ICS 35.020

CCS L77

T/ SIA

中 国 软 件 行 业 协 会 团 体 标 准

T/SIA056-2025

开源操作系统社区可控开源体系建设规范

security controllability requirements of open source operating system community construction

2025-9-4 发布　　2025-9-4 实施

中 国 软 件 行 业 协 会 发 布

目 录

前 言 II

1 范围 1

2 规范性引用文件 1

3 术语和定义 1

3.1 开源软件 1

3.2 开源社区 1

3.3 可控开源 2

3.4 开源许可协议 2

3.5 冗余代码 2

3.6 被动错误检测 2

3.7 主动错误检测 2

3.8 完整性 2

3.9 宽松型许可证 2

3.10 互惠型许可证 3

3.11 中危安全漏洞 3

4 缩略语 3

5 可控开源操作系统社区必要性 3

6 可控开源操作系统社区能力框架 4

7 可控开源操作系统社区规范 5

7.1 来源可控 5

7.2 设计可控 6

7.3 开发可控 7

参考文献 10

前 言

开源操作系统社区中的开源组件由于在软件来源、软件设计、软件开发的过程中缺乏相关的流程标准，使得开源操作系统的可控性难以得到保障。现有的开源规范主要是针对单个软件产品的规范，而开源操作系统社区需解决供应链协同等系统性问题，为此，提出“开源操作系统社区可控开源体系建设规范”，统一开源操作系统社区的可控开源要求，以促进开源操作系统社区高水平发展。

本标准由中国软件行业协会提出并归口。

本标准起草单位：麒麟软件有限公司、国防科技大学、联想开天科技有限公司、海光信息技术股份有限公司、沐曦集成电路(上海)股份有限公司、飞腾信息技术有限公司、北京中科通量科技有限公司、清华大学、国家工业信息安全发展研究中心、先进计算与关键软件海河实验室。

本标准主要起草人：毛周、刘敏、李剑峰、张超、朱晨、康艳红、余杰、马俊、刘晓东、曹先念、和志华、郑臣明、李伟、章津楠、黄勇才、刘勇鹏、胡湘华、吴冬冬、罗鑫、刘知远、辛晓华、赵娆、王文竹、田舒洋、谢炜、刘美燕、卢光明、张然、曾雪征。

本标准的某些内容可能涉及专利，本标准的发布机构不承担识别这些专利的责任。

本标准为首次制定。

本标准按照GB／T 1.1-2020 给出的规则起草。

开源操作系统社区可控开源体系建设规范

1 范围

本文件规定了桌面、服务器、嵌入式与实时、移动与互联网专用、分布式、安全强化型和车控开源操作系统社区建设在组件来源、软件设计、软件开发等方面的可控要求。

本文件适用于指导开源操作系统社区的规范化建设，也可对开发者向开源社区贡献安全的代码以及用户安全使用开源社区软件提供指导。

2 规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是标注日期的引用文件，仅标注日期的版本适用于本文件。凡是不标注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 15272-1994 程序设计语言C

ISO/TS 10303-27:2000 Industrial automation systems and integration — Product data representation and exchange — Part 27: Implementation methods: Java TM programming language binding to the standard data access interface with Internet/Intranet extensions

GB/T 16260.1-2006 软件工程 产品质量

ISO/IEC 25010:2011 软件质量评估模型

ISO/IEC 33004:2015 Information technology - Process assessment - Requirements for process reference, Process assessment and maturity models

ISO/IEC/IEEE 12207:2017 Systems and software engineering - Software life cycleprocesses

可靠性工程师手册

GB/T 44272-2024 开源许可证框架

ISO/IEC 5230:2020 Information technology - OpenChain Specification

H-202104251959 可信开源社区评估规范

T/NIFA 7-2021 金融行业开源软件评测规范

3 术语和定义

下列术语和定义适用于本文件。

3.1

开源软件 open source software

开源软件是指根据开源软件许可协议发布的、软件源代码可公开获得的软件。

3.2

开源社区 open source community

开源社区又称为开放源代码社区，是指开发/管理/推动开源软件项目、发布开源软件源代码及相关文档、分享/交流开源软件相关问题、托管开源软件源代码等功能的网络平台或者团体、组织等。

3.3

可控开源 controlled open source

可控开源是指在开源社区建设中，通过制定和实施一系列规章制度、技术方案，确保开源社区中的代码来源可控、系统设计可控、开发过程可控，确保使用户和开发者能够安全、稳定、可靠、持续地使用开源社区软件及开源软件涉及的支持性服务。

3.4

开源许可协议 open source license

开源软件许可协议是指软件权利人或开源社区/基金会在为了维护开源软件权利人的合法权利，在公开发布软件源代码时，对此软件源代码使用者授予一定的知识产权权能，并对使用者使用该开源软件的义务要求、使用条件等进行了相关规定的法律协议。

3.5

冗余代码 redundancy code

冗余代码，即编程时不必要的代码段。一般来说，一段程序能够执行既定的任务，但是经过优化，能够同样达到目的，执行效率增强，且代码数量减少了，则删除的代码就是程序的冗余代码，其主要分为多余执行冗余和代码数量冗余两部分。

多余执行冗余：如果在某段程序的函数中，出现的语句，在对返回的参数没有任何的影响，但是又执行了多次，是为多余执行，此冗余是对CPU的消耗，应该杜绝该种冗余，应该注释掉。

代码数量冗余：如果在代码中存在过多的注释，或者一些没有使用到的变量，函数而存在程序中，这种冗余会让代码的可读性降低，应该予以杜绝。

3.6

被动错误检测 passive error detection

被动式错误检测是在程序的若干部位设置检测点，等待错误征兆的出现。包括以下两个原则：

a) 相互怀疑原则：设计任何单元/模块时，假设其它单元/模块会出现错误；

b) 立即检测原则：错误征兆出现后尽快查明，限制错误损害并降低排错难度。

3.7

主动错误检测 active error detection

主动式错误检测是主动检查程序状态，例如软件BIT。举例如下：

a) 提供服务器定期监控功能，查看定位CPU高耗线程，提示系统管理员关注；

b) 提供服务端软件心跳监测功能，实时监控服务器的运行状态。

3.8

完整性 integrity

完整性是信息安全的三个基本要点之一，指用户、进程或者硬件组件具有能力，能够验证所发送或传送的东西的准确性，并且进程或硬件组件不会被以任何方式改变。

3.9

宽松型许可证 permissive license

宽松型许可证往往只要求被许可方保留原作品的版权信息，对其使用者施加的限制较少，基于其二次开发的软件可以不公布源代码，如Apache、MIT、BSD系列许可证。

3.10

互惠型许可证 copyleft license

要求对软件的修改和扩展，必须按照获得该软件的许可证进行开源，旨在促进开发人员的合作，保护源代码的自由共享，如GPL系列许可证。互惠型又可以分为强互惠型许可证和弱互惠型许证，前者包括AGPL、SSPL、GPL许可证等；后者包括LGPL系列许可证、MPL许可证等。强互惠型许可证要求对软件的修改和扩展，必须按照获得该软件的许可证进行开源，且不得违背原作品的限制条款。弱互惠型许可证要求对软件的修改、重新分发必须按照原软件的许可证进行开源，但合并这些软件代码的大型项目可以闭源。

3.11

中危安全漏洞 Medium security vulnerability

国标标准（CVSS v4.0）评分范围4.0－6.9分（满分10分），属于中危（Medium）级别。特征：可能导致敏感信息泄露（如用户数据、配置信息）；存在局部权限绕过（如绕过部分访问控制）；无法直接获取系统完全控制权，但可能为后续攻击创造条件。

4 缩略语

下列缩略语适用于本文件。

AGPL 通用公众特许条款（Affero General Public License）
GPL GNU通用公共许可证（GNU General Public License）
LGPL GNU宽通用公共许可证（GNU Lesser General Public License）
LTS 长期支持（Long-term Support）
SIG 特别兴趣小组（Special Interest Group）
API：应用程序编程接口（Application Programming Interface）
MD5: 信息摘要算法5 (MD5 Message-Digest Algorithm)
SHA-1: 安全散列算法1 （Secure Hash Algorithm 1）
Web：全球广域网（World Wide Web）
ABI：应用程序二进制接口（Application Binary Interface）
A11y：无障碍设计（Accessibility）
SSPL：服务器端公共授权（Server Side Public License）
MPL：Mozilla许可协议（Mozilla Public License）
BSD：BSD许可协议（Berkeley Software Distribution License）
MIT：MIT许可协议（Massachusetts Institute of Technology License）
Apache：Apache许可协议（Apache License）
HTTPS：超文本传输安全协议（Hypertext Transfer Protocol Secure）
URL：统一资源定位系统（Uniform Resource Locator）

5 可控开源操作系统社区必要性

开源软件相对闭源软件具有源代码开放、使用者可以二次开发、节约购买成本等优势。因此，近年来以开源软件为基础构建操作系统已成为主流趋势之一。然而，开源软件具有迭代速度过快、系统功能有限、开源协议繁杂、安全机制欠缺等“不可控”问题，导致全球开源风险事件频发，威胁着使用者的信息安全和产业链构建。因此，开源风险已成为全球化挑

战，是开源操作系统社区首要关注的风险点。总体而言，当前的操作系统开源社区建设中主要存在如下三个方面的问题：

a）外部来源不可控。开源操作系统通常是基于数千个开源组件构建而成，大量的开源组件是从上游社区直接引入。开源组件的代码质量、开源协议、安全机制各不相同、水平参差不齐，其来源的可控性是保证开源操作系统可控的基础。因此，需要针对开源组件在来源选型规范、可靠性检测、合规性检测、稳定性检测、安全性检测和可维护性检测等方面明确要求，确保引入的开源组件来源清晰、透明、合规和可靠。

b）系统设计不可控。开源操作系统中部分组件是由开源社区基于上游开源组件二次设计开发或者基于某个开源协议独立设计开发。一般而言，影响力越大的顶级开源社区，此类组件的占比越大，其可控程度在很大程度上决定了该社区的总体水平。为使此类组件在基础安全性、可靠性和鲁棒性上能够可控，需要在其需求分析、软件规划和系统设计等阶段遵守“设计可控”理念。

c）开发过程不可控。基于上游开源组件二次开发或者基于某个开源协议独立开发过程中，存在代码格式不统一、代码质量不合格、编译选项不完善、测试过程不全面、发布途径不可靠等风险，使得这些开源组件水平不可控，进一步影响到开源操作系统的整体稳定性和安全性。因此，制定开源组件开发规范，统一软件开发过程中从编码到编译、测试、发布的全流程规范，是保障开源操作系统高质量可持续发展的重要环节。

综上所述，开源操作系统社区中的开源组件由于在软件来源、软件设计、软件开发过程中缺乏相关的标准规范，使得开源操作系统的可控性难以得到保障。为此，本文提出“开源操作系统社区可控开源体系建设规范”，统一开源操作系统社区的可控开源要求，以促进开源操作系统社区高水平发展。

6 可控开源操作系统社区能力框架

可控开源操作系统社区能力框架围绕来源可控、设计可控、开发可控三个维度来树立可控开源操作系统社区的相关评价指标和衡量标准。具体框架如下图 1 所示,主要包括如下三大方面:

a）来源可控：从软件的源头出发，确保引入的开源组件来源清晰、透明、合规、安全和可靠：

b）设计可控：在开源软件设计阶段对兼容性、可靠性、易用性、安全性等方面的设计进行约束和规范，确保开源软件在安全性、可靠性、兼容性和易用性方面达到一定水平并拥有较高的统一性：

c）开发可控：提供统一的编程规范、开发合规检测、代码质量检测、编译构建检测、测试发布过程检测等机制，减少开源软件在开发过程中的潜在风险，提升开源软件产品质量，保障开源软件的稳定性和安全性。

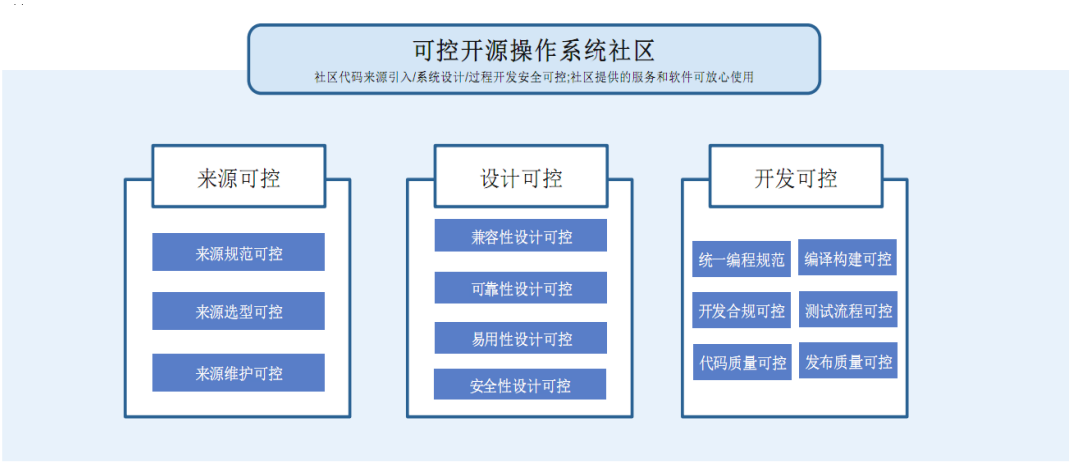

图 1 可控开源操作系统社区能力框架图

7 可控开源操作系统社区规范

7.1 来源可控

组件来源可控是确保开源操作系统社区可控的重要基础。来源可控规范从软件的源头出发，包括了来源选型规范、来源选型可靠性检测、合规性检测、稳定性检测、安全性检测、可维护性检测等方面，确保引入的开源组件来源清晰、透明、合规、安全和可靠。

7.1.1 基本原则

建立开源软件的来源管理规范体系，包括开源软件选型原则、开源软件使用规则、开源软件修改规则、以及开源软件生命周期管理规范等，保证开源软件的来源规范可控。

7.1.2 来源选型可控

7.1.2.1 来源可靠

需保证选型的开源软件来源可靠，应满足以下条件：

a) 选型的开源软件来自上游开源社区正式维护的下载地址；

b) 如果官方已提供文件哈希值（MD5或者SHA1值），选型软件的哈希值应与官方提供软件的哈希值完全一致；

c) 如果官方已提供数字签名，需保证通过数字签名校验工具来判断选型软件未被第三方篡改；

d) 确保获取选型软件的官网提供HTTPS协议。

7.1.2.2 来源合规

需保证选型的开源软件具有明确的开源许可证协议（包括但不限于GPL/LGPL/AGPL/SSPL /Mozilla/BSD/MIT/Apache 等），避免因许可证协议缺失造成相关的知识产权风险；同时，该协议与本开源操作系统社区的开源许可协议可兼容。开源许可证兼容性可根据《开源许可证兼容性指南》，或使用 Scanoss、Black Duck 等 SCA 工具进行判断。

7.1.2.3 来源稳定

需保证选型的开源软件版本是 LTS 版本、稳定版本或最新版本之一，版本的选型优先级策略应满足：LTS 版本>稳定版本>最新版本。有 LTS 版本优先选择最新 LTS 版本；无 LTS 版本，则选稳定版；如果有多个稳定版或无法明显判断稳定版，则选最新版本。

7.1.2.4 来源安全

需保证选型的开源软件来源安全，通过至少一款如 360 安全卫视、金山毒霸、奇安信等国产主流防病毒软件扫描，并保证防病毒软件不产生中等以上告警（不会导致系统完全崩溃或服务中断，但可能引起性能下降、或可能造成少量非敏感个人信息数据泄露，但不会涉及核心机密或大规模用户数据）。此外，需通过安全漏洞检测，保证选型的开源软件不存在已知的中危以上安全漏洞。

7.1.3 来源维护可控

7.1.3.1 来源升级策略

引入开源软件时，原则上其上游社区在 2 年内有持续的功能升级和安全更新发布。如无法满足此条件，需提前规划开源软件的升级策略，以避免开源软件因版本原因产生的无法正常使用风险。

7.1.3.2 来源漏洞修复策略

需持续跟踪开源软件的漏洞分析和报告，针对不同等级的漏洞进行分级漏洞修复，避免产生安全隐患。

高危漏洞（CVSS≥9.0）：社区发布补丁后 7 天内完成修复并推送更新，需同步提供临时缓解措施（如配置调整）。

中危漏洞（4.0≤CVSS<9.0）：15 天内修复，修复前需公示风险影响范围。

低危漏洞（CVSS<4.0）：纳入下一版本迭代修复，周期不超过 30 天。

7.1.3.3 来源漏洞应急响应机制

需建立安全应急响应中心，收到漏洞报告应在 24 小时内启动漏洞评估和分级响应。针对高危漏洞 48 小时内发布安全通告，提供热修复补丁或回滚方案。

7.2 设计可控

设计可控是指在开源软件设计阶段对兼容性、可靠性、易用性、安全性等方面的设计进行约束和规范，确保开源软件在安全性、可靠性、兼容性和易用性方面达到一定水平并拥有较高的统一性。

设计可控是指在开源软件设计阶段对兼容性、可靠性、易用性、安全性等方面的设计进行约束和规范，确保开源软件在安全性、可靠性、兼容性和易用性方面达到一定水平并拥有较高的统一性。

7.2.1 兼容性设计可控

7.2.1.1 共存性设计

建立共存性设计规范，要求开源软件能与系统平台、子系统和其他开源模块等兼容，同时针对国际化和本地化进行规范处理。

7.2.1.2 互操作性设计

建立互操作性设计规范，要求开源软件之间 API、ABI、文件格式和配置等能够有效对接。

7.2.1.3 许可证设计

建立许可证设计规范，开源软件项目的 License 应该与其依赖项目的 License 相兼容。

7.2.2 可靠性设计可控

7.2.2.1 避错设计

建立避错设计规范，要求在开源软件设计之初就能尽量避免故障的引入。

7.2.2.2 查错设计

建立查错设计规范，要求开源软件能通过主动错误检测和被动错误检测等手段，自动查找在运行中可能发生的错误。

7.2.2.3 容错设计

建立容错设计规范，要求开源软件在发生部分故障的情况下，在报告异常之后仍然能够正常工作。

7.2.3 易用性设计可控

7.2.3.1 易操作设计

建立易操作性设计规范，帮助用户方便地操作和控制开源软件各项功能。

7.2.3.2 易访问设计

建立易访问性设计规范，要求开源软件在设计时尽量考虑使用者的使用障碍，如年龄障碍、能力障碍等。

7.2.3.3 易理解设计

建立易理解设计规范，要求开源软件能让用户能理解其是否合适以及如何能用于特定的任务和使用条件。

7.2.4 安全性设计可控

7.2.4.1 系统安全设计

建立系统安全设计规范。系统安全设计规范主要在内核、功能库、基础服务运行环境和应用软件等操作系统各层次，从身份认证、访问控制、安全配置、系统漏洞和病毒防御等系统安全维度，同时结合硬件特性和应用场景开展安全体系设计，确保操作系统整体具有较高的安全等级。

7.2.4.2 应用安全设计

建立应用安全设计规范。应用安全设计规范内容包括但不限于：认证和鉴权、权限管理、敏感数据保护、业务安全、web 安全、数据库安全、容器安全等。

7.2.4.3 网络安全设计

建立网络安全设计规范。网络安全设计规范内容包括但不限于：设备安全、边界安全、远程访问安全等。

7.2.4.4 管理安全设计

建立管理安全设计规范。管理安全设计规范内容包括但不限于：用户管理安全、运维管理安全、操作维护安全等。

7.3 开发可控

提供统一的编程规范、代码质量检测、编译构建检测、测试发布过程检测等机制，减少开源软件在开发过程中的潜在风险，提升开源软件产品质量，保障开源软件的稳定性和安全性。

7.3.1 统一编程规范

7.3.1.1 通用编程规范

为开源软件的开发语言建立明确的通用编程规范，如针对 C/C++、JAVA、HTML5、Python 等语言的通用编程规范。通用编程规范主要包括变量、函数等命名规范、排版格式、注释规范、变量和类型规范等。

7.3.1.2 C/C++安全编程规范

制定开源软件 C/C++安全编程规范。C/C++安全编程规范包含变量、断言、函数（数组/字数串等特殊函数、安全函数使用、不安全函数使用等）、循环、类、安全退出、字符串、数组操作、整数、内存、文件、输入输出、敏感信息处理等安全规范。

7.3.1.3 JAVA 安全编程规范

制定开源软件 JAVA 安全编程规范。JAVA 安全编程规范包含数据校验、方法与表达式、数值与运算、线程同步、异常行为、I/O 操作、序列化和反序列化、平台安全等安全规范。

7.3.1.4 HTML5 安全编程规范

制定开源软件 HTML5 安全编程规范。HTML5 安全编程规范包含口令、认证、URL 输入校验、存储、权限控制、函数、变量等安全规范。

7.3.1.5 Python 安全编程规范

制定开源软件 Python 安全编程规范。Python 安全编程规范包含数据校验、异常行为、I/O 操作、运行环境等安全规范。

7.3.1.6 PHP 安全编程规范

制定开源软件 PHP 安全编程规范。PHP 安全编程规范包含 SQL 注入、XSS 跨站脚本攻击、CSRF 跨站请求伪造、漏洞、Cookie 管理、GC 垃圾收集等安全规范。

7.3.2 开发合规可控

7.3.2.1 开源许可证风险管理机制

开源软件在开发过程中具有完善的开源许可证风险管理机制，针对强互惠型、互惠型、弱互惠型、宽松型许可证的要求进行代码许可证申明和代码衍生分发，降低由于缺乏开源许可证风险管理机制导致的知识产权风险。

7.3.2.2 开源代码片段引用管理机制

开源软件在开发过程中具有完善的开源代码片段引用管理机制，降低由于缺乏开源代码片段引用管理机制导致的知识产权风险，具体包括：

a）不鼓励采用源代码片段引用的方式，尽量迁移成包管理器引用。

b）严格防止变更引入源代码片段Copyright（版权）声明或License（许可证）声明的情况。

c）严格防止所引入源代码片段的License和被引入开源项目的License不兼容的情况。

7.3.3 代码质量可控

7.3.3.1 代码审核机制

开源软件在开发过程具有完善的代码审核机制，针对代码审核流程、代码审核工具、参与代码审核的人员、代码审核标准等有明确规范，从而对合入的代码进行质量把控。

7.3.3.2 代码静态扫描

开源软件在开发过程中采用安全软件工具对代码编程规范、代码安全漏洞、代码合规性进行静态安全扫描，针对扫描结果，有完整的解读报告和相应的指导措施，确保代码不含已知安全漏洞。

7.3.3.3 代码冗余检查

开源软件在开发过程中采用工具对代码进行冗余度扫描，确保项目中的冗余代码比率不得过高，并有明确的原因记录。

7.3.3.4 代码清晰度检查

对开源软件代码的清晰度进行检查，发布的开源代码中应有合适的、易于理解的注释，且注释的总体比例不得过低，各主要编程语言注释比率建议如下：

JAVA：代码注释比率不低于 20%。

Python：代码注释比率不低于 30%。

C/C++：代码注释比率不低于 25%。

其他语言：代码注释比率不低于 15%。

7.3.4 编译构建可控

7.3.4.1 安全编译选项

开源软件在构建时开启安全编译选项，并针对编译过程中的警告及时处理。如果有特殊原因不能消除，则进行规避处理，并保留记录。

7.3.4.2 安全编译工具

开源软件在编译过程中，把编译构建工具的全架构、安全特性等安全因素加入到编译工具优选标准。

7.3.4.3 编译构建可重现

开源软件基于相同的源码、相同的环境，实现同一版本可重复构建，且用 Diffoscope 等工具分析除编译时间戳变化、寄存器分配顺序调整、调试路径偏移、地址随机化（ASLR）导致偏移的无害差异外，构建结果一致。

7.3.5 测试流程可控

7.3.5.1 测试规范统一

建立开源软件的测试规范。测试规范是针对开源软件的设计规范而输出的包含测试条件、测试计划、测试范围等方面的规范。

7.3.5.2 测试需求明确

在开源软件的需求确认阶段进行测试需求分析。测试需求分析主要是对开源软件的原始需求、开发设计文档、历史问题等进行分析，确定测试的内容及验收标准。

7.3.5.3 测试用例完备

开源软件测试用例包括：

a）对需求测试设计并输出测试用例。根据需求及开源软件原有设计等进行用例设计。

b）对软件使用设计并输出测试用例。根据开源软件的基础使用进行分析设计，如安装、升级等使用场景。

c）对历史问题测试设计并输出测试用例。结合开源软件历史问题，设计并完善用例。

d）测试用例进行评审且有明确记录。

7.3.5.4 测试过程可靠

开源软件测试执行过程包括：

a）测试用例100%执行且有明确测试结果。

b）测试过程中缺陷有记录、分析和跟踪。

c）测试提交缺陷有审核确认。

d）测试环境可重复构建，并且测试结果一致。

e）测试方案进行评审且有明确记录。

7.3.5.5 测试范围全面

开源软件测试过程遵守：

a）核心模块（如内核、安全协议）需求测试覆盖率≥95，非核心模块（如工具链、辅助功能）需求测试覆盖率≥85。

开源软件测试分析遵守：

a) 测试完成后对所有的需求进行测试分析，并对每条需求给出评估。

b) 测试完成后对整个版本质量情况进行评估。

c) 测试完成后发布测试报告。测试报告中包括用例执行情况、缺陷提交情况、遗留情况、测试结果评估及结论等。

d) 测试报告发布前经过评审。

7.3.6 发布质量可控

7.3.6.1 完整性检测

开源软件在版本发布时将软件包状态信息（例如：软件包清单、哈希/数字签名、测试状态，测试时间，变更时间等）与测试环节的软件包信息进行校验，保证对外发布的软件、补丁和镜像的唯一完整标识。

7.3.6.2 漏洞检测

开源软件发布前采用软件工具对二进制文件进行漏洞扫描，并给出详细的漏洞分析报告并进行修复。

7.3.6.3 补丁审核

开源软件对外发布的版本/补丁，通过版本/补丁发布流程评审、签发，并满足相关的质量要求。

7.3.6.4 角色分权

开源软件版本发布过程做到职责分离，发布申请和发布审核由不同的人员操作。

7.3.6.5 版本可追溯

开源软件发布流程平台化，基于平台系统管理版本发布的端到端流程，使版本发布历史可视化、可追溯。

参考文献

[1]Debian.Debian Developers' Manuals.
https://www.debian.org/doc/devel-manuals,2019-10-31.

[2]Debian.Debian Policy Manual.
https://www.debian.org/doc/debian-policy/,2019-10-03.

[3]《开源许可证兼容性指南》
https://max.book118.com/html/2021/0304/7021036010003063.shtm，2020-12

[4]ubuntu Developers' Manuals.
https://wiki.ubuntu.com/UbuntuDevelopment，2023-2-9