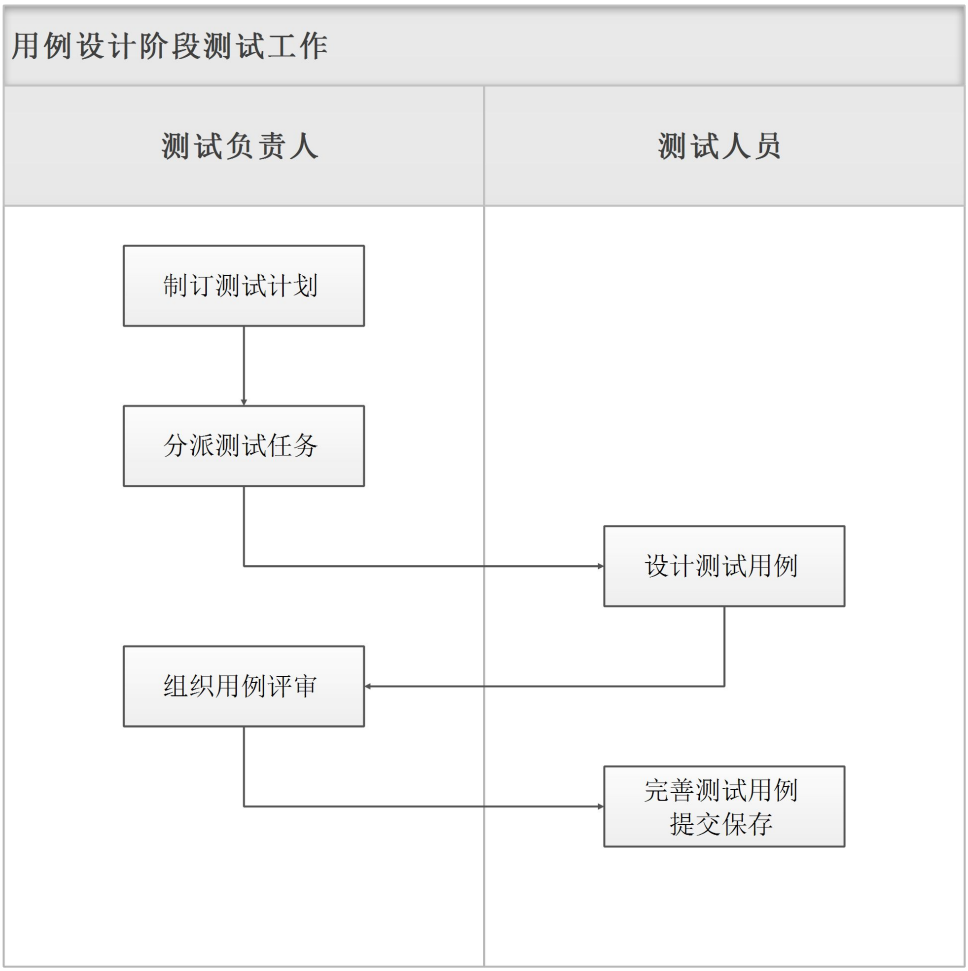
一 测试流程规范

1 需求分析阶段



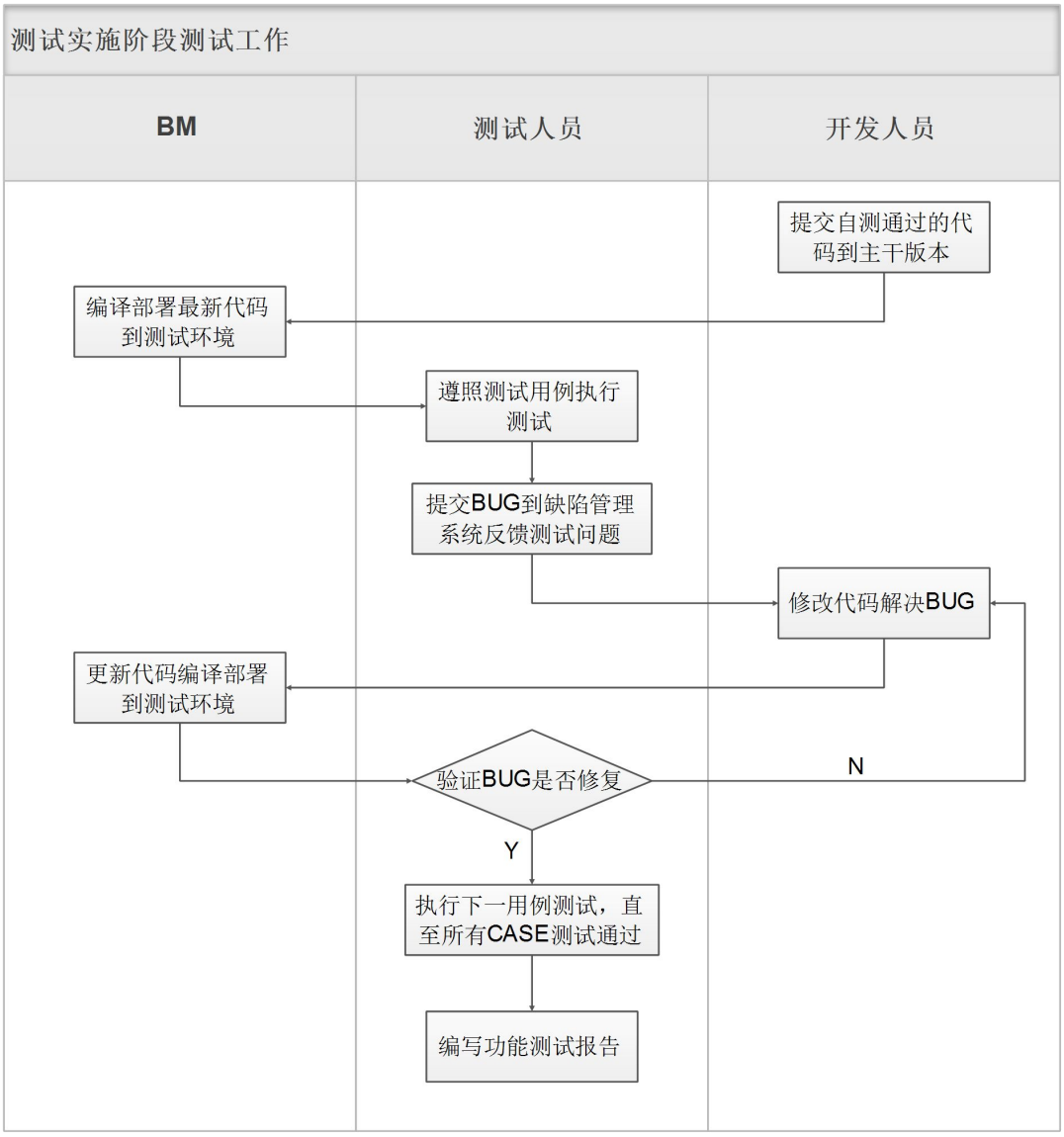
1. 需求分析人员搜集、提炼需求信息，形成初步的需求分析文档，发送给开发部门经理、项目经理、测试部门经理，及相关的开发人员和测试人员审阅。
2. 需求分析人员、开发部门经理、项目经理、测试部门经理、开发人员、测试人员组成需求评审小组，召开需求评审会议，确定需求的最终内容。
3. 需求分析人员根据需求评审会议决议，形成规范的需求分析文档，发送给开发部门经理、项目经理、测试部门经理，及相关的开发人员和测试人员。
4. 测试部门经理整合项目相关信息，指派项目测试负责人负责开展测试工作。

2 用例设计阶段



1. 项目经理根据需求分析文档制订出相应的开发计划后，测试负责人根据开发计划内容制订对应的测试计划，发送给开发部门经理、项目经理、需求分析人员、开发人员和测试人员。
2. 测试负责人将需求内容细分成任务单，分派给具体的测试人员。
3. 测试人员针对每一个测试任务单设计详尽的测试用例，发送给测试负责人及项目组内相关的测试人员。
4. 测试负责人组织项目组内测试人员及开发相关人员召开用例评审会议，对测试用例进行评审，提出完善方案。
5. 测试人员根据用例评审会议结果完善各自的测试用例，形成规范的系统测试用例文档提交到文档服务器保存。

3 测试实施阶段



1. 开发人员将自测通过的代码提交给 BM（Build Master），及将需求解决方案提供给项目测试人员。
2. BM 将最新的代码部署到测试环境，通知测试人员启动测试。
3. 测试人员检查需求解决方案是否符合规范、内容完整，否则退回开发人员重新完善。
4. 测试人员遵照测试计划、测试用例及需求解决方案对系统进行逐项测试，测试过程中发现的 BUG 通过缺陷管理工具反馈给对应的开发人员。

5. 开发人员将修复 BUG 的相关代码提交给 BM，并通过缺陷管理工具回复测试人员。BM 定时将代码更新到测试环境，通知测试人员测试。
6. 测试人员根据开发回复的 BUG 修复说明对 BUG 进行回归测试，直至 BUG 测试通过。
7. 系统功能测试通过后，测试人员根据测试结果编写各自任务单的测试报告，发送给测试负责人汇总。
8. 项目负责人汇总各测试人员的测试报告，形成完整的系统测试报告。

4 性能测试

系统功能测试通过之后，安排专业的性能测试人员使用性能测试工具对系统进行全面的性能测试，分析系统存在的瓶颈，协调相关人员采取相应措施进行优化，形成性能测试报告。

5 产品发布

产品经理确认产品达到测试计划所制定的产品质量目标和测试质量目标后，安排人员整理产品发布包及编写相关文档，汇总到 BM 处，择期正式发布。

产品发布包含的文档如下：

《产品白皮书》

《系统测试报告》

《性能测试报告》

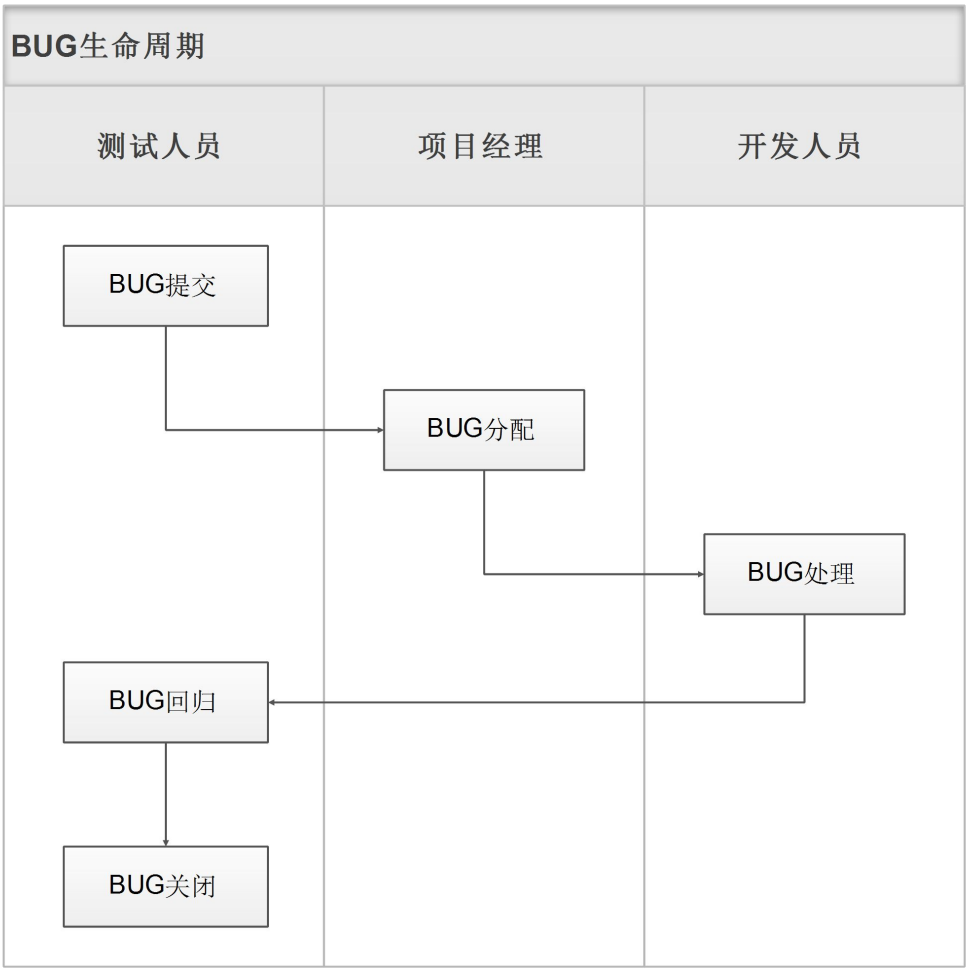
《产品安装手册》

《产品维护手册》

《用户操作手册》

二 缺陷管理规范

1 BUG 生命周期



2 BUG 判断规则

1. 软件未达到产品规格说明书标注的功能。
2. 软件出现了产品规格说明书指明不会出现的错误。
3. 软件功能超出了产品规格说明书指明的范围。
4. 软件未达到产品规格说明书未指出但应达到的目标（隐含需求）。
5. 测试人员认为软件难以理解、不易使用、不人性化、运行速度缓慢，或

用户体验需要改进。

3 BUG 分类

1. 功能问题
   - 功能未实现
   - 功能实现与需求设计不符
   - 功能重复
   - 功能多余
2. 界面问题
   - 界面风格、文字、按钮不符合规范
   - 操作繁杂
   - 提示信息模糊
   - 提示信息内容出现专业的代码错误信息
3. 数据问题
   - 数据类型错误
   - 数据计算错误
   - 数据初始化错误
   - 数据边界值相关错误
4. 安全性问题
   - 数据有效性检测不合理
   - 重要数据传输过程没有加密
   - 缺少身份认证机制或认证不合理
   - 系统日志未作级别控制
5. 性能问题
   - 并发量
   - 数据量
   - 压缩率
   - 响应时间
6. 配置问题

7. 兼容性问题
8. 编译问题
9. 设计问题

4 BUG 状态类别

序号	状态类别	说　明
1	新建状态（NEW）	BUG创建后的初始状态
2	已分配状态（ASSIGNED）	BUG经过确认后分配给具体开发人员的状态
3	待验证状态（RESOLVED）	BUG经开发人员修改、处理后的状态
4	退回（REJECT）	BUG描述不清晰，被开发人员退回
5	无需修复（INVALID）	提交的BUG确认不是问题，无需修改
6	重复提交（DUPLICATE）	BUG已经提交过，属于重复提交
7	后续处理（LATER）	因其他原因BUG暂时无法修复，后续进行修改。需明确修改的先决条件和修复时间
8	重打开状态（REOPENED）	回归不通过的状态，或延迟的BUG可进行处理的状态
9	修正状态（VERIFIED）	BUG回归通过后的状态
10	关闭状态（CLOSED）	BUG生命周期结束标识

5 BUG 严重级别

1. P1 级：严重问题

   软件非法退出

   软件运行导致机器死机

   软件出现死循环

   软件运行导致数据库死锁

   数据通讯错误

   数值计算严重错误

2. P2 级：较严重问题

   功能未实现

   功能实现不符合原始需求

数据流错误

程序接口错误

数值计算错误

3. P3 级：一般问题

界面错误

打印相关错误

附件上传失败

提示信息错误

字段类型控制错误

4. P4 级：轻微问题

数据格式不规范

页面显示不规整

信息提示不明确

必填字段未标识

附件格式未定义

5. P5 级：优化意见或建议

6 BUG 处理级别

紧急

优先

一般

低级

可延迟

7 BUG 提交内容规范

序号	内容信息	说　明
1	BUG ID	BUG的唯一标识，一般由缺陷管理工具自动生成
2	BUG标题	简明扼要描述BUG信息
3	测试版本	标注测试包的版本信息
4	测试平台	标注测试的环境信息，如Windows或Unix
5	BUG类型	明确BUG所属的类型
6	严重级别	明确BUG的严重级别
7	处理级别	明确BUG的处理级别
8	测试模块	标注功能测试所属的模块
9	测试CASE描述	描述测试case所验证的功能点及期望结果
10	测试问题描述	描述测试过程中发现的缺陷
11	测试步骤	详细描述测试过程中发现缺陷的步骤
12	测试数据	标注缺陷发现过程中的数据信息
13	测试日志/图片/附件	附上缺陷出现时的日志、图片等相关信息
14	备注	针对产生的缺陷附加的相关信息

8　有效的 BUG 填写原则

1	描述准确	只客观描述缺陷的内容和本质，不带主观性的评论
2	对象单一	一个错误报告仅对应一个BUG，避免出现一个错误报告对应多个BUG的情况
3	类型明确	根据BUG产生的现象，准确定位BUG所属的类型
4	步骤清晰	清楚地描述BUG产生的前置条件和详细操作步骤，尽可能的让BUG能够重现
5	附加必要的文档	附上截屏或日志信息，方便对问题迅速定位
6	附加个人建议或注释	沟通BUG修复期望达到的目标

9　BUG 修复信息规范

BUG产生原因：	
BUG解决方案：	
修改涉及文件：	
是否自测通过：	
开发人员：	
修改时间：	

10 BUG 回归信息规范

验证版本：	
验证结果：	
验证描述：	
注意事项：	
验证人员：	
验证时间：	

三 版本管理规范

以 SVN 作为版本管理工具为例制订以下管理流程。

代码库

代码库(repository)，也称为版本库，通常包含三个目录：主干（trunk）、分支（branches）、标签（tags）。这三个目录本质上一样都是分支，只是为了管理的方便，用来存放不同的版本。

主干（trunk）：开发组提交的稳定的版本。一般只有一个目录。

分支（branches）：开发组开发的版本目录。通常分为两类：

a、日常开发分支。如【./branches/dev】，一般只有一个。

b、紧急版本开发分支。如【./branches/dev_patch/XXX_V 版本 号 _ 日期_P_编译版本号_补丁序号】，该分支是基于标签版本建立的模块增量分支，

可有建立多个。

标签 tags：用来存放测试通过、已经发布的版本。通常分为两类：

a、大版本标签版本。如【ProName_V 版本号_日期】。

b、补丁标签版本，为紧急更新的版本。如【ProName _V 版本号_日期_P 补丁序号】。

代码库的管理一般如下范例所示：

CRM/				
	trunk/			主干版本，存放稳定的版本
		crm/		合版后的内核代码
			……	
	branches/			存放开发人员的代码库
		dev/		开发分支，仅一个
			……	
		dev_patch/		紧急更新开发分支，可多个
			crm_vX.X.X_20131030_p01	紧急更新开发分支
			……	
	tags/			存放测试通过、已经发布的版本
		2013/		
			crm_vX.X.X_20131030	大版本标签版本
			crm_vX.X.X_20131030_p01	紧急更新标签版本
			……	

Subversion 的版本号按代码库的全局版本号从 1 开始自增长分配，和其它版本控制系统不同的是，Subversion 的版本号是针对整个目录树的，而不是单个文件。每一个版本号代表了一次提交后版本库整个目录树的特定状态，需要注意的是，一个文件的版本 M 和 N 并不表示它们一定不同。

工作副本

工作副本是本地机器一个普通的目录，存放代码库上的文件拷贝，可以任意编辑。如果是源代码文件，可以直接进行编译。工作副本是私有工作区，在提交操作之前，Subversion 不会把当前的修改与其他人的合并，也不会把当前的修

改展现给他人。

可以使用【Checkout】从版本库创建一个工作副本，然后使用【Commit】将改动上传到版本库。

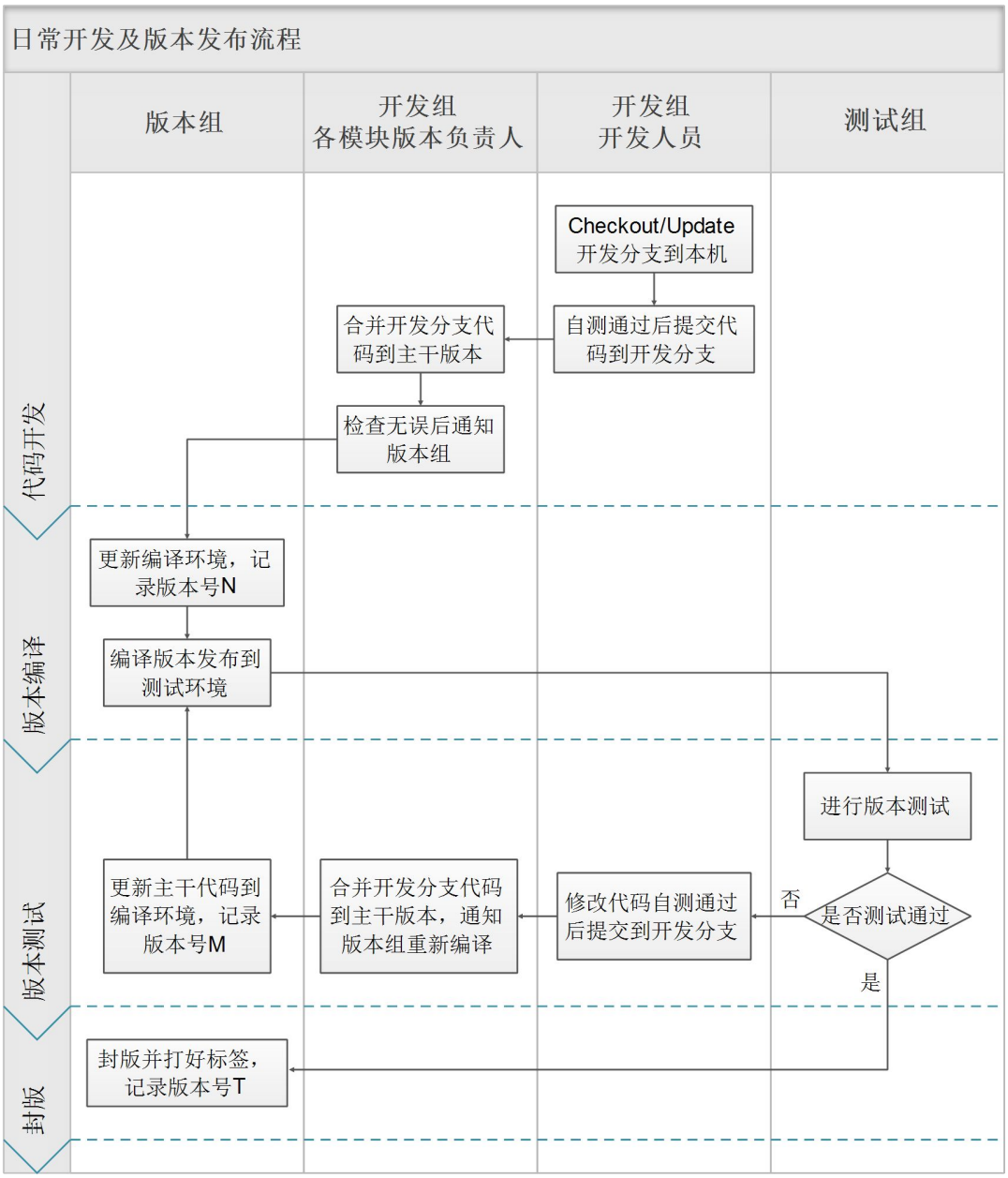
工作副本与目标代码库的分支对应，可通过【Switch】命令切换目标库的分支。

分支合并 merge

可从版本库的某个分支进行【merge】到工作副本的的修改，而不会破坏本地的修改，即自动合并。但当有冲突时，这些修改不会自动合并。

更新工作副本时若无冲突也会自动合并。

1 版本管理日常工作流程



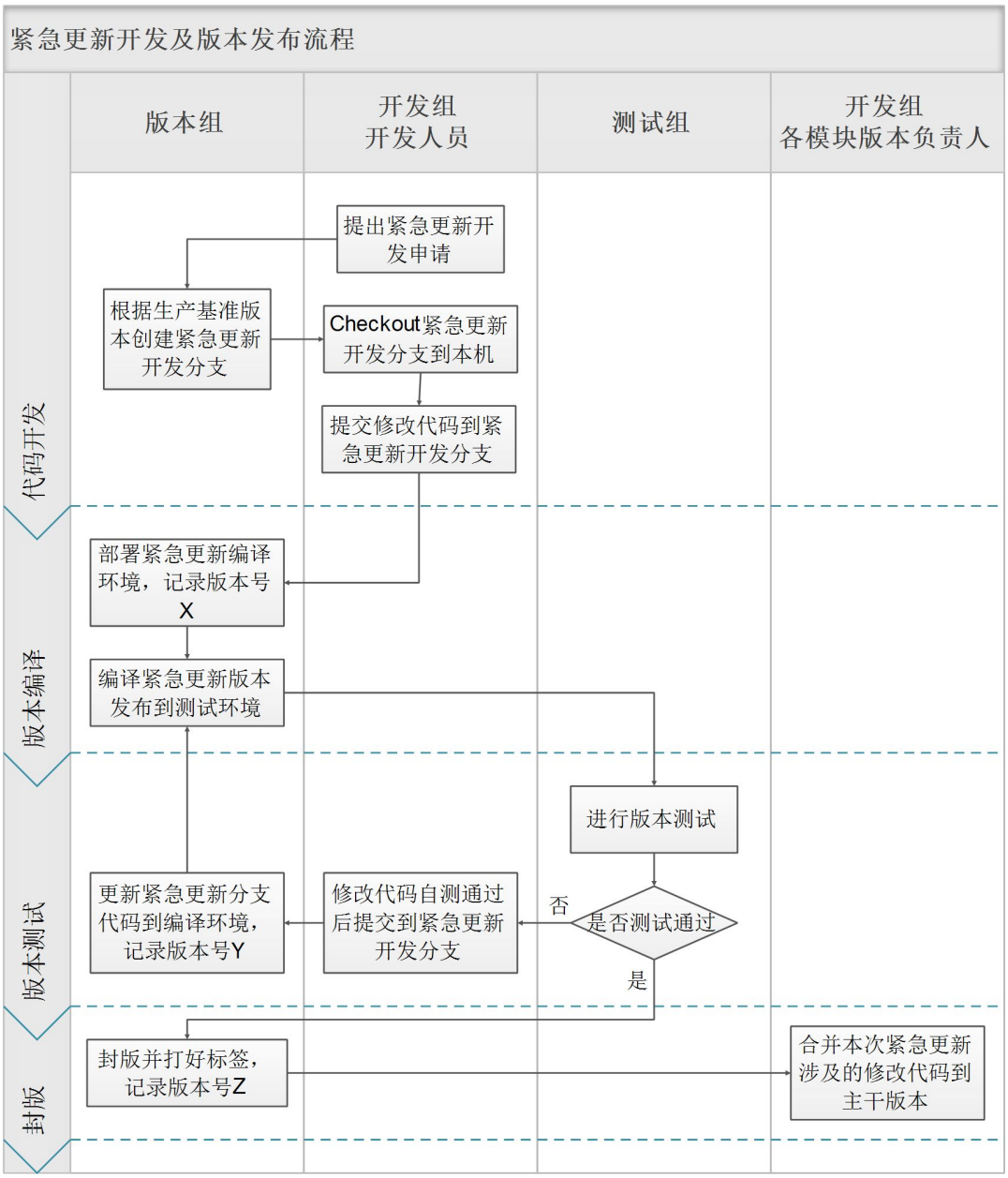
1. 版本组创建"主干版本"（基准版本）。
2. 开发组根据需要基于"主干版本"建立"分支"，一般有两类开发分支：
   - 日常开发分支，如：./branches/dev/
   - 紧急更新开发使用的分支，

     如：./branches/dev_patch/crm_vX.X.X_20131030_p01/
3. 开发人员提交自测通过后的代码到开发分支，各模块版本负责人合并模

块代码到主干，并添加更新说明文件。为了保持主干版本的稳定，只给版本负责人开放主干的写权限。

4. 开发提交截止后，版本组基于主干版本记录版本号 N。
5. 版本组从"主干版本"中提取代码进行编译，发布到测试环境。
6. 测试人员在测试环境进行测试，若测试中发现问题需要修改代码，由开发人员在开发分支进行修改好后，再由合版人员合并到主干版本。
7. 版本组更新主干代码到编译环境，记录版本号 M，编译最新版本更新到测试环境。
8. 测试完成版本组进行封版，为本次测试版本打好标签，并记录全局版本号 T。

2 紧急版本更新工作流程



1. 开发组提出紧急更新申请。
2. 版本组基于紧急更新对应的生产版本创建紧急更新的开发分支，如：./branches/dev_patch /crm_vX.X.X_20131030_p01/。
3. 开发人员基于紧急更新开发分支进行开发，自测通过后提交到紧急更新开发分支。
4. 版本组部署紧急更新编译环境，记录版本号 X，编译成功后发布到测试环

境。

5. 测试组进行测试，若发现问题通知开发修改，重复 3~5 步骤，直至测试通过。
6. 测试完成版本组进行封版，为紧急更新分支打好标签，并记录该补丁版本的全局版本号 Z。
7. 模块合版人员将本次紧急更新所涉及的修改代码合并到 trunk 主干版本，务必保持主干版本代码与发布的版本一致。