技　　术　　标　　准

研发运营一体化能力成熟度模型

第 3 部分：持续交付过程

The DevOps capability maturity model

Part 3: Continuous delivery process

（征求意见稿）

2017 年 11 月 15 日

插入目录

前　言

研发运营一体化是指在 IT 软件及相关服务的研发及交付过程中，将应用的需求、开发、测试、部署和运营统一起来，基于整个组织的协作和应用架构的优化，实现敏捷开发、持续交付和应用运营的无缝集成。帮助企业提升 IT 效能，在保证稳定的同时，快速交付高质量的软件及服务，灵活应对快速变化的业务需求和市场环境。

本标准是“研发运营一体化能力成熟度模型”系列标准的第 3 部分，该系列标准的结构和名称如下：

- 第 1 部分：总体架构
- 第 2 部分：敏捷开发管理过程
- 第 3 部分：持续交付过程
- 第 4 部分：技术运营过程
- 第 5 部分：应用架构
- 第 6 部分：组织结构

本标准按照 GB/T 1.1-2009 给出的规则起草。

本标准由中国通信标准化协会提出并归口。

本标准起草单位： DevOps 时代社区、高效运维社区、中国信息通信研究院、深圳优维科技有限公司、中兴通信有限公司

本标准主要起草人：石雪峰、张乐、景韵、王津银、鞠炜刚、萧田国、栗蔚

研发运营一体化

总体架构及能力成熟度模型

1 范围

本标准规定了研发运营一体化的持续交付过程及能力成熟度模型。本标准中的研发运营一体化包括IT软件及服务的需求、开发、测试、部署和运营五个环节，并实现敏捷开发、持续交付和技术运营的顺序闭环集成。

本标准适用于企业在实施IT软件开发和服务过程中实现研发运营一体化架构，提升IT效能。

2 规范性引用文件

下列文件中的条款通过本部分的引用而成为本部分的条款。凡是注日期的引用文件，仅所注日期的版本适用于本文件。凡是不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

[1] GB/T 32400-2015 信息技术 云计算 概览与词汇

[2] GB/T 32399-2015 信息技术 云计算 参考架构

[3] YD/T2441-2013 互联网数据中心技术及分级分类标准

[4] GB/T 33136-2016 信息技术服务数据中心服务能力成熟度模型

3 术语

下列术语和定义适用于本文件。

3.1 AB测试 ab test

为Web或App界面或流程制作两个（A/B）或多个（A/B/n）版本，在同一时间维度，分别让组成成分相同（相似）的访客群组随机的访问这些版本，收集各群组的用户体验数据和业务数据，最后分析评估出最好版本正式采用。

3.2 制品 artifact

即构建过程的输出物，包括软件包，测试报告，应用配置文件等。

3.3 代码复杂度 code complexity

主要度量指标为圈复杂度，即代码中线性独立路径的数量。

3.4 部署流水线 deployment pipeline

指软件从版本控制库到用户手中这一过程的自动化表现形式。

4 缩略语

下列缩略语适用于本文件。

CI Continuous Integration 持续集成

CD Continuous Delivery 持续交付

MVP Most Variable Product 最小可行产品

DEEP Principle Detailed Appropriately，Estimated，Emergent，Prioritized principle 适当细化的，有估算的，随时产生的，有优先级的原则

UI User Interface 用户界面

UAT User Acceptance Testing 用户验收测试

OS Operation System 操作系统

5 综述

持续交付是指以可持续的方式将各类变更（包括新功能、缺陷修复、配置变化、实验等）安全、快速、高质量地落实到生产环境或用户手中的能力。

持续交付的分级技术要求包括：配置管理、构建与持续集成、测试管理、部署与发布管理、环境管理、数据管理、度量与反馈等，如图1所示。

持续交付						
配置管理	构建与持续集成	测试管理	部署与发布管理	环境管理	数据管理	度量与反馈
版本控制	构建实践	测试分级策略	部署与发布模式	环境供给方式	测试数据管理	度量指标
版本可追踪性	持续集成	代码质量管理	持续部署流水线	环境一致性	数据变更管理	度量驱动改进
		测试自动化				

图 3 持续交付分级技术要求

6 配置管理

配置管理是指一个过程，通过该过程，所有与项目相关的产物，以及它们之间的关系都被唯一定义、修改、存储和检索，保证了软件版本交付生命周期过程中所有交付产物的完整性，一致性和可追溯性。

配置管理是持续交付的基础，是保障持续交付所有活动顺畅有效开展的前提。良好设计的配置管理策略，可以提高组织协作的效率，改善产品价值交付的完整流程。

配置管理可以分为版本控制和版本可追溯性两个维度表述。

6.1 版本控制

版本控制是指通过记录软件开发过程中的源代码、配置、工具、环境、数据等的历史信息，快速重现和访问任意一个修订版本。

版本控制是团队协作交付软件的基础，应支持团队间所有变更历史的详细信息查询及共享，包括修改人员、修改时间、文件内容以及注释信息等，通过有效信息共享，加快问题定位和沟通协作效率。

级别	版本控制系统	分支管理	构建产物管理	单一可信数据源
1	未使用统一的版本控制系统，源代码分散在研发本地设备管理	缺乏明确的分支管理策略，分支生命周期混乱	未使用统一的制品库，构建产物通过直接拷贝或本地共享等方式进行分发	无
2	使用集中式的版本控制系统并将所有源代码纳入系统管理	采取长周期和大批量的方式进行代码提交，代码合并过程存在大量冲突和错误	使用统一的制品库管理构建产物，有清晰的分级和目录结构及权限管控并通过单一制品库地址进行分发	无
3	使用分布式的版本控制系统，并将所有源代码、配置文件、构建和部署等自动化脚本纳入系统管理	采取短分支频繁提交的方式,研发人员至少每天完成一次代码提交,代码合并过程顺畅	使用统一的制品库管理构建产物，并将二进制库文件和三方依赖软件工具等纳入制品库管理	版本控制系统和制品库作为单一可信数据源，覆盖生产部署环节
4	将数据库变更脚本和环境配置等纳入版本控制系统管理版本控制系统支持自动化的变更操作	分支策略满足持续交付需求,可灵活适应产品交付	对制品库完成分级管理，有成熟的备份恢复清理策略，如采用使用分布式制品库	单一可信数据源进一步覆盖研发本地环境
5	将软件生命周期的所有配置项纳入版本控制系统管理，可完整回溯软件交付过程满足审计要求	持续优化的分支管理策略,可支持团队高效协作	同上	单一可信数据源贯穿整研发价值流交付过程在组织内部开放共享，建立知识积累和经验复用体系

6.2 版本可追溯性

版本可追溯性是指软件系统中的每一次变更都可以追溯变更的详细信息,并向上追溯变更的原始需求、流转过程等所有关联信息。

可追溯性也是版本回滚的历史依据和实施基础,建立良好的版本可追溯性可实现对任一版本完整环境流程的自动化，精确回滚，快速重现问题和恢复正常环境。

级别	变更过程	变更追溯	变更回滚
1	变更过程不受控且变更信息分散在每个系统内部，缺	变更缺乏基本的可追溯性	变更问题定位困难且回滚操作具有高风险

	乏信息的有效共享机制		
2	代码变更过程应附带变更管理信息	有清晰定义的软件版本号规则，实现版本和代码的关联，可追溯版本构建对应的完整源代码信息	可支持版本间差异对比和代码级别问题定位和回滚
3	所有配置项变更由变更管理系统触发，并作为版本控制系统的强制要求	实现版本控制系统和变更管理系统的自动化关联，信息双向同步和实时可追溯	实现变更管理系统和版本控制系统的同步回滚，保证状态的一致性
4	使用同一套变更管理系统覆盖从需求到部署发布全流程	变更依赖被识别和标记，实现数据库和环境变更信息的可追溯	可根据变更管理系统按需快速导出复用软件代码变更集，如建立从变更管理系统到软件代码变更集的关系数据库
5	可视化变更生命周期，支持全程数据分析管理和满足审计要求	实现从需求到部署发布各个环节的相关全部信息的全程可追溯	支持任何时间点全部状态的自动化回滚需求

7 构建与持续集成

构建是将软件源代码通过构建工具转换为可执行程序的过程，一般包含编译和链接两个步骤，将高级语言代码转换为可执行的机器代码并进行相应的优化，提升运行效率。

持续集成是软件构建过程中的一个最佳实践，在版本控制的基础上，通过频繁的代码提交，自动化构建和自动化测试，加快软件集成周期和问题反馈速度，从而及时验证系统可用性。

7.1 构建实践

构建实践关注软件代码到可运行程序之间的过程，通过规则、资源和工具的有效结合，提升构建质量和构建速度，使构建成为一个轻量级，可靠可重复的过程。同时构建产物被明确标识管理，采用清晰的规则定义版本号和目录结构有助于团队成员可以随时获取到可用版本，以及版本相关的信息，快速验证回溯版本变更。

级别	构建方式	构建环境	构建计划	构建职责
1	采用手工方式进行构建，构建过程不可重复	使用本地设备，构建环境不可靠	没有明确的版本号规则和构建任务计划	构建工具和环境受限于团队人员能力，频繁手动干预维护
2	实现脚本自动化，通过手工配置完成构建	有独立的构建服务器，多种任务共享构建环境	明确定义版本号规则，并根据发布策略细分构建类型，实现每日自动构建	构建工具和环境由专人负责维护，并使用权限隔离
3	定义结构化构建脚本，实现模块级共享复用和统一维护	构建环境配置实现标准化，有独立的构建集群，单次构建控制在小时级	明确定义构建计划和规则，实现代码提交触发构建和定期自动执行构建	构建工具和环境由专门团队维护，并细分团队人员职责

4	实现构建服务化，可按需提供接口和用户界面用于可视化构建编排	优化构建速度，实现增量化构建和模块化构建，单次构建控制在分钟级，如可采用分布式构建集群、构建缓存等技术	分级构建计划，实现按需构建并达到资源和速度的有效平衡	构建系统服务化提供更多用户使用，构建不再局限于专业团队进行
5	持续优化的构建服务平台，持续改进服务易用性	持续改进构建性能，实现构建资源共享和动态按需分配回收，如搭建基于云服务虚拟化和容器化的分布式构建集群	同上	将构建能力赋予全部团队成员，并按需触发构建实现快速反馈

7.2 持续集成

持续集成是软件工程领域中的一种最佳实践，即鼓励研发人员频繁的向主干分支提交代码，频率为至少每天一次。每次提交都触发完整的编译构建和自动化测试流程，缩短反馈周期，出现问题第一时间进行修复，从而保证软件代码质量，减少大规模代码合并的冲突和问题，让软件随时处于可发布状态。

级别	集成服务	集成频率	集成方式	反馈周期
1	没有搭建持续集成服务，团队成员缺乏对持续集成的理解	长期本地开发代码集成频率几周或者几月一次	代码集成作为软件交付流程中的一个独立阶段	每次集成伴随大量的问题和冲突，集成期间主干分支长期不可用
2	搭建统一的持续集成服务并对系统进行日常维护和管理	采用团队定期统一集成的策略，代码集成频率几天或者几周一次	在部分分支上进行每天多次的定时构建	集成问题反馈和解决需要半天或者更长时间
3	组建专门的持续集成团队，负责优化持续集成系统和服务	研发人员至少每天向代码主干集成一次	每次代码提交触发自动化构建，构建问题通过自动分析精准推送相关人员处理	集成问题反馈和解决可以在几个小时内完成
4	持续集成嵌入每个研发团队日常活动，实现持续集成系统服务化和自助化	研发人员每天多次向代码主干集成，每次集成代价较低	每次代码提交构建触发自动化测试和静态代码检查，测试问题自动上报变更管理系统，测试结果作为版本质量标准要求，如：采取质量门禁等方式强化主干代码质量	集成问题反馈和解决控制在 30 分钟以内完成
5	持续优化和改进团队持续集成服务，实现组织交付能力提升	任何变更（代码，配置，环境）都会触发完整的持续集成流程	实现持续集成分级和自动化测试分级，满足不同模块和集成阶段的差异化需求	集成问题反馈和解决控制在 10 分钟以内完成

8 测试管理

测试管理是指一个过程，通过该过程，所有与测试相关的过程、方法被定义。在产品投入生产性运行之前，验证产品的需求，尽可能地发现并排除软件中的缺陷，从而提高软件的质量。

测试管理又可以分为测试分层策略、代码质量管理、自动化测试等多个维度表述。

8.1 测试分层策略

测试分层策略是建立一种分层的测试体系，把测试作为一个整体来规划和执行，融入到持续交付的各个阶段中，达到质量防护的目的。

级别	分层方法	分层策略	测试时机
1	只进行用户/业务级的UI测试	尚未建立测试分层策略，测试不分层	测试在软件交付过程中在开发完成后才介入
2	采用接口/服务级测试对模块/服务进行覆盖全面的接口测试；采用代码级测试对核心模块的函数或类方法进行单元测试；对系统进行基本的性能测试	测试开始分层，但对测试分层策略缺乏系统的规划，对用户/业务级测试、接口/服务级、代码级测试分布比例由高到低，各层测试缺乏有效的设计	测试在持续交付过程中的介入时间提前到开发的集成阶段，接口/服务级测试在模块的接口开发完成后进行
3	采用代码级测试对模块的函数或类方法进行覆盖全面的单元测试；系统全面的进行性能、容量、稳定性、可靠性、易用性、兼容性、安全性等非功能性测试	对测试分层策略进行系统的规划，用户/业务级、接口/服务级、代码级测试分布比例由低到高，充分设计；对非功能性测试进行全面系统的设计	测试在持续交付过程中的介入时间提前到开发的编码阶段，代码级测试在模块的函数或类方法开发完成后进行
4	采用测试驱动开发的方式进行代码级、接口级测试；采用探索性测试方法对需求进行深入挖掘测试	测试分层策略的各层测试具有交叉互补性	代码级测试在模块的函数或类方法开发过程中同步进行和完成；接口/服务级测试在模块的接口开发过程中同步进行和完成
5	采用验收测试驱动开发的方式进行用户/业务级的UI测试	定期验证测试分层策略，是否完整、有效，持续优化策略	在需求阶段进行用户/业务级测试设计，在需求特性开发、交付整个过程中同步进行并完成测试

8.2 代码质量管理

代码质量管理是在软件研发过程中保证代码质量的一种机制，当代码变更后，可以对代码质量进行检查、分析,给出结论和改进建议，对代码质量数据进行管理，并可以对代码质量进行追溯。

级别	质量规约	检查策略	检查方式	反馈处理
1	代码质量检查无任何规约	代码质量检查无针对检查范围、质量门限等相关的策略	代码质量检查采用人工方式进行评审	对代码质量检查结果处理不及时，遗留大量技术债
2	代码质量检查具备基本规约，但还缺乏完整性和有效性	代码质量检查有针对检查范围、质量门限的策略，对代码规范、错误和圈复杂度、重复度等质量指标进行检查分析	代码质量检查采用自动化结合手工方式进行	对代码质量检查结果给出反馈，根据反馈进行处理，对遗留的部分技术债缺乏跟踪和管理，导致遗漏
3	代码质量检查具备完整、有效和强制执行的规约	代码质量检查将安全漏洞检查、合规检查纳入到检查范围	代码质量检查完全自动化，不需要手工干预	根据代码质量检查结果反馈及时处理，技术债仍有短期遗留，但进行有效的跟踪、管理和处理
4	代码质量检查规约根据需要可进行扩展和定制	代码质量检查针对检查范围、质量门限的策略可根据需要灵活调整	对代码质量检查发现的部分问题自动提出修改建议，支持可视化	将检查结果强制作为版本质量标准要求，根据代码质量检查提出的修改建议，对问题及时处理，在研发阶段主动解决技术债
5	定期验证代码质量规约的完整性和有效性，持续优化	定期验证代码质量策略的完整性和有效性，持续优化	具备企业级的代码质量管理平台，以服务的形式提供对代码质量的检查、分析	对代码质量数据进行统一管理，可有效追溯并对代码质量进行有效度量

8.3 自动化测试

自动化测试是把以人为驱动的测试行为转化为机器执行的一种过程，在预设条件下运行系统或应用程序，执行测试并评估测试结果，以达到节省人力、时间或硬件资源，提高测试效率和准确性。

级别	自动化设计	自动化开发	自动化执行	自动化分析
1	未采用自动化方式测试，纯手工测试	尚未对自动化测试脚本进行开发和管理，手工测试	手工测试执行效率低下，以周级为单位	手工对测试结果进行分析判断，错误高，可信度低
2	尚未对测试用例中自动化部分进行规划和设计，覆盖不完整	对自动化测试脚本进行开发和本地管理	对用户/业务级测试采用自动化测试，自动化测试的执行效率不高，以天级为单位	对自动化测试结果具备一定的自动判断能力，存在一定的误报，可信度不足

3	根据需求、接口和代码对不同测试分层中自动化测试用例进行规划和设计，自动化覆盖比较完整	自动化测试脚本开发采用数据驱动、关键字驱动等方法；使用版本控制系统对自动化测试脚本进行有效管理	从代码级、接口级到UI级测试实现了端到端的自动化测试打通；自动化测试执行效率较高，代码级测试分钟级,UI级测试小时级	对自动化测试结果具备较强的自动判断能力，误报少，可信度高
4	对性能、稳定性、可靠性、安全性等非功能性测试中自动化用例进行规划和设计，自动化覆盖完整	自动化测试用例脚本间具备独立性和大批量执行的健壮性	有组织级的统一自动化测试平台，和上下游需求、故障系统打通；可以根据需求针对性自动关联选择自动化测试用例脚本执行；可以将由于版本原因导致的失败用例和故障关联	自动化测试数据模型标准化，和上下游需求、故障等研发数据关联，可以对自动化测试效果进行度量分析。例如：需求测试覆盖率、测试通过率和测试效率等。
5	对故障和测试进行复盘，对遗漏的测试用例进行补充，不断优化和完善，持续提升覆盖率	自动化脚本是测试用例设计的活文档，自动化脚本开发和测试用例设计完全统一	采用企业级统一的自动化测试平台，以云化的方式提供测试服务，进行分布式测试调度执行，提高测试执行效率和资源利用率；定期验证自动化执行策略,持续优化	对自动化测试结果可以智能分析，自动分析失败用例的失败类型及原因，可以自动向故障管理系统提交故障，可信度高

9 部署与发布管理

部署与发布泛指软件生命周期中，将软件应用系统对用户可见，并提供服务的一系列活动，包括系统配置，发布，安装等。整个部署发布过程复杂，涉及多个团队之间的协作和交付，需要良好的计划和演练保证部署发布的正确性。

其中部署偏向技术实践，即将软件代码，应用，配置和数据库变更应用到测试环境、准生产环境和生产环境的过程。发布偏向于业务实践，指将部署完成的应用软件功能和服务正式对用户可见，提供线上服务的过程。部署和发布的有机结合，实现了软件价值向最终用户的交付。

9.1 部署与发布模式

部署和发布模式关注交付过程中的具体实践，将部署活动自动化并前移到研发阶段，通过频繁的演练和实践部署活动，成为研发日常工作的一部分，从而减少最终部署的困难和不确定性，可靠可重复的完成部署发布任务。部署发布模式通过合理规划，分层实施，一方面减少软件最终上线交付风险，同时可及时获取用户信息反馈，帮助持续改善整个软件交付过程和软件功能定义。

级别	部署方式	部署活动	部署策略	部署质量

1	运维人员手工完成所有环境的部署	部署过程复杂不可控，伴随大量问题和较长的停机时间	采用定期大批量部署策略	部署整体失败率较高，并且无法实现回滚，生产问题只能在线上修复，修复时间不可控
2	运维人员通过自动化脚本实现部署过程部分自动化	部署过程通过流程文档定义实现标准化整体可控	应用作为部署的最小单位，应用和数据库部署实现分离，实现测试环境的自动化部署	实现应用部署的回滚操作，部署失败率中等，问题可及时修复
3	部署和发布实现全自动化，同时支持数据库自动化部署	使用相同的过程和工具完成所有环境部署，一次部署过程中使用相同的构建产物	可运行的环境作为部署的最小单位，应用和配置进行分离	部署活动集成自动化测试功能，并以测试结果为部署前置条件 每次部署活动提供变更对象范围报告和测试报告
4	部署发布服务化，实现交付团队自助一键式多环境自动化	部署过程可灵活响应业务需求变化，通过合理组合高效编排	通过多种部署发布策略保证流程风险可控，如：蓝绿部署，金丝雀发布	建立监控体系跟踪和分析部署过程，出现问题自动化降级回滚，失败率较低
5	持续优化的部署发布模式和工具系统平台	持续部署，每次变更都触发一次自动化生产环境部署过程	软件交付团队自主进行安全可靠的部署和发布活动	持续优化的部署监控体系和测试体系，部署失败率维持在极低水平

9.2 持续部署流水线

持续部署流水线是DevOps的核心实践，通过可靠可重复的流水线，打通端到端价值流交付，实现交付过程中各个环节活动的自动化和可视化。部署流水线通过将复杂的软件交付流程细分为多个阶段，每个阶段层层递进，提升软件交付质量信心，并且在流水线过程中提供快速反馈，减少后端环节浪费。

可视化流水线可以增强跨组织的协同效率，提供有效的信息共享平台，从而统一组织目标，并且不断识别流水线中的约束点和瓶颈，以及潜在的自动化及协作场景，通过持续改进而不断提升软件交付效率。

级别	协作模式	流水线过程	过程可视化
1	整个软件交付过程严格遵循预先计划，存在复杂的部门间协作和等待，只有在开发完成后才进行测试和部署	软件交付过程中的大部分工作通过手工方式完成	交付过程中的信息是封闭的，交付状态不可追溯
2	通过定义完整的软件交付过程和清晰的交付规范，保证团队之间交付的有序	软件交付过程中的各个环节建立自动化能力以提升处理效率	交付过程在团队内部可见，信息在团队间共享，交付状态可追溯
3	团队间交付按照约定由系统间调用完成，仅在必要环节进行手工确认	打通软件交付过程中的各个环节，建立全流程的自动化能力，并根据自动化测试结果控制软件交付质量	交付过程组织内部可见，团队共享度量指标

4	团队间依赖解耦，可实现独立安全的自主部署交付	建立可视化部署流水线，覆盖整个软件交付过程，每次变更都会触发完整的自动化部署流水线	部署流水线全员可见，对过程信息进行有效聚合分析展示趋势
5	持续优化的交付业务组织灵活响应业务变化改善发布效率	持续部署流水线驱动持续改进	部署流水线过程信息进行数据价值挖掘，推动业务改进

10 环境管理

环境作为DevOps持续敏捷交付过程中最终的承载，环境的生命周期管理、一致性管理、环境的版本管理都变得非常重要。环境管理是用最小的代价来达到确保一致性的终极目标。

级别	环境类型	环境构建	环境依赖与配置管理
1	环境类型只有生产环境和非生产环境的划分	环境的构建通过人工创建完成	无依赖管理，环境的管理就是一个 OS 的交付
2	IT 交付过程意识到部分测试环境的重要性，开始提供功能测试环境。	环境构建通过一键化的脚本或者虚拟机来完成的，构建过程完全黑盒化完成。	以应用为中心有 OS 级别的依赖和配置管理能力，比如说操作系统版本、组件版本、程序包版本等等
3	持续交付过程意识到研发环境的重要性，开始提供面向各类开发者独立的研发工作区。	环境的构建通过资源交付平台来完成，并且底层是由云来交付	以应用为中心，有服务级依赖的配置管理能力，比如说依赖的关联服务，Mysql服务、cache服务、关联应用服务等等
4	全面的测试与灰度环境对于质量交付过程来说非常重要，有各类的环境类型划分，区分了开发者，技术测试及业务测试环境以及灰度发布环境等等	环境的构建可以通过 Docker 容器化快速交付，低成本构建一个新的环境	环境和依赖配置管理可以资源化描述，类似 dockerfile，大大提升其配置管理能力
5	根据业务与应用的需要，弹性分配各类环境	环境的构建结合底层 IT 资源状况，采用了各类混合 IT 技术，根据业务及应用架构弹性构建	环境依赖和配置可以做到实例级的动态配置管理能力，根据业务和应用架构的变化而变化

11 数据管理

系统开发过程中为了满足不同环境的测试需求，以及保证生产数据的安全，需要人为准备数量庞大的测试数据，需保证数据的有效性以适应不同的应用程序版本。另外应用程序在运行过程中会产生大量数据，这些数据天生有状态，同应用程序本身的生命周期不同，作为应用最有价值的内容需要妥善保存，并随应用程序的升级和回滚进行迁移。

11.1 测试数据管理

测试数据需要满足多种测试类型的需求（手工测试，自动化测试），覆盖正常状态，错误状态和边际状态，测试数据需同时满足测试效率和数据量的要求。测试数据的输入需要受控，并运行在受控环境中，保证输出的有效性，同时由于持久数据的必要行，要避免数据被未授权的篡改，以影响测试结果的客观一致性。为了模拟类生产环境系统运行情况，常采用仿生产环境数据，此类测试数据在使用时需要注意数据安全，避免敏感用户数据泄露，及时进行数据清洗和漂白。

级别	数据来源	数据覆盖	数据独立性	数据安全
1	每次测试时手工创建数据，测试数据都是临时性的	测试数据覆盖率低，仅支持部分测试场景，无法有效支持测试工作	测试数据没有版本控制和备份恢复机制	测试数据来源复杂，混入核心生产数据，带来信息安全风险
2	从生产环境导出一个子集并进行清洗后，形成基准的测试数据集，满足部分测试用例执行要求	测试数据覆盖主要场景，包括正常类型，错误类型以及边界类型，并进行初步的分类分级，满足不同测试类型需要	测试数据有明确备份恢复机制，实现测试数据复用和保证测试一致性	测试数据经过清洗，不包含敏感信息，有效避免信息安全风险
3	同上	建立体系化测试数据，进行数据依赖管理，覆盖更加复杂的业务场景	每个测试用例拥有专属的测试数据，有明确的测试初始状态 测试用例的执行不依赖其他测试用例执行所产生的数据	同上
4	每个测试用例专属的测试数据都可以通过模拟或调用应用程序 API 的方式自动生成	测试数据覆盖安全漏洞和开源合规等需求场景并建立定期更新机制	通过测试数据分级，实现专属测试数据和通用测试数据的有效管理和灵活组合，保证测试数据的独立性	同上
5	所有的功能、非功能测试的测试数据，都可以通过模拟、数据库转储或调用应用程序 API 的方式自动生成	持续优化的持续数据管理方式和策略	同上	同上

11.2 数据变更管理

数据变更管理主要关注应用程序升级和回滚过程中的数据库结构和数据的变更，良好的变更管理策略可以保证应用版本和数据库版本兼容匹配，以应对应用的快速扩容缩容等线上场景。通过应用变更和数据变更的解耦，减少系统变更的相互依赖，实施灵活的升级部署。

级别	变更过程	兼容回滚	版本控制	数据监控
1	数据变更由专业人员在后台手工完成数据变更作为软件发布的一个独立环节，单独实施和交付	没有识别数据库和应用版本，存在不兼容风险	数据变更没有纳入版本控制，变更过程不可重复	没有建立变更监控体系，变更结果不可见
2	数据变更通过文档实现标准化，使用自动化脚本完成变更	建立数据库和应用的版本对应关系，并跟踪变更有效性	数据变更脚本纳入版本控制，并与数据库版本进行关联	对变更日志进行收集分析，帮助问题快速定位
3	数据变更作为持续部署流水线的一个环节，随应用的部署自动化完成，无需专业人员单独执行	每次数据变更同时提供明确的恢复回滚机制，并进行变更测试，如：提供升级和回滚两个自动化脚本	同上	对数据变更进行流程分级定义，应对不同环境下的高危操作
4	应用程序部署和数据库迁移解耦，可单独执行	数据变更具备向下兼容性，支持保留数据的回滚操作和零停机部署	同上	对数据变更进行监控，自动发现异常变更状态
5	持续优化的数据管理方法，持续改进数据管理效率	同上	同上	监控数据库性能并持续优化

12 度量与反馈

DevOps基于精益思想发展而来，其中持续改进是精益思想的核心理念之一。DevOps主张在持续交付的每一个环节建立有效的度量和反馈机制，其中通过设立清晰可量化的度量指标，有助于衡量改进效果和实际产出，并不断迭代后续改进方向。另外设立及时有效的反馈机制，可以加快信息传递速率，有助于在初期发现问题，解决问题，并及时修正目标，减少后续返工带来的成本浪费。度量和反馈可以保证整个团队内部信息获取的及时性和一致性，避免信息不同步导致的问题，明确业务价值交付目标和状态，推进端到端价值的快速有效流动。

12.1 度量指标

度量指标的拣选和设定是度量和反馈的前提和基础，科学合理的设定度量指标有助于改进目标的达成。在拣选度量指标时需要关注两个方面，即度量指标的合理性和度量指标的有效性，合理性方面依托于对当前业务价值流的分析，从过程指标和结果指标两个维度来识别DevOps实施结果，以及整个软件交

付过程的改进方向；有效性方面一般遵循SMART原则，即指标必须是具体的、可衡量的、可达到的、同其他目标相关的和有明确的截止时间，通过这五大原则可以保证目标的科学有效。

级别	度量指标定义	度量指标类型	度量数据管理	度量指标更新
1	度量指标没有明确定义，对度量价值的理解是模糊的	无	度量数据是临时性的，没有收集管理	无
2	在持续交付各个阶段定义度量指标，度量指标局限于职能部门内部	度量指标以结果指标为主，如变更频率，需求交付前置时间，变更失败率和平均修复时间，	度量数据的收集是离散的不连续的，历史度量数据没有进行有效管理	度量指标的设立和更新是固化的，度量指标没有明确的优先级
3	建立跨组织度量指标，进行跨领域综合维度的度量	度量指标覆盖过程指标，客观反映组织研发现状	度量数据的收集是连续的，历史度量数据有明确的管理规则	度量指标的设立和更新是动态的，可以按照组织需求定期变更，度量指标的优先级在团队内部可以达成一致
4	整个研发团队共享业务价值导向的度量指标，实现指标的抽象分级，关注核心业务指标	度量指标覆盖探索性指标，关注展示趋势和识别潜在改进	度量数据的收集是连续且优化的，对历史数据数据进行有效的挖掘分析	建立完整的度量体系和成熟的度量框架，度量指标的设立和更新可按需实现快速定义并纳入度量体系，推动流程的持续改进
5	持续优化的度量指标，团队自我驱动持续改进	支持改进目标和试验结果的有效反馈，用于经验积累和指导下一阶段的改进工作	同上	度量指标可基于大数据分析和人工智能自动识别推荐，并且动态调整指标优先级

表1：部分参考过程度量指标

阶段	度量指标
需求	需求总数 各个状态需求数量 需求完成数量 需求平均时长
版本控制	代码仓库数量 代码提交数 代码提交频率 代码提交时间分布
构建	构建次数 构建频率 构建时长 构建失败率 构建修复时间

	构建类型
代码	代码行数 代码复杂度 代码重复率 单元测试覆盖率 单元测试用例数 单元测试成功率
环境	环境变更时长 变更频率 容器镜像更新 活跃容器数量 资源使用统计
部署	部署版本数量 部署时间 部署成功率 部署回滚率

12.2 度量驱动改进

度量驱动改进关注软件交付过程中各种度量数据数据的收集，统计，分析和反馈，通过可视化的度量数据客观反映整个研发过程的状态，以全局视角分析系统约束点，并在团队内部共享，帮助设立客观有效的改进目标，并调动团队资源进行优化.同时对行之有效的改进项目进行总结分享，帮助更大范围组织受益于改进项目的效果，打造学习型组织和信息共享，不断驱动持续改进和价值交付。

级别	报告生成方式	报告有效性	报告覆盖范围	反馈改进
1	度量报告通过手工方式生成，没有标准化的格式定义，内容缺乏细节	数据时效性无法保证	受众局限于报告生成人员及相关的小范围内部	报告发现的问题没有进行有效跟踪落实，问题长期无法改进
2	度量报告以自动化方式生成，通过预定义格式和内容标准化度量报告	数据体现报告生成时间点的最新状态	由预先定义的事件触发自动化报告发送，受众覆盖团队内部成员	测试报告中反馈的问题录入问题追踪系统，进行持续跟踪
3	度量报告进行分类分级，建立多种度量反馈渠道，内容按需生成	通过可视化看板实时展示数据	实现报告精准范围推送，支持主动订阅，受众覆盖跨部门团队	度量反馈问题纳入研发迭代的待办事项，作为持续改进的一部分
4	建立跨组织级统一的数据度量平台，数据看板内容可定制	通过可视化看板聚合报告内容，自动生成趋势图，进行趋势分析	多维度产品状态实时信息展示	度量反馈的持续改进纳入研发日常工作，预留时间处理非功能性需求和技术债务，并且识别有效改进并扩展到整个组织，作为

				企业级知识体系积累保留
5	持续优化的度量方法，平台和展现形式	同上	同上	通过数据挖掘实现跨组织跨流程数据度量分析，分析结果作为业务决策的重要依据，帮助组织持续改进价值交付流程