ICS 35.020
CCS

YD

中华人民共和国通信行业标准

YD/T 1755T—2018
[代替 YD/T]

研发运营一体化（DevOps）能力成熟度模型 第3部分：持续交付

The capability maturity model of DevOps Part 3: Continuous Delivery

（报批稿）

（本稿完成日期：2018.12.18）

[××××]-[××]-[××]发布　　[××××]-[××]-[××]实施

中华人民共和国工业和信息化部　发布

目　　次

前　　言 IV
1 范围 1
2 规范性引用文件 1
3 术语和定义 1
3.1 配置项 configuration item 1
3.2 制品 artifact 1
3.3 代码复杂度 code complexity 1
3.4 部署流水线 deployment pipeline 1
4 缩略语 1
5 持续交付 2
5.1 配置管理 2
5.1.1 版本控制 2
5.1.1.1 版本控制系统 2
5.1.1.2 分支管理 2
5.1.1.3 制品管理 3
5.1.1.4 单一可信数据源 3
5.1.2 变更管理 4
5.1.2.1 变更过程 4
5.1.2.2 变更追溯 4
5.1.2.3 变更回滚 4
5.2 构建与持续集成 5
5.2.1 构建实践 5
5.2.1.1 构建方式 5
5.2.1.2 构建环境 5
5.2.1.3 构建计划 5
5.2.1.4 构建职责 5
5.2.2 持续集成 6
5.2.2.1 集成服务 6
5.2.2.2 集成频率 6
5.2.2.3 集成方式 6
5.2.2.4 反馈周期 6
5.3 测试管理 7
5.3.1 测试分层策略 7
5.3.1.1 分层方法 7
5.3.1.2 分层策略 7
5.3.1.3 测试时机 7
5.3.2 代码质量管理 8

5.3.2.1 质量规约 8
5.3.2.2 检查方式 8
5.3.2.3 反馈处理 8
5.3.3 自动化测试 9
5.3.3.1 自动化设计 9
5.3.3.2 自动化开发 10
5.3.3.3 自动化执行 10
5.3.3.4 自动化分析 10
5.4 部署与发布管理 11
5.4.1 部署与发布模式 11
5.4.1.1 部署方式 11
5.4.1.2 部署过程 11
5.4.1.3 部署策略 11
5.4.1.4 部署质量 11
5.4.2 部署流水线 12
5.4.2.1 协作模式 12
5.4.2.2 流水线过程 12
5.4.2.3 过程可视化 12
5.5 环境管理 13
5.5.1 环境类型 13
5.5.2 环境构建 13
5.5.3 环境依赖与配置管理 13
5.6 数据管理 14
5.6.1 测试数据管理 14
5.6.1.1 数据来源 14
5.6.1.2 数据覆盖 14
5.6.1.3 数据独立性 14
5.6.2 数据变更管理 15
5.6.2.1 变更过程 15
5.6.2.2 兼容回滚 15
5.6.2.3 数据监控 15
5.7 度量与反馈 16
5.7.1 度量指标 16
5.7.1.1 度量指标定义 16
5.7.1.2 度量指标类型 16
5.7.1.3 度量数据管理 16
5.7.1.4 度量指标更新 16
5.7.2 度量驱动改进 18
5.7.2.1 内容和生成方式 18
5.7.2.2 数据时效性 18
5.7.2.3 覆盖范围 18
5.7.2.4 反馈改进 18

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（DevOps）能力成熟度模型”系列标准的第 3 部分：持续交付，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

第3部分：持续交付

第4部分：技术运营

第5部分：应用设计

第6部分：安全及风险管理

第7部分：评估方法

第8部分：系统和工具技术要求

本标准/本部分按照GB/T 1.1—2009给出的规则起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本标准/本部分由中国通信标准化协会提出并归口。

本标准/本部分起草单位：中国信息通信研究院、北京华佑科技有限公司、北京京东尚科信息技术有限公司、北京百度网讯科技有限公司、中国移动通信集团有限公司、中兴通讯股份有限公司、阿里巴巴（中国）有限公司、深圳市腾讯计算机系统有限公司、华为技术有限公司、中国联合网络通信集团有限公司。

本标准/本部分主要起草人：石雪峰、景韵、栗蔚、萧田国、雷涛、顾宇、牛晓玲、陈镔、张新、鞠炜刚、陈喻、黎嘉豪、毛茂德、李海传、徐奇琛、段新、孙辰星、程颖。

研发运营一体化（DevOps）能力成熟度模型
第3部分：持续交付

1 范围

本标准规定了研发运营一体化（DevOps）能力成熟度模型下持续交付过程的能力成熟度要求和评价方法。

本标准适用于具备IT软件研发交付运营能力的组织实施IT软件开发和服务过程的能力进行评价和指导；可供其他相关行业或组织进行参考；也可作为第三方权威评估机构衡量软件开发交付成熟的标准依据。

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 32399-2016 信息技术 云计算 参考架构

[3] YD/2441-2013 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 术语和定义

下列术语和定义适用于本标准。

3.1 配置项 configuration item

即纳入配置管理范畴的工作成果，是保存系统和项目的相关配置。

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
UI	User Interface	用户界面
API	Application Programming Interface	应用程序编程接口
SMART	Specific Measurable Attainable Relevant Time	具体的、可度量度、可实现的、相关性和时效性原则

5 持续交付

持续交付是指持续的将各类变更（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，如表1所示。

表1 持续交付分级技术要求

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	测试分层策略	部署与发布模式	环境管理	测试数据管理	度量指标
变更管理	持续集成	代码质量管理	部署流水线		数据变更管理	度量驱动改进
		自动化测试				

5.1 配置管理

配置管理是指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、存储和检索的过程，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，是保障持续交付正确性的前提，良好设计的配置管理策略，可提高组织协作的效率，改善产品价值交付流程。配置管理可以分为版本控制和变更管理两个维度表述。

5.1.1 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，加快问题定位速度和沟通协作效率，主要包括版本控制系统、分支管理、制品管理和单一可信数据源，如表 2 所示。

5.1.1.1 版本控制系统

版本控制系统是指通过记录一个或若干文件内容变化,能够查阅特定版本修订情况的系统。

5.1.1.2 分支管理

分支管理是对软件研发过程中的分支和集成策略的管理，分支策略代表了研发协作方式。

5.1.1.3 制品管理

制品管理是对软件研发过程中生成的产物的管理，一般作为最终交付物完成发布和交付。

5.1.1.4 单一可信数据源

单一可信数据源是一种信息数据模型和关联模式，保证每个数据元素只存储一份，确保数据的一致性。

表2 版本控制

级别	版本控制系统	分支管理	制品管理	单一可信数据源
1	源代码分散在研发本地自行管理	无	构建产物分散在研发本地自行管理	无
2	1) 使用统一的版本控制系统 2) 将全部源代码纳入版本控制系统管理	多条分支长期并行存在且分支合并到主干的周期长	1) 使用统一的制品库管理构建产物 2) 有清晰的存储结构 3) 有唯一的版本号 4) 通过统一的制品库地址进行构建产物分发	开发测试部署环节所用到的源代码来源于统一版本控制系统
3	1) 将配置文件、构建和部署等自动化脚本纳入版本控制系统管理 2) 有健全的版本控制系统管理机制，包括： 代码库命名规范 备份与可用性保障机制 权限模型 专人专岗管理	短周期分支 分支频繁地向主干合并	1) 将依赖组件纳入制品库管理 2) 将所有交付制品纳入制品库管理，比如：测试报告 3) 制品库读写有清晰的权限管控制度	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	1) 将数据库变更脚本和环境配置等纳入版本控制系统管理 2) 版本控制系统相关操作以自动化的方式实现，而非手工操作 3) 有针对版本控制系统的度量与监控机制	1) 分支策略满足持续交付需求，可灵活适应产品交付 2) 主干随时可进行指定版本的测试和发布 3) 特性代码可按需合并到主干进行验证和发布	对制品库完成分级管理 已建立体系化的制品库管理策略，包括：备份与恢复机制、策略管理，制品库完整性与一致性保障机制	单一可信数据源进一步覆盖研发本地环境
5	1) 将软件生命周	1) 持续优化的分支	持续优化的制品管理机	1) 单一可信数据源贯

	期的所有配置项纳入版本控制系统管理 2) 可完整回溯软件交付过程满足审计要求 3) 持续优化的版本控制系统	管理机制 2) 可以针对不同业务和技术要求，选用不同的分支策略，在指定时间发布	制	穿整研发价值流交付过程 2) 在组织内部开放共享，建立知识积累和经验复用体系

5.1.2 变更管理

变更管理指软件系统中的所有变更都可追溯变更的详细信息记录，并向上追溯变更的原始需求、流转过程等所有关联信息，主要包括变更过程、变更追溯和变更回滚三方面，如表 3 所示。

可追溯性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。

5.1.2.1 变更过程

变更过程是变更的触发条件和实施手段，覆盖变更的完整生命周期。

5.1.2.2 变更追溯

变更追溯是变更相关信息和状态的识别和查询，包括变更人员、变更时间、变更原因、变更内容等。

5.1.2.3 变更回滚

变更回滚是将变更恢复到变更之前的状态的过程。

表3 变更管理

级别	变更过程	变更追溯	变更回滚
1	1) 无变更过程 2) 变更信息分散在每个系统内部，缺乏信息的有效共享机制	无	无
2	1) 建立代码基线 2) 记录代码变更管理信息 3) 针对重点变更内容进行评审	1) 有清晰定义的版本号规则 2) 实现制品和代码基线的关联，可追溯指定版本的完整源代码信息	手工实现回滚
3	1) 所有配置项变更由变更管理系统触发 2) 针对每次变更内容进行评审，并使用自动化手段	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	1) 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 2) 回滚操作实现自动化
4	1) 使用统一的变更管理系统 2) 变更管理过程覆盖从	1) 变更依赖关系被识别和标记 2) 实现数据库和环境变更信息的可追溯	自动化回滚全流程的所有变更包括变更依赖

	需求到部署发布全流程 3) 建立变更的分级评审机制		
5	可视化变更生命周期，支持全程数据分析管理	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	同上

5.2 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。

5.2.1 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，主要包括构建方式、构建环境、构建计划和构建职责四方面，如表4所示。

5.2.1.1 构建方式

构建方式是源代码转变为可运行程序的方法和过程。

5.2.1.2 构建环境

构建环境是构建实际运行过程的设备和资源依赖的载体。

5.2.1.3 构建计划

构建计划是构建被触发的方式，频率和编排过程。

5.2.1.4 构建职责

构建职责是整个构建相关工具，系统和过程的责任主体。

表4 构建与持续集成

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	使用研发本地设备构建	构建任务不定期执行	无专人专岗负责构建
2	采用脚本实现构建过程自动化	有独立的构建服务器，多种任务共用构建环境	实现每日自动构建 根据发布策略细分构建类型，比如发布构建、测试构建	构建工具、环境与计划由专人负责维护并有权限管理
3	1) 定义结构化构建脚本，实现模块级共享复用	1) 构建环境配置实现标准化 2) 有独立的构建资源池	1) 实现定期自动执行构建和代码提交触发构建 2) 明确定义构建计	构建工具和环境由专门团队维护并细分团队人员职责

	2) 构建脚本由专人专岗统一维护		划和规则，并在研发团队内共享	
4	1) 实现构建方式服务化，可按需提供接口或用户界面，将构建能力赋予整个研发团队 2) 研发团队可以按场景实现构建过程可视化编排	实现构建资源动态弹性按需分配与回收，如搭建基于云服务虚拟化和容器化的分布式构建集群	实现按需制定构建计划	构建实现自服务，团队接口人可自主按需执行
5	持续优化的构建服务平台，持续改进服务易用性	持续改进构建环境以提高构建效能	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

5.2.2 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，及时修复问题，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，软件可按照指定时间发布，主要包括集成服务、集成频率、集成方式和反馈周期四方面，如表 5 所示。

5.2.2.1 集成服务

集成服务是指持续集成运行的系统和环境，以及集成团队的职责划分。

5.2.2.2 集成频率

集成频率是指研发编写的源代码向代码主干分支合并过程的方法和实施频率，是持续集成的核心指标和参考能力。

5.2.2.3 集成方式

集成方式是代码集成的触发条件和集成过程中的环节及输入输出。

5.2.2.4 反馈周期

反馈周期是集成过程中出现的异常状态通知到人为处理的时间周期，以及将集成状态恢复到正常状态的过程时长。

表5 持续集成

级别	集成服务	集成频率	集成方式	反馈周期
1	无持续集成服务	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，集成期间主干长期不可用
2	搭建统一的持续集成服务	采用团队定期统一集成的策略，代码集成	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

		频率几天或者几周一次		
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	1） 实现持续集成服务化和自助化，研发团队可自行使用持续集成服务 2) 具备集成队列管理能力，队列资源和优先级可控	研发人员具备每天多次向代码主干集成的能力，可按需集成 任何变更（代码，配置，环境）都会触发完整的持续集成流程	1) 每次代码提交构建触发自动化测试和静态代码检查 2) 测试问题自动上报变更管理系统， 3) 测试结果作为版本质量强制要求，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

5.3 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。

5.3.1 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，并融入到持续交付的各个阶段中，达到质量防护的目的，如表 6 所示。

5.3.1.1 分层方法

分层方法是测试体系按照不同的测试对象，类型进行分类聚合的方法，每一层对应了特有的测试需求。

5.3.1.2 分层策略

分层策略是指基于测试分层策略对每部分的测试比重和投入，以及覆盖度等的划分策略。

5.3.1.3 测试时机

测试时机是指测试接入软件研发过程的时间点和参与形式以及期望结果。

表6 测试分层策略

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	无测试分层策略	测试在软件交付过程中在开发完成后才介入

2	1) 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； 2) 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； 3) 对系统进行基本的性能测试	1) 已建立测试分层策略 2) 测试设计以对用户/业务级UI 测试为主，辅以少量的接口/服务级测试	1) 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 2) 接口/服务级测试在模块的接口开发完成后进行
3	1) 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； 2) 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	1) 测试设计以对接口/服务级测试为主，兼顾用户/业务级测试辅以少量的代码级测试 2) 对非功能性测试进行全面系统的设计	1) 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 2) 代码级测试在模块的函数或类方法开发完成后进行
4	1) 采用测试驱动开发的方式进行代码级、接口级测试； 2) 采用探索性测试方法对需求进行深入挖掘测试	1) 测试设计以对代码级测试为主，兼顾接口/服务级和用户/业务级测试 2) 对探索性测试进行全面系统的设计 3) 测试分层策略的各层测试具有交叉互补性	1) 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； 2) 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	1) 在需求阶段进行用户/业务级测试设计 2) 在需求特性开发、交付整个过程中同步进行并完成测试

5.3.2 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，主要包括质量规约、检查方式、反馈处理三方面，如表 7 所示。

5.3.2.1 质量规约

质量规约是指对软件代码质量和要求和规范，其涵盖了编码规范，复杂度，覆盖率，以及安全漏洞，合规性要求等多个方面。

5.3.2.2 检查方式

检查方式是指代码质量规约检查的执行手段，触发条件，对执行效率、易用性等方面的同样提出要求。

5.3.2.3 反馈处理

反馈处理是指代码质量检查结果的收集，跟踪，处理的完整流程，可通过代码技术债务的指标进行衡量。

表7　代码质量管理

级别	质量规约	检查方式	反馈处理
1	具备初始代码质量规约，规约停留在个人级	代码质量检查采用人工方式进行评审	无代码质量检查结果处理机制，存在大量技术债
2	1) 建立团队级代码质量规约 2) 规约范围覆盖部分代码质量指标，如：代码规范、错误和圈复杂度、重复度等质量指标	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，只处理部分检查结果
3	1) 建立组织级代码质量规约 2) 建立完整的质量规约，将安全漏洞检查、合规检查纳入规约 3) 建立强制执行的质量门禁体系 4) 建立规约固定更新机制	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，根据质量规约维持一定的技术债
4	1) 建立公司级代码质量规约 2) 可根据业务需要灵活扩展和定制	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	1) 团队在研发阶段主动解决技术债 2) 整体技术债呈下降趋势
5	定期验证代码质量规约的完整性和有效性，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	1) 对代码质量数据进行统一管理 2) 可有效追溯并对代码质量进行有效度量

5.3.3　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，主要包括自动化设计、自动化开发、自动化执行和自动化分析，如表8所示。

5.3.3.1　自动化设计

自动化设计是指测试分层中各种测试类型的自动化设计方法，用于指导自动化测试工作的有效执行。

5.3.3.2 自动化开发

自动化开发是指依据自动化设计进行自动化测试工具、脚本、用例、框架、系统等不同层面的开发水平。

5.3.3.3 自动化执行

自动化执行是指自动化测试的执行条件和触发机制，以及测试问题的跟踪处理机制，从而满足自动化设计的目标。

5.3.3.4 自动化分析

自动化分析是指自动化测试结果的准确性，数据分析能力，以提供更多的反馈信息用来优化和持续改进自动化测试流程。

表8 自动化测试

级别	自动化设计	自动化开发	自动化执行	自动化分析
1	无	自动化测试脚本与工具分散管理	1) 手工测试为主，自动化程度低 2) 测试执行以周为单位	手工对测试结果进行分析判断
2	对用户/业务级的UI测试进行自动化设计	1) 设置专人专岗统一管理自动化测试脚本与工具 2) 使用版本控制系统对自动化测试脚本进行有效管理	1) 对用户/业务级UI测试采用自动化测试 2) 自动化测试的执行效率较低，以天为单位	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
3	1) 对接口/服务级测试进行自动化设计 2) 对代码级测试进行自动化设计	1) 建立统一的自动化测试框架，统一管理自动化测试用例 2) 自动化测试脚本开发采用数据驱动、关键字驱动等方法；	1) 对接口/服务级与代码级测试采用自动化测试 2) 自动化测试由流水线自动化触发	对自动化测试结果具备较强的自动判断能力，误报少
4	对性能、稳定性、可靠性、安全性等非功能性测试进行自动化设计	1) 建立自动化测试自服务平台 2) 优化自动化测试执行效率 3) 自动化测试资源池化	1) 有组织级的统一自动化测试平台，和上下游需求、变更管理系统打通； 2) 可以根据需求选择关联的自动化测试用例执行； 3) 可以将由于版本原因导致的失败用例	自动化测试数据模型标准化，和上下游需求、缺陷等研发数据关联，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。

			和缺陷关联	
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	持续优化的自动化测试平台	定期验证自动化执行策略，持续优化测试执行效率和资源利用率	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，可以自动向缺陷管理系统提交缺陷

5.4 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。

5.4.1 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，主要包括部署方式、部署过程、部署策略和部署质量四个方面，如表9所示。

5.4.1.1 部署方式

部署方式指软件包部署到线上生产环境或者交付用户的过程所采用的工具和方法。

5.4.1.2 部署过程

部署过程是指软件上线部署环节的实践方法以及完成部署活动的能力。

5.4.1.3 部署策略

部署策略是指部署过程的执行频率和部署内容以及部署手段来保证安全快速顺畅的生产部署。

5.4.1.4 部署质量

部署质量是指部署活动的成功率和确保部署质量提升的机制和能力。

表9 部署与发布模式

级别	部署方式	部署过程	部署策略	部署质量
1	运维人员手工完成所有环境的部署	部署过程存在较长的服务中止时间	1) 采用定期部署策略，部署频率以月为单位 2) 单次部署包含大量需求	1) 部署整体失败率较高 2) 部署无法实现回滚，生产问题只能在线上修复，修复时间不可控
2	1) 运维人员通过	部署过程通过流程文	1) 采用定期部署策	1) 部署失败率中等

	自动化脚本实现部署 2) 部署过程部分自动化	档实现标准化	略，部署频率以周为单位 2) 应用作为部署的最小单位 3) 应用和数据库部署实现分离 4) 实现测试环境的自动化部署	2) 实现应用部署的回滚操作，问题可及时修复
3	部署和发布实现全自动化	1) 使用相同的过程和工具完成所有环境部署 2) 一次部署过程中使用相同的构建产物	1) 采用定期部署策略，具备按天进行部署的能力 2) 应用和环境整体作为部署的最小单位 3) 应用和配置进行分离	1) 部署失败率低 2) 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 3) 每次部署活动提供变更范围报告和测试报告
4	1) 部署发布服务化，实现团队自助一键式多环境自动化部署 2) 同时支持数据库自动化部署	部署过程可灵活响应业务需求变化，通过合理组合实现灵活编排	1) 采用按需部署策略，具备一天部署多次的能力 2) 通过低风险的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
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	团队自主进行安全可靠地部署和发布	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

5.4.2 部署流水线

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，主要包括协作模式、流水线过程和过程可视化三方面，如表10所示。

5.4.2.1 协作模式

协作模式是指软件从需求到上线交付各个环节中各责任主体之间的信息传递和交互方式，体现整体交付过程顺畅程度。

5.4.2.2 流水线过程

流水线过程是指软件交付过程中各个环节活动的实现机制和整体交付的触发条件。

5.4.2.3 过程可视化

过程可视化指软件交付过程中信息的可见程度，以及所展现数据对于业务价值的展现能力。

表10　部署流水线

级别	协作模式	流水线过程	过程可视化
1	1) 整个软件交付过程严格遵循预先计划 2) 存在复杂的部门间协作和等待 3) 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	1) 交付过程中的信息是封闭的 2) 交付状态追溯困难
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	1) 交付过程在团队内部可见，信息在团队间共享 2) 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保障软件交付质量	1) 交付过程组织内部可见 2) 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	1) 建立可视化部署流水线，覆盖整个软件交付过程 2) 每次变更都会触发完整的自动化部署流水线	1) 部署流水线全员可见 2) 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

5.5　环境管理

环境作为DevOps持续敏捷交付过程中最终的承载，包括环境的生命周期管理、一致性管理、环境的版本管理。环境管理是用最小的代价来达到确保一致性的终极目标，主要包括环境类型、环境构建、环境依赖与配置管理三方面，如表11所示。

5.5.1　环境类型

环境类型是指研发环境种类的齐备性，用于满足不同阶段业务需求的能力。

5.5.2　环境构建

环境构建是指环境的生成方式和交付能力，从交付过程和交付速度中体现。

5.5.3　环境依赖与配置管理

环境依赖与配置管理是指环境所依赖的内容的识别和管理方法，以及环境变更的有效跟踪反馈，用于确保环境的一致性和受控。

表11　环境管理

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	1) 人工创建环境 2) 环境准备时间长，需要几周	1) 无依赖管理 2) 环境的管理为操作系统的

		完成	交付方式
2	建立功能测试环境	1) 环境构建通过自动化来完成 2) 环境准备时间以天为单位	通过配置管理工具实现操作系统级别的依赖管理，比如说操作系统版本、组件版本、程序包版本等等
3	建立标准的研发环境	1) 环境的构建通过自服务的资源交付平台来完成 2) 环境准备时间小时级	以应用为中心，有服务级依赖的配置管理能力，比如：依赖的关联服务，数据库服务、缓存服务、关联应用服务等等
4	建立全面的测试与灰度环境包括：开发环境，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容器化快速交付 环境准备实现分钟级	环境和依赖配置管理实现代码化描述
5	根据业务与应用的需要，弹性分配各类环境	环境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弹性变化

5.6 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。

5.6.1 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，包括数据来源、数据覆盖和数据独立性三方面的内容，如表12所示。

5.6.1.1 数据来源

数据来源是指测试数据的生成方式，用以满足不同测试类型的需求。

5.6.1.2 数据覆盖

数据覆盖是指测试数据对于各种测试类型需求的支持能力。

5.6.1.3 数据独立性

数据独立性是指测试数据在测试执行各阶段的完整性和一致性，不会受到其他任务执行结果的影响。

表12 测试数据管理

级别	数据来源	数据覆盖	数据独立性
1	测试时手工创建临时数据	测试数据覆盖部分测试场景	无
2	导出部分生产环境数据形成基准的测试数据集	1) 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 2) 进行初步的分类分级，满足不同测试类型需要	1) 测试数据有明确备份恢复机制 2) 实现测试数据复用和保证测试一致性
3	导出部分生产环境数据并清洗敏感信息后形成基准的测试数据集	建立体系化测试数据，进行数据依赖管理，覆盖全部测试分层策略要求的测试类型（注明引用内容的序号）	1) 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 2) 测试用例的执行不依赖其他测试用例执行所产生的数据
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	1) 测试数据覆盖安全漏洞和开源合规等需求场景 2) 建立定期更新机制	对测试数据分级，形成元数据和测试用例专用数据
5	所有的功能、非功能测试的测试数据，均可通过模拟、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

5.6.2 数据变更管理

数据变更管理主要是指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应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，主要包括变更过程、兼容回滚和数据监控三方面，如表13所示。

5.6.2.1 变更过程

变更过程是指数据库相关信息的更新方法和实现机制。

5.6.2.2 兼容回滚

兼容回滚是指数据库变更的向下兼容性以及回退变更的能力和方法。

5.6.2.3 数据监控

数据监控是指对数据变更过程的日志、状态、以及数据指标的收集分析和辅助决策的能力。

表13 数据变更管理

级别	变更过程	兼容回滚	数据监控
1	1) 数据变更由专业人员在后台手工完成 2) 数据变更作为软件发布的一个独立环节，单独	没有建立数据库版本，数据库和应用存在不兼容风险	数据变更结果需要访问数据库进行验证

	实施和交付		
2	1) 数据变更通过文档实现标准化 2) 使用自动化脚本完成数据变更	建立数据库和应用的版本对应关系，并持续跟踪版本变更	收集和分析数据变更日志，实现变更问题快速定位
3	将数据变更纳入持续部署流水线，经人工确认后自动完成	每次数据变更同时提供明确的回滚机制，并实现进行变更测试，如：提供升级和回滚两个自动化脚本	针对不同环境和危险程度对数据变更建立分级监控机制
4	1) 应用程序部署和数据库变更解耦，可单独执行 2) 数据变更随应用的部署自动化完成，无需专业人员单独执行	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	对数据变更进行监控，自动发现和修复异常变更
5	持续优化的数据管理方法，持续改进数据管理效率	同上	持续监控和优化数据变更机制

5.7 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。强调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。

5.7.1 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，主要包括度量指标定义、度量指标类型、度量数据管理和度量指标更新四个方面，如表14所示。

5.7.1.1 度量指标定义

度量指标定义是指度量指标设计的依据和生效领域，用于识别符合业务需求的度量指标。

5.7.1.2 度量指标类型

度量指标类型是指度量指标的覆盖和完整度。

5.7.1.3 度量数据管理

度量数据管理是指度量数据的收集，分析和管理。

5.7.1.4 度量指标更新

度量指标更新是指度量指标的更新机制，范围和频率。

表14　度量指标

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	在持续交付部分阶段定义度量指标	无	度量数据是临时性的	无
2	在持续交付各个阶段定义度量指标，度量指标仅限于部门内部	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	度量数据采用抽样方法收集	度量指标的设立后变更周期较长
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	持续收集度量数据 历史度量数据有明确的管理规则	1) 度量指标可以按照需求定期更新 2) 度量指标的优先级在团队内部达成一致
4	1) 整个团队共享核心业务度量指标 2) 建立基于能力成熟度模型的完整度量体系	度量指标覆盖探索性指标，并展示趋势，预测潜在问题，并及时预警	对历史度量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并持续更新
5	持续优化的度量指标，团队自我驱动持续改进	建立度量指标的有效反馈机制，并持续优化度量指标分类	同上	度量指标可基于大数据分析和人工智能自动识别和推荐动态调整指标优先级

表15　部分参考度量指标

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

5.7.2 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机制，不断驱动持续改进和价值交付，主要包括内容和生成方式、数据时效性、覆盖范围和反馈改进四个方面，如表16所示。

5.7.2.1 内容和生成方式

内容和生成方式是指度量报告的生成手段和数据展示能力。

5.7.2.2 数据时效性

数据时效性是指度量报告所体现结果的及时性以及实时更新能力。

5.7.2.3 覆盖范围

覆盖范围是指可查看度量报告的人员范围。

5.7.2.4 反馈改进

反馈改进是指度量发现的问题的处理方式。

表16 度量驱动改进

级别	内容和生成方式	数据时效性	覆盖范围	反馈改进
1	度量报告通过手工方式生成	数据和度量报告的时效性存在差异	仅部分人员可查看报告	度量反馈问题解决周期较长
2	度量报告以自动化方式生成 度量报告具有标准格式定义	数据报告可展现最新状态	团队内部成员均可查看报告	度量反馈的问题录入系统跟踪，并定期实施改进
3	度量报告进行分类分级并按需生成内容	通过可视化看板实时展示数据	全部团队成员均可查看报告	度量反馈问题纳入研发迭代的待办事项列表，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	通过可视化看板聚合报告内容多维度展示	实现报告精准范围推送，支持主动订阅	建立度量问题持续改进机制

	度量报告基于平台定制化生成，支持多种类型，如表格、看板等	实时状态 支持自动生成数据趋势图和趋势分析		团队预留工作时间用于解决度量反馈问题 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	持续优化度量方法，度量平台和度量展现形式	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