ICS
点击此处添加中国标准文献分类号
备案号：

中 华 人 民 共 和 国　　行 业 标 准

XX/T XXXXX—XXXX

研发运营一体化（DevOps）能力成熟度模型
第 5 部分：应用设计

The capability maturity model of DevOps

Part 5: Application Design

点击此处添加与国际标准一致性程度的标识

（征求意见稿）

－XX－XX 发布　　XXXX－XX－X

发 布

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（DevOps）能力成熟度模型”系列标准的第 5 部分 应用设计，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：安全风险管理

第7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： 待完善

本标准主要起草人：待完善

研发运营一体化（DevOps）能力成熟度模型 第5部分：应用设计

1 范围

本标准规定了研发运营一体化（DevOps）能力成熟度模型中应用设计能力的成熟度要求。

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度的标准依据。

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] YD/T 1171-2001 IP网络技术要求——网络性能参数与指标

[2] YD/T 1823-2008 IPTV业务系统总体技术要求

[3] YD/T 1489-2006 数字蜂窝移动通信网移动流媒体业务总体技术要求

3 术语

下列术语和定义适用于本文件。

3.1 软件架构 Software Architecture

软件架构是计算系统的软件架构是解释该系统所需的结构体的集合,其中包括软件元素,元素之间的相互关系和二者各自的属性。

3.2 应用程序 Application program

指研发团队生产的可基于运行时环境运行的软件包。

3.3 运行时环境 Runtime Environment

运行时环境指应用程序进入运行态的软件环境，包括操作系统、中间件、计算机程序设计语言编译器、计算机程序设计语言解释器、环境变量、SDK 等非研发团队的应用程序产出。

3.4 软件包 Software Package

通过计算机程序设计语言编写并生成的可运行计算机的代码集合。

4 缩略语

下列缩略语适用于本文件。

DevOps a portmanteau of development and operations 一组过程、方法与系统的统称

JSON JavaScript Object Notation JS 对象标记

HTTP HyperText Transfer Protocol 超文本传输协议

MTTF Mean Time To Failure 平均失效前时间

MTTR Mean Time Between Failures 平均恢复前时间

RPC Remote Procedure Call 远程过程调用

TCP Transmission Control Protocol 传输控制协议

XML eXtensible Markup Language 可扩展标记语言

5 应用设计

DevOps技术能力包括开发技术、测试技术、运维技术等能力，其中开发技术中最核心的是应用设计相关技术，应用设计的分级技术要求包括：应用接口、应用性能、应用扩展和故障处理，如表1所示。

表1 应用设计分级技术要求

应用设计			
应用接口	应用性能	应用扩展	故障处理
传输协议	实际性能	水平扩展	日志
数据协议	可用性	垂直扩展	监控
内容协议			故障追踪
接口治理			故障修复

5.1 应用接口

是指软件系统不同组成部分衔接的约定。

5.1.1 接口规范

是指通过接口标准化制定统一的规范和处理方式，降低接口的复杂度，减少接口对接的工作量，从而提升应用交付的速度和效率。

5.1.1.1 传输协议

指应用系统间传输数据所用的协议，例如TCP、HTTP、RPC等。

5.1.1.2 数据协议

指应用系统间传输的数据所采用的格式，例如 JSON、XML、私有协议等。

5.1.1.3 内容管理

指应用系统间传输的数据内容有统一的标准管理，例如JSON数据应该包含哪些字段。

表2 应用接口

级别	传输协议	数据协议	内容协议

1	各应用系统分别对外提供不同的传输协议，例如 A 系统提供 HTTP、B 系统提供二进制，C 系统提供 RPC	各应用系统分别对外提供不同的数据协议，例如 A 系统对外提供 XML 数据,B 系统对外提供 JSON 数据	各应用系统分别对外提供不同的内容协议，例如同样是 JSON 格式，A 系统 Json 数据键命名风格是下划线,B 系统 Json 数据键命名风格是驼峰式
2	各应用系统约定了统一的传输协议，例如指定采用 HTTP 传输协议	各应用系统约定了接口间传输的数据协议，例如指定数据协议为 XML	各应用系统约定了接口间传输的内容协议，例如 Json 的命名规范，最大长度等
3	各应用系统约定了统一的传输协议和协议规范，例如采用 HTTP 协议时，指定 Content-Type、Connection 等 header 的配置	各应用系统约定了统一的数据协议和数据规范，例如采用 XML 格式时，哪些信息用属性表示，哪些信息用元素表示	各应用系统约定了接口间传输的内容协议规范，例如指定 JSON 数据必须包含这些字段：requestID，caller，source，time，response
4	提供了统一的接口开发库或者开发包，各应用系统统一使用开发库或者开发包来完成协议解析和处理	提供了统一的数据编解码开发库或者开发包，各应用系统统一使用开发库或者开发包	提供了统一的内容编解码开发库或者开发包，各应用系统统一使用开发库或者开发包
5	提供了统一的开发框架，开发框架集成了协议处理，可以通过简单的方式就能够对外提供接口，例如通过 SpringBoot 的 @RestController 注解方式提供接口	提供了统一的开发框架，开发框架集成了协议处理，可以通过简单的方式就能够对外提供接口，例如通过 SpringBoot 的注解方式	提供了统一的开发框架，开发框架集成了协议处理，可以通过简单的方式就能够对外提供接口，例如 SpringBoot 的注解方式

5.1.2 接口管理

接口规范是指利用统一的接口规范约束各个系统按照统一的标准规范操作，并对应用系统提供的接口进行管理，主要内容包括接口查询和权限控制，如表 2 所示。

5.1.2.1 接口查询

应用系统需要查询其它应用系统提供的接口，应确保接口符合规范。

5.1.2.2 权限控制

应用系统对外提供的接口，需要进行权限控制，例如部分敏感数据的访问。

表3 接口管理

级别	接口查询	权限控制	接口规范管理
1	各应用系统自己维护接口文档，通过邮件或者即时通信的方式传送接口文档	各应用系统对外提供的接口没有权限控制能力	自行定义的格式的接口规范，接口规范相对松散。

2	提供统一的接口查询平台，各个应用系统开发人员在这个平台上手工填写自己系统的接口信息	各应用系统自己实现基本的访问控制权限	有统一的接口规范，但接口由标准自由定义。例如：无标准的 XML ，Json
3	提供统一的接口查询平台，各个应用系统开发人员可以从代码导出接口信息，然后发布在接口信息查询平台	各应用系统自己实现了访问控制和细粒度的数据控制权限	有统一的接口规范，接口遵循相应的标准规范。例如：wsdl，swagger.io
4	提供统一的接口查询平台，各个应用系统开发人员可以从代码导出接口信息，然后发布在接口信息查询平台	各应用系统自己实现了访问控制和细粒度的数据控制权限	a） 有统一的接口规范，接口遵循相应的标准规范。 b） 接口规范由统一的基础设施中心化管理。 例如：依照规范设立的集中式 API Gateway
5	提供统一的接口平台，各个应用系统可以自动注册接口相关信息，接口平台可以自动测试接口是否符合规范要求。	提供统一的接口权限管理平台，平台统一管理接口的访问控制权限和数据控制权限。	a） 有统一的接口规范，接口遵循相应的标准规范。 b） 接口规范由 API 按照标准的方式去中心化管理。 例如：依照规范设立的分布式服务网格(Service Mesh)

5.2 应用性能

应有性能是对应用实际性能（Real Performance，与感知性能 Perceived performance 相对）和可用性（Availability）的度量，是衡量应用服务水平的重要指标。

5.2.1 实际性能

实际性能是指用户实际体验的性能，例如在正常载荷或者最大载荷情况下平均响应时间。通过两个指标进行度量，如下：

a） 应用在一定载荷（Load，例如请求数/秒、事务数/秒、页面数/秒）情况下对最终用户请求的响应时间；

b） 应用在一定载荷情况下计算资源消费情况，包括CPU、内存、IO、网络带宽等。通过计算资源消费情况判断计算资源是否支持指定的载荷，或者建立资源消费情况基线，在应用生命周期中追踪应用性能变化。

5.2.2 可用性

可用性是指在要求的外部资源得到保证的前提下，产品在规定的条件下和规定的时刻或时间区间内处于可执行规定功能状态的能力。它是产品可靠性、可维护性和维护保障性的综合反映。可用性一般通过冗余和故障转移的方式获得。

5.2.2.1 系统可用性

是指系统服务不中断运行时间占实际运行时间的比例。

5.2.2.1.1 系统可用性指标

系统可用性指标定义为：MTTF/(MTTF+MTTR) * 100% ，其相关的两个指标定义如下：
MTTF： mean time to failure，平均失效前时间，也就是平均正常运行的时间。
MTTR： mean time to restoration， 平均恢复前时间,也就是平均故障时间。

表4 应用性能

级别	实际性能	可用性
1	a) 各模块独自处理性能问题，采用不同的性能方案，性能结果是被动获得； b) 性能指标散布在日志中，采用手工方式计算性能指标。	a) 应用没有进行冗余设计，不支持高可用； b) 故障问题没有设计统一的故障恢复方案；
2	a) 对于典型的性能问题进行了个性化设计，解决局部性能问题； b) 对部分性能指标进行度量； c) 部分性能指标采用个性化方式可视化；	a) 应用部分子系统（例如应用服务器、数据库、web 服务器等）或者 IT 设施（网络、存储、主机等）进行冗余和故障转移设计，支持部分高可用； b) 对于部分可用性故障问题有故障恢复方案；
3	a) 站在最终用户的视角，对应用进行系统化性能设计； b) 支持对性能进行全方位度量，从端到端性能到系统各层性能； c) 支持性能指标实时采集； d) 设计或者采用第三方工具对性能进行可视化，并支持性能问题追溯；	a) 应用子系统和 IT 设施有系统化的冗余和故障转移设计，整个系统全面支持高可用； b) 应用子系统和 IT 设施的设计完整的故障恢复方案，人工可按照恢复方案进行故障恢复； c) 设计应用子系统和 IT 设施可用性检查（健康检查）方案； d) 设计或者采用第三方工具对可用性可视化；
4	a) 设计或外购统一性能管理支持平台，支持性能管理循环； b) 建立性能管理制度化机制，平台支持制度化性能设计流程，支持性能度量、分析、追溯、定位、响应、	a) 建立高可用管理支持平台，统一支持应用子系统和 IT 设施的可用性监控和预警； b) 建立高可用制度化机制，高可用管理支持平台可用性管理机制实施；

	评估性能管理循环； c） 平台支持性能问题预警；	c) 支持自动化故障恢复过程； d) 支持制度化进行灾备演习，确保故障恢复方案有效，缩短故障恢复时间；
5	a) 支持实时性能指标自动分析能力，对于一些常见性问题能够进行问题自动定位； b) 对于一些常见的自动定位问题能够进行智能处理（例如扩容等）； c) 支持制度化评估性能管理机制和平台，支持卓越性能管理；	a) 支持实时可用性自动分析，支持常见问题自动定位； b) 自动定位的常规问题执行故障自动恢复； c) 制度化评估可用性管理机制和平台，支持高可用卓越管理；

5.3 应用扩展

应用程序在达到最大负载时，能够支持以下方式进行扩展，以保证系统稳定运行。如表 5 所示。应用扩展性是应对高并发的重要手段，扩展包括三个维度，如下：

a) X 轴 - 是否支持水平扩张（容量扩展），应用可以复制多个实例，共同提供服务；

b) Y 轴 - 是否支持垂直扩展（服务资源），将应用的不同模块部署在不同的进程中；

c) Z 轴 - 是否支持数据扩展（数据存储），将数据分散在多个存储单元中；

应用系统的容量需求会随着业务的发展而增加，容量扩展与应用架构相关，当应用架构具备容量扩展的能力，才能完成容量扩展操作。

表5 应用扩展

级别	水平扩展	垂直扩展
1	a) 系统容量不支持水平扩展。 b) 扩展时系统性能受影响。	a) 应用没有进行切分，采用一个巨石架构，所有功能归集在一个发布包中； b) 应用内部没有或者进行了简单的逻辑分层。 c) 部署不可回滚，或者回滚后需要人工进行数据修复 d) 单个子系统部署耗时 30 分钟以上。
2	a) 系统容量支持水平扩展，能够根据业务的需要通过手工的方式扩展容量，但扩展到一定容量后无法继续	a) 应用按照经验进行了简单拆分，将大应用分为若干独立的子系统，各个子系统独立部署。 b) 子系统职责定义清晰，子系统没有分层，控制

	扩展。 b) 扩展时系统性能受影响。 例如：负载均衡的业务系统可以扩展，但扩展到一定程度后数据库成为瓶颈，单独扩展业务系统容量无法提升系统容量。	循环依赖； c) 单个子系统部署耗时不超过 30 分钟 d) 80%业务版本需要多个子系统联动升级，升级有先后顺序
3	系统容量支持水平扩展，能够根据业务的需要通过手工的方式按需扩展。	a) 由架构设计人员采用领域驱动设计方法对应用进行拆分，定义各个拆分的职责，各个拆分可以独立继续部署。 b) 架构设计人员采用领域驱动定义应用切分分层及各层职责，各层之间严格控制调用关系（例如上层可以调用下层或者同层），控制依赖的复杂性。 c) 单个服务系统部署时间不超过 10 分钟。 d) 30%以下的业务版本需要多个服务系统联动升级。
4	a) 系统容量支持水平扩展，系统能够自动的根据业务需要扩展容量。 b) 扩展时系统性能不受影响。 例如：使用 Docker、虚拟机等	a) 对应用架构的拆分定义明确的指标和评估方法，有完整的指标收集方法和流程，并对拆分后的应用健壮性进行评估； b) 对模块耦合性定义指标和评估方法，有完善的流程对指标数据进行评估。 c) 单个子系统部署耗时 5 分钟以内。 d) 10%以下的业务版本需要多个服务系统联动升级。
5	a) 系统具备动态容量管理能力，能够根据系统负载 1 分钟内自动扩容，并且在容量过剩的时候根据一定的策略进行缩容。 b) 扩展时系统性能不受影响。	a) 依据指标对架构拆分情况进行评估，并提出实施架构改进措施; b) 依据耦合性指标数据评价耦合性现状，有完善的流程和方法对耦合性进行控制。 c) 单个子系统部署耗时 1 分钟之内。

	例如：电商大促、秒杀等场景根据流量自动扩容	d) 没有业务版本需要多个服务系统联动升级。

5.4 故障处理

故障处理（Troubleshooting）是指在系统失效、停止响应或出现异常时识别、规划和解决系统问题的过程，帮助修理和恢复系统。在系统运行过程中，由于运行环境的变化、软件本身的缺陷等原因，可能造成系统运行故障。在系统出现故障时，要求运维人员能够快速发现和解决故障，即快速的故障处理。故障处理过程循环包括五个步骤，即故障发现、故障追踪、故障解决方案设计、故障排除和故障记录。

复杂系统往往由多个子系统构成，任何一个最终用户的操作可能涉及多个子系统之间复杂的协作，故障发现、追踪和排除是一个复杂的过程，快速故障处理需要应用设计提供基础故障处理能力。

应用设计从以下几个方面支持故障处理过程，包括日志（记录故障现场）、监控（发现故障）、故障追踪（定位故障）和故障修复，如表 6 所示。

5.4.1 日志

日志是指对系统运行过程的记录，分为开发日志和业务日志。开发日志是记录代码调用过程和状态，例如 JAVA 程序员一般用 log4j 工具输出开发日志。业务日志是对用户业务操作行为及结果的记录，一般需要设计独立的工具进行支持。

业务系统拆分为多个子系统后，如果需要了解整个业务的运行情况，需要综合多个子系统的监控信息，将多个子系统的监控信息关联起来才能得出业务整体的运行状态信息。

通过监控标准化制定统一的监控规范、监控对象、监控数据，监控系统能够将多个子系统上的监控信息关联起来形成业务整体监控信息。

5.4.2 监控

应用设计需要支持将系统的运行情况实时展示监控信息。系统能够通过工具随时查看当前系统的运行情况；以便系统运行出现故障时，研发、测试、运维人员等能够及时的获取整个系统的运行情况，进行快速的故障判断和处理。

根据运维提供的监控规范、监控对象、监控指标，应用设计需要支持运维系统能将分散在多个子系统上运行情况实时的展现出来，并关联起来形成整体监控信息，保证系统可监控。

5.4.3 故障追踪

故障追踪是指应用设计能够支持对实时监控或其它方式获得的问题进行调用链分析，定位故障根源，为故障处理提供依据。例如，对于复杂的分布式系统，一个客户请求涉及来自多个机器的多个进程的多个服务协作，故障的定位是一个复杂过程。

5.4.4 故障修复

故障修复是指应用设计支持自动修复常见的故障，例如流量突增、硬件损坏、网络拥塞等，当故障发生时，系统除了进行告警外，还能够自动采取一定应对措施及时处理，降低故障影响范围和程度，减少故障影响时长。对于一些不常见的故障，支持人工故障修复。在故障处理完成之后，还需要记录故障，以便后期进行故障分析和积累故障管理知识。

表6　故障处理

级别	日志	监控	故障追踪	故障修复
1	a) 各应用模块/层采用不同的规范和格式输出开发日志； b) 开发日志存储在各个运行环境中； c) 日志记录随意，缺乏完整性；	开发和运维分离，开发独立完成监控设计。	系统是黑盒状态，运行状态无法获取，出现问题仅靠运维人员经验猜测或者看源代码推断。	系统无故障处理能力，出现故障后需要人工处理。
2	a) 模块/层并制定并实施了开发日志规范； b) 开发日志完整记录程序调用过程；	a) 制定了架构层面标准化监控的规范 b) 各系统按照规范独立实现监控功能	依靠人工方式进行日志分析。	系统提供故障处理能力，但需要人工触发操作，例如手工降级、手工倒换、手工切换冷备机房。
3	a) 开发日志记录统一追溯标示，开发日志支持调用链分析； b) 有统一开发日志存储环境集中日志实时存储； c) 设计专门业务日志系统，系统支持业务日志收集；	a) 运维人员在应用设计前，提供完善的监控规范、监控对象和监控指标； b) 运维人员全程参与应用设计； c) 按照监控规范要求进行完整的监控设计；	a) 有专门的故障追踪工具； b) 系统的关键信息能够通过日志查看； c) 单个子系统提供输入输出相关的监控，图形化展示，例如：吞吐量、响应时间、错误码分布。	当出现硬件级别的故障时，系统能够在 5 分钟内自动恢复业务。常见手段如主备、集群
4	a) 支持端到端开发日志记录； b) 开发日志与业务日志能相互贯通； c) 业务日志支持敏	支持监控系统能够基于不同系统的监控信息，构建全链路、全业务监控。此项能力关注的是监控信息关联和展示	a) 单个子系统提供处理流程相关的监控，图形化展示，例如：输入（请求量）、输出（响应时间）、处	当出现机房级别的故障时（机房断电、机房被攻击、流量突发等），系统能够在 5 分钟内自动自动恢复业务。常见手段如同城双活、限流。

	感业务审计；		理步骤（访问数据库的性能）等； b) 多个子系统的监控信息能够关联起来形成整个业务系统的监控全景。	
5	建立制度化日志评审机制，定期对日志质量进行评估，提高日志有效性。	监控系统能够基于全链路、全业务监控进行自动化预警、故障定位。此项能力关注的是监控信息分析和诊断	a) 整个业务系统的监控能够进行自动预警； b) 多个业务子系统能够针对具体的请求进行细粒度的监控，监控信息能够关联起来，形成业务请求全链路监控。例如微服务系统里面的全链路跟踪。	当出现地理位置级别的故障时，系统能够在 5 分钟内自动恢复业务。常见手段如异地多活。

参 考 文 献