# **固件版本发布环境规则**

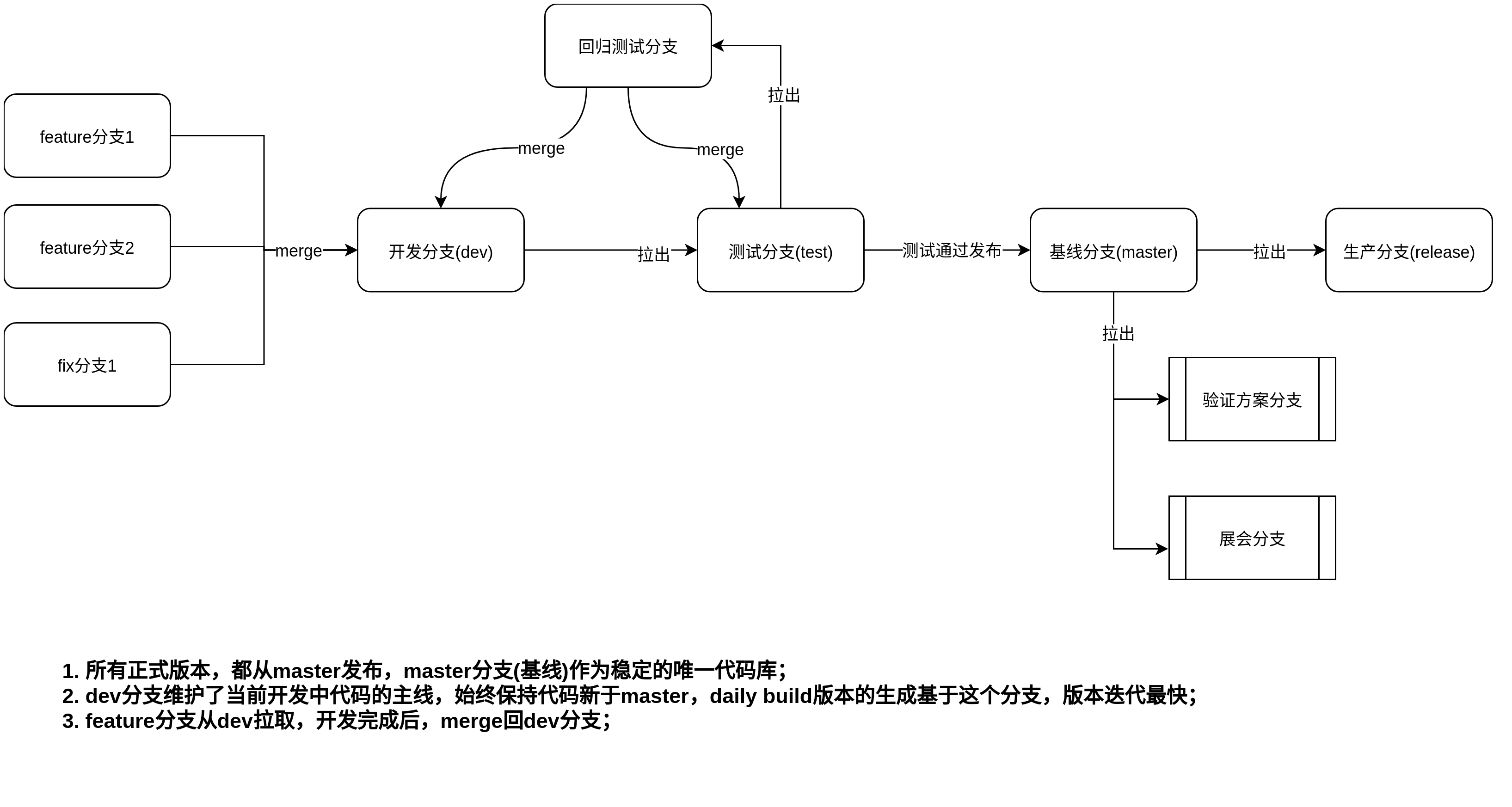
|  |  |  |
| --- | --- | --- |
| 版本编号 | 变更内容 | 备注 |
| V0.1 | 初版框架，确定环境、发布逻辑、发布计划、多规则的优劣式 |  |
| V0.2 | 会议详细确定开发环境 |  |
| V0.3 | 新增基线环境 |  |
| V0.4 | 新增分支命名 |  |
| V0.5 | 新增版本命名 |  |

# **方案**

**背景：**X9软件分支较少，不能及时响应版本测试、展会等场景，相较于APP+云平台在发布流程缺失基础三环境规则，导致软件目前的规则仅有DEV+发布，现象会产生测试阻塞、产品点检阻塞、修改后BUG较多等情况。

**目标**：建立固件版本发布环境的规则，隔离DEV环境的错误影响正常版本，确保发布到TEST++Master+Release环境的版本经过充分的测试

## **初步环境规则**



**开发环境（Dev）：**在开发环境中，开发人员进行新功能的开发、测试和调试

* 可以频繁进行小范围的代码更改和测试，不必严格遵循正式的版本发布流程
* 可能会使用一些临时的配置和工具，以方便开发和调试
* 开发环境中的固件版本通常带有特定的开发标识，如 “DEV” 或开发人员自定义的标记，以便与其他环境区分开来

**测试环境（Test）：**测试环境用于对固件进行全面的测试，确保其稳定性和功能完整性

* 测试环境应该尽可能模拟实际的使用场景，包括硬件配置、网络环境等
* 固件版本在测试环境中应该经过多轮的功能测试、性能测试、兼容性测试等
* 测试环境中的版本可以使用类似 “TEST” 的标识，或者加上测试版本号，如 “V1.0.0\_Test\_1” 等
* 只有通过了严格测试的固件版本才可以进入下一阶段

**基线环境（master）：**提供了一个可靠的代码基础，开发人员可以在这个基础上进行新的开发

* 通常被认为是相对稳定的分支，代表了当前项目的主要版本。
* 只有经过充分测试和验证的代码才能合并到这个分支。
* 可能会定期从这个分支创建发布版本。

注：在基线环境中，可以开辟多个分支如：展会分支、方案验证分支、生产等

**生产环境（Release）：**生产环境是最终用户使用的环境，对稳定性和可靠性要求最高

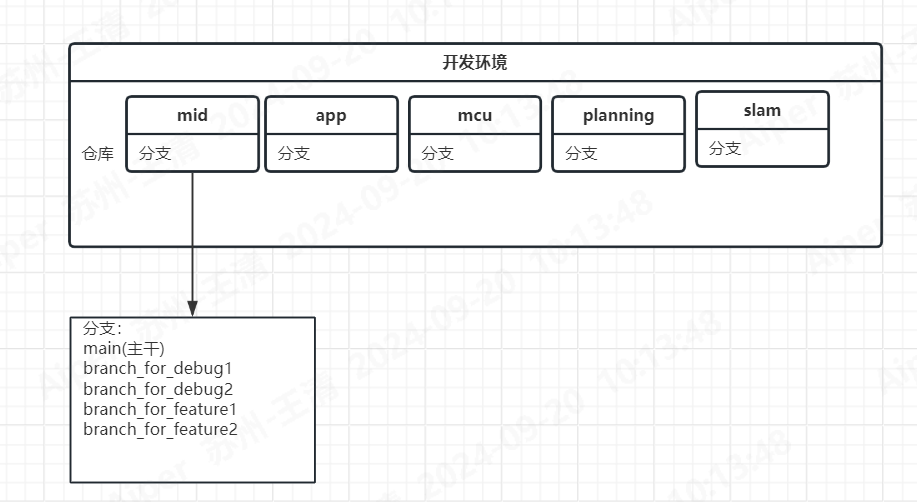
* 只有经过充分测试和验证的固件版本才能发布到生产环境
* 生产环境中的固件版本应该是明确的、稳定的版本号，如 “V1.0.0” 等
* 在生产环境中，应建立严格的版本控制和回滚机制，以应对可能出现的问题。如果出现严重问题，需要能够快速回滚到上一个稳定版本

**环境切换规则**

* 当在开发环境中完成了一定的功能开发和初步测试后，可以将固件版本部署到测试环境进行全面测试。
* 只有在测试环境中通过了所有测试的版本，才能进入生产环境
* 在不同环境切换过程中，确保数据的安全性和完整性，以及配置的正确迁移。同时，要对切换过程进行详细的记录和监控，以便在出现问题时能够快速定位和解决

**各环境之间的关系与区别**

|  |  |  |  |
| --- | --- | --- | --- |
| 环境 | 主要用途 | 配置特点 | 数据特点 |
| 开发环境 | 开发、调试 | 灵活、不稳定 | 测试数据 |
| 测试环境 | 功能测试、性能测试 | 与生产环境相似 | 测试数据或部分生产数据 |
| 基线环境 | 稳定的代码基础 | 与生产环境相似 | - |
| 生产环境 | 用户访问 | 稳定、安全 | 真实生产数据 |



云平台的地址变更是否会影响到固件环境，eg：云平台的开发环境变更到生成环境？

a：不会影响，打包也不影响。

b：固件不关心app，固件只关系云平台，机器有唯一sn，sn绑定到哪个平台（开，试，生）环境下，就往哪传。

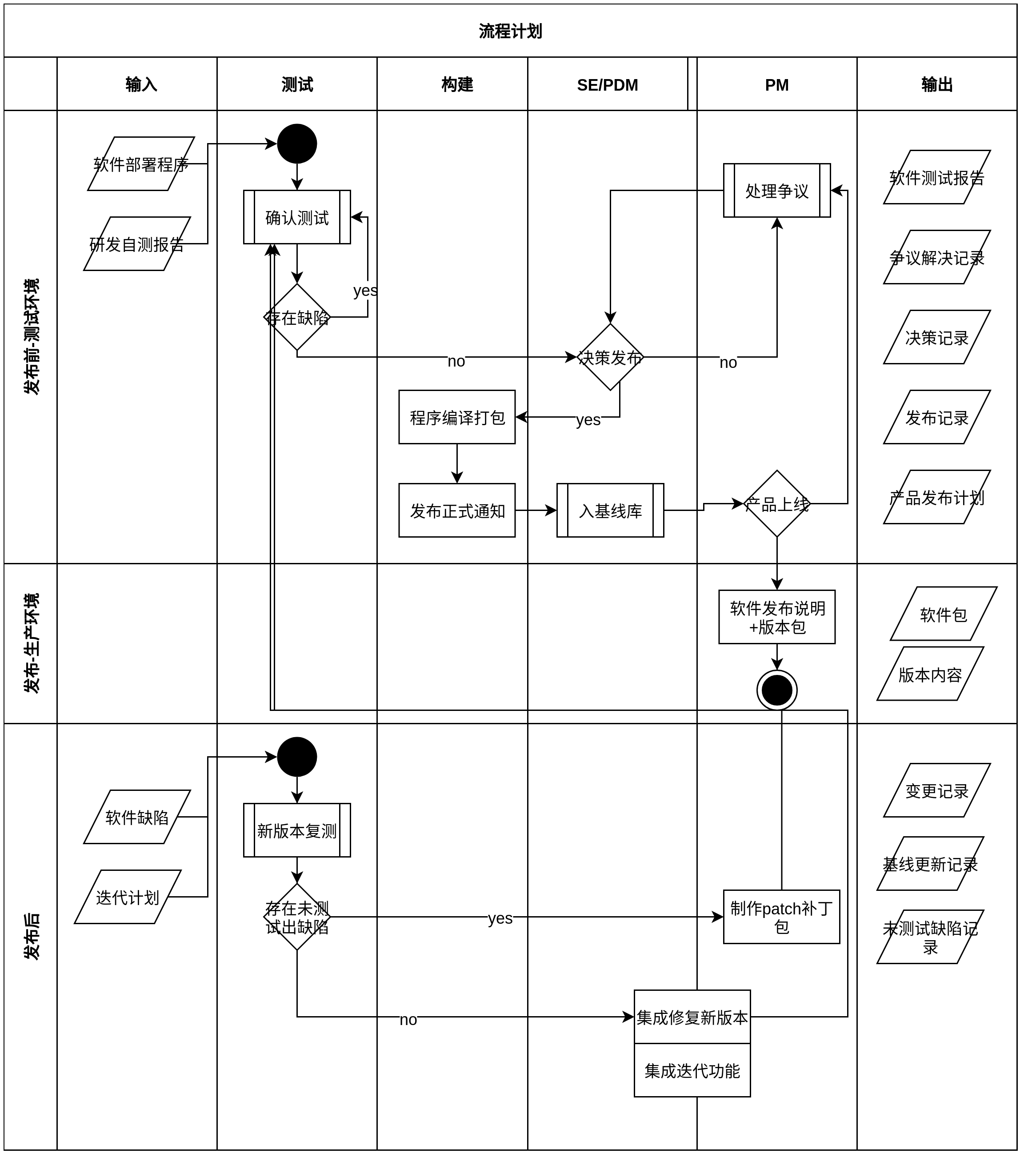
c：建议一个组一个仓库

**补充**

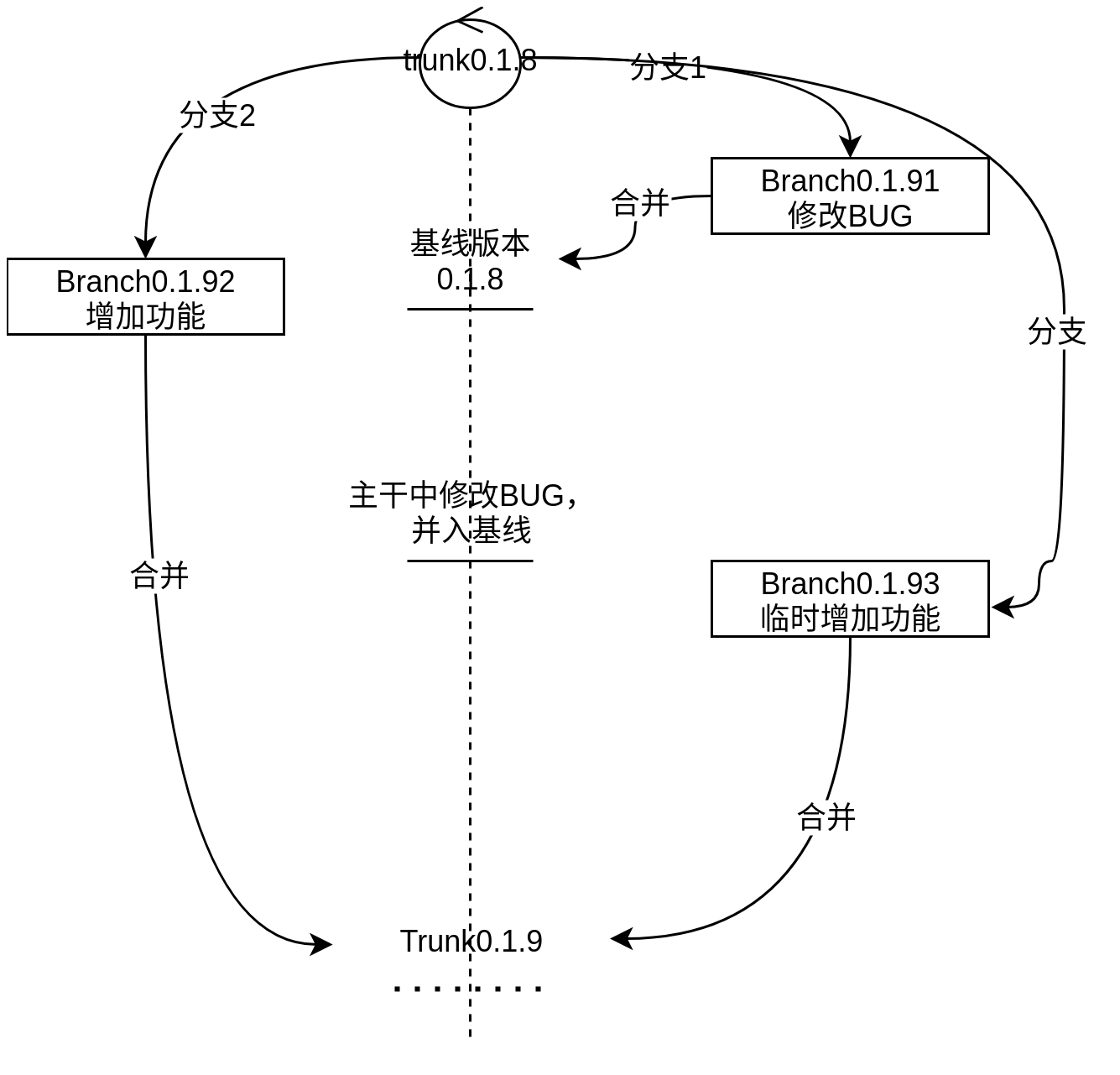
1、【各组自测】开发环境打包前，研发需要组内交叉code review，然后合包自测

2、发版前单模块SE验收，MCU、中间件、SOC、应用负责人王清；算法负责人梁振

**产品发布前准备**



**产品版本发布计划**



|  |  |
| --- | --- |
| 版本名称 | 演化说明 |
| trunk0.1.8 | 第1个未通过测试的基线版本 |
| 基线版本0.1.8 | 第1个通过测试的基线版本 |
| branch0.1.91 | 修改分支，修改禅道中与已知的BUG |
| branch0.1.92 | 基于基线版本新增功能，需要交付功能 |
| branch0.1.93 | 新增临时性功能，紧急加入主干或临时性验证的新功能 |
| trunk0.1.9 | 第2个未通过测试的基线版本 |

# **优劣**

**偏向基础环境**，因为改动小，符合当前软件流程，上游研发的代码管理没有变更，同时也贴合深圳的“开发、测试、生产”三环境规则，便于后期三端系统联调

后期我们可以慢慢向devops模式转变，可以借鉴部分devops的灵活性的

|  |  |  |
| --- | --- | --- |
| 方案 | 优势 | 劣势 |
| 基础环境 | 稳定性和可控性   * 传统的软件发布环境通常经过严格的测试和验证流程，能够确保软件在特定环境下的稳定性。在发布前，会进行多轮的功能测试、性能测试和安全测试等，以减少发布后的风险 * 可以更好地控制发布的时间和节奏，避免因匆忙发布而导致的问题 | 发布周期长   * 传统的软件发布往往需要经过多个阶段的测试和审批，导致发布周期较长。这可能会影响企业对市场变化的响应速度，无法快速推出新功能或修复问题 * 长时间的发布周期也可能导致开发人员的积极性降低，因为他们需要等待很长时间才能看到自己的成果被发布 |
| 明确的职责分工   * 在传统发布环境中，通常有明确的角色和职责分工，如开发人员、测试人员、SE、PM等。每个人都清楚自己的任务和责任，有利于提高工作效率和质量 * 这种明确的分工可以确保每个环节都有专业的人员进行处理，减少错误的发生 | 缺乏灵活性   * 传统发布环境通常比较固定，难以适应快速变化的需求和技术环境。一旦出现新的需求或问题，可能需要重新进行整个发布流程，增加了成本和时间 * 对于一些需要快速迭代的项目，传统发布环境可能无法满足要求 |
| devops | 快速迭代和交付   * DevOps 强调自动化、持续集成和持续部署，能够大大缩短软件的发布周期。开发人员可以更快地将代码集成到主干分支，并通过自动化测试和部署流程将软件发布到生产环境 * 这种快速迭代的能力使企业能够更快地响应市场变化和用户需求，提高竞争力 | 技术门槛高   * 实施 DevOps 需要一定的技术基础和投入，包括自动化工具的使用、容器化技术、持续集成和持续部署平台等。对当前我们来说是个负担，可能存在一定的技术门槛 * 需要团队成员具备新的技能和知识，如自动化测试、容器编排等，这可能需要一定的培训和学习成本 |
| 增强协作和沟通   * DevOps 打破了传统开发、测试和运维之间的壁垒，促进了团队之间的协作和沟通。开发人员、测试人员和运维人员可以共同参与整个软件开发生命周期，提高工作效率和质量 * 通过共享工具和流程，团队成员可以更好地理解彼此的工作，减少误解和冲突 | 团队流程变革挑战   * DevOps 不仅仅是技术的变革，还涉及到管理的转变。需要打破传统的部门壁垒，建立协作、共享和创新的文化。这对于我们来说可能是一个挑战，需要时间和努力来实现 * 团队成员可能需要适应新的工作方式和流程，这可能会引起一些抵触情绪 |
| 提高软件质量   * DevOps 中的自动化测试和持续监控可以帮助及时发现和修复问题，提高软件的质量。自动化测试可以在每次代码提交后进行，确保新功能不会破坏现有功能 * 持续监控可以实时监测生产环境中的软件运行状态，及时发现并解决潜在问题 |  |

# **分支命名规则**

## **命名约定**

* **分支命名：**使用小写字母、短横线（）分隔词， 使用斜杠分隔不同级别的信息，例如 。
* **使用有意义的名称:** 分支名称应该清晰地反映分支的目的。
* **保持简洁:** 避免过长的分支名称。
* **避免特殊字符:** 不要在分支名称中使用空格、逗号等特殊字符。

## **命名规则**

#### **功能分支：**

* **用途:** 开发新的功能。
* **命名:** ，例如 、。
* **注意事项：**feature分支从开发分支创建，开发完成后合并回开发分支。

#### **修复分支：**

* **用途:** 修复bug。
* **命名:**，例如 。
* **注意事项:**分支通常从开发分支创建，开发完成后合并回开发分支。

#### **开发分支：**

* **用途:** 所有的功能分支和修复分支都会合并到这里。
* **命名:** 。

#### **测试分支：**

* **用途:** 所有的功能分支和修复分支都会合并到这里。
* **命名:** 、 或 。

#### **主分支：**

* **用途:** 存放稳定的、可发布的代码。
* **命名:** 。
* **注意事项:** 此分支上不能直接进行开发。

#### **发布分支：**

* **用途:** 即将发布的版本。
* **命名:** ，例如 。
* **注意事项:** 发布分支从主分支创建。

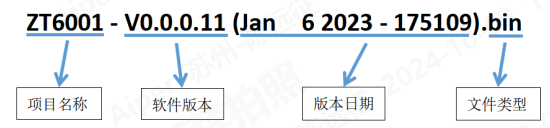
#### **特殊分支：**

* **用途:** 特殊版本，特殊场景，例如：展会、方案验证等。
* **命名:** 、，例如 ，。
* **注意事项:** 特殊分支从主分支创建。

# **版本命名规则**

## **深圳已有命名规则，采纳4位规范**

文件名称由五部分组成：第一部分为项目名称，第二部分为当前软件版本，第三部分为版本  
日期，第四部分为文件后缀，例如：项目发布版本 ZT6001 - V1.0.1 (Jan 6 2023 - 175109).bin



（1）项目名称：项目名称由当前测试项目为准，例如：ZT2001,ZT3001,ZT6001 等。  
（2）软件版本：以软件提测发布版本定义即可，根据本文档软件部分介绍为准。  
（3）版本日期：用于记录修改项目的当前日期，每天对项目的修改都需要更改日期版本号。  
（4）文件类型：针对软件内测和产线蓝牙烧录，例如：ZT6001 - V0.0.0.11 (Jan 16 2023 -  
185307).bin 或 PCBA 贴片厂烧录决定文件类型，例如：ZT6001 - V0.0.0.11 (Jan 16 2023 -  
185307).hex。

软件版本号由四部分组成，第一个 0 为主版本号，第二个 0 为子版本号，第三个  
0 为修正版本号，第四个 11 为内测版本，例如：V0.0.0.11。

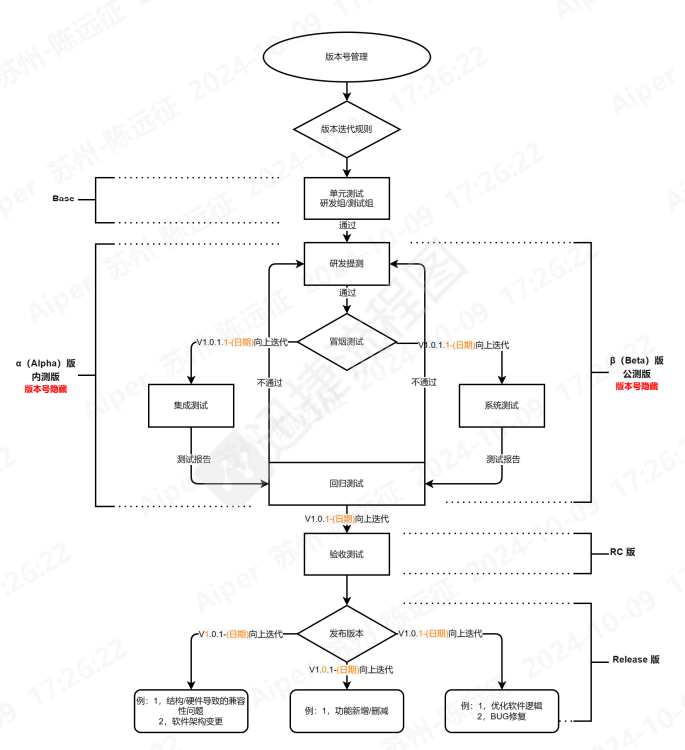


软件初始版本为：V1.0.0.00  
4.2.2 版本号变更规则  
⑴ 主版本号（对外）  
当功能模块有较大的变动，比如结构/硬件导致的兼容性问题或者整体架构发生变化。此版  
本号由研发决定是否修改。  
⑵ 子版本号（对外）  
相对于主版本号而言，子版本号升级对应的是软件功能有一定新增或删减，比如增加脱困功  
能、移除外浸水探针检测功能。此版本号由研发决定是否修改。  
⑶ 修正版本号（对外）  
一般是修复紧急性高风险 bug 或者累积到一定的中低风险 Bug 进行集中修复，发布生产版  
本。此版本号由研发决定是否修改。  
⑷ 内测版本号（对外隐藏）  
此版本号为在软件正式发布前，做的内部迭代测试版本号，对外隐藏，测试人员可输入命令  
in test 进行查看，即单元测试，集成测试，系统测试等阶段迭代。此版本号由研发决定是否  
修改。  
4.2.3 版本号修改举例说明  
比如版本号为：V1.0.0.01 (日期) ，此时为内测阶段，同时隐藏内测版本号；上一级有变动  
时，下级要归零，例：V1.1.1.4，版本号可改为 V1.2.0.0  
⑴ 内测：开发人员修复了测试人员提交的 bug 并经测试人员测试验证关闭 bug 之后，发布  
提测时，此时就进入了软件的下一个阶段，版本号可改为：V1.0.0.2 (日期) ，如当前日期跟  
上一个版本号的日期不一样，版本号可改为：V1.0.0.3 (日期)。  
⑶ 发布：如果对软件进行了一些功能上的软件逻辑优化，严重 BUG 的修复，进行了一些  
局部变动的时候要修改阶段版本号，如：V1.0.1(日期)。  
⑷ 发布：当功能模块有较大变动，功能新增/修改时要修改子版本号，如新增加了功能，则  
版本号要改为： V1.1.0 (日期)。  
⑷ 发布：软件涉及结构/硬件变更导致的兼容性问题，软件架构的变更时需要修改主版本号：  
V1.1.0 (日期）

**软件版本阶段说明**

⑴ Base：  
此版本表示该软件仅仅是一个假页面链接，通常包括所有的功能和页面布局，但是页面中的  
功能都没有做完整的实现，只是做为整体网站的一个基础架构。  
⑵ α（Alpha）版：内测版。  
软件的初级版本，表示该软件在此阶段以实现软件功能为主，通常只在软件开发者 内部交  
流，或者专业测试人员测试用，一般而言，该版本软件的 Bug 较多，需要继续修改，是测  
试版本。测试人员提交 Bug 经开发人员修改确认之后，发布到测试网址让测试人员测试，  
此时可将软件版本标注为 alpha 版。  
⑶ β（Beta）版：公测版。  
该版本相对于α版已有了很大的改进，消除了严重的错误，但还是存在着一些缺陷，需要经  
过多次测试来进一步消除，此版本主要的修改对像是软件的 UI，供专业爱好者大规模测试  
用。  
⑷ RC 版：  
是 Release Candidate 的缩写，意思是发布倒计时，候选版本，该版本已经相当成熟了，完  
成全部功能并清除大部分的 BUG，基本上不存在导致错误的 BUG，与即将发行的正式版相  
差无几。  
(5) Release 版：  
该版本意味“最终版本”，在前面版本的一系列测试版之后，终归会有一个正式版本，是最  
终交付用户使用的一个版本。该版本有时也称为标准版。一般情况下，Release 不会以单词  
形式出现在软件封面上，取而代之的是符号(R)

**软件版本流程管理**



## **自有命名规范，采纳全球化规范**

新增：

Debug版本，包含调试信息，供开发者内部使用，便于排错。

命名示例：在版本号后添加 \_debug，如 1.0.0\_debug，清楚地标识这是一个包含调试信息的版本

test版本，包含修复信息，供测试者内部使用，便于修复与记录

命名示例：在版本号后添加\_master，如1.0.0\_test，清楚的标识这是一个测试版本

master版本，包含基线包，供测试、生产、展会、验证方案适应，便于版本管理，可自由更改配置文件、配置参数

命名示例：在版本号后添加\_master，如1.0.0\_master，清楚的标识这是一个基线版本

Release版本，准备向公众发布的稳定版本，经过彻底测试，无调试代码，性能优化。

命名示例：可能在版本号后加上 \_release，如 1.0.0\_release（预发布），或者直接用 1.0.0 表示正式发布版本

全球化版本规则：可以组合规则①\_②\_③\_④

①使用国家或地区代码

格式示例：软件名\_版本号\_国家代码

举例：RPC\_1.0.0\_US, RPC\_1.0.0\_EU, RPC\_1.0.0\_CN

说明：在版本号后添加ISO标准的国家或地区代码，明确表示该版本适用的法律或市场。

②使用法规遵从性标签

格式示例：软件名\_版本号\_遵从性标签

举例：RPC\_1.0.0\_GDPR, RPC\_1.0.0\_CCPA

说明：指明该版本符合特定法规，如欧盟的GDPR或加州的CCPA，适用于法规对数据处理有具体要求的情况。

③语言和地区组合

格式示例：软件名\_版本号\_语言-地区

举例：RPC\_1.0.0\_en\_US, RPC\_1.0.0\_zh\_CN

说明：结合语言代码和国家代码，用于区分语言和地区特定的版本

④使用功能性标签

格式示例：软件名\_版本号\_功能标签

举例：RPC\_1.0.0\_NoCrypto, RPC\_1.0.0\_Encrypted

说明：针对某些国家或地区对特定技术（如加密技术）有限制的情况，使用功能性标签来区分是否包含这些技术的版本。