ICS 03.060
CCS A 11

JR

中华人民共和国金融行业标准

JR/T 0351—2026

证券期货业研发运营一体化体系建设指南

Guidelines for the construction of integrated R&D operation system in the securities and futures industry

2026-05-09 发布　　2026-05-09 实施

中国证券监督管理委员会　发布

目　次

前言 III
引言 IV
1　范围 1
2　规范性引用文件 1
3　术语和定义 1
4　缩略语 5
5　概述 5
5.1　综述 5
5.2　文化理念 7
5.3　组织架构 7
5.4　制度体系 7
6　需求管理体系 8
6.1　需求管理概述 8
6.2　需求管理体系组织保障 8
6.3　需求收集与分析 9
6.4　需求过程管理 12
6.5　需求的价值管理 14
6.6　需求管理工具链与平台 15
7　技术实现体系 17
7.1　技术实现概述 17
7.2　技术实现体系组织保障 17
7.3　研发迭代管理 18
7.4　研发过程管理 19
7.5　研发流水线管理 22
7.6　技术实现工具链与平台 24
8　质量管理体系 25
8.1　质量管理概述 25
8.2　质量组织体系保障 25
8.3　质量体系建设 26
8.4　质量分级管控 28
8.5　质量的价值管理 29
8.6　质量管理工具链与平台 30
9　技术运营管理体系 32
9.1　技术运营管理概述 32
9.2　技术运营体系组织保障 32
9.3　监控管理 33
9.4　可用性管理 34
9.5　事件管理 36
9.6　应急管理 37
9.7　变更管理 37

9.8 性能容量管理 39
9.9 运营配置管理 41
9.10 技术运营管理工具链与平台 41
10 安全管理体系 44
10.1 安全管理体系概述 44
10.2 组织保障 44
10.3 需求管理体系中的安全 45
10.4 技术实现体系中的安全 45
10.5 质量管理体系中的安全 48
10.6 技术运营体系中的安全 49
10.7 新技术安全管理 51
10.8 安全管理工具链与平台 51
11 效能度量 53
11.1 效能度量概述 53
11.2 组织保障 53
11.3 度量目标与指标 53
11.4 度量可视化分析与决策 54
11.5 度量驱动改进 57
11.6 度量指标实践参考 59
12 一体化协同 66
12.1 需求管理与技术实现的协同 66
12.2 需求管理与质量管理的协同 68
12.3 需求管理与安全管理的协同 68
12.4 技术实现与质量管理的协同 69
12.5 技术实现与技术运营管理的协同 69
12.6 技术实现与安全管理的协同 70
12.7 质量管理与技术运营管理的协同 70
12.8 质量管理与安全管理的协同 71
12.9 技术运营管理与需求管理的协同 72
12.10 技术运营管理与安全管理的协同 72

前　　言

本文件按照GB/T 1.1—2020《标准化工作导则　第1部分：标准化文件的结构和起草规则》的规定起草。

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专利的责任。

本文件由全国金融标准化技术委员会证券分技术委员会（SAC/TC 180/SC4）提出。

本文件由全国金融标准化技术委员会（SAC/TC 180）归口。

本文件起草单位：上海证券交易所、中泰证券股份有限公司、中国金融期货交易所、上交所技术有限责任公司、上证所信息网络有限公司、国泰海通证券股份有限公司、华泰证券股份有限公司、申万宏源集团股份有限公司、东方证券股份有限公司、光大证券股份有限公司、国金证券股份有限公司、富国基金管理有限公司、中欧基金管理有限公司、中银国际证券股份有限公司、江海证券股份有限公司、易方达基金管理有限公司。

本文件主要起草人：黄天寿、王泊、唐忆、高剑、张永启、吴鑫、俞枫、王洪涛、王东、王忭思、张禄旭、庄飞、杨忠琪、朱震宇、李强、贾建国、葛浩、李丹、陈冬严、张勇、王征宇、向元武、杜铁绳、张嵩、王波、唐永鹏。

引　言

在全球化和信息化的今天，金融科技的创新浪潮正以前所未有的速度重塑着证券期货业的面貌。随着人工智能、大数据、云计算等新兴技术的广泛应用，传统的金融服务模式和运营流程正面临着重大的转型压力。研究如何在确保研发运营过程安全、合规的前提下，提升研发交付效率，使产品和服务需求能够得到及时响应，是当前行业面临的重要挑战。

本文件旨在规范证券期货业的研发运营过程，为研发运营一体化体系建设提供科学指导。本文件从需求管理、技术实现、质量管理、技术运营和安全管理等多个角度，阐述了研发运营一体化的具体要求和实施方法。本文件编写过程中广泛收集了来自监管机构、行业协会、金融机构以及技术专家的意见和建议，力求适应行业发展的实际需求和未来趋势，通过构建高效、协同、创新的研发运营模式，提升行业的整体技术水平和服务能力。

证券期货业研发运营一体化体系建设指南

1 范围

本文件提供了证券期货业研发运营一体化体系建设领域的一套参考性标准指南，包括规范性引用文件、术语和定义、缩略语、概述、需求管理体系、技术实现体系、质量管理体系、技术运营管理体系、安全管理体系、效能度量和一体化协同等内容。

本文件适用于证券期货业相关机构建设研发运营一体化体系作为指导性文件，为相关咨询、评估、认证、完善活动提供系统化、规范化参考条目。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；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。

GB/T 25000.1—2021 系统与软件工程 系统与软件质量要求和评价（SQuaRE） 第1部分：SQuaRE指南

GB/T 25069—2022 信息安全技术 术语

GB/T 29246—2023 信息安全技术 信息安全管理体系 概述和词汇

GB/T 38634.1—2020 系统与软件工程 软件测试 第1部分：概念和定义

GB/T 43698—2024 网络安全技术 软件供应链安全要求

中国证监会令第179号（2018年12月19日发布，2021年1月15日修订） 证券基金经营机构信息技术管理办法 第二十九条、第三十一条

3 术语和定义

下列术语和定义适用于本文件。

3.1

需求管理 requirement management

确保需求被及时、准确地理解与满足的系统性过程。

3.2

合规性 compliance

遵循相关法律法规、行业标准、内部政策等要求的特性。

3.3

风险管理 risk management

指导和控制组织相关风险的协调活动。

[来源：GB/T 29246—2023，3.69]

3.4

持续交付 continuous delivery

持续、可靠地向用户交付产品或服务。

3.5

持续测试 continuous testing

产品开发过程中持续进行的测试活动。

3.6

持续安全 continuous security

软件开发和运维过程中持续实施的安全措施。

3.7

需求规格化 requirement specification

将用户需求转化为清晰、具体、可执行和衡量的规范的过程。

3.8

需求条目化 requirement itemization

将管理需求通过标签和层次划分的过程。

3.9

需求过程管理 requirement process management

将需求按计划推进的管理活动。

注1：包括需求的提出、分析、规格化和实现等环节。

3.10

需求变更管理 requirement change management

在项目开发过程中，针对需求发生变化时采取的系统化管理过程。

3.11

需求验收 requirement acceptance

验证交付物与需求规格一致性的活动。

3.12

需求池 requirements pool

按照一定规则汇总记录产品（系统）相关需求信息的数据库。

3.13

需求价值管理 requirements value management

项目开发过程中，通过对需求进行评估、优先级排序、资源分配和跟踪等操作进行管理的实践活动。

3.14

迭代管理 iterative management

遵循敏捷原则交付产品和服务的管理活动。

3.15

技术实现 technical implementation

基于业务需求，通过软件开发或系统集成流程，生产交付满足需求的可执行代码、系统或服务的过程。

3.16

辅助编程 assisted programming

利用编码工具、库或现有代码片段来加速和简化编程过程的方法或技术。

3.17

制品管理 artifact management

在软件开发过程中对各种软件制品（如编译后的代码、库、文档等）进行全面管理和控制的活动。

3.18

研发配置管理 R&D configuration management

对软件产品或系统的资源和配置项进行全面管理和控制的活动。

3.19

研发提测 R&D testing submission

在研发阶段完成后，将阶段性成果提交给测试团队进行评估和验证的过程。

3.20

构建环境管理 build environment management

对用于软件构建的环境进行配置、更新、伸缩等管理活动。

3.21

流水线原子操作能力 pipeline atomic operation capability

流水线完成特定单元任务操作的能力。

注1：如代码检出、编译、测试的能力等。

3.22

测试管理 test management

计划、估算、监控和控制测试活动。

注1：需要对质量、进度、资源、风险进行管理。

3.23

缺陷管理 defect management

通过跨团队协作与自动化工具，对软件生命周期中的缺陷进行系统化识别、跟踪、修复和验证，以持续改进质量的过程。

3.24

自动化测试 automated testing

应用软件来执行或支持测试的活动。

注1：如测试管理、 测试设计、测试执行和结果检验。

3.25

测试左移 shift-left testing

在软件开发生命周期（SDLC）的早期阶段系统化地引入测试的活动。

注1：如需求分析、设计、编码阶段

3.26

测试右移 shift-right testing

在软件发布后的生产环境中持续进行的质量保障活动。

注1：通过监控、用户反馈、混沌工程等手段，验证系统在真实场景下的性能、稳定性及用户体验，并快速响应和修复问题。

3.27

测试用例 test case

前置条件、输入（包括操作）和预期结果的集合，用于驱动测试项的执行以满足测试目标，测试目标包括正确实现、错误识别、检查质量和其他有价值的信息。

注1：测试用例是测试子过程的最低测试输入级别（即，测试用例无法再划分为更细的测试用例）。

注2：测试用例的前置条件包括测试环境、已有数据（如数据库）、被测软件、硬件等。

注 3：输入是用于驱动测试执行的数据信息。

注 4：预期结果包含通过的准则、失效的校核。

[来源：GB/T 38634.1-2020，3.48]

3.28

质量门禁　quality threshold control

在软件生命周期中，通过一定的规则和测试技术，对软件质量要求进行限制的过程。清晰地定义从需求到线上发布交付这一整个流程中，每个环节的准入准出标准，在实施中可建立各项指标以及综合性评价指标。

3.29

软件供应链　software supply chain

需方和供方基于供应关系，开展并完成软件采购、开发、交付、获取、运维和废止等供应活动而形成的网链结构。

[来源：GB/T 43698-2024，3.7]

3.30

安全架构　security architecture

一种由多个相互协作的安全模块构成的体系结构。

[来源：GB/T 25069-2022，3.14]

3.31

安全测试　security testing

使评估对象按预定方法/工具产生特定行为，以获取证据来证明其安全确保措施是否有效的过程。

3.32

渗透测试　penetration testing

以未经授权的动作绕过某一系统的安全机制来检查信息系统的安全功能，以发现信息系统安全问题的手段。

3.33

安全评估　security assessment

按安全标准及相应方法，验证某一安全可交付件与适用标准的符合程度，及其确保安全程度的过程。

3.34

风险评估　risk assessment

将风险分析的结果与风险准则比较，以确定风险大小是否可接受或可容忍的过程。

[来源：GB/T 29246-2023，3.67]

3.35

风险接受　risk acceptation

在充分了解风险的基础上，承担特定风险的知情决定。

[来源：GB/T 29246-2023，3.62]

3.36

风险识别　risk identification

发现、识别和描述风险的过程。

[来源：GB/T 29246-2023，3.68]

3.37

服务水平协议　Service-Level Agreement

规定技术支持、业务性能目标及责任承担的文件，包括服务提供方能为其客户提供保证性能和承担失败后果的措施。

3.38

服务可用性 Service Availability

服务满足性能标准的时间百分比。

3.39

灾难恢复计划 disaster recovery plan

信息系统灾难恢复过程中筹划所需的任务、行动、数据和资源，用于指导相关人员在预定的灾难恢复目标内恢复信息系统所支持关键业务功能的文件。

4 缩略语

下列缩略语适用于本文件。

R&D：Research and Experimental Development

API：应用程序编程接口（Application Programming Interface）

CD：持续部署（Continuous Deployment）

CI：持续集成（Continuous Integration）

CRM：客户关系管理（Customer Relationship Management）

CTDD：持续测试驱动开发（Continuous Test-Driven Development）

DAG：有向无环图（Directed Acyclic Graph）

DAST：动态应用程序安全测试（Dynamic Application Security Testing）

IaC：基础设施即代码（Infrastructure as Code）

IAST：交互式应用程序安全测试（Interactive Application Security Testing）

IT：信息技术（Information Technology）

MAST：移动应用程序安全测试（Mobile Application Security Testing）

MTTR：平均故障恢复时间（Mean Time To Recovery）

SAST：静态应用安全测试（Static Application Security Testing）

SCA：软件成分分析（Software Composition Analysis）

SLA：服务水平协议（Service Level Agreement）

5 概述

5.1 综述

在技术系统或相关服务的研发与交付过程中，研发运营一体化体系将需求管理、技术实现、测试验证、技术运营管理、质量管理、安全管理和组织保障及文化等环节有机整合起来，基于整个组织的协作和应用架构优化，实现需求管理、敏捷开发、持续交付和技术运营等各环节在相对独立的基础上无缝衔接，帮助企业提升技术运营效能，在确保安全和合规的同时，快速交付高质量的系统或服务，灵活应对快速变化的业务需求、市场环境和通用技术变迁。研发运营一体化体系是企业内部技术生态基因，其结构具有较强的稳定性并会随时间发生进化，同类企业不同个体间具有较高相似性，较小差异会衍射出个体层面的巨大不同，每个环节和研发运营一体化的文化理念、组织架构和制度体系密切相关。研发运营一体化作用域参考图1。

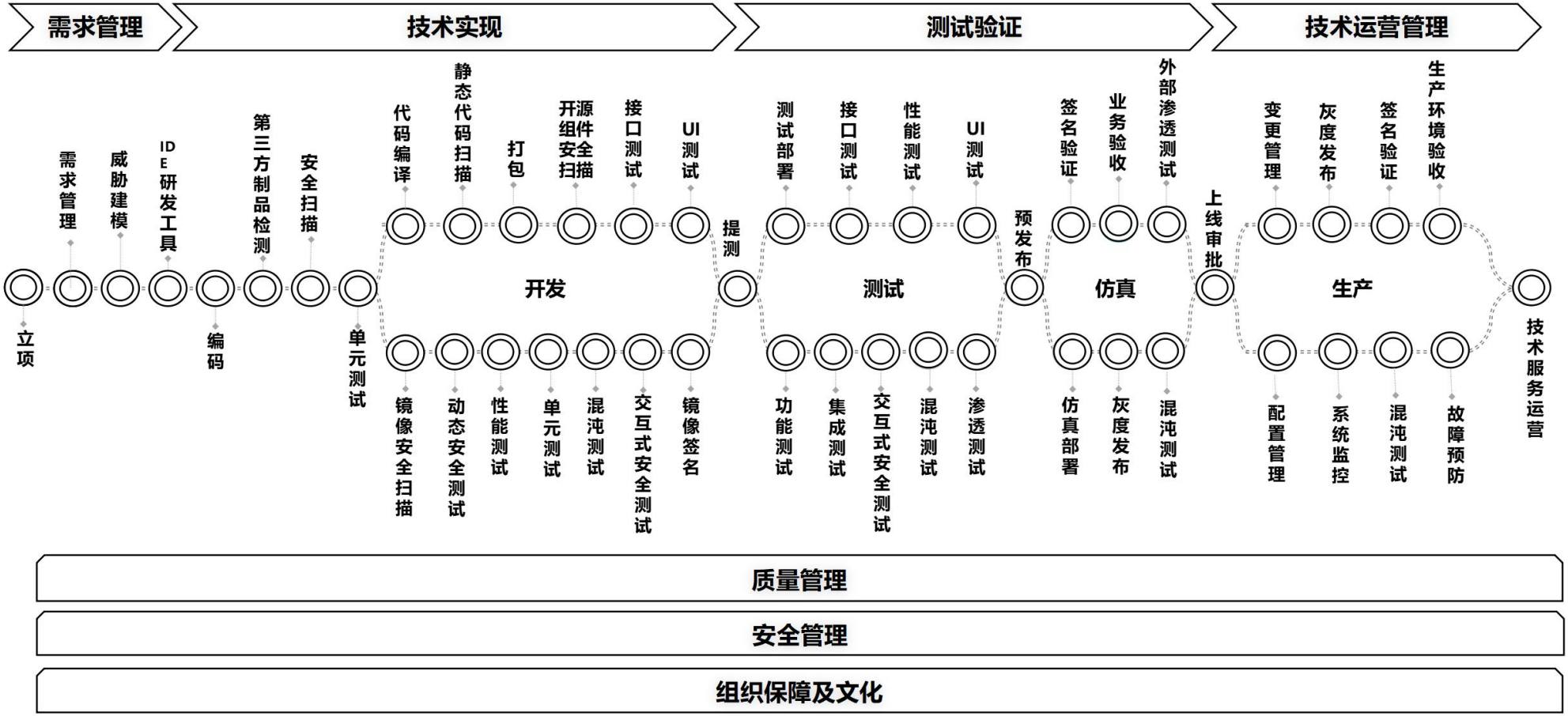

图 1　研发运营一体化作用域

5.2 文化理念

企业文化是企业发展的灵魂，对于证券期货业而言，构建与研发运营一体化相适应的企业文化至关重要。这种文化强调创新、协作、学习和持续改进的理念，通过敏捷机制实现价值的高效交付，激发员工的积极性和创造力，为行业的持续发展提供强大的精神动力。文化建设的路径主要包括以下几点：

a）树立创新理念：鼓励员工敢于尝试、勇于创新，营造宽松的创新氛围，使创新成为企业发展的核心驱动力；

b）强化团队协作：建立跨部门、跨领域的敏捷协作机制，采用敏捷项目管理框架打破职能壁垒，促进信息共享和资源整合，提高整体工作效率；

c）倡导学习精神：倡导终身学习、持续学习的理念，加强敏捷方法论和数字化转型的知识储备，为员工提供多样化的学习机会和平台，不断提升员工的专业素养和综合能力；

d）追求持续改进：建立持续改进的文化氛围，鼓励员工发现问题、分析问题、解决问题，不断优化工作流程和业务模式；

e）建立敏捷响应机制：培养全员危机意识，通过价值流分析识别关键节点，建立快速决策通道和弹性资源调配机制，确保组织能够快速应对监管政策调整和技术变革；

f）践行价值驱动交付：推行最小可行产品（MVP）理念，建立业务－技术协同的敏捷交付体系，通过持续客户反馈和动态优先级调整，确保每个迭代周期都能产出可衡量的业务价值。

5.3 组织架构

组织架构是企业运行的基础框架，对于研发运营一体化而言，一个高效、灵活的组织架构是确保各项工作顺利开展的关键。一个合理的组织架构能够充分发挥各部门的优势，促进资源的优化配置和高效利用，组织架构的设计原则如下：

a）扁平化管理：设置合理的管理层级，如专业序列与管理序列并行的双通道体系，控制层级深度，确保决策效率和执行力，使信息能够迅速传递和反馈；

b）跨部门协作：打破部门壁垒，组建跨职能敏捷团队，建立跨部门协作机制，如在部门墙之间建立“铰链式”接口岗位，通过轮岗机制培养 T 型人才，实现资源的共享和互补；

c）灵活性与适应性：构建“稳态+敏态”双模组织结构，根据市场变化和业务需求及时调整组织架构，保持组织的灵活性和适应性；

d）领域专业化发展路径：按业务条线建立专业能力中心（CoE），设计覆盖前中后台的任职资格体系，实施专业人才梯队建设计划，通过行业资格认证、专业案例库建设持续提升职能领域专业度；

e）跨领域协同创新机制：设立数字化转型联合实验室，建立跨部门创新协同制度，通过定期举办业务技术融合工作坊，搭建知识图谱驱动的协同平台，促进智能化、合规科技等交叉领域的突破性创新。

5.4 制度体系

制度体系是企业运行的规范保障，对于研发运营一体化而言，一套完善、有效的制度体系是确保各项工作有序开展的重要保障。完善的制度体系应涵盖以下方面：

a）需求管理制度，明确需求获取、分析、评审、跟踪及变更等流程，规范需求管理过程。通过详细全面的需求管理制度，确保需求的业务价值与质量；

b）技术实现管理制度，明确技术实现流程、规范技术实现行为、保障技术实现质量。通过制定详细的技术实现管理制度，确保该工作有序进行，减少不必要的风险和浪费；

c）质量管理制度，建立全面的质量分层测试体系，包括单元测试、集成测试、系统测试等环节。

通过制定详细的测试管理制度，确保产品质量的可靠性和稳定性；

d）运维管理制度，规范运维行为、保障系统稳定运行。通过制定详细的运维管理制度和应急预案，确保系统能够及时响应和处理各种突发情况，保障业务的连续性和稳定性；

e）安全管理制度，重视信息安全，制定严格的安全管理制度。通过制定详细安全管理制度，确保信息系统的安全、合规运行，防范潜在的安全风险。

6 需求管理体系

6.1 需求管理概述

需求管理是证券期货业研发运营一体化体系的核心组成部分，涉及需求的识别、分析、规划、跟踪、实施和评估的全过程。其核心目标是确保需求被及时准确理解和满足，同时符合合规要求，促进团队沟通与协作，推动组织持续增长。需求管理涵盖需求的全生命周期，与持续交付、测试、安全和运营等关键领域紧密协同，支持业务创新，提升效率，促进数字化转型。它的重要性体现在确保合规性、提升市场适应性和客户满意度，以及构建风险防控体系。成功实施需求管理需遵循以客户为中心、合规先行、风险意识、透明沟通、持续改进、跨部门协作、需求条目化和优先级排序、标准化和系统化方法，以及数据驱动决策等基本原则。

6.2 需求管理体系组织保障

6.2.1 需求管理组织架构设计

高效的需求管理组织架构是确保需求能够有效收集、分析、实施和跟踪的关键。在证券期货业进行需求管理组织架构设计时，需要考虑行业的特殊性、监管要求以及业务复杂性，确保架构既能满足严格的合规要求，又能快速响应市场变化和技术创新。

在设计需求管理组织架构时，包括但不限于产品经理、需求分析经理、合规风控经理和项目经理 4 类角色，更好地助力实现需求的高效收集、深入分析和严格跟踪，各角色的职责描述如下：

a）产品经理：搜集需求，细化产品特性和用户体验流程；

b）需求分析经理：深入分析需求，评估技术可行性和合规性；

c）合规风控经理：评估潜在风险和合规性，确保符合监管要求；

d）项目经理：监控项目时间线和预算，协调团队沟通，管理风险。

在上述组织架构的基础上，需求的实施和维护同样需要一个跨职能团队的紧密协作。在需求管理架构中，开发、系统集成测试（SIT）、用户验收测试（UAT）和运维等角色通过紧密协作，确保产品从设计到上线的每个环节都高效、稳定且符合用户和监管的需求。

6.2.2 需求管理流程机制

基于 6.2.1 的需求管理组织架构设计，还需要定义清楚需求管理流程机制，这对于确保需求管理的效率和质量至关重要，可参照图 2 的流程框架设立需求管理流程机制。

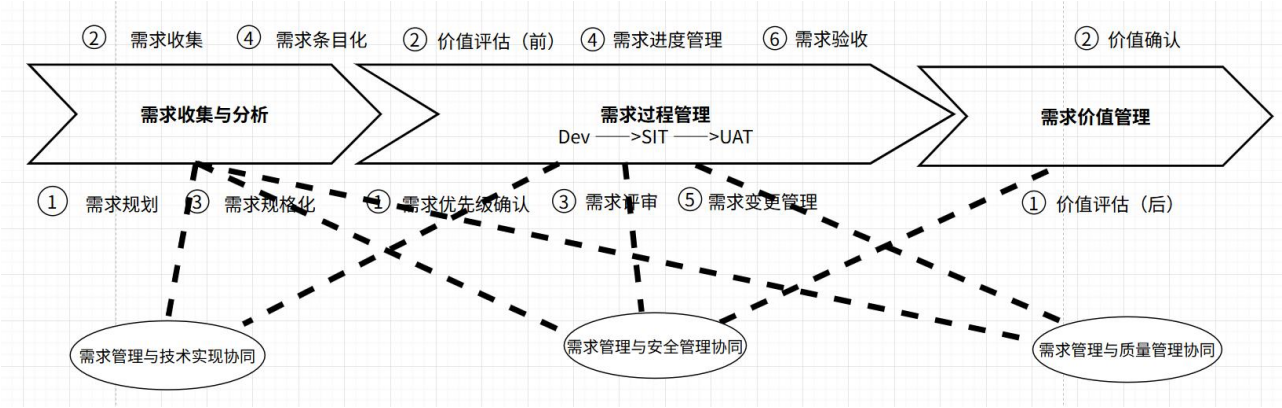

图 2　需求管理流程机制

规范的需求管理流程机制对提升需求管理效率、识别防范潜在风险具有深远意义。全面捕捉用户需求并准确理解，结合合规风控团队的早期介入，保障需求符合法律法规，降低风险。明确的角色分配和责任划分促进跨团队合作，整合不同视角，为决策者提供关键信息，支持明智的产品开发决策。

6.3　需求收集与分析

6.3.1　产品规划

产品规划是需求管理的源头，它结合了组织战略和市场洞察。在证券期货业，产品规划宜采取以下策略：

a）市场研究与分析：深入研究行业趋势，理解市场动态和消费者行为。分析不同客户群体的需求，包括个人和机构投资者；

b）业务战略制定：明确业务方向和产品策略，考虑新交易品种或服务。通过技术提升业务效率，降低成本；

c）合规与风险管理：预判监管政策，确保产品规划合规。融入风险管理，加强信息安全和合规交易流程；

d）产品概念与定位：根据市场和公司优势，构思产品概念，明确特色和优势。制定产品路线图，规划功能迭代和演进方向。

这些策略有助于企业确定产品规划，以适应市场变化，满足用户需求，并为后续的需求收集与分析提供指导，确保需求管理流程的连贯性和执行的精准度。

6.3.2　需求来源与收集

6.3.2.1　需求来源分析

需求管理的有效性依赖于对需求的细致分类和明确标记。需求可分为 4 大类：业务需求、技术需求、监管需求和运维需求。每类需求在提交时都宜标记，以快速识别其来源和影响范围，促进资源合理分配和项目优先级确定。

a）业务需求：源于市场和用户，关注产品功能和服务改进。通常由业务团队、客户反馈和市场调研提出，与商业目标和用户满意度直接相关。通过市场调查、用户访谈和数据分析收集，结合行业趋势判断；

b）技术需求：由业务发展、技术更新和系统性能优化驱动。技术团队需与业务团队合作，评估技术风险和解决方案可行性。通过技术调研、竞品分析和趋势预测识别需求；

c）监管需求：来自政府监管机构的法规和标准。企业需建立监管动态监测机制，快速响应监管要求。跨部门小组负责将监管要求转化为操作步骤并监督执行；

d）运维需求：关注系统的稳定性、安全性和可维护性。这些需求通常由运维团队提出，以确保系统的持续运行和优化。

通过需求来源的分类分析，旨在为证券期货业的需求分类管理提供参考。这将促进需求的高效识别、精准管理和及时响应，从而保障企业运营的合规性、增强市场适应能力，并推动技术革新和进步。

6.3.2.2 合规、安全风险评估

在证券期货业，需求管理的安全合规评估对企业稳健运营至关重要。需求收集时，必须考虑合规性和安全性。合规风险评估通过法规检索、咨询和法务审核，确保需求符合法规和标准。安全风险评估关注数据和系统安全，在需求分析阶段识别风险，使用工具如风险矩阵和威胁建模进行评估。业务部门提出新需求前，需经内部讨论和合规风控团队审查，确保合规性。合规部门出具确认文件，作为 IT 部门受理需求的前提。IT 部门再次审查合规性，确保流程合规。需要保持合规性和风险控制的需求为如下 3 类：

a）业务需求：来自市场和用户，如交易服务和客户管理。合规风控团队确保这些需求合规，控制业务风险；

b）技术需求：涉及系统升级和性能改进。合规风控团队评估技术变更对安全性和合规性的影响；

c）监管需求：关联外部监管变动，要求金融机构调整。合规风控团队解读监管要求，确保及时实施并监控风险。

6.3.3 需求规格化

需求规格化是确保研发团队和业务团队对需求有共同理解的关键步骤。在这一过程中，需求不仅被详细描述，而且与业务模型紧密关联，以确保需求的实现能够直接支持和推动业务目标的达成。一个完善的需求规格宜包括以下要素：

a）需求描述：明确描述需求的功能和目的；

b）业务价值：解释需求对业务目标的贡献；

c）业务模型描述：详细阐述需求如何与企业的业务模型相结合，包括业务流程、业务规则、业务实体及其相互关系。这一描述宜足够具体，以便于需求可以直接映射到业务模型中，形成规格化的需求；

d）用户故事/用例：从用户角度描述需求的使用场景。以简洁的叙述形式描述用户的需求和期望。用户故事通常遵循“作为[某种类型的用户]，我希望[某种功能]，以便于[某种收益]”的模式。这种方法有助于更好地理解用户的真实需求，并促进需求的快速迭代和研发；

e）功能需求：列出需求具备的功能点；

f）性能需求：定义性能标准，如响应时间和并发用户数；

g）安全需求：描述安全标准和隐私保护措施；

h）合规性需求：确保需求符合法律法规和行业标准；

i）原型设计：将需求可视化，通过创建产品或功能的初步设计或模型，帮助团队和用户更直观地理解和探讨需求。原型可以是草图、线框图、交互式模型等不同形式；

j）技术限制：识别影响需求实现的技术限制或依赖；

k）完成标准：定义需求满足的标准和完成条件。

l）非功能性需求：明确系统在性能、安全性、可靠性、可扩展性等方面的需求，包括但不限于：

系统可用性要求、系统性能要求、灾难恢复和业务连续性需求、系统兼容性要求等。

在证券期货业，需求规格化还需考虑以下特定要素：

a）风险管理：评估和管理潜在金融风险；

b）监管遵从性：明确满足监管机构要求的策略；

c）数据保护：制定严格的数据保护和隐私策略；

d）市场数据准确性和实时性：确保数据的准确性和实时更新；

e）系统稳定性和可靠性：稳定性和可靠性的要求描述；

f）审计跟踪：记录操作的审计跟踪日志；

g）灾难恢复和业务连续性：制定相关计划和策略。

如果需求没有达到准入标准，宜有一个完善的改进过程：

a）需求审查：定期审查需求文档，确保其完整性和准确性；

b）反馈循环：建立反馈机制，收集来自各方的意见和建议；

c）需求优先级调整：根据业务价值和资源情况，调整需求优先级；

d）风险评估更新：持续评估需求实施过程中的风险，并更新管理策略；

e）合规性检查：确保需求持续符合最新的法律法规和行业标准；

f）技术可行性分析：定期评估技术限制，寻找解决方案或替代方案；

g）持续沟通：保持与所有利益相关者的沟通，确保需求的透明度和可理解性；

h）质量保证：通过持续的质量保证活动，确保需求规格的高标准。

通过这一过程，可以确保需求在进入开发流程之前经过充分考虑和详细规划，提高项目成功率，降低返工风险，并最终交付高质量的产品。

6.3.4 需求条目化

需求条目化是一种需求管理策略，它通过为需求设置标签和进行层次划分，以实现用户故事的有效实施，帮助解决需求优先级不明确、需求跟踪困难、需求与实际开发脱节等痛点问题。在这种策略下，标签的设定有助于从多个角度全面审视需求之间的相互联系，层次划分则有助于深入追踪需求的具体情况。此外，该策略还包括对用例和用户故事进行条目化的列表管理，这有助于直接从这些条目中分解出具体的开发任务。

通过需求条目化与拆解，需求得以细化和清晰化，有助于团队更好地理解每个需求的组成部分，并为开发和测试工作提供明确的指导。这不仅提高了项目管理的效率，还确保了需求能够按照既定的优先级和标准得到实现，主要包括如下措施：

a）需求归类与整理：将收集到的需求进行分类整理，根据需求的重要性、紧急性、影响范围和资源可用性对需求进行优先级排序。对于每一个需求，都要明确其业务背景、目标用户群体、预期目标和衡量标准。并根据需求的性质和来源，对其进行分类（如功能性、非功能性、用户界面（UI）等），添加标签以便于跟踪和过滤；

b）需求条目清单：创建需求条目清单，每个需求条目都宜有一个唯一的需求标识码，包括需求名称、需求所属系统、需求详情、优先级、相关方、预期结果和验收标准等。这样可以让每个需求成为一个独立的单元，便于跟踪和管理；

c）需求拆分：对于大型或复杂的需求，将其细分为一系列较小、可独立实现的功能模块或用户故事，这有助于团队更好地理解每个需求的组成部分，并为开发和测试工作提供明确的指导。

6.3.5 需求资产沉淀管理

需求资产沉淀管理是组织内部对研发过程中产生的需求相关文档、知识、经验和技术成果进行系统化收集、标准化处理、安全存储、共享利用以及持续维护的过程。建议采取一系列措施来管理

和沉淀研发过程中产生的需求资产：

a）需求资产的识别与收集：公司在需求启动初期定义了需求资产的范围，包括市场分析、用户故事、规格说明等，并使用统一模板和工具记录；

b）资产的标准化与存储：需求资产按公司标准命名分类，存储于中央知识库，通过权限管理确保信息安全，便于索引和授权访问；

c）知识共享与培训：通过内部培训和知识库平台，鼓励员工学习并应用需求资产，提高培训效率，促进知识共享和创新；

d）持续的需求资产维护与合规性：指定团队负责需求资产的定期审查更新，确保与业务需求和技术趋势一致，并由合规部门确保遵守法规标准。

6.4 需求过程管理

6.4.1 需求过程管理概述

需求的过程管理对于确保项目目标的达成至关重要，同时也是推动项目高效执行的关键。在开展需求的过程管理时，建议关注以下五个组成部分：优先级确认、需求评审、进度管理、需求变更和需求验收。

6.4.2 优先级确认

需求优先级是指对收集到的需求进行评估、排序以确定需求的相对重要性，需求优先级确认是需求过程管理的首要步骤，建议由需求提出方、需求关联方、需求实施方一起开展。通常可基于需求的业务分类、紧迫性、重要性、资源限制以及对业务目标的贡献程度等维度进行评估，可以采取定性的评估方法，根据这些维度的定性描述确认需求优先级。也可以采取定量的评估方法，对这些维度赋予不同权重，然后使用加权法计算每个需求得分，按得分排序后确认需求优先级。优先级确认建议遵循和需求所属的业务分类相匹配的确认机制，具体分类与策略如下：

a）对于监管合规类需求可以从合规重要性和紧迫性考虑，建议给予较高优先级，及时投入 IT 资源进行排期开发；

b）对于技术类需求可以从 IT 资源的整体排期出发，统筹考虑技术改造、技术债务等方面的投入产出情况确认优先级；

c）对于业务类需求，可以依据需求的业务分类和业务目标，设立与业务目标一致的优先级，比如交易的需求、权益研究的需求所对应的业务目标是最高优先级的话，那么这些业务需求宜确认为最高优先级。如部分公司在年初依据公司年度业务目标规划年度项目，年度项目中的需求优先级宜与之匹配。

6.4.3 需求评审

需求评审是需求提出方、需求实施方等多方共同对符合需求评审条件的需求进行检查、分析和确认的活动。需求评审的准入是必须满足需求规格化中的要素完整性才能进入评审。需求评审可以采用需求评审会的形式，组织需求的相关专家对需求进行充分交流、讨论、分析，确保即将投入实施的需求的正确性、完整性、一致性、可行性、可测试性、合规性等，降低后续的研发成本和风险。需求评审时建议对需求实现的工作量进行预估，作为评审结果的一部分。

需求评审要包含合规审核环节以确保业务合规性。合规审核的方式建议与公司的监察稽核机制相匹配。例如，有的公司将需求统一由合规部门进行业务合规评审，有的公司将需求的合规评审由相应职能部门里的专职或兼职的合规人员进行。需求评审的内容可依据需求的业务分类、业务复杂度、技术复杂度、影响范围等进行裁剪。

对监管合规类需求，要重点关注监管的核心诉求，突出强调业务合规性。比如要求信息系统的运行日志记录的完整性、可审计性，可能需要由合规部的人员评估日志完整性、可审计性的具体要求和检查方式，再由信息技术部的人员评估对信息系统的日志进行统一收集和标准化处理的影响范围和复杂度。

对技术类需求，主要关注可行性、合理性、必要性。可以从所涉及的数据、系统、架构、网络等方面进行评估。比如有一项技术改造的要求是从数据库直连取数改为从数据服务的接口取数，则可以按实际情况引入应用架构设计评审、网络架构设计评审的内容。

对于业务类需求，要重点考虑业务复杂度和重要性。交易、行情、风控等必须确保业务连续性、稳定性，办公场景的业务需求相对要求较低。比如交易条线的业务需求可能涉及在投资交易系统中处理敏感数据，在需求评审时建议由合规人员给出评审意见，安全人员在需求评审时要给出信息安全评审意见，项目团队在需求评审时要结合交易业务模型，分析意向、询价、委托、指令下达、指令分发、执行、风控、成交、清算结算等各环节的影响，确保需求交付后达成预期效果、不影响交易业务的连续性、稳定性。

此外，为了确保网页（Web）应用和移动端应用在多样化的设备上均能提供一致且友好的用户体验，建议增加对视觉交互设计和设备兼容性的评审。这包括对 UI 的视觉元素、布局适应性、交互逻辑以及跨平台一致性的细致检查。评审团队将评估应用在不同操作系统、浏览器和设备上的表现，确保设计满足广泛的用户需求和提供无缝的交互体验。通过这一过程，我们能够提前识别并解决可能影响用户体验的兼容性问题，从而在产品研发的早期阶段就为提供优质的用户体验打下坚实基础。

6.4.4 进度管理

进度管理是为了确保需求按照预定的时间计划顺利推进和交付，开展进度管理可分为以下 3 个过程：

a）估算需求的工期。建议由经验丰富的人员来估算，可以单人估算也可以多人估算。估算出较大的工期时，建议将需求拆分为子需求再估算每个子需求工期。估算工期可采用历史经验、专家判断、三点估算等技术；需求工期的估算范围宜涵盖完整的研发运营一体化周期，包括开发、测试（SIT/UAT）、部署和技术运营（如监控配置、运维手册编写等）全流程。由产品经理牵头，技术经理配合，共同确定估算范围。对于涉及生产环境变更的需求，必须包含技术运营环节的工期预估；

b）制定整体进度计划。建议先通过专家评估、项目计划会等方式形成进度基准，这有助于为项目团队提供一个衡量进度绩效的标准。再依据需求优先级确定每个需求的计划开始时间、计划完成时间、里程碑节点、前置依赖等，最终的进度计划可以通过图表形式呈现，比如甘特图、关键路径图、里程碑计划表等，这些图表的创建和维护通常可以借助需求管理平台来实现。整体进度计划由产品经理主导制定，技术经理负责提供技术实施层面的时间评估。在证券期货业，进度计划需要特别考虑监管报送时间窗口、业务周期特点、技术运营准备期（如灾备演练窗口）等因素。制定整体进度计划时，可以包含项目团队和其他需求相关方的进度沟通机制，包括沟通频率、沟通内容、沟通形式：比如每两周开展一次邮件进度沟通，沟通内容是汇总项目里的每个需求的实际进度、进度偏离情况，依据需求进度估算项目进度，分析潜在的进度延期风险；

c）实施进度控制。包括设立进度控制点、收集实际数据、分析进度偏差、制定调整方案、实施调整措施、持续监控与反馈。在分析进度偏差时，可以将需求的实际进度数据与进度计划中的进度基准进行比较，通过比较计划进度和实际进度的差异，发现问题所在并确定偏差的原因，可以借助需求管理平台来实现进度偏差提醒，进度数据展示。一旦发现进度偏差，项目团队需要开展相应的进度调整措施。比如重新安排需求优先级、调整资源分配、增加人力投

入或延迟项目里程碑等，调整措施建议综合考虑项目的整体目标、资源可用性和时间约束等因素。

6.4.5 需求变更管理

需求变更管理是确保需求能够适应变化并持续推进的关键过程。需求变更通常会影响需求的优先级、进度计划、目标交付物、测试计划等，因此需要建立变更管理机制应对需求变更的情况。合理地变更管理机制建议包含以下组成部分：

a）明确变更申请准入要求。需求的变更申请需要满足需求规格化的要求，确保要素的完备性，建议包含变更的原因，变更的必要性，变更的范围，变更的期望结果。要避免出现一句话的变更申请，比如对于交易、行情等业务相关的需求变更，要素的完备性通常比办公、销售等业务的需求变更更加严格；

b）变更的合规性审核。需求变更要根据所需的业务分类、优先级、风险等级等要素确认相匹配的合规管理过程。比如业务类的需求变更，合规人员的审核内容可以与实际的行情、交易、风控等业务条线的运行方式相结合，判断业务合规风险。比如监管合规类的需求变更，合规人员可以根据监管发文单位、监管要求等进行审核。比如技术类的需求变更，合规人员可以依据影响系统的重要性、影响系统或组件的范围、持续时间、数据敏感性等维度进行审核；

c）清晰地变更管理流程。可以包括 5 个步骤：发起变更申请、评估变更内容、审批变更、实施与控制变更、变更后评估与验收。

6.4.6 需求验收

需求验收是需求在经过研发、部署、测试等技术实现过程后的关键步骤，目的是确保需求的交付物符合预定的需求规格，并且得到了需求相关方的认可。需求验收的主要过程如下：

a）确定验收标准和范围。在前期的需求评审环节，宜明确定义需求验收的标准、验收范围和验收人员，与需求提出方、需求实施方等需求相关方协商一致，并记录在项目文档中；

b）实施需求验收工作。只有经过验收的需求才能进行生产上线或发布。验收人员要根据前期确定的验收标准和范围进行验收。例如可以评审需求的测试计划和测试报告，开展 UAT 测试，检查需求的交付物是否符合预期的质量标准、是否实现了预期的功能；对于关键业务需求或高风险变更，在发布上线后宜进行生产环境验收测试；

c）形成验收报告。完成需求验收后，由验收人员编写一份需求验收报告，记录验收过程中的结果。最后还要请需求相关方对验收报告进行评审和确认。

需求验收建议依据需求所属的业务分类、重要性的不同而采用不同的验收标准。业务类需求的验收，在验收报告中可能要重点关注需求交付成果对业务连续性的影响、对业务运作效率的影响。监管合规类需求的验收，在验收报告中可能要重点关注是否切实满足了监管发文中的各项要点。技术类需求的验收，在验收报告中可能要关注需求交付物所影响的系统或组件，网络权限、数据权限是否符合预期。

6.5 需求的价值管理

6.5.1 需求价值管理概述

需求价值管理是对研发运营一体化各环节持续开展需求价值评估、需求价值确认的活动，是一项指导、促进需求的高效开展与交付，支撑业务价值的持续提升的活动。目的是确保需求交付物能够为客户和需求相关方带来预期业务价值。

6.5.2 整体业务价值管理

整体业务价值管理是一项依据公司的整体业务目标分解成各业务条线的业务目标后，对每个需求建立关联业务目标、分析需求的业务目标贡献度的过程。

需求价值可以体现在产品或服务的功能、性能、质量提升，业务运行成本的降低，组织运作效率的提高，以及对用户体验、市场竞争力等方面的积极影响。建议项目团队要和需求提出方进行详细地沟通，共同确认需求的所属业务条线、业务分类、业务目标、业务目标贡献度，从而确保需求在交付后可以实现预期的业务价值。

6.5.3 需求价值管理流程

需求价值管理流程主要包含两个方面：

a）需求价值评估。每个公司可以形成与各业务条线实际情况一致的、清晰的、可量化的、分级分类的价值评估方法与模型。建议在需求评审环节开展需求价值评估，评估过程的参与人员要包含需求提出方或需求所属业务条线的相关人员。可以从提升运营效率、提高合规水平、提高客户满意度、降低操作风险、降低运维成本等方面评估需求价值；

b）需求价值确认。在需求交付后的一段时间内，建议项目团队和需求提出方共同完成需求价值确认。项目团队可以收集并提供需求交付上线后的用户访问数据（比如页面使用频次、停留时长等），业务执行时长、业务成功率等客观数据，需求提出方可以依据这些客观数据和主观感受，判断需求对业务的实际贡献情况，比如降低操作耗时，降低操作风险，提高用户转化率等。对于产生负面价值的情况，可以将其纳入下一轮需求管理的流程中，从而不断优化产品或服务。

6.5.4 需求价值管理的持续改进和优化

为了确保需求可持续地满足业务目标，同时实现研发资源的最优配置，需求价值管理需要不断地进行持续改进和优化。为此，建议围绕需求价值管理的各个环节，建立一个可持续的需求价值评估和优化机制。在价值评估时，可以通过公司内部在线协同的知识库，维护价值评估方法和模型，并对价值评估过程进行完整记录，可以收集需求相关方对价值评估的意见反馈。在价值确认时，可以持续引入自动化、可视化的客观数据采集工具，辅助需求提出方进行更高效、更合理的价值确认。

在需求评审中进行价值评估时，需求提出方在需求评审会上围绕需求价值、期望和目标进行论述。项目团队成员在需求评审会上进行充分沟通讨论，确保理解需求价值，并确定价值确认的数据要素，如设立业务价值类的监控指标：总交易数、异常交易拦截数、异常交易拦截率等，设立成本类的监控指标：工时投入，从而可以在需求的技术实现过程中，要求项目团队及时登记这个需求的工时投入情况，以此判断工时投入是否有偏离。在需求交付上线后，通过需求价值复盘会的形式，分析这些指标数据，请相关业务部门的人员依据价值评价方法确认这些需求价值。

6.6 需求管理工具链与平台

6.6.1 需求管理工具链与平台概述

需求管理体系涵盖需求的整个生命周期，从需求收集、分析、文档化，到需求追踪、变更管理、验证与审批，常用的工具链与平台都具备相关功能支持这些流程。

6.6.2 需求收集与线上化

需求收集与线上化相关平台宜包含如下功能：

a）需求收集：项目启动阶段，业务分析师通过需求管理工具收集各部门的业务需求。利用企业

内部需求管理工具创建需求文档，详细记录每个需求的背景、目标和验收标准；

b）文档化：将收集到的需求按照功能模块进行分类，使用企业内部需求管理工具的模板统一格式，确保文档的可读性和一致性；

c）线上化：需求的收集录入线上化需求管理系统；

d）集中记录：所有需求集中在需求管理工具中，便于管理和查找；

e）详细描述：通过文档管理工具详细记录需求细节，确保开发团队理解准确；

f）协作编辑：团队成员可以实时协作编辑需求文档，提升沟通效率。

6.6.3 需求分析与优先级设定

需求分析与优先级设定相关平台宜包含如下功能：

a）需求分析：项目经理和技术团队在需求管理工具中对收集到的需求进行分析，评估其可行性和影响；

b）优先级设定：根据业务价值、紧急程度和资源可用性，使用需求管理工具的优先级功能对需求进行排序，确保高优先级需求优先实施；

c）可视化管理：通过看板或甘特图展示需求优先级和进度；

d）关联分析：将需求与业务目标、功能模块关联，便于评估其重要性。可以通过需求影响地图能力进行辅助分析；

e）协作评审：团队成员可以在需求管理系统中评论和讨论需求，达成共识。

6.6.4 需求审批与确认

需求审批与确认相关平台宜包含如下功能：

a）需求评审：在需求管理系统中组织跨部门的需求评审会议，业务、技术和合规团队共同审查需求的完整性和可行性；

b）审批流程：通过需求管理系统的工作流配置，设定需求审批流程，确保每个需求在开发前经过必要的审批。对于监管合规类需求，需提级管理，确保审批流程中覆盖合规性检查节点；

c）签署确认：使用需求管理系统中的任务流转与任务确认功能，进行需求确认，也可以集成电子签署工具，对关键需求进行正式签署，确保责任明确；

d）工作流管理：配置多阶段审批流程，确保需求经过各级审核；

e）电子签署：集成电子签署工具，提升审批效率和正式性；

f）审计追踪：记录审批过程中的所有操作，满足合规性要求。

6.6.5 需求追踪与变更管理

需求追踪与变更管理相关平台宜包含如下功能：

a）需求状态追踪与管理：根据需求全生命周期定义需求状态，依据状态变化能够及时追踪需求进度；

b）需求追踪：在需求管理工具中创建需求与开发任务的关联，确保每个需求都有对应的代码实现和测试用例。通过需求管理工具与版本控制系统相结合，实现需求在不同形态下的双向追踪；

c）变更管理：当业务需求发生变化时，通过需求管理系统提交变更请求，评估其影响并更新相关任务；

d）版本控制：使用管理代码版本，确保每个需求的变更都有代码版本对应，便于追溯；

e）双向追踪：需求与代码、测试用例的双向关联，确保需求的完整实现；

f）变更记录：记录需求变更的历史，便于回溯和审计；
g）影响分析：自动识别变更对其他需求和模块的影响，降低风险。

6.6.6 跨部门协作与沟通

跨部门协作与沟通相关平台宜包含如下功能：
a）信息共享：通过企业内部工具创建和分享需求文档、项目进展报告等，促进信息透明；
b）实时沟通：使用需求管理系统与即时通讯工具进行集成，进行实时有效的通知、沟通，快速解决需求相关的问题。可以在需求生命周期中的关键节点，设置关键角色进行通知，如：流转、审批、打回和异常状态等，按照角色、模板等进行通知；
c）会议与讨论：组织线上或线下会议，利用工具记录会议纪要和决策；
d）文档集中化：所有相关文档集中存储，方便团队成员查阅和更新；
e）即时通讯：实时交流功能，提升团队协作效率；
f）集成通知：将需求管理系统和版本控制系统等工具的通知集成到即时通讯工具中，及时获取项目动态。

7 技术实现体系

7.1 技术实现概述

技术实现部分是研发运营体系的重要一环，该部分关注基于各类需求进行快速、有序地推进技术实现。主要内容涵盖组织保障、研发迭代管理、研发过程管理和研发流水线管理等主要章节。

7.2 技术实现体系组织保障

7.2.1 组织保障

为保障研发运营一体化技术实现体系的有效实施，需建立完善的组织结构、明确的角色划分和在企业形成对应的流程保障机制，以确保各部门之间不同角色能够高效协同。组织保障宜包含以下三个方面：
a）关键角色：指参与产品技术实现的相关角色人员，技术实现主要角色包括：
——产品经理：负责收集和整理业务需求，与开发团队紧密合作，确保产品需求被准确理解和实现；
——项目经理：负责协调资源、管理进度、控制风险，确保项目按计划成功交付；
——技术经理：统筹技术资源、协调技术方向，确保项目技术目标和进度按计划实现；
——开发人员：根据产品需求进行具体开发工作，与产品经理和测试人员保持密切沟通；
——测试人员：在技术实现阶段，测试人员、产品经理和开发人员等角色进行沟通，准备测试用例。
b）组织结构：在产品技术实现对应的团队中需包含相应角色人员，同时根据项目规模，形成对应数量和不同角色人员比例的组织团队：
——角色构成：宜覆盖含产品经理、技术经理、开发人员等角色。不同角色之间宜有明确的责任边界，以便在出现问题时能够迅速定位并解决；
——人员比例：根据项目规模，不同团队形成不同角色比例的人员数量组织，对应角色人员数量需能够满足日常的快速迭代需要。
c）协同方式：
——定期会议：定期召开会议，共同讨论项目进度、问题和解决方案；

——信息共享：通过信息共享平台，确保所有团队成员能够实时获取项目相关信息以及研发迭代的进度。

7.2.2 流程保障

为确保研发过程顺畅地进行以及组织的高效运转，需要制定完善的流程保障措施。为了保障流程落地需要明确各个环节的职责和输出标准，建立流程监控和评估机制，以及提供必要的流程培训和支持。流程保障宜包含：

a）流程制定：制定详细的研发流程图，明确每个环节的任务、责任人、输入和输出；

b）流程落地：研发流程与管理系统融合，结合质量门禁措施，保障流程落地；

c）流程监控：对研发过程中的关键节点进行实时跟踪和监控，定期评估流程执行情况，及时发现问题并进行调整，确保流程的高效执行。

7.3 研发迭代管理

7.3.1 需求池管理

需求池是按照一定规则汇总记录产品（系统）相关的需求信息的地方，目的是确保需求被及时、完整、有序地描述、排序、推进等，实现对需求集中化、统一化和透明化的全生命周期管理。需求池管理包括需求收集、需求整理、需求评估、需求拆分和需求跟踪等内容，具体描述如下：

a）需求收集：需求收集是需求池管理的基础，用于收集来自不同来源（例如客户、相关方及其他团队）的需求，并按照统一的标准模板进行录入，以确保需求的完整性、一致性和可分析性。需求模板主要包括需求描述、需求背景、需求提出人、需求来源、需求领域、需求类型、需求价值及期望交付时间等要素；

b）需求整理：需求整理是对已收集到的需求，通过整理、分类、去重、合并以及进一步明确需求等操作，以确保需求的清晰度和可实现性；

c）需求评估：需求评估是对需求进行深入分析，评估其重要性、复杂性、紧急性和可行性等，根据需求的价值以及评估结果，对需求进行优先级排序，确定开发的先后顺序；

d）需求拆分：需求拆分是将整体需求分解成更小、更具体的需求单元，每个需求单元都有明确的定义和范围。需求拆分有助于降低需求复杂度、提高可追踪性，便于尽早实现和交付。需求拆分可以按照产品/系统、时间、层次或模块等维度进行拆分，并对拆分后的需求规模进行初步估算，以便为后续迭代规划提供依据；

e）需求跟踪：需求跟踪是对需求进行跟踪和管理，透明化其状态，及时发现和解决问题，以确保需求在整个产品生命周期中得到适当的关注、更新和管理，以避免遗漏、误解或重复处理。

7.3.2 迭代管理

7.3.2.1 迭代管理概述

迭代管理是一种敏捷开发方法，将软件研发过程划分为若干迭代周期且每个迭代周期都有需求分析、设计、编码、测试和部署等活动。迭代管理的目的是以增量的方式构建软件，使团队能够快速灵活地响应变化、及时修复缺陷、尽早地交付可用的软件版本，提高软件质量和稳定性。需要说明的是，迭代管理是可选的，是否设置迭代，根据项目类型自行选用。

7.3.2.2 迭代规划

迭代规划是迭代管理中的一个关键活动，它确保在每个迭代中能够交付有价值的需求，并确保

迭代能够持续、稳定地推进。以下是迭代规划活动的主要内容：

a）确定迭代周期和迭代目标。迭代周期通常是固定的（如两周或一个月）。迭代目标宜明确且可衡量，并与项目目标一致；

b）从需求池中选择需求。根据需求的优先级、业务价值以及团队的能力，从需求池中挑选要在当前迭代中完成的需求。优先选择对用户最有价值、最能够提升产品功能或用户体验的需求。除了新需求，迭代中还宜包括需在本迭代中修复的已知缺陷，以及其他任务；

c）评估需求风险和工作量。对每个选定的需求进行风险评估，识别可能的技术难题、依赖关系或其他潜在问题，并对每个需求进行任务拆分和工作量预估，确保迭代目标的合理性和可行性；

d）资源分配。根据团队成员的经验和能力，以及任务工作量预估情况，为每个任务分配适当的成员。确保每个团队成员都有明确的任务和责任，并且工作量相对均衡；

e）制定迭代计划和日程表。根据需求优先级和依赖关系，制定迭代计划和日程表，加强团队成员之间的沟通协作，以便有效地完成迭代周期的工作。

7.3.2.3 迭代执行与监控

迭代执行和监控是迭代管理中至关重要的一环，目的是更好地掌握进展情况、及时发现和解决问题、加强团队成员之间的沟通和协作、及时响应变化和风险、提高软件开发的质量和效率。以下是迭代执行与监控活动的主要内容：

a）任务执行与监控：团队成员按照迭代计划执行任务，并及时更新任务状态，确保团队成员和管理层了解项目的最新状态。团队监控任务进展情况，及时调整计划和资源，以确保任务顺利完成；

b）定期会议：团队召开定期会议（如每日站会）来同步工作进展，及时识别潜在风险并讨论如何消除风险；

c）风险和问题管理：团队识别、记录及管理风险和问题。对风险进行分类和评估，确定优先级并制定缓解措施，并在执行过程中加以监控和调整。对迭代过程中发现的问题进行跟踪，以确保问题及时解决；

d）迭代计划监控：团队监督迭代计划执行情况，识别偏差并分析原因，并根据实际情况及时调整迭代计划，以确保能够按时交付需求并满足质量要求，并向相关利益相关方定期进行进度反馈。

7.3.2.4 迭代总结与回顾

迭代总结与回顾是迭代管理中的重要环节，旨在帮助团队不断地提高工作效率和质量，不断完善软件研发过程，同时增强团队的凝聚力和协作能力。

在迭代结束后，团队对迭代进行全面的工作总结，分享成果，并回顾团队在迭代中的表现，识别优点和问题，并对问题进行深入分析，制定相应的改进措施，并计划在未来的迭代中采取相应的行动。

7.4 研发过程管理

7.4.1 研发过程管理概述

为了有组织、有规划地推进各项研发活动，确保项目按时、高质量地完成，需要在产品的研发过程中，通过科学、有效的管理手段和技术方法进行研发过程管理。在证券期货业的产品研发过程中，研发过程管理关键步骤主要包括系统架构设计、接口设计与管理、代码实现、代码管理、代码编译、制品管理、研发配置管理和研发提测等多个部分。

7.4.2 系统架构设计

在系统阶段，为了更好地设计系统的整体结构、模块之间的交互方式以及系统的可扩展性、可维护性等关键特性，在系统架构设计的过程中，基于企业规范及流程对相应需求进行业务分析与技术架构设计，可参考如下：

a）系统架构设计规范：在系统架构设计过程中，基于架构设计原则与公司实际情况，形成公司内部架构设计规范；

b）系统架构方案评审：根据业务需求形成的架构设计方案需经过架构评审组评审，评审形成的架构设计方案形成文档后进行管理。

7.4.3 接口设计与管理

接口设计与管理贯穿整个产品研发过程，为对接口进行全生命周期管理，在接口设计与管理部分，可参考如下：

a）接口设计：确保不同软件或系统组件之间能够顺畅通信和交互的关键环节。在设计过程中，需要遵循一定的规范和标准，以确保接口的通用、稳定和安全性；

b）接口管理：确保接口设计得到有效实施和持续维护的过程。在接口管理的过程中，需要基于接口工具能力对接口的全生命周期进行有效管理和跟踪。

7.4.4 代码实现

代码实现阶段，为了加快编码速度和统一编码规范，可参考如下：

a）代码实现：在代码的编写过程中，可以借助编码工具以及企业内部编码规范编写代码，达到实现目标需求的目的。此外，可以结合工具实现代码管理仓库和需求管理工具打通，实现需求与代码关联；

b）辅助编程：在代码实现过程中，可利用一些辅助编程工具、库或者已有的代码片段来加速和简化编程的过程。在辅助编程工具的选择上，需满足自动补全、调试、响应速度快等基础要求。

7.4.5 代码管理

在软件开发中扮演着至关重要的角色，对软件开发过程中产生的代码进行有效的管理和维护，确保软件开发的顺利进行和代码质量的高度稳定。代码管理包括如下五方面：

a）分支管理：通过划分不同的分支来管理开发任务、特性和修复工作，在分支管理上，需要依据项目特性采用具体的分支管理策略。此外，在分支的命名上遵循对应的分支命名规范；

b）版本控制：代码管理需要可以追踪代码的变更历史，记录每个修改的细节。需要借助代码管理工具的能力识别每次提交的研发人员、修改内容等关键信息，同时对于每次代码提交信息宜采用代码提交规范；

c）代码审查：借助代码管理工具，团队成员之间互相进行代码审查来提高代码质量，并促进知识共享。此外，也可以通过将大语言模型（LLM）与版本控制系统相结合，可以实现智能化、自动化的代码评审功能；

d）代码安全与质量：对于增量提交的代码宜进行代码检测，通过后才能合并到主干分支；对于存量的代码定期进行检测，发现漏洞宜尽快修复。检测的过程可以采用静态扫描工具提高效率；

e）权限管理：为维护代码安全、项目完整性，对代码参与人员需要明确的角色划分，并对应具体代码仓库权限。代码权限管理需形成规范文档，并在研发部门内部推广。

7.4.6 代码编译

对于编译型语言，代码编译是技术实现的必须环节，随着项目代码规模的提升，会导致编译速度的下降，影响整体交付效率。为提升编译效率，可以采用增量编译、并行编译、编译依赖缓存、分拆代码库等方式提升编译效率，并将编译工具与流水线结合实现自动化的代码编译。

7.4.7 制品管理

混乱的制品管理对研发规范、环境稳定性和安全带来挑战，通过规范、有序的制品管理方式，能够为研发运营提供必要保障。制品包括编译后的二进制包及对应的源代码、依赖库、文档等，在制品与制品库管理过程中，宜包括如下六方面：

a）制品管理规范：根据实际情况制定制品管理规范并形成标准文档，为后续与制品对接的相关操作提供基础依据；

b）制品推拉方式管理：制品库支持多种类型制品上传和下载的方式；

c）制品晋级：在研发的过程中，涉及制品在多个环节的流转，需要基于公司的安全与合规策略形成对应的制品从左到右的晋级策略；

d）制品分类管理：在制品的管理过程中，依据制品来源和类型对制品进行分类。对于外部制品入库前需进行相应的安全检测、合规检测和评审；

e）制品库与流水线融合：制品库需与流水线相融合，能够自动地对制品库进行高效管理；

f）基础制品引入标准：对基础依赖制品形成明确的引入标准，如第三方依赖、基础镜像等底层常用制品，从合规、安全与效率等角度形成引入评估标准。

7.4.8 研发配置管理

研发配置管理是在研发过程中，对软件产品或系统的各种资源和配置项进行全面管理和控制，确保研发过程中的数据一致性、可追踪性和有效性。该部分配置管理主要侧重于应用配置的管理。配置管理工具可以落地配置管理思想，在具体能力上涵盖如下几个方面：

a）版本控制：对于研发应用配置的变更，通过配置管理工具，能够进行识别并追踪修改历史，同时需支持回滚到以前的版本；

b）应用配置：能够基于应用在不同的环境进行应用配置项的管理，使应用适应特定环境进行运行，通常包括配置数据库连接、设置系统参数、调整性能参数等。应用配置主要内容：

——配置文件管理：将所有配置项集中管理在配置文件中，避免硬编码在代码中。同时根据环境（开发、测试、生产等）使用不同的配置文件；

——配置项的命名与格式：采用统一的命名规范，确保配置项的名称清晰易懂；

——配置项的变更与审查：任何配置项的变更都宜经过审查，并将变更原因和结果记录到版本控制工具里；

——配置文件的备份与恢复：定期备份配置文件，以防止意外丢失。同时制定恢复策略，以便在配置文件损坏或丢失时能够快速回滚恢复。

7.4.9 研发提测

研发提测为研发阶段性成果完成的一个节点，在该部分需要基于公司的合规和研发流程特性，提交给测试具体的交付物，交付物宜符合如下两方面要求：

a）提测内容规范：根据公司合规、测试要求，对提测内容进行规范，明确描述变动的范围以及对应的需求项；

b）质量门禁：提测时需依据公司流程合规规范，对相应检测结果进行质量门禁结果检测，没有通过的不能发起提测。

7.5 研发流水线管理

7.5.1 研发流水线管理概述

流水线作为软件交付过程中的关键环节，通过自动化的方式将技术实现的各个阶段—包括代码编写、构建、测试、部署等紧密地串联起来，确保流程的高效、有序，减少了人为错误，提升交付质量。通过流水线的自动化功能，可以将开发过程中的变更内容，包括源代码、数据库脚本、系统配置、外部依赖库以及相关文档，转换成最终的交付成果，如软件制品包、容器镜像、环境配置文件、自动化部署脚本和质量测试报告等。此外，流水线内嵌了组织的质量控制标准和合规性要求，通过数字化手段对整个流程进行严格管控。流水线管理部分主要包括构建环境管理、流水线原子操作能力、流水线编排能力、性能和容量、安全与合规性以及反馈与审计等部分。

7.5.2 构建环境管理

7.5.2.1 构建环境管理概述

构建环境管理是实现整个持续集成服务的基础。环境管理主要涉及环境的制备、更新、伸缩等，并需要考虑环境的一致性、安全性、隔离性等问题，以确保构建环境的稳定性、安全性和可维护性，同时支持高效的构建流程。

7.5.2.2 环境制备

构建环境配置实现标准化，例如构建环境的操作系统、编译器、构建工具、依赖管理工具等选型和版本宜按照组织的需求形成支持矩阵，并控制各版本的碎片化倾向。

构建环境的环境变量宜预先按照基线设置，保证环境制备后即可使用，并使用受控的配置基线以保证安全性。

构建环境宜使用预设的专用账户，避免越权风险，如：运行构建的账号、下载代码的账号等。

此类环境宜独立于生产环境，以消除对生产环境产生的潜在风险。

建设基础环境模板，可快速克隆开发、测试所需环境，提高环境分配效率。

7.5.2.3 环境更新

构建环境宜定期或按需更新。宜定期检查如操作系统、基础镜像和依赖的安全性更新，并更新构建环境以包含这些更新。

宜具备构建环境回滚能力，确保新版本出现问题时可以及时回滚。

宜实现构建环境的版本化管理，以及滚动式更新、回滚等能力，如：采用基础设施即代码等技术。

7.5.2.4 扩缩容

宜实现构建资源快速、高效地分配与回收，以充分利用资源并快速高效地支撑构建服务。如搭建基于云原生的分布式构建集群来实现构建资源动态弹性扩缩容。

7.5.3 流水线原子操作能力

流水线的原子操作是指流水线完成某项特定任务的操作单元。根据交付的不同阶段的活动划分，有如下流水线原子操作能力：

a）开发阶段：代码检出、编译、代码扫描、单元测试、质量门禁、制品打包、容器镜像构建、容器镜像扫描、制品上传等活动；

b）测试阶段：流水线宜支持不同测试级别活动的自动化执行，主要包括单元测试、集成测试、系统测试、性能测试和安全测试等多种测试活动。流水线在不同测试级别活动执行时同时支持对应的关键操作，包括环境制备、制品部署、数据准备、用例执行、报告生成等，从而实现对应测试级别活动的高效执行。例如系统测试中，冒烟测试、集成测试、接口测试、性能和负载测试等可以通过流水线自动化执行；验收测试阶段，用户验收测试（UAT）可模拟最终用户的操作，确保产品满足用户需求；

c）发布阶段：包括制品晋级到发布仓、代码分支锁定和标签以及发布门禁报告等活动；

d）交付（部署上线）阶段：变更审批、预发布、正式上线、滚动上线、灰度发布等活动。可根据需要，通过一个或者多个流水线来完成一次交付过程，支持并行、串行上线部署方式，提高上线部署效率。

7.5.4 流水线编排能力

流水线编排是持续集成/持续交付（CI/CD）实践的核心，它定义了从代码提交到产品部署的整个流程。以下是流水线编排能力的一些关键要求：

a）可视化：提供图形化界面，展示流水线的结构和状态；

b）原子化：允许流水线由独立、可复用的原子组件构成，如一个典型的流水线可以分为阶段、任务及任务组、步骤等要素；

c）编排管理：支持以有向无环图（DAG）、阶段等方式表达流水线各个任务和步骤之间的前后依赖、分组、串行/并行执行等方式，以及有条件触发、暂停、终止，以实现各种流水线的运行需求，并实现流水线自身的高效运行；

d）参数管理：允许在任务、步骤之间传递参数，以及对步骤和任务在流水线执行时的参数化配置，以提升流水线的灵活性，提高流水线的复用度；

e）模板化能力：为应用、组件和典型流水线场景预先定义好流水线模板，简化创建和维护流水线的过程，并且能将组织级的必要管控措施（如质量门禁等）内置其中，保障措施的落地；

f）自定义脚本：支持自定义脚本编写，可根据项目要求，进行简单的逻辑处理，如目录迁移、接口刷新等操作；

g）触发方式：支持多种触发方式，如定时、手工触发等。

7.5.5 性能和容量

快速高效的流水线可以提供快速反馈、提高资源利用率、减少等待时间进而让研发人员更高效地工作，避免浪费，提高流水线用户的满意度。流水线宜从以下的角度来保证流水线的性能和容量达到设计预期：

a）并行处理：平台宜支持流水线之间、同一流水线内不同任务和步骤之间的并行执行；

b）多任务并发：针对同一流水线在短时间内重复执行的场景，可通过支持同一流水线并行执行来降低后续执行请求的等待时间；

c）弹性伸缩：平台宜实现弹性伸缩机制，根据负载自动调整资源；

d）缓存应用：合理使用缓存，如代码、依赖、镜像、编译等缓存的使用，以提高流水线速度，减少等待时长。

此外，还宜定期进行负载测试，确保流水线能够处理预期的负载，通过性能和容量监控来识别瓶颈。对于部分耗时长的流水线宜优化流水线编排、减少不必要步骤、提升步骤并行度、选择更优的工具或方案等方式降低执行耗时。还宜对流水线进行容量规划，以适应不断变化的需求。

7.5.6 安全与合规性

流水线安全与合规性能够避免环境稳定性和技术合规性可能面临的风险。通过安全与合规性的实施，并将对应权限控制与效能平台结合，能够对研发资源和流程实现更细粒度的控制。实施措施宜包含以下三点：

a）安全性：确保流水线的安全性，包括安全的凭据管理和执行过程敏感信息的安全性；

b）权限控制：按需实施访问控制，以保护流水线和相关资源的有效管控；

c）逐级验证：宜设置合理的验证流程和环境隔离，对变更进行充分的测试和验证，确保生产环境的安全稳定。

7.5.7 反馈与审计

通过反馈与审计构建流水线闭环，能够帮助研发人员更快发现构建中的问题，为产品交付提速和流水线优化提供帮助。具体措施宜包含以下三点：

a）日志记录：记录流水线执行的详细日志，便于问题追踪和调试；

b）通知机制：流水线宜以多种渠道、多种方式向相关人员发送通知，如邮件、聊天工具等。通知方式包括群发以及点对点等。此外，平台宜允许用户自行订阅；

c）审计跟踪：平台宜记录流水线的变更历史，包括谁、何时、如何进行了修改。平台宜能支持回溯流水线执行记录、门禁结果等数据且不可更改；

d）流水线执行统计：平台统计流水线的执行成功率，针对失败率较高的数据进行分析总结，提高构建编译效率；

e）大模型监控分析：基于大模型能力对流水线阻塞、异常记录进行根因分析和故障定位。

7.6 技术实现工具链与平台

7.6.1 代码管理

代码管理工具宜包含如下能力：

a）通过代码管理工具，能够实现对产生的代码进行维护，且有利于团队的协作；

b）选择代码管理工具时，除宜满足代码管理章节内容的必要要求外，还宜支持接口的形式对外扩展、代码的备份等能力。

7.6.2 制品管理

制品管理工具宜包含如下能力：

a）通过制品管理工具，能够实现制品在应用不同生命周期阶段的管理，能够满足制品管理章节要求的基本能力；

b）选择制品管理工具时，还需考虑自动备份、权限管控、版本记录和开放 API 的能力。

7.6.3 接口管理

接口管理工具宜包含如下能力：

a）接口管理贯穿整个应用研发过程，接口管理工具需满足接口管理章节的使用能力；

b）接口管理工具需支持模拟能力，提升研发过程中前后端研发的协作能力；

c）接口管理工具需支持快速导入现有 API 接口文档的能力，避免手动导入的维护成本；

d）接口管理工具还需考虑版本控制、权限管控和接口统计的能力。

7.6.4 流水线工具

流水线工具宜包含如下能力：

a）流水线工具需具备研发流水线管理章节要求的基本能力；
b）流水线工具需具备开放 API 接口或者用户自定义流水线阶段脚本的能力，通过灵活性提升流水线在不同场景下的适用性；
c）流水线需具备监控能力，能够监控流水线耗时和交付吞吐率；
d）在日常的运营上，流水线可支持模板管理、快速复用现有流水线等特性能力。

7.6.5 大模型工具能力

使用大模型工具提升研发效率，需要大模型对接研发多个阶段，在技术实现阶段主要考虑如下：
a）模型维护能力，能够对大模型工具的底层模型进行维护和管理；
b）支持多种接入方式，能够与技术实现阶段各工具快速对接；
c）模型工具运营分析，能够对大模型工具产生的结果进行价值分析。

7.6.6 应用配置参数管理

应用配置参数管理工具宜包含如下能力：
a）通过应用配置管理工具，能够集中维护应用系统参数配置项；
b）能够对不同环境的配置参数进行隔离管理，并能够实现权限管控；
c）支持参数的版本管理，并基于版本能够实现回滚。

8 质量管理体系

8.1 质量管理概述

质量管理贯穿于软件研发与运营的各个阶段，包括需求、开发、测试、部署和维护等阶段，通过实施标准、流程和策略来确保软件满足用户需求和预期质量要求。它是质量共建的过程，强调所有相关方在研发运营一体化中的共同参与和合作协同。质量管理依托组织支持，从测试赋能入手，共建质量保障体系，其作用域广泛，涉及软件研发全生命周期管理、跨部门协同、组织及能力建设等多个方面。质量管理策略旨在通过持续执行测试提供高效反馈，是软件交付流水线中的连续性质量保障活动，常见方法与实践包括测试左移、测试右移、持续测试和质量门禁等。

8.2 质量组织体系保障

8.2.1 质量组织体系保障概述

质量管理不仅包括技术层面的质量控制，还涵盖了管理、运营和用户体验等多个维度，其价值在于通过多方面的合作和共同努力，建立起一种质量文化，使得产品质量成为团队共同的责任和追求。质量组织保障是在众多团队的协同下，统一规划，形成可横向（跨项目）、可纵向（跨时间）的质量控制管理能力。

8.2.2 质量组织的定位

质量组织是围绕高质量软件交付目标构建的跨职能协作体系，涵盖开发团队、测试团队、产品团队等核心部门，通过整合组织资源、制定质量策略、落实管理、推进流程执行与持续改进，形成覆盖质量管理全周期的基础架构。质量组织是质量守护者，同时也是质量策略的制定者、质量保证的执行者、质量风险的监控者、质量改进的推动者以及跨部门的协作者。质量组织的作用不仅在于确保软件产品的质量，还在于提升开发效率、优化开发流程、增强客户信任与满意度，以及促进团队协作与沟通。

8.2.3 质量组织的特点

质量组织具有系统性，是一个从组织结构、方法、过程到资源的相互管理和作用的组合体。同时在设定的范围内宜具有唯一性，质量组织可根据合作方需求、项目背景、质量目标、职能分工拆解成具有唯一性的最小单位。质量组织还宜具有动态性，综合考虑利益、成本和风险，宜定期进行内部体系审核、管理评审和调整完善，进一步配合改进质量管理体系，使其持续有效最优地运行。这些特点共同构成了质量组织的基本框架和运行方式，为质量管理提供了坚实的基础。

为保障研发运营一体化质量管理体系的有效实施，需要建立完善的组织结构、明确的角色划分，确保各部门之间不同角色能够高效协同。基于全员参与质量的基本理念，参与质量保障的主要角色包括：

a）产品经理：负责需求质量保障，包括需求描述准确性、需求验收条件清晰化、参与需求验收等；

b）技术经理：负责技术实现的整体质量保障，包括功能质量、非功能质量等；

c）开发人员：负责具体代码实现的质量保障，包括开展单元测试、开发自测等；

d）测试人员：负责对功能、非功能进行质量验证；

e）质量保障人员：负责保障质量管理体系的准确有效实施，从流程机制上保障质量；

f）应用系统运维：负责保障生产运维的质量，包括部署后验证、应用状态监控等。

8.3 质量体系建设

8.3.1 质量管理组件

在质量体系建设中，宜基于用户需求、业务目标和行业规范明确质量目标和标准。通过制定清晰的质量指标和评估标准，为整个团队提供一个明确的方向和参考。同时建立包括质量计划、质量保证和质量控制等方面的管理体系，确保从需求分析、设计、编码、测试到维护每个阶段都有相应的质量管理措施。建立持续测试、持续改进过程，不断优化质量管理体系。通过引入质量管理工具和技术，提高质量管理的效率和准确性。

在研发运营一体化的建设中，建立测试需求组件、测试设计组件、测试执行组件、自动化测试组件、非功能测试组件、测试用例库、测试数据管理、测试环境管理等。包括使用线上化管理平台进行需求过程管理、使用测试设计规范分类管理测试策略、测试优先级等测试前置活动、使用测试管理平台进行测试过程管理、使用配置管理平台进行测试周边活动管理。

8.3.2 测试通用基础能力

测试通用基础能力是质量全面保障的基础，包括流程制度规范、测试人员管理、测试环境管理、持续测试平台建设、测试资产和数据管理 5 方面。

a）流程制度规范方面需要包括：测试流程与人员职责、测试用例设计规范、测试准入准出规范等；

b）测试人员管理方面需要包括：测试人员能力模型、测试人员培养机制等；

c）测试环境管理方面需要包括：独立完整的测试环境提供能力、测试人员可以方便地自助申请测试资源；

d）持续测试平台建设方面需要包括：测试用例管理、测试计划管理、自动化接口测试、自动化 UI 测试、性能测试、移动应用测试等平台；

e）测试资产和数据管理方面需要包括：各类测试资产的统一管理、测试数据生成、测试数据脱敏等能力。

8.3.3 测试左移

测试左移是为了提升研发和测试效率与质量，测试不断向左移动，向需求、研发等环节“左移”，参与需求评审以及内部需求版本管理、建立软件准入准出等测试管理流程规范等环节，测试左移通过前置自动化测试能力至需求与研发阶段，将“自动化”前置到系统测试阶段之前，对需求进行尽早测试，鼓励测试人员提前参与需求评审和开发设计评审等活动，将自动化测试能力（如接口自动化、UI 自动化、性能自动化）与流水线 CI/CD 集成。研发人员以极低的成本从研发运营一体化平台上获得自动化测试的相关能力。测试左移宜关注以下 3 方面：

a）在测试左移中，重点是对需求质量和开发质量两方面的把控，从而提高提测通过率、提高应用本身的质量；

b）在需求质量的把控方面，测试人员需参与需求评审，明确需求的验收条件，建立需求与测试用例的关联等；

c）在开发质量的把控方面，开发人员需通过单元测试及自测保障代码功能质量，通过代码评审与代码扫描等操作来保障代码符合规范。

8.3.4 测试右移

测试右移是为了不断提高交付质量和实现持续改进，测试不断向右移动，向运维运营等环节“右移”，及时获取线上问题和用户反馈，持续优化和改进产品。测试右移关注持续部署和发布，部署过程宜实现高度自动化，配置部署的流水线能够快速反馈部署的问题，去除手动配置的成本和风险。可以在不同环境（如：开发环境、测试环境、预发环境和生产环境等）分别部署流水线，并将测试检查内容纳入部署流水线门禁中，实现测试右移。

8.3.5 持续测试

持续测试是在整个软件研发生命周期中持续执行测试以提供有关被测系统质量高效反馈的过程，是软件交付流水线中的一种可以随时开展且具有连续性的测试活动。持续测试基于具体项目的流水线，拉通各角色的协作配合，提倡持续测试驱动开发，促进质量管理在各环节的实践落地，切实以质量为出发点渗透全员质量意识，实现贯穿软件交付过程的持续测试。

持续测试管理通过端到端的流程，在研发流水线中使各环节的流程能够流动起来，从而具备更高的价值，形成需求特性提出到运营数据、用户反馈验证的持续交付闭环，从而基于实际用户反馈调整计划和需求。持续测试能避免问题在后期积累，有效减少了整个研发生命周期中时间和资源的消耗，能使问题在软件研发过程中得到实时反馈，促进软件研发过程的持续改进。持续测试提倡敏捷化和自动化测试，测试敏捷化是通过流水线中各环节的前置反馈和度量，推动业务价值快速交付的能力，从而建立更好的质量保障。自动化测试是指测试用例可按需被频繁、高效且自动化地执行，并能满足跨环境灵活地执行。自动化测试用例的开发采用数据驱动、关键字驱动等方法，自动化测试用例执行集成在流水线中执行，自动化测试执行用于冒烟测试、回归测试、功能验收测试等多个场景。持续测试实践环节参考表 1。

表 1　持续测试实践环节

持续测试实践举例	说明
单元测试	一般在软件研发过程中，对软件中的最小可测试单元进行检查和验证。
代码扫描	依靠扫描工具，通过解析源码语法，在交付初期发现源码存在的问题，持续改进代码质量，提升软件开发过程的稳定性和高效性。
功能测试	根据产品、用户等要求检查产品是否满足要求。

表 1　持续测试实践环节（续）

持续测试实践举例	说明
接口自动化	用代码或工具实现接口自动发起请求、自动验证返回结果、自动生成测试报告。
UI 自动化	通过编写代码或脚本来模拟人类用户与软件界面进行交互。
精准测试	精准测试建立在业务功能点（测试用例）和业务代码相互关联的基础之上，获取功能点的代码执行步骤和覆盖率，进行测试精准覆盖，测试缺陷精准定位。
安全测试	对产品进行检验以验证产品符合安全需求定义和产品质量标准。
性能测试	通过自动化的测试工具模拟多种正常、设计峰值以及异常负载（大压力）条件来对系统的各项性能指标进行测试，重点关注时延、容量、系统资源等指标。
模糊测试	根据一定的规则自动或半自动地生成随机数据，然后将这些产生的随机数据输入到动态运行的被测程序入口，同时监控被测程序是否有异常情况出现。
混沌测试	通过实验对系统注入故障来提升系统稳定性的赋能活动，通过主动地在系统中模拟故障，制造出系统异常，支持以可回放、持续性的故障演练验证系统稳定性，提前暴露出潜在问题并进行修复。
全网测试	市场中的所有参与者能够顺畅地连接和交互，验证交易系统、结算系统、监察系统等是否能够正常运行，各机构之间的网络连接是否正常等。
A/B 测试	通过科学的实验设计、采样样本代表性、流量分割与小流量测试等方式来获得具有代表性的实验结论，并确信该结论再推广到全部流量可信。
反演测试	通过反转系统输入、业务规则或假设条件，验证软件在“非常规场景”下逻辑正确性与健壮性的测试方法。通过验证系统在异常、错误或反向场景下的表现，检验软件功能的健壮性、边界处理能力及逻辑的严谨性，聚焦“否定性需求”，强调“逻辑闭环”。

8.4　质量分级管控

8.4.1　质量服务分级定义

质量分级管控是通过数据建模方式，将测试服务分级分类的方法，以及基于测试能力域评估模型进行维度平衡适配，用以寻求不同级别测试服务的最佳响应方式，形成完整且具备实际指导意义的测试服务响应能力评估体系。质量分级管控旨在为测试机构、企业、组织在面对某一测试需求时，通过对被测需求进行分类分级的分析，结合被测需求的实际属性，确定测试团队需要提供对应测试服务响应等级的参考方法。

8.4.2　质量服务分级策略

被测系统按照重要程度由高到低分为核心系统类、业务系统类、业务内管类和基础服务类，一般按照系统的重要程度高低来分配测试资源，重要程度越高则分配测试资源越多。从被测系统分类、研发模式、服务对象这三个基准属性以及紧急程度、影响范围这两个外部客观条件识别系统层级特性，结合行业测试服务现状，指导质量服务分级分类。

8.4.3　测试服务响应分级模型

测试服务响应分级模型结合测试服务需求分类分级方法和测试能力域评估模型，通过将不同类

型的需求及能力域的平衡适配，寻求不同级别测试需求的最佳响应方式，测试服务相应分级模型参考图 3。

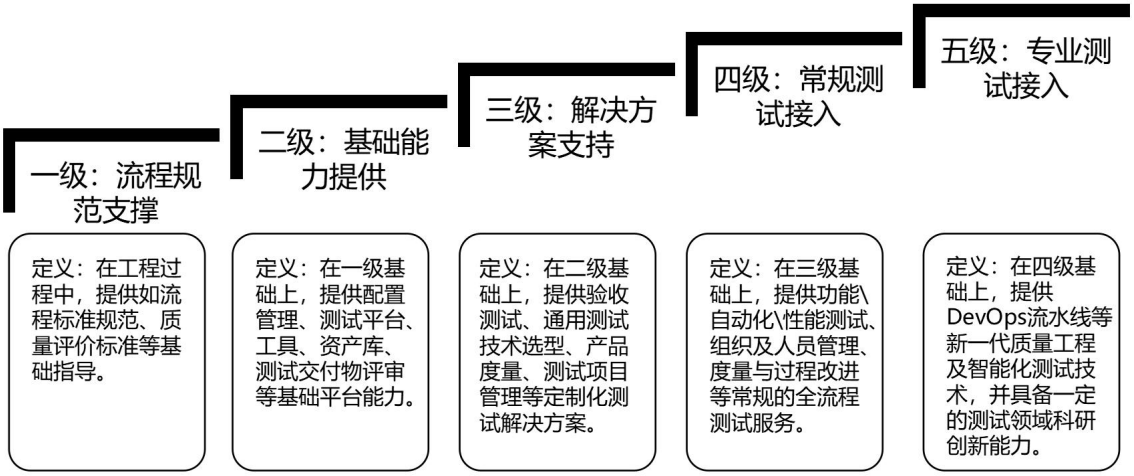

图 3　测试服务响应分级模型

8.5　质量的价值管理

8.5.1　质量的价值管理概述

质量的价值管理对研发运营一体化各环节持续开展质量管理、业务价值确认的活动，是持续改进质量、促进业务价值高质量交付的实践。目的是确保最终交付能满足业务需求提出方、客户及业务价值的相关方的需求。质量的价值管理通过质量的持续反馈、质量的分析度量及质量的度量改进来实现，其中质量的分析度量与改进见后续效能度量章节。

8.5.2　质量持续反馈

8.5.2.1　质量持续反馈概述

持续反馈是质量管理的重要部分，对软件全生命周期的各个阶段进行测试的结果、流程、效率和质量进行反馈，帮助快速定位问题，预防缺陷故障，提升软件研发效能。持续反馈提倡全面质量管理思想贯穿整个软件研发生命周期，采用自动化的手段发现产品交付过程中的潜在风险。持续反馈贯穿于研发交付流水线中，可采用质量门禁、测试报告、缺陷管理、生产事件复盘等优秀实践。

8.5.2.2　质量门禁

在研发流水线的各个环节中加入质量门禁能及时反馈软件在各阶段的质量，通过指标设定和权重计算量化质量标准，好的质量门禁能提前发现和预防风险、提高交付质量。质量门禁的制定需考虑全局最优，在建立之初就要做好清晰的定义和规划，但在落地管控过程中，要先从局部的质量门禁实践做起，如：冒烟通过率、单元测试覆盖率等。采用渐进管控的方式，不断加深质量门禁的实践，并不断拓展门禁的作用范围，通过持续改进和优化，最终达到全局最优的质量保障目的。实施过程中宜持续跟进执行和检查的过程和结果，对反馈的结果宜尽快响应并处理，才能更好地利用质量门禁来把控每个环节的产出物质量。

8.5.2.3　测试报告

测试报告作为测试产出物之一，主要展现测试结果、测试过程数据、缺陷数据、风险与建议等，

测试报告需满足内容完善准确和结果可追溯，测试报告需归档保存。在研发流水线中测试报告宜以邮件或其他形式自动发送给相关人员，同时支持测试报告与测试执行日志的在线查看，并将重要信息进行高亮显示。关系到重要业务场景的错误结果及风险预测宜通过实时通信方式发送并提醒项目负责人及时跟进并解决。

8.5.2.4 缺陷管理

缺陷管理包括缺陷定义识别、跟踪修复、预防、追溯，测试需对产品出现的所有失效进行识别和管理。项目团队记录缺陷发生场景与步骤并采集缺陷发生过程的日志和数据，确保缺陷尽可能被复现，定位缺陷原因，提供缺陷修复方案，并能举一反三推理出类似的缺陷，提高缺陷预防能力。

8.5.2.5 生产事件复盘

做好生产事件复盘是生产事件预测和预防的重要一环，通过复盘能有效地避免类似问题再次发生。生产事件复盘是持续测试右移的体现，通过对生产事件持续不断反馈、分析、总结和复盘，促进需求侧和开发侧不断优化，减少生产事件发生。

8.5.2.6 风险管理

风险管理能够提前识别潜在问题，评估风险的严重性，并制定相应的应对策略，从而有效降低风险发生的概率和影响程度。风险管理通过风险识别、风险评估、风险应对来保障质量。常见风险识别方法有头脑风暴法、检查表法和流程图分析法。风险评估是对已识别的风险进行分析和评估，确定其发生的可能性和影响程度。风险评估的标准通常包括风险发生的可能性、影响程度、紧急程度和可控性。根据风险评估的结果，制定相应的风险应对策略，并加以实施。常见的风险应对策略包括风险规避、风险降低、风险转移、风险接受。在实施风险管理过程中，需要持续跟踪和监控风险的变化情况，及时调整风险应对策略。

8.5.2.7 用户验收测试

用户验收测试是质量持续反馈的关键环节，由最终用户或业务方在类生产环境中执行，通过真实业务场景验证软件是否满足实际需求和用户期望。用户验收测试聚焦业务功能完整性、操作流程合理性和用户体验适配性，是软件交付前从用户视角进行的最终质量验证，其结果直接反映产品的业务价值和用户满意度。

在研发流水线中，用户验收测试需与需求管理、缺陷管理流程紧密衔接，形成闭环管控。测试前需明确验收标准和通过准则，测试过程中需详细记录测试场景、执行步骤和实际结果，测试后需形成验收报告并跟踪问题闭环。用户验收测试宜采用自动化工具辅助执行重复性场景，同时保留人工验证关键业务流程和用户体验细节，确保测试效率与质量的平衡。

用户验收测试的实施需注重场景化和真实性，测试用例宜覆盖核心业务流程、异常处理场景和边界条件。测试环境宜模拟生产配置，测试数据宜采用脱敏后的真实业务数据，确保测试结果的参考价值。通过用户验收测试，可及时发现需求理解偏差、功能实现缺陷和用户体验问题，避免问题流入生产环境。测试过程中收集的用户反馈需纳入产品改进计划，形成“需求－开发－测试－反馈－优化”的持续改进闭环，促进产品质量和用户满意度的不断提升。

8.6 质量管理工具链与平台

8.6.1 质量管理工具链与平台概述

质量管理体系涵盖软件研发的整个生命周期，为推动质量效率提升，提倡利用规范化、自动化

和体系化的方法推动质量的各个活动，可以从测试环境管理平台、自动化验收测试平台、测试数据管理平台等多方面分别进行探索及实践。

8.6.2 测试环境管理平台

测试环境管理主要用于管理被测环境，需通过一系列方法来规划和管理被测环境，提高测试环境的稳定性，减少因环境问题对测试活动的影响，以提升测试工作的质量和效率。

测试环境在软件研发的各个阶段都发挥重要作用，但不同阶段的环境有所不同，各测试环境的说明如下：

a）开发自测环境：基于本地开发机或集成环境进行编码、自测、部署和联调测试；

b）系统测试环境：基于功能环境进行系统测试；

c）验收测试环境：在发版审查正式之前，为了保证用户使用进行功能确认和质量确认的过程，可基于预发布环境；

d）线上生产环境：线上运行环境，供用户真实业务使用的环境。

8.6.3 自动化验收测试平台

自动化验收测试是利用自动化执行替代手工执行的用户验收活动。用户验收是端到端测试活动，从用户角度验证软件是否满足业务需求。

自动化验收测试平台搭建思路如下：

a）明确自动化验收测试的对象：自动化验收测试适合用于软件中的批量回归测试，或固定场景的功能测试、性能测试、容量测试或安全测试，对于业务逻辑经常变更的部分则需要手工测试来完成；

b）选择合适的自动化测试框架：选择自动化测试框架需要结合项目团队人员的特点、上手难易程度、维护成本等多方面考虑；

c）测试数据管理：对于自动化验收测试用例需要满足能被重复执行，因此测试数据的准备、清理和还原也要自动化完成，并且测试数据宜以文件或者数据库表的方式保存；

d）自动化测试用例管理：自动化测试用例宜划分优先级，并按照优先级加入研发流水线中，充分提升测试效率；

e）自动化测试用例更新：需保证自动化测试用例与代码变更的频率一致，当版本迭代更新时，保证自动化测试用例执行的准确性，避免缺陷误报。

8.6.4 测试数据管理平台

测试数据管理是为了保证在软件研发过程中能方便、快速地进行测试验证，结合软件功能维护相关测试数据。测试数据是质量活动的重要数据资产之一，对于软件质量至关重要。测试数据在存储、管理和使用过程中宜与其他系统、模块以及不同测试数据之间保持相互独立的特性，独立性包括逻辑独立性、存储独立性、环境独立性和业务独立性。逻辑独立性可以通过统一数据模型抽象和数据模板化配置支持解耦数据与应用逻辑。存储独立性可以通过标准化中间数据格式和设计存储抽象层来支持存储引擎和格式。环境独立性可以通过环境配置、数据分离、数据备份支持跨环境无缝迁移。业务独立性可以通过模块化数据组装和动态数据切片支持数据按需组合和裁剪。

测试数据管理平台搭建思路如下：

a）测试数据的来源：测试数据的来源一般是生产环境的脱敏数据、测试过程中的模拟数据、历史业务流程遗留的数据；

b）测试数据的存储：测试数据通常按照产品类别、测试类型、测试场景和数据来源进行分类、管理和存储以满足不同产品和不同阶段测试的需要；

c）测试数据的管理：存储后的测试数据为了适应各种测试场景和后续软件迭代和升级，需要进行版本管理，帮助清晰地区分有价值的测试数据，剔除冗余无效的测试数据。

9 技术运营管理体系

9.1 技术运营管理概述

技术运营是研发运营一体化的重要环节之一，承载着将需求、技术实现、安全与质量控制成果转化为稳定、高效服务的重要任务。不仅需要通过优化组织管理、强化系统监控、优化变更流程、管理性能和容量来确保服务连续性，保障信息系统的稳健运行，还需要体现研发运营一体化的核心理念——持续集成、持续交付与持续改进。

9.2 技术运营体系组织保障

9.2.1 组织架构

组织架构宜包含以下方面：

a）扁平化团队结构：在研发运营一体化实践中，运维组织倾向于采用更为扁平化的管理结构，以增强团队的灵活性和响应速度。核心团队或领导者仍然负责制定整体的战略方向和运维政策，但具体的运维计划和执行更多地依赖于自组织、跨功能的团队来完成。这些团队能够根据实际情况快速调整计划和执行任务，确保运维工作的高效和敏捷；

b）跨部门协作：研发运营一体化理念强调开发与运维的紧密合作。因此，运维组织内部宜设立与开发团队紧密对接、制定团队间协同规则的部门或小组，确保多部门之间在工作上的顺畅沟通；

c）专业化分工：根据运维工作的不同领域如基础设施运维、应用系统运维、运维工具开发等，运维组织内部会有进一步的专业化分工，以提高工作效率和质量，主要角色说明如下：

——基础设施运维：负责除应用系统外的基础设施运维，包括：操作系统、服务器、云平台、硬件平台、集成开发环境等；

——应用系统运维：负责应用系统的运维，包括应用系统，及其依赖的中间件、数据库等；

——运维工具开发：负责基础设施与应用运维所需工具的开发与维护，如：监控工具、信息技术服务管理平台等。

9.2.2 工作机制

工作机制宜包含以下方面：

a）快速恢复服务机制：运维组织需要快速响应各种突发情况，包括系统故障、性能下降等。因此需要建立完善的应急响应机制，确保在出现问题时能够迅速定位并处理，防止故障扩大或影响扩大；

b）主动预防机制：通过对系统各项指标的持续监控，将焦点集中在风险防范，通过全面地监控、深入地分析和精准地预测，及时发现潜在风险，并采取有效的预防措施，确保系统稳定运行，避免故障的发生；

c）评估与改进机制：运维组织会定期进行工作评估，总结经验教训，并针对发现的问题进行改进。保持对新技术、新方法的探索和学习，持续优化运维流程和工具，持续提升运维工作的效率和质量。

9.2.3 资源协同

资源协同宜包含以下方面：

a）统一资源管理：运维组织宜建立统一的资源管理与配置管理平台，对硬件、软件和网络等资源进行集中管理和调配，确保资源的高效利用和合理分配；

b）协同工具平台：运维组织合理利用协同工具平台，如项目管理软件、即时通讯工具、一体化运维平台等，提高团队之间的协作效率；

c）知识共享：运维组织内部及与其他团队之间建立知识共享机制，通过定期交流、培训、建立知识库等方式提升整个团队的技术水平和应对问题的能力。

9.2.4 流程管控

流程管控宜包含以下方面：

a）标准化流程：制定并执行标准化的运维流程，包括故障处理流程、系统升级流程等。这有助于确保工作的规范性和一致性；

b）流程服务化：运维流程除完成审批功能外，在一体化实践中强调流程的服务化，如可通过与CI/CD流水线、自动化运维平台等工具的集成，完成配置信息的更新、实现例行标准操作的自动化等；

c）反馈与改进：流程执行完毕后，需要收集反馈并进行评估。针对发现的问题和不足，及时调整和优化流程。

9.3 监控管理

9.3.1 监控管理概述

监控管理是以运维数据管理为基础，通过对日志、链路信息和运行指标等多种数据的综合收集和分析，实现对系统可观测性建设的过程。通过将系统的状态和运行指标等信息通过服务化的方式开放给业务、研发、测试等团队，是研发运营一体化中推进相关环节持续改进的重要手段。

9.3.2 运维数据管理

运维数据管理是从运维数据的采集、清洗、预处理直到存储的全生命周期的过程管理，宜包含以下方面：

a）数据采集：数据采集是运维数据管理的起点。需要确保所有关键系统和应用的日志、性能指标和交易数据等都能被有效收集。确保采集的数据类型和存储方式符合《证券基金经营机构信息技术管理办法》第二十九条的规定，同时宜采用安全、合规的数据存储解决方案，符合证监会要求的数据加密和备份机制。这些数据包括但不限于系统日志、性能指标、网络流量和用户行为等，对后续的数据分析、问题诊断和系统优化至关重要；

b）数据清洗：数据清洗是确保数据质量的关键步骤，通过自动化工具去除无效、重复或不完整的数据记录，以提高数据的准确性和可用性；

c）数据预处理：预处理包括数据转换、归一化和聚合等操作，以便数据能够适应后续的分析和处理流程；

d）数据存储：数据存储解决方案宜确保数据的持久化、安全性和可访问性。宜采用符合证监会要求的数据加密和备份机制。例如：可使用相关分布式存储数据库作为时序数据库存储长时间序列数据；

e）数据安全：数据安全是运维数据管理的核心，需要遵循相关的法律法规和标准，实施严格的访问控制和监控策略；

f）监控管理的数据安全宜遵循《证券基金经营机构信息技术管理办法》第三十一条的规定，实施网络隔离、用户认证、访问控制等安全保障措施。

9.3.3 数据可视化

数据可视化是对运维数据的特征，通过可被快速解读的方式进行呈现的过程。可使用开源工具辅助数据可视化工作的实现。最终可通过监控大屏，实时展示关键指标，便于运维团队及时发现和解决问题。数据可视化宜包含以下方面：

a）图表类型：选择合适的图表类型对于有效展示数据至关重要。宜根据数据的特点和监控需求选择条形图、折线图、饼图等；

b）数据一致性：确保不同来源和不同时间点的数据具有一致的格式和度量标准，以便于比较和分析。

9.3.4 告警管控

告警管控主要包括告警各项要素定义，如触发条件、告警级别等。此外还宜涵盖告警如何触达运维人员，如何记录以及事后分析。告警管控宜包含以下方面：

a）告警分类定义：明确告警的类型和触发条件，如设备故障告警、系统性能阈值、核心业务指标异常告警等，以便及时响应潜在的问题。例如：可使用告警工具进行告警定义管理；

b）告警级别：根据告警的严重性和紧急性划分不同的级别，如警告、严重和紧急，以指导运维团队的响应优先级；

c）告警记录和审计：所有告警事件都宜记录在案，并定期进行审计，以便于事后分析和改进。

d）趋势分析：通过趋势分析，可以预测潜在的问题和风险，从而采取预防措施；

e）告警收敛：采用基于事件、时间、根因等方式进行告警信息的收敛，提高告警的精准度和有效性。

9.3.5 巡检管理

巡检作为监控的补充手段，是运维工作不可或缺的部分。巡检管理包含对计划、内容、结果记录的相关要求。巡检管理宜包含以下方面：

a）巡检计划：制定详细的巡检计划，包括巡检的时间、频率和责任人。例如：可使用自动化巡检工具进行巡检脚本编写；

b）巡检内容：明确巡检的具体内容，如系统状态检查、配置验证和性能测试等；

c）巡检记录：记录每次巡检的结果，包括发现的问题和采取的措施。例如：可通过表格软件或者在线巡检管理系统进行巡检结果的记录以备审计和分析。

9.4 可用性管理

9.4.1 可用性管理概述

可用性管理是一系列策略、流程和技术，旨在确保 IT 服务能够满足预定的性能标准和业务需求。可用性管理的核心目标是确保 IT 服务在用户需要时能够按预期可靠地提供，同时最大化服务的正常运行时间，并最小化服务中断的影响。

9.4.2 可用性评估

可用性评估的目的是确保 IT 服务的可用性满足业务需求并符合服务水平协议（SLA）的要求。可用性评估宜包含以下关键方面：

a）明确服务可用性目标：与利益相关方协商确定 IT 服务的可用性目标，这些目标通常在服务水平协议（SLA）中定义；

b）可用性度量：使用关键指标来量化服务的可用性，例如：

——正常运行时间（Uptime）：服务正常运行的总时间；

——服务可用性（Service Availability）：服务满足性能标准的时间百分比；

——故障间隔时间（MTBF）：两次连续故障之间的平均时间；

——平均故障恢复时间（MTTR）：检测故障到恢复服务所需的平均时间。

c）容量管理：对 IT 基础设施的容量进行规划、分析、计划和管理，以确保 IT 服务能够满足当前和未来的业务需求。主要由以下部分组成：

——业务视角：容量管理宜与业务目标和战略保持一致，理解业务需求和工作负载模式，预测未来的服务需求；

——性能数据收集：收集和分析 IT 基础设施的性能数据，包括硬件、网络、存储和应用程序；

——性能优化：通过技术改进和资源优化，提高现有基础设施的性能和效率；

——风险管理：识别容量相关的潜在风险，如性能瓶颈或资源短缺，并制定相应的缓解措施。

9.4.3 容灾备份

容灾备份是确保业务连续性和灾难恢复的关键组成部分。容灾备份的目的是减少因意外事件（如硬件故障、数据丢失、网络攻击或自然灾害）导致的服务中断和数据丢失的风险。要求组织采取全面的、预防性的措施来保护关键数据和系统，以最小化潜在的业务影响。容灾备份宜包含以下关键要素：

a）容灾规划：制定用于指导组织在发生灾难性事件后如何恢复 IT 服务和运营的计划，需要考虑在不同级别的灾难情况下的应对策略和恢复优先级；

b）数据备份：考虑各种因素，根据需求设计不同的备份策略。可以参考以下关键因素：备份频率、存储介质、备份类型（冷备或热备）等；

c）异地备份：将备份数据存储在不同于主要数据中心的物理位置，以保护数据不受单一地点灾害的影响，如异地多活。

9.4.4 故障恢复

故障恢复是 IT 服务管理中的一个过程，旨在发生 IT 服务故障或中断后，尽快恢复服务的正常操作，以最小化对业务活动的影响。故障恢复宜包含以下方面：

a）事故恢复流程：

——准备阶段：确保所有必要的资源、工具和文档都已准备就绪；

——识别阶段：快速识别事故的范围和性质，以便于采取适当的恢复措施；

——遏制阶段：采取措施防止事故影响进一步扩散；

——恢复阶段：执行恢复活动，以恢复受影响的服务或功能；

——后续阶段：事故解决后，进行彻底的审查和复盘以识别根本原因。

b）关键衡量指标：

通过明确定义 RPO 和 RTO，可以更好地规划和实施适当的备份、冗余和恢复策略，以保护其关键业务数据和服务，确保在面对各种潜在的 IT 灾难时能够迅速恢复。

——恢复点目标（RPO）：在恢复过程中可以接受的数据丢失的最大量，通常表示为时间间隔。RPO 的设定还可以帮助组织确定备份数据的频率，以确保在故障发生时能够恢复到最近的状态；

——恢复时间目标（RTO）：发生服务中断后，服务必须恢复到可接受水平的最大时间限制。RTO 关注的是服务的可用性和恢复速度。

c）制定灾难恢复计划：

制定计划文档，需要详细说明组织如何应对和从灾难性事件中恢复关键的 IT 服务和业务操作。内容包括：明确灾难恢复计划的目标，包括保护关键业务功能和数据，确定哪些 IT 系统和服务将被包括在内。确定恢复过程中所需的资源，包括备用硬件、软件、数据备份和备用数据中心。详细描述故障发生时的步骤和程序，包括事故检测、遏制、恢复和后续行动。

9.4.5 高可用架构

高可用架构是确保 IT 系统和服务能够在面对硬件故障、软件崩溃、网络问题等各种故障情况下，仍然保持运行或快速恢复的有效手段。构建高可用架构宜包含以下的关键方法和策略：

a）冗余设计：对每个层级的 IT 设施做适当的冗余。包括网络、服务器、存储或其他计算资源等。确保任何组件失败时，有备选可以立即接管其工作；

b）负载均衡：利用负载均衡器分散请求和工作负载，减少单点过载的风险，并提高系统的整体性能；

c）弹性伸缩：实现系统的弹性伸缩，需要基于预设指标来动态调整资源分配，如 CPU 使用率、请求延迟或队列长度等，并设定相应的阈值。一旦指标超出这些阈值，系统将自动执行预定义的伸缩行为，可能是增加或减少实例的数量，以此来适应负载变化。为了避免因频繁伸缩操作而引发的系统不稳定，可以引入冷却时间机制，确保系统在每次伸缩决策后都能有一段稳定运行期；

d）数据复制：通过将数据复制到多个存储位置，以确保在发生故障时，数据仍然可以被访问和使用。数据复制是实现高可用性和灾难恢复的关键技术，但它也增加了系统的复杂性。在设计数据复制策略时，需要在数据一致性、系统可用性、分区容错性之间做出权衡；

e）容错机制：系统需要具备在面对故障或不利情况时，仍能继续提供服务或执行任务的能力。实现容错有以下关键措施：

——健康检测：系统必须能够及时检测到故障的发生，这是启动容错机制的必要条件；

——服务降级：系统可以暂时降低非关键性服务水平，优先将资源向核心功能倾斜，保证正常使用。可采用但不限于限流、熔断等方式实现；

——失败转移：当检测到故障时，系统能够自动将工作负载从故障节点转移到健康节点。

9.5 事件管理

9.5.1 事件管理概述

事件是指在信息技术（IT）服务的预期操作中发生的任何不符合标准的情况。这些情况可能会影响服务的可用性、性能或安全性。事件管理是一个核心流程，旨在快速恢复服务到正常运行状态，以最小化对业务的影响。

9.5.2 事件分类分级定义

事件宜根据不同的特性，做合理的分类，分类的目的是用于快速识别事件的性质，从而采取适当的处理措施。可参考的事件分类包括：安全、网络、软件、硬件等，IT 组织依据实际情况做合理定义。

宜根据事件的严重性、影响范围和紧急程度对事件进行定级，事件分级的目的是确保 IT 组织能够合理分配资源，优先处理那些对业务影响最大的事件。通常事件按严重程度从高到低，分为 1 至 5 级。每个 IT 组织根据自己的业务需求、服务水平协议（SLA）和服务台的工作流程来定制具体的分级标准。

9.5.3 事件应用

需要使用标准化的事件管理流程来处理 IT 服务中的意外中断或性能下降，以确保服务的连续性和可靠性。事件应用宜满足以下常用原则：

a）快速响应：事件发生时，IT 团队能够迅速响应，以最小化对业务活动的影响；

b）完整记录：所有事件都被记录在事故管理系统中，包括详细信息和处理步骤；

c）合理分类定级：事件被分类，并根据严重性、影响范围和紧急程度被赋予优先级；

d）复盘：对事件进行分析，以确定其根本原因，将事件处理过程中获得的信息和解决方案记录在案，形成知识库。

9.6 应急管理

9.6.1 应急管理概述

应急管理是指针对可能影响 IT 服务的突发事件或紧急情况所采取的准备、响应和恢复措施的一系列管理过程。应急管理的目的在于确保组织在面对可能影响 IT 服务交付的突发事件或紧急情况时，能够有效地维持关键业务流程的连续性，并迅速恢复其运营。其中对已知标准故障场景实现故障自愈恢复是常见的实践之一。

9.6.2 应急预案

为确保 IT 服务的连续性，组织宜定期执行风险评估，以识别可能的威胁，并据此制定详细的应急预案。这些预案宜包括明确的前置条件、指定的操作人员和具体的执行步骤。同时，宜指派专人负责预案的持续更新和维护，确保其反映最新的业务和技术环境。应急预案宜整合进组织的知识管理体系。其中鼓励对已知标准故障场景进行识别，通过完善系统架构或自动化工具辅助，实现应急预案的自动化，完成已知故障场景的故障自愈。

9.6.3 应急演练

应急演练是模拟紧急情况，以评估和提升组织对 IT 事件的响应能力的过程。应急演练需定期执行，以适应业务与技术的最新变化。宜采用真实 IT 系统和数据，模拟各种紧急情境，包括常见与极端事件，确保全面覆盖。所有相关方都宜参与其中。基于演练反馈，采取改进措施，并将经验整合入知识管理。同时，确保演练的合规性。

9.6.4 应急响应与处置

应急响应与处置宜包含以下关键步骤：

a）初步诊断：一线支持团队在接到事故报告后，立即进行初步诊断，以确定事故的性质和影响范围；

b）上报：如果一线团队无法解决问题，需将详细情况记录并上报给合适的专业团队；

c）沟通：在整个过程中，保持与所有相关方沟通，及时更新事故状态和任何重要信息；

d）调查和诊断：进一步调查事故原因，进行深入分析，以确定根本原因，必要时引入外部专家；

e）解决方案和恢复：一旦确定了事故原因，制定实施解决方案以恢复服务；

f）关闭：事故解决后，由服务台或指定团队正式关闭事故记录；

g）后续复盘：记录事故处理过程中的所有活动，包括采取的措施和取得的结果，以便未来可以改进响应流程；

h）知识管理：将事故处理的经验教训整合到组织的知识库中。

9.7 变更管理

9.7.1 变更过程管理

9.7.1.1 变更过程管理概述

变更管理工作是指对证券期货业系统的软硬件、网络、数据等各个方面的变更进行全面管理和控制，以确保系统的稳定性和安全性。变更管理工作的目的是全面管控信息系统变更风险，确保变更符合相关规定和标准，避免因变更不当导致系统故障、数据丢失等问题，保障证券期货市场的稳定运营。变更管理需要确保在保持速度和敏捷性的同时跟踪和管理变更。

9.7.1.2 变更评审

变更评审主要目标是评估变更请求的合理性、可行性和风险，确保变更符合业务需求和技术标准。主要内容包括：

a）审查变更请求的内容，包括变更目的、影响范围、预期结果等；

b）分析变更可能带来的风险，并制定相应的风险应对措施；

c）评估变更所需资源和时间成本；

d）确定变更的优先级。

变更评审参与人员包括：项目经理、开发团队成员、测试团队成员、运维团队成员及相关业务代表。

9.7.1.3 发布计划

发布计划主要目标是规划和管理变更的发布过程，确保变更能够按照计划顺利上线。在实践中CI/CD流水线或运维工具支持多种发布方式。发布计划宜包含以下方面：

a）制定详细的发布计划，包括发布时间、发布策略、步骤和预期结果；

b）协调各团队之间的合作，确保发布过程的顺利进行。准备发布所需的文档和工具，如发布说明、配置文件等。

9.7.1.4 变更实施与核对

变更实施与核对的主要目的是按照既定计划实施变更，并确保变更内容的准确性和完整性。在实践中CI/CD流水线或运维工具辅助对变更进行自动化执行，并对变更结果数据进行收集与核对。变更实施与核对宜包含以下方面：

a）在预定的时间窗口内执行变更操作；

b）核对变更内容是否与计划一致，确保没有遗漏或错误；

c）记录变更过程中的所有操作和遇到的问题。

9.7.1.5 验证测试

验证测试的主要目标是验证变更后的系统功能和性能是否满足预期要求。在实践中CI/CD流水线或运维工具支持对验证测试结果的采集与统计。验证测试宜包含以下方面：

a）确保测试覆盖受变更影响的部分，以及可能引发的连锁反应；

b）按需进行性能测试，评估变更对系统性能的影响；

c）记录测试结果，评估变更效果；

d）如果测试发现问题，及时通知相关团队进行修复，并重新进行验证测试。

9.7.1.6 回滚计划

回滚计划的主要目标是在变更出现问题时，能够迅速恢复到变更前的状态，保证系统的稳定性和可用性。在实践中 CI/CD 流水线或运维工具支持对变更回滚的自动化执行。回滚计划宜包含以下方面：

a）制定详细的回滚计划，包括回滚步骤、时间点和预期结果；

b）准备回滚所需的文档和工具，如备份文件、恢复脚本等；

c）在变更实施前进行回滚演练，确保在紧急情况下能够迅速执行回滚操作。

9.7.2 变更管理效果回顾

9.7.2.1 变更管理效果回顾概述

变更效果回顾用以评估变更管理的效果，并为变更管理的改进提供数据支持和方向指引。

9.7.2.2 效果数据收集

变更管理效果数据收集的主要目标是收集变更活动过程中的相关数据，以量化方式评估变更管理的效果。并且提供客观的数据支持，帮助团队识别变更管理中的优点和不足。效果数据收集宜包含以下方面：

a）收集变更请求的数量、类型、处理时间和结果等关键数据；

b）记录变更过程中的延误、失败、回滚等异常情况，并分析其原因；

c）形成变更管理量化指标，如变更引起的事件数量、变更引入故障（事件）占比、变更回滚率等。

9.7.2.3 制定改进计划

制定改进计划的主要目标是基于统计数据和分析结果，制定具体的改进计划，以提升变更管理的效率和质量。并且确保改进计划具有可行性和针对性，能够解决实际问题。制定改进计划宜包含以下方面：

a）分析统计数据，识别变更管理中的瓶颈和问题点；

b）确定改进目标，如缩短变更实施时间、降低变更失败率等；

c）提出具体的改进措施，如优化变更流程、提升团队成员技能、引入自动化工具等；

d）制定实施时间表和责任人，确保改进计划的顺利执行。

9.8 性能容量管理

9.8.1 性能容量目标管理

9.8.1.1 性能容量目标管理概述

性能容量管理目的是确保系统在不同负载下都能保持稳定、高效地运行，以满足不断变化的业务需求。性能容量管理主要通过规划、监测、优化等手段。其中，通过容量规划确定性能容量目标是性能容量管理的基础，涉及对系统资源、业务需求和发展趋势的全面了解与分析，以制定出合理的性能和容量规划方案。

9.8.1.2 业务容量规划

业务容量规划是为了确保系统的能力能够满足业务在未来一定期限内的服务需求，对系统性能、容量进行规划、设计的过程。主要根据业务运营、交易系统需求，确定系统业务容量，业务容量规划可从预期的总用户数、在线用户数、业务订单、交易量、请求数、请求时长等方面进行规划。

9.8.1.3 信息系统容量规划

信息系统容量规划是为了确保系统的能力能够满足业务在未来一定期限内的服务需求，对基础资源容量进行规划、设计的过程，具体说明如下：

a）信息系统容量规划是依据业务容量规划，转化形成信息系统容量需求的过程，信息系统容量规划可根据应用时延、请求响应、服务调用频次等方面关键性能容量指标进行规划；

b）基础资源容量规划。基础资源容量规划是依据信息系统容量设计，转化形成基础资源容量需求的过程，基础资源容量规划可从主机拥有的计算、存储、网络和输入输出（IO）四大类资源进行评估，包括相关的中央处理器（CPU）使用率、内存使用率、磁盘 IO、存储空间、网卡流量、网络相关的带宽、网络吞吐量，以及数据库、中间件相关的技术指标等方面进行规划。

9.8.2 日常监测与特别保障

日常监测是实时了解系统运行状态的关键，包括对各项性能指标和容量使用情况的持续监控，确保系统始终在健康状态下运行。当发现系统性能不足或容量瓶颈时，优化工作就显得尤为重要。通过调整系统配置、优化代码和资源使用等手段，可以有效提升系统的整体性能和容量利用率。日常监测与特别保障宜包含以下方面：

a）日常容量与性能管理：

——进行信息系统的容量资源监控、预警、告警；

——各信息系统的容量指标不断完善，容量监控质量持续提高；

——通过容量数据分析，预测容量发展趋势；

——发现版本发布前后的性能容量变化。

b）年度容量与性能管理：

——对重要系统的性能容量使用进行分析总结；

——分析容量短板，制定年度容量计划，并作为 IT 资源计划制定的依据。

c）特别保障：

——主要包括重大变更、重要业务投产等的容量预估、容量压测、营销后数据分析等工作；

——评估相应信息系统的容量满足特别保障活动需要，及时提出扩容需求并进行扩容。

9.8.3 技术运营反馈

9.8.3.1 技术运营反馈概述

技术运营团队通过对系统运行的质量、效能数据进行统计分析，向开发与业务团队进行非功能需求、业务数据分析等内容的闭环反馈，是实现技术运营从运维到运营的转变的关键环节，提高系统的整体质量和效益。

9.8.3.2 持续改进与优化

向开发与业务团队进行闭环反馈包括系统稳定性反馈、性能反馈、数据分析反馈。持续改进与优化宜包含以下方面：

a）系统稳定性反馈：技术运营团队提供关于系统的稳定性反馈，包括系统的故障频率、MTTR、交易中断次数等。识别和解决交易系统的稳定性问题，提高客户的满意度；

b）性能反馈：技术运营团队提供关于系统的性能反馈，包括系统的响应时间、吞吐量、并发处理能力等。识别和解决性能瓶颈，提高系统的效率和用户体验；

c）数据分析反馈：技术运营团队提供关于交易系统的数据分析反馈，包括对交易数据等用户的分析结果、市场趋势的预测结果等。可以辅助公司业务团队更好地理解市场状况和客户需求，并制定更加精准的业务策略。

9.8.3.3 反馈渠道

向开发与业务团队进行闭环反馈渠道包括专项会议、非功能需求、定期运营报告。反馈渠道宜包含以下方面：

a）专项会议：专项会议是一种高效、直接的反馈方式。技术运营团队会定期组织与研发和业务团队的专项会议，针对生产运维中的关键问题、新出现的技术难题或重大变更进行讨论；

b）非功能需求：非功能需求是技术运营团队向研发团队反馈的重要方式，需求主要关注系统的性能、安全性、可靠性等非业务功能方面。技术运营团队根据生产环境的实际情况，提出信息系统的非功能需求，如提高系统的响应速度、可观测性等；

c）定期运营报告：定期运营报告是技术运营团队向研发、业务团队反馈生产运维情况的主要方式之一。这些报告通常包括关键性能指标（KPIs）、故障分析、用户反馈汇总等信息。通过定期（如每周、每月）报告，相关团队及管理层能够全面了解信息系统的技术运营状况。

9.9 运营配置管理

9.9.1 运营配置管理概述

运营配置管理是识别和确认系统的配置项、记录和报告配置项状态和变更请求、检查配置项的正确性和完整性等活动构成的过程，其目的是提供 IT 基础架构的逻辑模型，支持技术运营管理体系的各项工作。

9.9.2 运营配置对象管理

运营配置对象是指与技术运营相关的配置项，包括如下内容：

a）对运营配置对象全生命周期进行管理，配置对象状态更新可通知；

b）运营配置对象的变更可以关联技术运营事件；

c）自动发现配置对象间的关联关系，智能识别配置对象间的内建关联关系，动态更新配置库；

d）根据运营配置对象间的关联关系动态生成配置视图或架构图，辅助绘制业务功能架构视图。

9.9.3 运营配置数据管理

运营配置数据是指与配置项相关的数据，记录配置项的状态、变更及配置项间的关系，可覆盖配置项的全生命周期。包括如下内容：

a）统一运营配置数据管理，实时反馈在线运维对象的运行状态；

b）运营配置数据接口化，对外提供 API 调用接口；

c）通过流程管理配置项变更，部分采用人工维持数据的准确性，部分采用自动化变更方式；

d）运营配置数据变更日志可维护管理，可支持变更回溯和日志审计；

e）基于单一可信数据源自动发现和更正关键运营配置数据；

f）运营配置数据变更可以联动触发其他技术运营系统，如：部署系统中的目标机器信息等。

9.10 技术运营管理工具链与平台

9.10.1 自动化环境部署

开发、测试与生产环境的一致性，通过 IaC，运维团队可以使用统一的代码模板快速创建和配

置开发、测试、生产等不同环境。这种方法确保了各个环境之间的配置一致性，避免了“在开发环境正常运行，但在生产环境出错”的问题。

快速创建临时环境，在需要进行特定测试或演示时，IaC 允许快速创建临时环境，测试完成后再自动销毁，节省资源并提高灵活性。

在开发、测试、预生产和生产等多个环境中部署应用时，确保各环境配置一致，避免环境一致性的问题。

工具结合（多环境一致性部署）宜包括如下能力：

a）使用工具编写基础设施配置文件，定义不同环境所需的资源；

b）通过代码化文件编写配置管理文本，自动化安装和配置应用所需的软件和服务；

c）在 CI/CD 流水线中集成这些工具，自动化创建和配置各环境。

工具结合（快速创建临时测试环境）宜包括如下能力：

a）定义临时环境的基础设施和应用部署流程；

b）在代码库中配置 CI/CD 流水线，当有新分支或拉取请求时，自动运行脚本创建基础设施；

c）使用容器技术部署应用，并通过容器编排框架管理临时集群；

d）测试完成后，自动销毁临时环境，释放资源。

9.10.2 配置管理

统一管理服务器配置，使用 IaC 工具可以统一管理和配置服务器，确保所有服务器的一致性和标准化，减少人为配置错误。

动态配置更新，通过 IaC，可以轻松地对基础设施配置进行更新和修改，确保所有变更能够自动化地应用到所有相关资源上。

工具结合（基础设施与应用代码同步部署）宜包括如下能力：

a）在代码仓库中将基础设施代码和应用代码存放在不同目录；

b）配置 CI/CD 流水线，先运行脚本部署或更新基础设施；

c）使用配置管理工具，自动化配置新部署的基础设施；

d）部署应用代码到新配置的环境中，完成同步部署。

工具结合（自动化回滚与灾难恢复）宜包括如下能力：

a）在 CI/CD 流水线中记录每次部署的基础设施和应用代码版本；

b）当检测到部署失败或系统异常时，自动触发回滚流程；

c）使用脚本回滚基础设施到先前的稳定状态；

d）利用发布管理工具或 CI/CD 工具回滚应用代码到之前的版本。

9.10.3 基础设施版本控制

变更追踪与审计，将基础设施定义存储在版本控制系统中，可以跟踪所有变更历史，进行审计，确保每一次变更都有章可循。

支持回滚操作，如果某次基础设施变更导致问题，可以通过版本控制系统快速回滚到之前的稳定版本，减少停机时间和影响范围。

工具结合（代码化版本管理）：将基础设施代码化文件通过版本系统进行管理。

9.10.4 自动化扩展与弹性管理

根据负载自动调整资源，利用 IaC 脚本，结合自动化工具，可以根据实时负载情况自动扩展或缩减计算资源，确保系统的高可用性和性能。

实现弹性伸缩，通过 IaC 定义的策略，基础设施能够根据预设的规则自动进行弹性伸缩，适应业务需求的变化。

工具结合（基础设施监控与告警）宜包括如下能力：

a）实时监控基础设施和应用的性能指标，自动触发告警并执行修复操作；

b）使用监控和可视化系统性能指标采集基础设施和应用的性能指标；

c）在展示工具中配置仪表盘，实时可视化监控数据；

d）配置告警管理工具，根据设定的阈值自动发送告警通知；

e）在告警触发时，利用代码化文件自动执行修复脚本（如扩展资源、重启服务）。

工具结合（自动化运维任务）宜包括如下能力：

a）自动化执行日常运维任务，如系统更新、备份、日志清理等，减少手动操作，提高效率；

b）编写代码化文件，定义需要自动化的运维任务；

c）使用 CI/CD 工具定时触发这些自动化任务，确保按计划执行；

d）利用工具管理与运维相关的基础设施资源，确保自动化任务的执行环境一致。

9.10.5 灾备恢复与高可用性

快速重建基础设施，在灾难发生后，IaC 使得基础设施能够通过代码快速重建，缩短恢复时间，提升系统的可靠性。

高可用架构的自动部署，通过 IaC，可以定义和部署高可用架构（如多区域部署、负载均衡配置），确保系统具备容错能力，提升服务的连续性。

工具结合（自动化灾难恢复流程）宜包括如下能力：

a）在发生灾难性故障时，自动化执行恢复流程，确保系统快速恢复正常运行；

b）使用代码化基础设施文件定义备用环境的基础设施配置；

c）在主环境发生故障时，触发 CI/CD 流水线，自动创建备用环境；

d）使用域名服务器工具（自动切换流量到备用环境）；

e）确保应用在备用环境中自动部署并恢复服务。

工具结合（高可用架构的自动化部署）宜包括如下能力：

a）部署高可用性架构，确保系统在部分组件故障时仍能正常运行；

b）使用基础设施配置定义跨多个区域和可用区的基础设施资源，如负载均衡器、数据库主从配置；

c）部署容器编排管理集群，配置自动伸缩和故障转移策略；

d）利用环境代码化配置高可用性应用，确保在一个节点故障时，其他节点能够自动接管服务。

9.10.6 多云与混合云管理

跨云平台的一致管理，IaC 工具支持多个云服务提供商，能够统一管理跨云平台的基础设施资源，实现多云环境的一致性和可控性。

混合云环境的集成，在混合云环境中，IaC 可以帮助整合本地数据中心与云资源，实现资源的无缝管理与协同运作。

工具结合（跨云平台的基础设施管理）宜包括如下能力：

a）在多个云服务提供商之间统一管理基础设施，实现资源的跨平台一致性和可控性；

b）使用环境代码化文件编写跨云平台的基础设施定义，管理不同云提供商的资源；

c）利用代码化环境的配置管理和部署统一配置跨云平台的服务器和服务，确保一致性；

d）在 CI/CD 流水线中集成环境代码化文件，实现自动化的跨云部署和管理。

工具结合（混合云环境的自动化集成）宜包括如下能力：

a）将本地数据中心与云资源无缝集成，统一管理，实现资源的高效利用和灵活调度；
b）使用代码化本地和云端的基础设施文件定义本地数据中心和云端的基础设施资源；
c）通过统一代码化文件管理配置本地和云端服务器，确保配置的一致性；
d）部署容器编排管理集群，实现本地与云端资源的统一编排与管理，支持跨环境的容器部署与扩展。

9.10.7 自动化网络安全配置

自动化安全配置，通过 IaC，可以自动配置安全组、网络策略、身份与访问管理（IAM）等安全措施，确保基础设施符合企业的安全标准。

合规性检查，IaC 可以在部署过程中自动检查和强制执行合规性规则，避免违规配置的产生。

工具结合（自动化安全配置与扫描）宜包括如下能力：
a）确保基础设施和应用配置符合安全标准，自动化检测和修复安全漏洞；
b）在 IaC 脚本中定义安全策略，如防火墙规则、IAM 角色等；
c）在 CI/CD 流水线中集成静态代码分析工具，自动扫描代码中的安全漏洞；
d）编写安全策略，确保基础设施配置符合企业安全标准；
e）自动修复检测到的安全漏洞或配置不符合问题。

工具结合（合规性审计与报告）宜包括如下能力：
a）定期审计基础设施配置和应用部署，生成合规性报告，满足行业和法律法规要求；
b）使用工具管理基础设施，并通过版本控制系统跟踪变更；
c）配置云服务提供商的合规性工具，自动监控基础设施是否符合规定的政策；
d）利用日志收集工具汇总基础设施和应用的配置变更日志；
e）定期生成合规性报告，自动发送给相关审核人员。

10 安全管理体系

10.1 安全管理体系概述

安全管理是研发运营一体化体系的重要组成部分，贯穿研发体系的全生命周期。它涵盖了组织保障、安全工具链、需求管理、技术实现、质量管理、技术运营及新兴技术中的安全等多个方面。在安全体系的构建过程中，企业宜以相关法律法规为基础，如《中华人民共和国网络安全法》《中华人民共和国数据安全法》《中华人民共和国个人信息保护法》等，确保企业安全和业务合规。企业宜加强安全意识提升，通过定期培训、培育安全文化、设立奖惩机制等措施强化员工的安全防范能力。公司宜建立完善的安全制度体系，包括软件安全开发生命周期管理办法、项目安全评估流程、软件供应链及开源软件安全管理规范和第三方软件的安全管理办法等，明确各阶段的安全管理要求，确保软件使用的安全性和合规性，并制定应急处理机制，以应对潜在风险。

10.2 组织保障

组织保障是安全体系中重要的一个环节，通过对团队职责的定义，实现对安全体系的组织级保障。组织保障宜包含以下方面：

a）应用安全团队职责：主要有软件安全开发生命周期管理及安全管理体系建设；系统建设过程中的安全专家支持，包括安全需求分析、威胁建模、安全设计、安全架构、代码审计、风险决策等；应用安全标准化架构、应用安全解决方案及安全组件的交付；端到端应用安全测试工具链建设；安全工具及产品的研发；应用安全风险常态化运营；应用安全相关新领域安全

研究和实践等；

b）数据安全团队职责：数据安全治理、风险监控，系统的数据安全和个人信息保护措施建设，业务使用数据场景的能力建设；数据全生命周期的风险管理体系建设项目、业务场景的数据安全风险评估，数据安全方案设计和建设。及时识别业务可能存在的数据安全风险，提出解决方案，并制定和维护数据安全标准和基线；数据安全规范、规范和流程的制定和优化；以及公司内部数据安全相关标准、制度和安全基线的落地、安全策略实施和风险事件处置；

c）网络安全团队职责：网络安全设施维护；网络安全风险常态化运营；对网络服务器，网络设备，以及网络系统等进行安全维护，包括日常的安全巡检，以及策略维护管理，安全故障的处置等；对服务器进行漏洞扫描、端口扫描、弱密码扫描，整理报告；对各类安全事件主导跟进。

10.3 需求管理体系中的安全

10.3.1 安全需求

行业内信息系统都会面临各种不同类型、不同程度的安全风险，包括：法律法规风险、内容风险、个人隐私和数据安全风险、Web 应用和移动应用风险等。通过持续的风险分析活动，可以不断收集、形成、完善安全需求库，以便在具体信息系统建设初期，对照和形成有针对性的安全建设需求列表，作为系统后期详细设计、编码、测试、部署、运行的基础。从而有效地降低信息系统上线后的安全风险，提升信息系统的安全保障能力。

信息系统安全需求描述了为实现信息系统安全属性所需要满足的功能性和非功能性要求。安全需求来源可以包括 4 个层面：

a）法律法规，国家标准；

b）行业监管法令、指引、技术标准；

c）公司自有信息安全策略、标准、指南；

d）安全界实践。

构建安全需求库。将各来源的需求，经过分类、去重后，按统一的方式进行编号和入库，确定对应的信息系统场景（即适用条件）（包括：系统的业务属性、用户属性、重要功能属性、安全功能相关属性、网络属性、终端属性、架构属性、系统获取方式、数据敏感级别等），以便进行自动化的，基于场景的轻量级威胁建模。

10.3.2 威胁建模

威胁建模是分析应用程序安全性的一种结构化方法，通过识别威胁，定义防御或消极威胁控制措施的一个过程。

在需求阶段，建议通过威胁建模，系统地分析系统面临的安全威胁。威胁建模通常包括以下步骤：确定系统边界、识别系统资产、识别安全威胁、评估风险等。在识别安全威胁时，要全面考虑系统面临的各种威胁，如恶意攻击、数据泄露、权限滥用等。风险评估要考虑威胁发生的可能性和影响程度。威胁建模的结果宜形成安全威胁列表和风险评估报告，为制定安全需求和设计安全方案提供依据。威胁建模宜与需求分析紧密结合，并在系统生命周期中持续进行。

10.4 技术实现体系中的安全

10.4.1 安全架构与设计

技术实现阶段，建议引入安全架构与设计活动，在系统总体架构和设计中体现安全需求，采用适当的安全机制和技术，使系统满足安全目标。

应用系统的架构设计人员运用基本的安全设计原则来进行与开发语言、平台无关的架构级系统设计，保护整个信息系统避免因意外错误到恶意入侵在内的各种安全故障，帮助开发人员和安全专业人员构建安全的应用程序。安全设计原则包括开放设计原则、默认安全原则、权限分离原则、最小权限原则、最小公共化原则、不要轻信原则、保护薄弱原则及提升隐私原则等。

安全设计宜合理采用各种安全技术，如访问控制、加密、日志审计等。宜权衡安全性、可用性、性能等因素，实现安全与业务的平衡。安全架构与设计宜经过评审和验证，以确保其有效性。

10.4.2 安全编码

安全编码用于提高代码本身的安全性，具体的措施宜包含以下方面：

a）制定适用于企业自身的安全编码规范和指南，包括最佳实践、安全漏洞防范措施和相关代码样式案例等，并推动研发人员严格遵守相关编程语言的安全编码规范进行软件开发；

b）通过培训等一系列手段提升研发人员的安全意识，确保研发人员了解常见的安全漏洞和攻击技术，并知道如何编写安全的代码；

c）采用安全的开发框架和库，以减少漏洞引入的风险；

d）使用集成开发环境（IDE）安全编码插件和相关源码安全审计工具，以便研发人员能够及时知晓所写代码的安全漏洞；

e）对于涉及比较关键的安全相关功能，比如用户注册与认证、忘记与修改密码、文件上传下载等，应对相关源码进行人工审计；

f）定期开展代码评审和安全评估以识别和解决源代码中的安全问题。

10.4.3 构建安全管理

构建安全管理确保在代码构建阶段的安全，具体的措施宜包含以下方面：

a）在整个构建过程中，宜采取必要的安全控制措施，确保构建环境和构建输出的安全；

b）对构建服务器进行环境隔离，限制构建服务器的物理访问。宜加强构建环境的访问控制，设置强密码和多因素认证，限制访问权限和操作范围，记录访问日志和操作历史。对构建环境定期备份重要信息。宜定期扫描和修复构建服务器的安全漏洞；

c）通过数字签名、加密存储、访问控制等手段，确保构建输出的完整性、可信性和可追溯性，避免构建输出被篡改、泄漏或滥用；

d）通过日志记录及安全审计等手段，记录和监控构建过程中的关键数据。

10.4.4 安全加固/签名

安全加固/签名用于进一步提高开发环境、运行环境的安全性，具体的措施宜包含以下方面：

a）制定操作系统、Web 中间件、数据库等安全配置基线，并持续进行更新；

b）基于安全配置基线，对操作系统、Web 中间件、数据库等进行安全加固；

c）引入移动应用安全加固工具，对移动应用（包括 Android、iOS 及 H5 开发客户端）进行安全加固保护，防止反编译、动态调试、二次打包、数据窃取等风险；

d）对源代码进行混淆，以加大逆向难度；

e）采用可信代码签名证书，对面向客户的视窗可执行软件，在发布之前进行代码签名。

10.4.5 配置安全管理

配置安全用于提升应用配置的规范性，具体的措施宜包含以下方面：

a）制定配置管理安全管理规范，如配置数据访问策略、配置变更管理规范等；

b）对数据库变更脚本、环境配置脚本、应用配置等配置项进行安全管控，宜有严格的变更流程和操作记录。

10.4.6 环境安全管理

环境安全管理用于提升应用运行环境的安全性，具体的措施宜包含以下方面：

a）采取网络安全域、防火墙及入侵检测等技术，防范网络威胁；

b）定期对主机进行安全扫描，发现主机安全漏洞及安全配置问题，并及时修复主机安全漏洞及安全配置问题；

c）制定操作系统补丁更新策略，定期进行操作系统补丁更新；

d）采取访问控制及日志审计等安全措施保障运行环境安全。

10.4.7 技术栈安全管理

技术栈安全管理用于提升技术栈的安全性，具体的措施宜包含以下方面：

a）根据系统采用的技术栈，采取相应的安全控制措施。比如，对于 Web 应用，要防范常见安全漏洞；对于移动应用，要关注基于不同平台的特定安全要求；对于大数据平台，要注意各类组件的安全加固；

b）密切关注技术栈的安全动态，及时更新和修复有安全风险的版本。

10.4.8 源代码安全管理

源代码管理用于提升源代码的安全性，具体的措施宜包含以下方面：

a）制定和实施源代码安全策略，防止逆向工程和代码篡改；

b）对代码仓库进行访问控制管理，包括强身份认证和基于角色的访问控制，以确保访问者有基于其角色和工作职责的适当权限；定期评估访问权限；及时回收过期权限等；

c）使用代码签名证书完成对源代码的数字签名，确保源码的可信性和完整性；

d）保护源代码的知识产权，包括建立法律协议（如保密协议、员工合同等），以保护源代码的机密性和所有权；

e）加强终端设备安全，防止用户通过不安全的或非授权的终端设备访问源代码；

f）对源代码库中的数据，在存储和传输过程中进行加密；

g）对开发人员进行源代码风险和最佳实践的安全意识教育，鼓励员工对可疑的源代码安全事件及时报告。

10.4.9 制品库安全管理

制品库安全管理用于提升制品库的安全性，具体的措施宜包含以下方面：

a）制定制品库安全管理规范，对软件开发过程中涉及的各种制品的安全进行管控，制品包括二进制程序包、镜像文件、配置文件、脚本等；

b）对制品库的访问进行严格的权限控制和审计，防止未授权的访问和操作；

c）采用加密、数字签名等技术手段，确保制品在存储和传输过程中的完整性和机密性；

d）对于关键制品，宜有可信的来源渠道，并对其进行安全审查，以防止引入恶意代码等安全隐患；

e）在使用开源组件时，要注意开源组件的安全性，选择可信的来源，并及时更新到安全的版本；

f）宜定期对制品库进行安全审计，发现和处置安全隐患。

10.4.10 安全组件和软件开发工具包（SDK）管理

安全组件和软件开发工具包（SDK）管理用于提升 SDK 的安全性，具体的措施宜包含以下方面：

a）引入安全组件和 SDK，如密码键盘、统一认证、数据加密等；

b）选择安全组件时，要充分评估其安全性和可靠性，优先选择经过安全认证或广泛使用的成熟产品；

c）使用集成安全组件时，宜正确配置其安全参数，并与系统的其他安全机制协调一致；

d）对于对外提供的安全 SDK，宜提供清晰、易用的接口，并配备详尽的安全使用指南；

e）对安全组件和 SDK 的漏洞进行持续跟踪和修复，并及时通知和协助用户进行更新。

10.4.11 开源及软件供应链安全管理

开源及软件供应链安全管理用于管理开源软件的安全性，具体的措施宜包含以下方面：

a）建立开源软件管理规范，规范开源软件的引入、使用和管理过程；

b）引入软件成分分析工具（SCA），识别开源软件的组件及其构成和依赖关系，并识别开源组件存在的已知安全漏洞或者潜在的许可证授权问题；

c）将开源组件安全分析工具集成到 CI 持续集成流水线中；

d）制定开源组件漏洞响应和处理流程，持续监测开源组件漏洞情报，及时进行开源组件漏洞修复；

e）建立软件材料清单（SBOM），记录软件中包含的各种组件及其版本、许可证等信息，以便追溯和问题定位；

f）建立软件供应链安全事件应急响应机制，监测软件供应链的安全事件，并及时采取应对措施。

10.5 质量管理体系中的安全

10.5.1 安全左移测试

10.5.1.1 安全左移测试类型

安全左移测试宜包含以下类型：

a）开发阶段安全测试：宜利用端到端安全测试工具链对开发过程中的各种产物（代码、制品、镜像等）进行静态或动态的安全测试和合规扫描。该部分的测试嵌入到原有的开发流程中，对于开发过程中的漏洞及不合规项实现及时反馈和修复。安全测试工具链选择与本身开发流程和技术栈适配的安全测试类型，例如：静态代码安全扫描工具、开源组件安全扫描工具、容器安全扫描工具、灰盒安全测试工具、黑盒安全测试工具等安全测试工具。部分工具可以通过在流水线中集成自动化安全测试，安全测试结果自动化反馈研发处理。同时，通过制定相关质量门限来实现流水线阻断，从而保证整体安全质量；

b）上线前安全测试：在系统上线前开展上线前安全测试，包括基于测试用例安全功能和风险测试、自动化工具测试、人工渗透测试。根据不同系统类型，建议灵活设置不同的上线前安全质量标准。安全测试用例要覆盖包括身份认证安全、口令安全、访问权限安全、会话管理安全、通信安全、业务逻辑安全、数据安全、配置安全等各项测试点；

c）重要系统常态化安全测试：重要系统除了在上线前开展安全测试，定期开展安全测试以应对相关风险。其测试手段和质量标准可以参考上线前安全测试；

d）应用专项测试：有应用产品的场景，需针对应用客户端安全、应用隐私合规开展安全测试。

10.5.1.2 安全左移测试能力持续提升

安全左移测试能力的持续提升，宜包括以下方面：

a）建立工具和安全测试策略优化机制，持续降低误报率与漏报率；
b）建立安全测试用例定期更新机制，通过测试结果、外部反馈、业界技术发展定期优化安全测试用例。建立质量门禁持续跟踪和改进机制，对上述相关阶段的质量门禁实施效果进行监控和评估，并进行必要的修改和改进。

10.5.2 安全度量反馈

10.5.2.1 安全度量指标

安全度量指标的目标之一是建立跨领域、跨组织、端到端的安全度量指标体系并持续更新，具有部分预测性指标，度量指标宜与业务发展状态相关的。同时，安全度量指标覆盖整个软件开发生命周期，包括但不限于：开发交付指标、安全运营指标和安全管理指标等度量指标。

10.5.2.2 安全度量平台

安全度量平台建立的流程可以参考如下：定义安全指标、各系统数据收集、数据可视化展示、问题发现与反馈、改进效果展示。

平台的数据来源可以是多个平台的，通过平台管理数据可以保障数据时效性。如研发效能平台元数据、安全工具链扫描结果、外部录入数据、运维台账等。

10.5.2.3 其他建议事项

安全度量反馈宜包括其他建议事项如下：
a）指标完备性：跨领域、跨组织、端到端的安全度量指标体系；
b）强调阈值和预警：对重点指标设立相关预警；
c）数据时效性：度量平台架构，采集方式等合理设计；
d）问题自动推送：自动化地反馈改进，修复问题。

10.5.3 发布安全管理

10.5.3.1 上线前验证测试和风险评估

应用的部署与发布具有基础的安全检查点，对于不同类型系统制定不同的安全发布策略，提供相关的安全检查清单，阻断不安全应用发布上线。应用在各个环境中的部署与发布过程采用统一且安全的自动化工具与同等标准人工测试开展。

10.5.3.2 剩余风险确认

建立剩余风险确认机制，对于产品或系统不进行安全左移建设、测试，或因为上线时间等原因承担漏洞产生的风险而暂时不进行修复。风险接受宜经过与风险相对应级别的审批。

剩余风险报告宜包括产生剩余风险的漏洞的详情、风险接受原因、临时处置措施、业务、系统下线时间表或平替计划、最终修复漏洞的计划和时间表等要素，信息系统开发部门或信息系统运维部门等责任团队宜根据风险接受审批矩阵提请相应级别的审批。

10.6 技术运营体系中的安全

10.6.1 安全漏洞运营管理

建立安全漏洞运营管理流程，通过一系列有组织的步骤来识别、评估、报告、修复以及重新评估信息系统中的安全漏洞。安全漏洞运营通常包括以下步骤：
a）漏洞发现：通过主动扫描、被动监控或接收第三方安全公告等方式，识别系统、网络或应用

程序中存在的安全弱点；

b）漏洞验证：在发现潜在的漏洞后，进行进一步的分析以确认漏洞的真实性和可利用性；

c）漏洞评估：对已验证的漏洞进行风险评估，确定其严重性和优先级；

d）漏洞修复：根据漏洞评估的结果，采取措施修复或减轻漏洞的影响；

e）漏洞复测：在实施修复措施后，重新评估漏洞以确保修复是有效的；

f）漏洞复盘：通过总结经验教训来提升未来的漏洞管理能力。

10.6.2 安全措施的有效性验证

验证安全措施的有效性是一个与安全运营活动紧密结合的持续过程。通过对安全策略的实施效果进行定期检查，组织能够确保其防御措施能够适应不断演变的威胁环境。安全有效性验证不宜被视为一次性任务，而宜是一个动态的、持续的过程。它要求组织不断审视和更新其安全措施，以应对不断变化的安全挑战。

可以采用多种方法来检验安全措施的有效性。其中包括渗透测试、策略基线检查、安全基础设施的完整性校验等。

10.6.3 定期安全风险评估

通过定期安全风险评估，用于识别组织中的信息系统可能遭受的安全威胁，并确定这些威胁对组织运营和资产可能造成的影响。进行安全风险评估通常包括以下步骤：

a）评估准备：确定评估的范围和目标，收集必要的信息和资源；

b）风险识别：找出潜在的威胁源和脆弱点，以及可能受到威胁的资产；

c）风险分析：分析和评估风险发生的可能性以及其对组织的潜在影响；

d）风险评价：根据风险分析的结果，对风险进行排序，并确定哪些风险需要优先处理。

10.6.4 安全通报预警

建立网络安全事件通报预警机制，及时向相关人员传达潜在的安全威胁和漏洞，以便采取必要的预防和应对措施，减少安全事件对组织的影响。

安全通报预警的步骤通常包括：

a）预警发布：这是当检测到潜在的安全威胁或漏洞时，相关团队会发出通知；

b）预警处置：接收到预警的个人或团队需要评估威胁的严重性，并根据预定的响应计划采取行动；

c）预警升级或降级：根据威胁的发展和对组织的影响，相关团队可能需要调整预警的级别；

d）预警解除：当威胁被消除或风险降至可接受水平时，可以宣布预警结束。整个过程需要跨部门的协作和快速的信息共享，以确保组织能够有效地应对安全威胁。

10.6.5 安全告警监控和研判

建立安全告警监控和研判能力，对网络环境中的潜在威胁和异常行为进行识别、分析和响应。安全告警监控的目标是实时收集和分析安全事件，以便及时发现和响应安全威胁。而研判则是在监控基础上，通过专业知识和技术手段对告警进行深入分析，判断其是否为真实的安全事件，并确定相应的处置措施。

安全告警监控和研判的具体操作包括但不限于：配置和维护安全监控设备，如入侵检测系统（IDS）和入侵防御系统（IPS）；收集和分析告警日志，包括源地址、目的地址、端口、事件名称等信息；利用威胁情报和安全分析平台进行数据关联和行为分析；对疑似攻击行为进行追踪和溯源；以及制

定和执行响应计划，包括隔离受影响系统、修补漏洞和更新安全策略等。为了提高研判的准确性和效率，可以参考一些专业的安全事件研判指南和最佳实践。

10.6.6 安全事件预案和处置

建立网络安全事件预案，通过事先准备好的步骤和程序，用于识别、评估和应对网络安全事件以减少潜在的损害，保护组织的资产和声誉。

网络安全事件处置则涉及对已发生的安全事件进行管理和修复，以恢复正常运营。具体流程通常包括：建立应急响应团队，制定通信计划，识别和分类安全事件，实施初步控制措施，调查和分析事件，以及恢复受影响系统。此外，还宜包括对事件的后续审查和改进措施，以增强未来的安全防护。这个过程需要跨部门合作，确保所有关键人员都了解他们的角色和责任。

10.6.7 安全技术运营基础设施

安全技术运营基础设施包括物理安全设施、网络安全设施、数据安全设施、应用安全设施等。

安全技术运营基础设施在运维和维护过程中宜注意：

a）建立常态化检查、维护机制，及时发现及解决相关问题，确保安全技术运营基础设施正常运行；

b）针对不同类型的基础设施建立台账记录、制定合理地升级和更新安全机制，提高安全技术运营基础设施的有效性；

c）针对安全基础设施本身，宜与其他系统一致，通过合理安全配置、网络隔离及其他安全控制手段，确保系统及数据的可用性、完整性、机密性。

10.6.8 重要技术运营基础设施的安全防护

针对重要技术运营基础设施开展专项安全防护。重要技术运营基础设施包括在网络、主机、终端、应用、分支机构等各类型系统中，具有管理、监控、指挥等较高层级权限的应用系统。如堡垒机等运维操作系统、域控系统、各类云管平台、各类运维管理平台、文件传输（FTP）类文件系统、网络设备集中管理系统、无线局域网、虚拟专网（VPN）、统一身份认证系统、统一安全管控平台等。对于该类设备需要进行定期漏洞扫描和漏洞修复、弱密码整改、网络隔离等。

10.7 新技术安全管理

随着人工智能、大数据和区块链等新兴技术在证券期货业研发运营一体化体系领域的深化应用和持续发展，包括：人工智能预训练数据应用、人工智能大模型架构参数配置、人工智能生成内容监管、大数据证券期货业应用、区块链证券期货业应用等，将对该领域安全管理的目标、架构、规范和执行框架等方面都带来新的、建设性的积极影响，可以结合新技术特性、公司安全规范与安全团队建立符合公司实际的新技术安全管理规范。

10.8 安全管理工具链与平台

10.8.1 静态应用安全测试（SAST）

使用静态应用安全测试（SAST）时宜关注如下方面：

a）引入 SAST 工具，在不运行应用程序的情况下，通过分析源代码、字节码或二进制文件来识别潜在安全漏洞；

b）在软件开发的早期阶段引入 SAST，及时发现和修复安全漏洞，减少后期修复的成本和风险；

c）定期对 SAST 工具的检测规则进行更新和优化，以适应不断变化的安全威胁形势；

d）选择 SAST 工具时，宜考虑以下因素：支持的编程语言、检测规则的可定制性、与开发工具

的集成能力、报告的可读性和可操作性等。

10.8.2 软件成分分析（SCA）

使用软件成分分析（SCA）时宜关注如下方面：

a）引入 SCA 工具，识别、跟踪和管理应用程序中使用的开源组件和第三方库，识别其存在的安全漏洞及许可证合规风险；

b）定期更新 SCA 工具的漏洞库和许可证数据库，确保及时发现新的安全漏洞和合规风险。

10.8.3 交互式应用程序安全测试（IAST）

使用交互式应用程序安全测试（IAST）时宜关注如下方面：

a）引入 IAST 工具，通过插桩或代理的方式，在应用程序运行时实时监测和分析应用程序的行为，识别潜在的安全漏洞；

b）选择与应用程序技术栈兼容的 IAST 工具，确保工具能够有效覆盖应用程序的所有功能；

c）对 IAST 工具的检测结果进行分析和优化，不断提高检测的准确性和效率。

10.8.4 动态应用程序安全测试（DAST）

使用动态应用程序安全测试（DAST）时宜关注如下方面：

a）引入 DAST 工具，通过模拟攻击者的行为，对正在运行的应用程序发送各种恶意输入，并分析应用程序的响应，来识别潜在的安全漏洞；

b）选择 DAST 工具时，宜考虑以下因素：支持的技术栈、扫描范围、自动化程度、报告和分析功能、集成能力、性能和扫描速度以及支持和维护等。

10.8.5 移动应用安全测试（MAST）

引入移动应用程序安全测试工具，通过静态分析和动态测试相结合的方式，识别移动应用程序中的安全风险，包括数据存储、通信传输、身份验证、访问控制、逆向防护、安全配置、隐私保护、合规检查等。MAST 工具宜支持目标移动平台（如 Android、iOS）。

10.8.6 容器镜像安全扫描

使用容器镜像安全扫描时宜关注如下方面：

a）引入容器镜像安全扫描工具，识别容器镜像中的安全漏洞、恶意软件、配置错误等问题；

b）将容器镜像安全扫描工具集成到研发效能平台流水线，在容器镜像构建过程中自动化进行容器镜像安全扫描，及时识别容器镜像安全漏洞，并通过安全门限控制镜像是否入库。

10.8.7 混沌安全测试

使用混沌安全测试时宜关注如下方面：

a）引入混沌安全测试，通过故意引入故障和异常情况，测试系统在非预期条件下的安全性和恢复能力的方法。混沌安全测试宜模拟现实世界中的各种安全威胁和故障场景，如网络攻击、服务中断、数据损坏等，以验证系统的鲁棒性和容错能力；

b）混沌安全测试宜在隔离的测试环境中进行，避免对生产环境造成影响。

10.8.8 自动化渗透测试

使用自动化渗透测试时宜关注如下方面：

a）引入自动化渗透测试，利用自动化工具和脚本，模拟攻击者的行为，对目标系统进行全面安全测试；
b）实施自动化渗透测试时，宜选择成熟稳定的自动化测试工具和框架，自动化渗透测试宜覆盖目标系统的各个层面，如网络层、应用层、数据层等，并针对不同的安全威胁和漏洞类型设计相应的测试用例。

11 效能度量

11.1 效能度量概述

效能度量是研发运营一体化体系的重要组成部分，是基于需求管理、技术实现、质量管理、技术运营管理、安全管理全过程的效能指标进行度量，以实现度量驱动持续改进的目标。

11.2 组织保障

效能度量工作由专门团队或人员负责，主要负责度量指标的设计、度量可视化报表的设计与实现、度量数据的处理、度量报告分析与决策支持，度量驱动改进机制的维护和落地等。

11.3 度量目标与指标

11.3.1 度量目标

效能度量是有效衡量业务和技术团队表现、优化流程、提升竞争力的重要工具。特别是在研发运营建设实践中，科学的效能度量不仅能够促进团队协作，还能推动持续改进，确保系统的高效运行和业务目标的实现。下面将详细分析效能度量在证券期货业中的目标和价值，探讨度量指标的设计原则，推荐研发运营一体化建设过程中适用的度量指标，以供参考。

在证券期货业，效能度量的主要目标包括：

a）提升业务绩效，通过量化业务流程的效率，识别瓶颈，优化资源配置，提升整体业务表现；
b）保障系统稳定性，监控系统性能，确保交易平台的高可用性和低延迟，避免因系统故障导致的经济损失；
c）支持决策制定，提供数据支持，帮助管理层作出基于事实的战略决策；
d）促进持续改进，通过持续监测和反馈，推动流程和技术的不断优化，适应市场变化。

效能度量在证券期货业中的价值体现在：

a）数据驱动决策，通过客观的数据分析，减少决策的主观性，提高决策的准确性和有效性；
b）风险管理，及时发现和应对潜在的系统风险和业务风险，提升整体风险控制能力；
c）提高竞争力，通过优化业务流程和信息技术架构（对信息技术服务、信息技术组件及其交互关系进行全面描述的结构化框架），提高运营效率，增强市场竞争力；
d）增强透明度，为各级管理层和团队成员提供清晰的绩效指标，促进信息共享和透明管理。

11.3.2 度量指标

在证券期货业，软件交付过程的效能度量对于确保系统的高可用性、稳定性、安全性以及满足业务需求至关重要。度量指标是衡量、评估实施结果重要的标准和参考依据，在研发运营一体化建设过程中可以对关键阶段设置度量指标，包括每个指标的目标、定义、价值、计算公式及展现形式。

11.3.3 度量指标设计的基本原则

度量指标设计的基本原则包括相关性、可衡量性、可操作性、简洁性、实时性等方面，具体描

述如下：

a）相关性（Relevance）

度量指标宜与业务目标和技术目标紧密相关，确保所测量的内容能够直接反映出团队或系统的表现对业务成功的影响。例如，在高频交易系统中，延迟时间是一个关键指标，因为它直接影响交易的执行速度和准确性；

b）可衡量性（Measurability）

指标宜具体且可量化的，能够通过数据准确测量。避免使用模糊或主观的指标，如“代码质量好”，而宜使用具体的数值指标，如“代码覆盖率达到80%”；

c）可操作性（Actionability）

度量指标宜具备可操作性，即当指标偏离预期时，团队能够采取具体的行动进行调整和改进。例如，如果系统的CPU使用率持续超过90%，运维团队宜立即分析原因并优化资源配置；

d）简洁性（Simplicity）

指标设计宜尽量简洁，避免过多复杂的指标导致信息过载。重点关注关键绩效指标（KPI），确保管理层和团队能够轻松理解和应用。例如，选择“部署频率”和“平均恢复时间”作为核心指标，而不是追踪所有可能的指标；

e）实时性（Timeliness）

在系统运行环境中，实时或近实时的度量指标尤为重要，以便快速响应系统状态和业务需求的变化。例如，实时监控交易系统的响应时间，能够及时发现并解决延迟问题。

11.4 度量可视化分析与决策

11.4.1 明确目标与需求

11.4.1.1 业务目标对齐

度量系统的建设宜与证券公司的业务目标和战略方向高度对齐。明确软件交付过程中的关键目标，如提升开发效率、保证软件质量、缩短交付周期等，为度量指标的选择和系统设计提供指导。

11.4.1.2 利益相关方需求分析

识别和分析不同利益相关方（如开发团队、项目管理团队、质量保证团队、业务部门、客户等）的需求，确保度量系统能够满足各方的关切点和信息需求。

输出明确的项目目标和各利益相关方的需求文档。

11.4.2 数据收集与集成

11.4.2.1 数据源识别

确定内部和外部的数据源，如版本控制系统、CI/CD工具、项目管理工具、CRM系统等。

11.4.2.2 数据集成与清洗

使用数据清洗工具将数据从各源系统提取、转换并加载到统一的数据仓库中。确保数据的一致性、准确性和完整性，处理缺失值和异常值。

输出集成的数据仓库和清洗后的高质量数据集。

11.4.3 数据分析与可视化

11.4.3.1 可视化数据分析

数据分析是将收集到的原始数据转化为有意义的信息和洞察的过程。首先，明确分析目标，确保数据分析与业务需求和战略目标一致。接着，进行数据清洗，处理缺失值、异常值和重复数据，确保数据的准确性和完整性。然后，使用适当的统计方法和工具（如描述性统计、回归分析、数据可视化等）对数据进行探索和建模，以识别模式、趋势和关联关系。通过可视化仪表盘和报告，将分析结果以直观的方式呈现，便于决策者理解和应用。最后，根据分析结果制定和优化业务策略，持续监控指标，进行迭代改进，确保数据驱动的决策能够有效支持业务目标的实现。

11.4.3.2 设计可视化度量仪表盘与报告

可视化度量仪表盘与报告宜包含以下元素：

a）图表与图形：使用图表（如折线图、柱状图、饼图等）直观展示数据；

b）表格：展示精确的数据和比较；

c）图像与图标：增强视觉效果，帮助解释复杂信息；

d）输出各职能部门专属的图表、图形和直观易懂的数据展示。

11.4.3.3 添加叙述与解释

添加叙述与解释宜包含以下元素：

a）数据解读：对图表和数据进行解释，说明其意义和影响；

b）业务洞察：基于数据分析提供业务洞察和发现；

c）行动建议：根据分析结果提出具体的行动建议；

d）输出详尽的文字解释和分析，以及对应数据关联的业务目标解释。

11.4.3.4 确保可读性与易用性

确保可读性与易用性宜包含以下方面：

a）排版与布局：使用清晰的排版，合理分布内容和图表；

b）字体与颜色：选择易读的字体和协调的颜色；

c）输出视觉上整洁、易于阅读的报告，提供友好的阅读体验；

d）自动化报告生成，配置数据分析模块功能，自动生成定期报告（如周报、月报），减少手动工作，提高效率；

e）为不同层级职能部门设计专属的仪表盘，实时展示关键指标。生成定期的业务报告，汇总关键指标的变化趋势和分析结果。

11.4.3.5 高级分析

高级分析宜包含以下方面：

a）进行趋势分析、关联分析和预测分析，提供更深入的业务洞察；

b）输出各职能部门专属的仪表盘、定期业务报告、高级分析报告。

11.4.3.6 质量的度量分析与反馈

分析度量是通过定性和定量方法，对软件生命周期产研各阶段进行质量分析与评估，通过建立不同维度不同层级的质量分析指标，找出质量瓶颈点和效率阻塞点，促进过程改进和效率提升。通过合理的度量体系使研发产出更直观，推动研发流程的改进，质量的度量分析方法参考表 2。

度量反馈是通过对度量指标数据的收集、清洗、整理、计算等过程收集度量结果数据，对结果数据进行趋势分析、下钻分析、相关性分析、累积流图分析、流效率分析、流负载分析等度量分析方法来分析质量问题、指导团队改进。

表 2　质量的度量分析方法

度量分析方法	说明
趋势分析	趋势分析是随时间推移的变化获取度量结果的变化值，对度量结果的趋势分析比绝对值分析更有价值。
下钻分析	下钻分析一般包括按软件交付阶段下钻、按聚合维度下钻、按制品进行下钻等。下钻分析可获得微观数据，从表象到根因逐层排查问题，找到影响质量的瓶颈点。
相关性分析	影响质量的因素很多，通过找到各因素之间的相关性关系并进行分析，找到真正影响质量的因素进行持续干预。
累积流图分析	累积流图的横轴是日期，纵轴是工作项数量。把工作项每个阶段的流动情况，按照时间维度绘制出来累积流图，来识别当前交付的进展、瓶颈、问题和需要进一步探查的内容。
流效率分析	流效率是工作项在流管道中处于活跃工作状态的时间与总交付时间的比率，通过明确各个阶段的准入准出，规范各个团队的操作规范，来计算流效率的度量数据。
流负载率分析	通过分析在制品数量得到流负载率，在制品数量多则流负载率高，流负载率过高会导致后续的交付效率下降、交付周期变长，流负载率分析可以帮助找到阻塞的工作项。

有效的度量反馈能推动研发团队改进，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。

度量可视化是将度量指标和计算结果展示出来，使相关人员能直观地感知度量数据和结果，合理的度量指标能客观反馈研发过程是否高效，组织内各角色成员能力素质的高低，产品质量的好坏以及潜在的风险。度量结果的计算需要考虑数据的时效性以及数据覆盖的范围。

度量改进是通过对产品、系统、流程等进行持续不断地测试以发现潜在的问题和改进点，基于度量结果提供快速准确直观的趋势和质量数据，指导研发团队优化改进，推动企业数字化智能化转型与决策的进程，组织从度量反馈中不断汲取经验教训，并不断优化软件全生命周期中的各项活动，实现度量数据驱动持续改进。

11.4.3.7　技术运营度量分析与预警

11.4.3.7.1　技术运营度量分析与预警概述

为了对信息系统进行运营与监测，确保系统满足服务需求，通过主动收集基础资源、信息系统和业务相关信息进行度量分析，发现潜在系统问题并预警的过程。

11.4.3.7.2　建立容量监测指标

容量监测指标宜与容量规划相匹配，可从系统资源监控、性能指标监控方面进行监测：

a）系统资源监控：

——CPU 利用率监控；

——内存使用监控；

——磁盘空间使用率监控：监控磁盘空间的使用，避免存储空间不足导致的系统故障；

——网络带宽监控；

——应用程序性能监控。

b）性能指标监控与度量：

——设定并监控关键性能指标，如响应时间、吞吐量、数据库查询时间等；

——使用专门的监控工具来度量和跟踪这些性能指标。

11.4.3.7.3 建立容量预警基线

容量预警基线宜结合容量规划、业务特点进行设置，可参考监管要求、系统设计容量、历史峰值、压力测试极限值、平均值、静态经验阈值、历史同比与环比值、历史动态基线等设定预警基线，可形成以下预警与报警机制：

a）设定性能阈值，当性能达到或超过这些阈值时触发预警或报警；

b）报警可以通过邮件、短信或特定的监控界面通知相关人员。

11.4.4 报告生成与传播

11.4.4.1 自动化报告生成

配置数据分析工具，自动生成定期报告（如周报、月报）。

11.4.4.2 信息共享与传播

通过内部网络、邮件或协作工具分享报告。组织定期的报告会议，向管理层和团队展示分析结果，包括自动化生成的定期报告、共享的业务洞察与分析结果。

11.4.5 决策制定

11.4.5.1 数据驱动的决策制定

度量系统提供的数据和分析结果使决策过程更加科学和精准。通过实时监控和历史数据分析，证券公司可以识别软件交付过程中的问题和机会，作出数据驱动的决策。

11.4.5.2 优化软件交付流程

通过分析开发效率和软件质量指标，识别开发过程中的瓶颈和低效环节，优化开发流程。例如，如果代码审查通过率低，可能需要改进代码审查流程或加强代码质量培训。

11.4.5.3 提升软件质量

通过缺陷密度和缺陷修复时间等指标，监控软件质量，及时发现和修复缺陷，降低生产环境中的故障率。通过提高测试覆盖率，确保软件功能的全面性和可靠性。

11.4.5.4 加强风险管理

通过监控风险管理指标，及时识别和应对潜在风险，降低业务运营中的风险暴露。度量系统能够提供实时的风险数据支持，帮助风险管理团队做出快速反应。

11.4.5.5 提升团队协作与效率

通过团队协作指标和开发效率指标，评估团队的协作效果和工作效率。根据分析结果，优化团队结构、改进沟通方式和提升团队成员的技能，提升整体开发效率和协作水平。

11.5 度量驱动改进

11.5.1 度量驱动改进概述

度量驱动改进（Metric-Driven Improvement）是一种系统化的方法，通过定义、收集、分析和应用关键指标来持续优化软件交付流程、提高效率和提升产品质量。一个全面的度量驱动改进解决方案设计，涵盖明确改进目标、建立基线与设定目标、数据收集与管理、数据分析与洞察、实施改进措施、改进问题跟踪与验收、监控与反馈等关键步骤。

11.5.2 明确改进目标

度量驱动改进的首要步骤是明确改进目标，确保所有改进活动与企业的业务战略和团队需求紧密对齐。通过与利益相关者（如管理层、开发团队、质量团队和运维团队）沟通，识别具体且可衡量的目标，采用具体、可衡量、可实现、相关、时限原则来设定目标。例如，目标可以包括将部署频率从每周一次提升至每天一次，或将缺陷密度从每千行代码 5 个减少到 2 个。明确的改进目标为后续的度量和优化提供了清晰的方向，确保团队集中资源和努力在最关键的改进领域。

11.5.3 建立基线与设定目标

在明确改进目标之后，下一步是建立基线并设定具体的改进目标。通过分析某段时间内的关键指标数据，确定各项指标的当前表现和历史趋势，作为评估改进效果的参考点。同时，参考行业标准和最佳实践，设定合理且具有挑战性的目标。例如，将交付周期从两周缩短至一周，或将系统 MTTR 从 30 分钟减少至 10 分钟。建立基线和设定目标不仅帮助团队了解当前的绩效水平，还为持续改进提供了明确的衡量标准。

11.5.4 数据收集与管理

可靠的数据收集与管理是度量驱动改进的基础，确保数据的准确性和及时性至关重要。首先，识别与软件交付过程相关的各类数据源，如 CI/CD 工具、项目管理工具、监控工具和日志管理系统。通过集成插件或 API 等实现自动化数据采集，并建立统一的数据仓库来整合和存储数据。实施严格的数据清洗和预处理流程，处理缺失值、异常值和数据噪声，确保数据的高质量和一致性，为后续的分析提供可靠的数据基础。

11.5.5 数据分析与洞察

在数据收集与管理之后，深入的数据分析是识别问题、趋势和改进机会的关键。首先进行描述性分析，了解当前状况和历史趋势，识别指标的变化模式。接着，通过关联分析和因果推断，找出问题的根本原因，识别关键影响因素。此外，利用时间序列分析和机器学习模型进行预测性分析，预测未来的交付趋势和潜在问题。最终，通过规范性分析，基于分析结果提出具体的改进建议和优化措施。使用统计分析工具和可视化工具展示分析结果，提供有价值的业务洞察，支持数据驱动的决策。

11.5.6 实施改进措施

基于数据分析的洞察，制定和执行具体的改进措施，以实现预定的改进目标。首先，制定详细的行动计划，明确具体的改进措施、责任人和时间表。例如，优化 CI/CD 流水线以缩短构建时间，或增加自动化测试覆盖率以降低缺陷率。然后，通过流程再造和自动化工具提升流程效率，减少手动操作，提高团队的工作效率。此外，引入新工具和技术，提升系统性能和稳定性，同时为团队提供必要的培训和支持，确保改进措施的有效实施。通过优化流程、技术升级和团队培训，确保改进措施能够切实提升软件交付的效率和质量。

11.5.7 改进问题的跟踪与验收

在实施改进措施的过程中，有效的问题跟踪与验收机制至关重要，以确保改进措施的执行效果和持续优化。利用项目管理工具记录和跟踪改进过程中遇到的问题和挑战，确保每个问题都有明确的解决路径和责任人。设定具体的验收标准和成功指标，确保改进措施达到预期效果。例如，确保部署频率提升至每天一次，或将 MTTR 减少至 10 分钟以内。定期召开评审会议，评估改进措施的执行进度和效果，通过数据对比和用户反馈验证改进成果。同时，记录改进过程和解决方案，建立知识库，支持未来的改进活动，确保持续优化和最佳实践的应用。

11.5.8 监控与反馈

持续的监控与反馈机制是度量驱动改进的最后环节，确保改进措施的长期效果和持续优化。通过实时监控工具跟踪关键指标的表现，及时发现和响应异常情况。定期收集团队和利益相关者的反馈，通过团队会议、调查问卷等方式了解改进措施的实际效果和潜在问题。基于监控数据和反馈，持续优化改进措施，确保软件交付过程始终保持高效、可靠和高质量。同时，培养持续改进的文化，通过激励机制、透明沟通和知识共享，确保度量驱动改进成为团队日常工作的核心部分，推动企业实现持续改进和卓越运营。

11.6 度量指标实践参考

11.6.1 研发过程相关指标

11.6.1.1 需求响应时间（Response Time to Requirements）

目标：评估需求从提出到被响应和开发的时间；

定义：从需求提出到开发团队开始实施的时间差；

价值：反映团队对业务需求的响应速度，帮助识别需求管理和沟通的瓶颈；

计算公式：需求响应时间=需求实施开始日期-需求提出日期；

展现形式：以平均天数或工作日的形式展现，通常通过甘特图、折线图等展示不同需求的响应时间。

11.6.1.2 开发周期（Development Cycle Time）

目标：评估从需求分析开始到软件交付的总时间；

定义：从需求分析阶段开始，到代码交付和上线的整个开发过程所耗费的时间；

价值：衡量团队在整个开发过程中的工作效率，帮助识别瓶颈和提升开发效率；

计算公式：开发周期=交付日期-需求分析完成日期；

展现形式：以时间段形式（如天、周、月）展示，可以用甘特图、进度条或折线图展示每个项目的开发周期。

11.6.1.3 代码提交频率（Code Commit Frequency）

目标：评估开发人员的编码活动频率，反映开发的活跃度；

定义：开发人员向版本控制系统提交代码的次数；

价值：高频次的提交通常表明开发过程更加稳定，减少了冲突和回滚的可能性，同时提高了代码质量的可追溯性；

计算公式：代码提交频率=总提交次数/时间周期；

展现形式：以图表（如柱状图、折线图）展示不同时间周期内的提交次数，帮助追踪开发活动的频率。

11.6.1.4 缺陷密度（Defect Density）

目标：衡量每千行代码中出现的缺陷数量，反映代码质量；

定义：每千行代码中检测到的缺陷数；

价值：该指标帮助团队识别代码中的质量问题，过高的缺陷密度通常意味着开发过程中存在严重的质量问题；

计算公式：缺陷密度=（缺陷总数/总代码行数）×1000；

展现形式：以每千行代码为单位的数值展示，常用柱状图或折线图展示不同模块或项目的缺陷密度。

11.6.1.5 代码覆盖率（Code Coverage）

目标：衡量单元测试对代码的覆盖程度；

定义：测试代码中覆盖的源代码行数与总代码行数的比例；

价值：高代码覆盖率有助于确保代码质量，发现潜在的缺陷；

计算公式：代码覆盖率=（已测试代码行数/总代码行数）×100%；

展现形式：以百分比的形式展示，通常通过仪表盘或报告中显示，也可以用折线图来展示代码覆盖率随时间的变化。

11.6.1.6 缺陷修复时间（Defect Fix Time）

目标：评估缺陷被发现到修复完成所需的时间；

定义：从缺陷报告到修复完成的时间间隔；

价值：反映开发团队响应问题的速度，帮助优化问题处理流程；

计算公式：缺陷修复时间=缺陷修复时间-缺陷报告时间；

展现形式：以天数为单位显示缺陷修复时间，使用柱状图或折线图展示不同时间段的修复时间。

11.6.1.7 持续集成成功率（CI Success Rate）

目标：评估整体持续集成结果的成功率，反应代码变更触发自动化集成后，所有测试（如集成测试等）均通过所占的比率；

定义：持续集成过程中，能够成功完成完整持续集成的数量与占集成总次数的比例；

价值：高成功率表示团队的构建稳定性较高，系统集成较为顺畅。失败率高则可能指示存在代码质量问题或集成流程问题；

计算公式：CI 成功率=（成功集成次数/总集成次数）×100%；

展现形式：以百分比形式展示成功率，通常以折线图、柱状图或饼图形式呈现。

11.6.1.8 持续构建成功率（Continuous Build Success Rate）

目标：持续构建是持续集成的一部分，评估持续构建过程的稳定性和成功率，能够反应代码变更经过预设步骤（如编译、打包等）均通过，无错误或失败的比率；

定义：持续集成过程中成功构建的次数占总构建次数的比例；

价值：高构建成功率表明持续集成流程稳定，代码质量高，减少集成过程中的问题和返工；

计算公式：持续构建成功率=（成功构建次数/总构建次数）×100%；

展现形式：

a）折线图：展示不同时间段的构建成功率趋势；

b）柱状图：对比不同项目或团队的构建成功率。

11.6.2 质量与安全管理相关指标

11.6.2.1 需求交付周期（Requirement Delivery Cycle Time）

目标：衡量从需求提出到需求完成交付的周期时间；

定义：从需求提出，到完成开发、测试、验收、上线的时间周期；

价值：需求交付周期可以反映研发团队响应市场需求的速度，短的交付周期能够帮助企业更快响应市场变化，提高竞争力；

计算公式：需求交付周期=需求交付完成时间-需求提出时间；

展现形式：以天、周或月为单位展示，通常使用甘特图、折线图或柱状图展示不同需求的交付周期。

11.6.2.2 产研交付周期（Product R&D Delivery Cycle Time）

目标：衡量从产品研发开始到最终交付（包括测试、上线等）的整个周期时间；

定义：从需求被产研团队确认，到完成开发、测试、验收、上线的时间周期；

价值：该指标帮助评估研发团队的整体效率，缩短产研交付周期可以加快产品上线和市场反馈，提高企业的响应能力；

计算公式：产研交付周期=产品交付时间-产品需求明确时间；

展现形式：以天、周或月为单位展示，可以通过甘特图、折线图或柱状图展示不同项目的产研交付周期，帮助分析不同阶段的时间消耗。

11.6.2.3 需求吞吐量（Requirement Throughput）

目标：衡量团队在特定时间内处理和交付的需求数量；

定义：统计周期内交付的需求个数/统计周期，即单位时间内交付的需求个数；

价值：需求吞吐量直接反映团队处理需求的能力。高吞吐量表示团队能够快速有效地交付需求，而低吞吐量则可能表示需求管理、开发或资源调度上的瓶颈；

计算公式：需求吞吐量=成功交付的需求数/时间周期；

展现形式：以数量为单位展示，通常使用柱状图、折线图或积累图展示每个时间周期内的需求吞吐量。

11.6.2.4 线上缺陷密度（Online Defect Density）

目标：衡量产品上线后在生产环境中出现的严重缺陷的密度；

定义：统计周期内线上或单个版本严重级别缺陷数量/需求个数；

价值：该指标能反映上线后产品的质量水平。较高的线上缺陷密度表明产品在上线后存在质量问题，可能需要迅速修复，避免影响用户体验或业务正常运行；

计算公式：线上缺陷密度=严重级别缺陷数量/需求个数；

展现形式：以每个需求为单位，常用柱状图、折线图等可视化形式展示不同版本或不同模块的缺陷密度。

11.6.2.5 上线变更成功率（Deployment Success Rate）

目标：衡量在生产环境中部署变更时成功的比例；

定义：上线部署成功，且没有导致服务受损、降级或需要事后补救的比例；

价值：该指标能反映发布流程的成熟度、稳定性和风险管理能力。高成功率表明变更管理流程规范，系统更稳定；

计算公式：上线变更成功率=（成功上线次数/总上线次数）×100%；

展现形式：以百分比形式展示，通常使用饼图或柱状图展示每次发布的成功与失败比例。

11.6.2.6 部署频率（Deployment Frequency）

目标：衡量软件部署到生产环境的频率；
定义：在一定时间内，有效部署次数；
价值：高频次的部署表明团队能够快速响应需求和修复问题，支持持续交付的能力；
计算公式：部署频率=总部署次数/时间周期；
展现形式：以次数为单位展示，通常通过柱状图或折线图展示不同时间段内的部署频率。

11.6.2.7 提测变更前置时间（Test Change Lead Time）

目标：衡量从提测到变更实施的时间跨度；
定义：制品提交到制品库到功能上线的时长；
价值：有助于评估团队响应提测频率变化的能力，缩短前置时间可以加快响应速度；
计算公式：提测变更前置时间=变更实施开始时间-制品送测时间；
展现形式：以天或工作日为单位，通常以折线图或甘特图展示变更前置时间随时间的变化。

11.6.2.8 工作项吞吐量（Work Item Throughput）

目标：衡量团队在一定时间内完成的工作项数量；
定义：单位时间内流经交付管道的工作项数量；
价值：该指标帮助评估团队的工作效率和生产力，吞吐量越高，表示团队处理工作的能力越强；
计算公式：工作项吞吐量=完成的工作项数/时间周期；
展现形式：以数量为单位，通常使用柱状图或折线图展示不同时间段内的工作项吞吐量。

11.6.2.9 在制品数量（Work in Progress-WIP）

目标：衡量在任何时刻团队正在处理的工作项数量；
定义：在交付管道中已经开始、尚未完成的工作项的数量；
价值：有助于识别工作负载和潜在的瓶颈，过高的在制品数量可能导致流程瓶颈或开发延迟；
计算公式：在制品数量=当前进行中的工作项数；
展现形式：以数量为单位展示，通常使用柱状图或堆积条形图来显示每个阶段的在制品数量。

11.6.2.10 流效率（Flow Efficiency）

目标：衡量工作项从开始到完成所需的时间与在制时间之间的比例；
定义：在交付管道中工作项处于活跃工作状态的时间与总交付时间的比率；
价值：高流效率表明工作流程更加顺畅，低流效率表明存在阻塞和延迟；
计算公式：流效率=活跃工作时间/总工作时间×100%；
展现形式：以百分比形式展示，常用折线图或柱状图展示流效率的变化趋势。

11.6.2.11 类深度（Class Depth）

目标：衡量类的继承层次深度；
定义：类的继承深度，从叶子节点到根节点的最大深度，从对象层次结构的顶部开始的继承层数；
价值：深层次的继承结构可能导致代码的可维护性差，复杂度较高。减少类深度有助于提高代码的可读性和易于维护；
计算公式：类深度=类的继承层级数；

展现形式：以数字形式展示，通常用于静态代码分析报告中。

11.6.2.12 耦合数（Coupling）

目标：衡量类与其他类之间的依赖关系；
定义：调用其他类的成员方法或实例变量，对象类的耦合程度；
价值：高耦合性通常会导致系统的灵活性和可维护性下降，优化耦合度可以降低系统的复杂性；
计算公式：耦合数=直接依赖于该类的数量；
展现形式：以数字形式展示，常用静态代码分析工具生成报告展示。

11.6.2.13 聚合数（Aggregation）

目标：衡量类之间的聚合关系；
定义：统计类中包含其他类成员变量、成员方法的数量；
价值：低聚合度有助于提高系统的模块化，减少相互依赖，使系统更易于扩展和维护；
计算公式：聚合数=类拥有的对象数量；
展现形式：以数字形式展示，常用于静态分析报告中。

11.6.2.14 模块复杂度（Module Complexity）

目标：衡量一个软件模块的复杂程度；
定义：类的每个成员方法的圈复杂度之和；
价值：复杂度高的模块更难维护，容易引发缺陷。降低模块复杂度有助于提升系统的可维护性和可扩展性；
计算公式：模块复杂度=模块中方法数+类之间依赖数；
展现形式：以数字或分数形式展示，通常使用热图、条形图等可视化方式呈现不同模块的复杂度。

11.6.2.15 自动化测试覆盖率（Automated Test Coverage）

目标：衡量自动化测试覆盖的代码或功能比例，确保代码质量和功能完整性；
定义：自动化测试覆盖的代码行数或功能点数占总代码行数或功能点数的比例；
价值：高测试覆盖率有助于提前发现和修复代码缺陷，提升代码质量和系统稳定性，减少生产环境中的故障率；
计算公式：自动化测试覆盖率=（被自动化测试的代码行数/总代码行数）×100%；
展现形式：
a）折线图：展示测试覆盖率随时间的变化趋势；
b）堆叠柱状图：对比不同模块或功能的测试覆盖率。

11.6.2.16 代码质量指标（Code Quality Metrics）

目标：评估代码的复杂度、可维护性和健壮性，确保代码符合质量标准；
定义：包括代码复杂度、代码重复率、静态代码分析得分等多个方面的综合指标；
价值：高代码质量降低了维护成本，提高了系统的可扩展性和稳定性，减少了技术债务；
计算公式：
a）代码复杂度=每个函数的平均圈复杂度；
b）代码重复率=（重复代码行数/总代码行数）×100%；
c）静态代码分析得分=工具通过计算规则给出的综合评分。

展现形式：

a）仪表盘：实时显示各类代码质量指标；

b）热力图：展示不同模块的代码质量分布。

11.6.2.17 安全漏洞检测（Security Vulnerability Detection）

目标：识别和修复系统中的安全漏洞，保障系统安全性，防止数据泄露和攻击；

定义：自动化工具扫描出系统中的安全漏洞数量和严重程度；

价值：及时发现和修复安全漏洞，保障系统安全，避免潜在的安全风险和法律合规问题；

计算公式：安全漏洞数量=不同严重程度的漏洞总数；

展现形式：

a）堆叠柱状图：按严重程度分类展示漏洞数量；

b）折线图：展示不同时间段的漏洞检测和修复情况。

11.6.2.18 变更失败率（Change Failure Rate）

目标：评估变更引入的故障比例，反映变更过程的稳定性和质量；

定义：部署到生产环境后出现故障或回滚的变更次数占总变更次数的比例；

价值：低变更失败率表明变更过程稳定，减少因变更引发的系统故障，提升系统可靠性；

计算公式：变更失败率=（失败变更次数/总变更次数）×100%；

展现形式：

a）饼图：展示成功变更与失败变更的比例；

b）折线图：展示变更失败率随时间的变化趋势。

11.6.2.19 缺陷密度（Defect Density）

目标：评估每千行代码（KLOC）中的缺陷数量，反映代码质量和开发过程的有效性；

定义：每千行代码中的缺陷数量；

价值：低缺陷密度表明代码质量高，开发过程成熟，减少后期维护成本和系统故障风险；

计算公式：缺陷密度=缺陷总数/（总代码行数/1000）；

展现形式：

a）条形图：展示不同模块或团队的缺陷密度；

b）折线图：展示缺陷密度随时间的变化趋势。

11.6.2.20 自动化部署成功率（Automated Deployment Success Rate）

目标：衡量自动化部署流程的稳定性和可靠性，确保自动化部署的成功率；

定义：自动化部署过程中成功部署的次数占总部署次数的比例；

价值：高自动化部署成功率表明部署流程稳定，减少人为干预和错误，提升部署效率和系统稳定性；

计算公式：自动化部署成功率=（自动化成功部署次数/总部署次数）×100%；

展现形式：

a）饼图：展示成功部署与失败部署的比例；

b）折线图：展示不同时间段的自动化部署成功率趋势。

11.6.3 技术运营领域相关指标

11.6.3.1 部署频率（Deployment Frequency）

目标：衡量软件变更被成功部署到生产环境的频率，反映团队的交付速度和敏捷性；

定义：单位时间内（如每周、每月）成功部署到生产环境的次数；

价值：高部署频率意味着团队能够快速响应业务需求和市场变化，提升竞争力。同时，频繁的小规模部署有助于降低每次部署的风险，便于快速发现和修复问题；

计算公式：部署频率=单位时间内的成功部署次数；

展现形式：

a）折线图：展示不同时间段的部署次数趋势；

b）柱状图：对比不同周期（如周、月）的部署频率。

11.6.3.2 部署失败率（Deployment Failure Rate）

目标：评估部署过程中出现故障或需要回滚的比例，反映部署过程的稳定性和可靠性；

定义：单位时间内失败部署次数与总部署次数的比率；

价值：低部署失败率表明部署过程成熟，系统稳定性高，减少因部署失败带来的业务中断和经济损失；

计算公式：部署失败率=（失败部署次数/总部署次数）×100%；

展现形式：

a）饼图：展示成功部署与失败部署的比例；

b）折线图：展示部署失败率随时间的变化趋势。

11.6.3.3 平均故障恢复时间（MTTR）

目标：衡量系统从故障发生到恢复正常运行所需的平均时间，反映团队的响应和恢复能力；

定义：系统从故障发生到完全恢复正常服务所需的平均时间；

价值：较短的 MTTR 意味着团队能够快速响应和解决问题，降低系统停机时间，保障业务连续性和用户体验；

计算公式：MTTR=故障恢复总时间/故障次数；

展现形式：

a）折线图：展示不同时间段的 MTTR 变化；

b）柱状图：对比不同周期（如月、季度）的 MTTR。

11.6.3.4 服务可用性（Service Availability）

目标：评估系统或服务在规定时间内的正常运行比例，确保关键业务系统的高可用性；

定义：系统或服务在特定时间内正常运行的时间比例；

价值：高服务可用性是证券交易系统的核心要求，保障交易活动不受系统故障影响，提升用户信任度和满意度；

计算公式：服务可用性=[（总时间-停机时间）/总时间]×100%；

展现形式：

a）仪表盘：实时显示服务可用性百分比；

b）折线图：展示不同时间段的可用性趋势。

11.6.3.5 用户满意度（User Satisfaction）

目标：衡量用户对系统和服务的满意程度，指导产品和服务的改进方向；

定义：通过用户调查或反馈收集的满意度评分；

价值：高用户满意度反映系统和服务满足用户需求，提升用户忠诚度和市场竞争力；

计算公式：用户满意度=（满意用户数/总用户数）×100%；

展现形式：

a）柱状图：展示不同时间段的用户满意度评分；

b）饼图：展示满意用户与不满意用户的比例。

11.6.3.6 应用响应时间（Application Response Time）

目标：监控和优化应用的响应速度，提升用户体验和系统性能；

定义：应用在处理请求时的平均响应时间；

价值：较短的响应时间提升用户体验，确保交易系统的实时性和可靠性，避免因延迟导致的交易失败或用户流失；

计算公式：应用响应时间=请求处理总时间/请求次数；

展现形式：

a）折线图：展示不同时间段的响应时间变化；

b）热力图：展示不同功能模块的响应时间分布。

11.6.3.7 系统性能指标（System Performance Metrics）

目标：监控系统的整体性能，确保系统能够高效处理交易和业务需求；

定义：包括 CPU 使用率、内存使用率、磁盘 I/O、网络吞吐量等多个方面的综合性能指标；

价值：全面的性能监控有助于及时发现系统瓶颈，优化资源配置，提升系统响应速度和处理能力，确保交易系统的高效运行；

计算公式：

a）CPU 使用率=（CPU 使用时间/总时间）×100%；

b）内存使用率=（已用内存/总内存）×100%；

c）磁盘 I/O=每秒读写次数；

d）网络吞吐量=每秒传输的数据量。

展现形式：

a）仪表盘：实时展示各项性能指标的当前值；

b）折线图：展示不同时间段的性能指标变化趋势；

c）热力图：展示不同服务器或组件的性能状态。

12 一体化协同

12.1 需求管理与技术实现的协同

12.1.1 以需求为线索贯穿交付过程

需求管理与技术实现的协同主要体现为技术实现过程要能够与需求提出以及需求变更进行协同。尤其在项目的实施过程中，需求变更是一个不可避免的现象。为了应对这一挑战，技术实现过程宜具备一定的灵活性和适应性，能够及时响应需求的变更。这要求项目团队遵循严格的变更管理流程，确保每一次需求变更都能得到及时、有效地沟通和处理。同时，技术实现要适应需求变更的情况，技术团队需要与需求相关方积极沟通，共同评估变更对技术实现的影响，并制定相应的调整策略。对于监管合规类需求的协同，需要技术团队主动地、积极地了解监管合规的最新动态，与合规人员

紧密配合，及时推进监管合规类需求的进度。对于业务类需求的协同，需要技术团队全面细致地评估需求变更对于业务系统稳定运行的影响，分析潜在的风险和应对措施，尤其要向业务人员揭示和确认信息安全风险、敏感数据外泄风险等不易被业务人员关注的风险；需求变更对于上下游系统的影响，不局限于技术类需求的协同，对于关键的需求变更都建议评估，对上下游系统的影响、对数据完整性的影响、对后期运维成本和操作难度的影响。

最佳实践：

a）需求即代码（Requirements as Code）：将需求文档以代码形式管理，与代码仓库集成，确保需求变更可追溯；使用版本控制跟踪需求变更历史，与代码提交关联；

b）用户故事映射与持续分解：通过敏捷方法（如用户故事映射）将高层需求分解为可执行的任务，确保每个迭代周期内需求明确；在工具中维护用户故事与开发任务的链接，实现需求与实现的双向追溯；

c）自动化需求验证：将验收标准转化为自动化测试用例，嵌入 CI/CD 管道，确保每次代码提交符合需求；通过测试结果反馈需求实现状态，及时调整开发优先级；

d）持续反馈与迭代调整：通过持续交付获取用户反馈，快速调整需求优先级和实现计划；利用 A/B 测试或功能开关逐步验证需求实现效果。

12.1.2 需求与工程化对接

需求管理与代码管理的协同是软件开发生命周期中，通过系统化的方法和工具，将业务需求的定义、分解、变更与代码的编写、版本控制、测试及部署紧密关联，确保需求的高效实现和全程可追溯。其核心在于建立需求与代码之间的双向追溯链路，使得每个需求都能对应到具体的代码变更，同时代码的修改也能反向关联到原始需求。这种协同通过自动化工具链（如需求管理系统与版本控制系统的集成）和标准化流程（如需求即代码、自动化测试验证）实现，从而提升交付质量、加快迭代速度，并满足金融等行业对合规性和安全性的严格要求。主要体现在以下 3 点：

a）需求与代码的严格追溯性：每个业务需求宜与对应的代码变更建立可追溯的关联，确保需求实现过程全程透明。使用需求管理系统与代码仓库集成，通过用户故事标识码或需求标识码标记代码提交。在代码提交信息中引用需求标识码进行关联；

b）版本一致性：需求文档的版本宜与代码版本保持同步，任何需求变更需触发代码评审和版本更新。将需求文档以“代码化”形式管理，并纳入代码仓库，与代码同步版本控制；在 CI/CD 流水线中嵌入需求版本检查，确保部署的代码版本与已批准的需求文档版本匹配；

c）安全与合规审计：所有需求变更和代码修改需记录审计线索，以满足金融监管的合规要求。签名提交（Signed Commits）确保代码变更的完整性。定期生成需求与代码的关联审计报告，包含需求状态、代码覆盖率及安全测试结果。

最佳实践包括如下方面：

a）双向追溯工具链：通过工具链实现需求与代码的双向追溯，提升协作效率。选择支持需求与代码关联的平台；在需求管理工具中展示关联的代码提交、测试结果及部署状态；通过 API 等在需求变更时触发代码评审流程；

b）自动化需求验证：将需求验收标准转化为自动化测试用例，确保代码实现符合业务需求。使用行为驱动开发工具编写需求对应的测试场景；将测试用例集成到 CI/CD 流水线，每次代码提交需求自动运行需求验证测试；通过测试结果反向追踪需求实现进度，标识未覆盖的需求点；

c）合规驱动的代码审查：通过代码合规性审查流程确保需求实现的合规性。在代码评审（Code Review）环节加入合规检查清单（如数据隐私、加密标准）；使用静态代码分析工具检测安全漏洞与需求偏离；记录审查结果并与需求文档关联，形成合规审计证据。

12.2 需求管理与质量管理的协同

需求管理与质量管理之间的协同对于确保项目交付物的高质量至关重要。这种协同不仅能够提升项目团队的效率，还能提高最终产品的质量和客户满意度。需求管理与质量管理的协同主要体现在以下方面：

a）需求管理与质量管理密切相关。在需求阶段，质量管理人员要和业务、产品经理人员紧密配合，参与需求的收集、优先级确认、需求评审过程，充分理解合规要求，完成测试用例设计，建立质量标准，在开发、测试等环节开展用例测试、系统集成测试（SIT）等质量保证活动；

b）需求管理与质量管理相互促进。清晰的需求定义和优先级排序为质量管理提供了明确的方向。同时，质量管理过程中通过严格的测试和验证，能够及时发现并修正需求实现中的问题和缺陷，确保产品符合高质量标准。需求管理与质量管理的协同还体现在持续改进和反馈机制上。通过收集用户反馈和需求实现过程中的数据，项目团队可以及时调整需求管理方式，提升产品质量。质量管理中发现的问题和改进点也为需求管理提供了宝贵的经验和教训，促进了持续改进。

为了实现需求管理与质量管理的有效协同，公司层面宜建立完善的需求管理和质量管理制度，确保项目团队在需求的收集分析、评审、变更管理、开发和测试等方面遵循有效的规范，形成良性循环。同时，宜注重培养团队成员的技能和意识，加强沟通和协作，最大化地发挥需求管理与质量管理的协同作用。通过这种协同，项目团队能够更好地理解和管理需求，同时保持对质量的持续关注，从而提高项目的整体成功率。

最佳实践包括如下方面：

a）需求与测试用例的联动：将需求分解为可测试的用户故事或功能点，确保每个需求对应明确的验收标准；使用工具将需求与测试用例关联，实现需求覆盖率的追踪；

b）自动化测试嵌入 CI/CD 流水线：在持续集成阶段运行单元测试、集成测试，确保代码变更不破坏现有功能；在持续交付阶段实施端到端（E2E）测试，验证需求实现的完整性；

c）需求变更的回归测试：每次需求变更后，自动触发相关回归测试，防止引入副作用；使用测试自动化框架快速验证需求变更的正确性；

d）用户反馈驱动的质量改进：通过监控用户行为日志和反馈识别未满足的需求或质量问题；将用户反馈纳入需求优先级调整，形成闭环优化。

12.3 需求管理与安全管理的协同

需求管理与安全管理的协同主要体现在以下 3 点：

a）对需求进行分级分类是进行有效安全管理的前提。首先宜根据需求所属业务条线、业务目标、关联的业务系统的不同进行分级分类，安全管理时根据需求的分级分类确定需要投入的安全资源。在需求过程管理中要随时能够对安全问题进行识别和评估。例如在需求分析评审环节有专业的安全人员参与，识别和评估安全问题，可以有效避免在技术实现后或系统上线前暴露安全问题导致的返工或交付延期的情况；

b）安全管理要随时适应需求管理的动态性。例如需求变更时原有的安全管理方式要跟着变化，需求优先级和进度计划提前时，安全管理的资源投入也要跟着提前。依据需求所属业务条线、重要性，采取不同的安全管理措施。对于监管合规类需求、业务类需求、技术类需求，安全管理的关注重点要有所不同。比如对监管合规类需求要随时了解监管动态，及时响应监管要求；

c）需求管理要兼顾技术运营管理中的安全措施可执行。比如需求评审时要考虑关联系统是否运行在安全可控的应用基础环境、潜在的安全漏洞能否被识别、修复、复测；要兼顾系统运维

和监控能力，确保需求的实现不影响关键的系统运行日志、链路信息、运行指标等监控要素的完整性。

最佳实践包括以下方面：

a）安全需求前置化：在需求分析阶段引入安全需求（如数据加密、访问控制），将其作为用户故事或验收标准的一部分；使用威胁建模识别潜在风险，并将其转化为具体的安全需求；

b）安全评审自动化：在 CI/CD 流水线中嵌入自动化安全测试工具，确保每个需求实现时符合安全规范；通过静态分析和动态分析工具检查代码漏洞；

c）安全需求的可追溯性：使用需求管理工具追踪安全需求的实现进度，确保每个安全需求与对应的代码变更和测试结果关联；通过需求追踪矩阵（RTM）确保安全需求在开发、测试和部署阶段不被遗漏；

d）安全即代码（Security as Code）：将安全策略（如防火墙规则、权限策略）以代码形式嵌入基础设施即代码（IaC）中，确保版本控制和自动化部署；使用策略即代码工具动态验证需求实现是否符合安全策略；

e）持续安全反馈：在部署后通过监控工具（如 SIEM）实时检测安全需求的实际效果，例如检测未授权的访问尝试；将安全事件反馈至需求管理流程，持续优化安全需求的设计。

12.4 技术实现与质量管理的协同

研发运营一体化中交付过程与质量管理的协同主要体现在持续集成与持续交付（CI/CD）和自动化测试的深度融合。通过将质量管理嵌入研发运营一体化流水线，确保每次代码提交都经过严格的质量验证，这不仅提高了交付速度，还大幅提升了代码质量和产品稳定性，具体表现在以下方面：

a）持续集成与持续交付（CI/CD），通过自动化构建、测试和部署流程，确保代码变更能够快速、高效地集成和交付。质量管理通过自动化测试（如单元测试、集成测试、端到端测试）嵌入 CI/CD 流水线，确保每次代码提交都经过严格的质量验证；

b）提高交付速度，自动化流程减少手动操作，加快从开发到部署的周期，使团队能够更快地响应市场需求和用户反馈；

c）提升代码质量，持续的质量检测和反馈机制确保代码始终符合质量标准，减少缺陷，降低后期维护成本；

d）减少缺陷，通过早期发现并修复问题，避免缺陷积累到后期，确保产品的高质量交付。

最佳实践包括以下方面：

a）集成质量工具，在流水线中嵌入代码扫描、单元测试等质量工具，自动化执行代码质量检测和测试；

b）制定明确的质量标准，确保团队成员理解并遵守代码覆盖率、静态代码分析等质量要求，设定质量门槛，未达标的变更将被阻止；

c）持续监控与反馈，实时监控质量指标，通过仪表盘展示质量状态，及时反馈和改进，确保质量管理与开发流程紧密结合。

12.5 技术实现与技术运营管理的协同

研发运营一体化中交付过程与运维管理的协同主要通过基础设施即代码（IaC）和自动化监控与报警实现。这种协同确保了基础设施的自动化部署与管理，提高了系统的一致性和可重复性，增强了系统的可用性和稳定性，具体表现在以下方面：

a）基础设施即代码（IaC）：运维管理通过基础设施代码化 IaC 工具与流水线集成，实现基础设施的自动化部署和管理，确保不同环境配置的一致性，减少人为错误；

b）自动化监控与报警：运维团队通过监控工具与研发运营一体化的流程集成，实时监控系统状

态，自动触发告警与修复，提高系统的可用性和稳定性；

c）一致性与可重复性：自动化部署确保不同环境配置一致，减少人为错误，提升部署效率和可靠性；

d）快速响应与恢复：自动化监控和修复机制提高系统的可用性，能够快速响应故障，减少停机时间；

e）优化资源利用：自动化工具帮助运维团队高效管理资源，提升运维效率，降低运营成本。

最佳实践包括以下方面：

a）持续交付（CD）的自动化运维：将部署、监控和回滚流程自动化，通过 CD 流水线确保每次代码变更快速、可靠地交付到生产环境；使用工具实现构建、测试和部署的自动化；

b）基础设施即代码（IaC）的统一管理：使用工具以代码形式管理服务器、网络和资源配置，确保开发、测试和生产环境的一致性；通过版本控制追踪基础设施变更，实现可追溯和快速回滚；

c）监控与日志的整合分析：集成监控工具，实时收集系统性能指标和日志数据；通过自动化警报和仪表盘快速定位故障，缩短 MTTR；

d）混沌工程与韧性测试：定期实施混沌工程实验，模拟生产环境中的故障，验证系统的容错能力；通过自动化测试确保系统在异常条件下的稳定性；

e）协作与知识共享：使用工具促进开发与运维团队的实时沟通，确保知识共享和问题协同解决；建立共享的运维手册，标准化故障处理流程；

f）容量与成本优化：通过自动化伸缩动态调整资源使用，优化成本；使用工具监控资源消耗，防止超额支出。

12.6 技术实现与安全管理的协同

研发运营一体化交付过程与安全管理的协同通过持续安全体系建设实践和自动化安全扫描实现，确保安全性贯穿于整个开发和交付流程中。这种协同不仅提升了系统的安全性，还确保了合规性要求的及时满足，具体表现在以下方面：

a）持续安全体系的建设，将安全集成到研发运营流程中，在开发和交付的每个阶段都考虑安全性，确保安全性不被忽视；

b）自动化安全扫描，在 CI/CD 流水线中集成静态应用安全测试（SAST）、动态应用安全测试（DAST）等安全工具，自动检测代码和应用中的安全漏洞，及时发现并修复安全问题；

c）权限与合规管理，通过 IaC 和自动化工具管理基础设施和应用的安全配置，确保系统符合合规要求；

d）早期安全介入，在开发初期就发现并修复安全问题，减少后期修复成本，提升系统的整体安全性；

e）持续安全保障，通过自动化工具持续监控和保护系统，确保系统在整个生命周期中的安全性；

f）合规性遵循，自动化合规检查确保系统始终符合行业和法律法规要求，降低合规风险。

最佳实践包括以下方面：

a）集成安全工具：在 CI/CD 流水线中嵌入安全扫描工具，自动化执行代码和应用的安全检测；

b）安全培训：定期对开发和运维团队进行安全培训，提升团队成员的安全意识和技能，确保安全实践得到有效执行；

c）制定安全策略：明确安全标准和最佳实践，确保团队成员在开发和运维过程中遵循统一的安全规范，降低安全风险。

12.7 质量管理与技术运营管理的协同

质量管理与技术运营管理的协同通过质量反馈循环和缺陷管理实现，确保产品在开发和运行阶段均能保持高质量和高可靠性。这种协同不仅提升了产品质量，还增强了系统的稳定性和用户体验，具体表现在以下方面：

a）质量反馈循环，运维团队通过监控和日志分析提供运行时质量反馈，质量管理团队利用这些数据改进测试和开发流程，形成持续改进的闭环；

b）缺陷管理，运维发现的缺陷通过缺陷跟踪工具反馈给质量管理，促进缺陷的及时修复和质量提升；

c）性能优化，运维团队提供性能数据，质量管理团队根据数据优化测试用例和质量标准，确保系统性能符合预期；

d）全面质量覆盖，结合运行时数据和测试结果，全面覆盖产品质量，确保系统在各种环境下的稳定性和可靠性；

e）快速问题定位，运维提供的实时数据帮助质量管理团队快速定位和解决问题，提升问题解决的效率；

f）持续改进，基于实际运行数据持续优化质量管理流程，提高产品的稳定性和用户体验。

最佳实践包括以下方面：

a）集成监控与测试工具：将监控工具与质量管理系统集成，实现数据共享和自动反馈，提升问题发现和解决的效率；

b）跨职能团队协作：建立开发、质量和运维的跨职能团队，促进信息共享和协同工作，确保质量管理与运维管理的紧密结合；

c）定期回顾与改进：通过定期的质量与运维回顾会议，识别问题和改进机会，持续优化质量管理与运维管理的协同流程，提升整体项目管理水平。

12.8 质量管理与安全管理的协同

质量管理与安全管理的协同通过安全测试纳入质量管理和风险评估与管理实现，确保产品在功能和安全性上均符合高标准要求。这种协同不仅提升了产品的整体质量，还增强了系统的安全性和合规性，具体表现在以下方面：

a）安全测试纳入质量管理，将安全测试作为质量保证的一部分，确保产品在功能和安全性上均符合标准，通过严格的测试和验证，发现并修复安全漏洞；

b）风险评估与管理，质量管理通过安全风险评估识别潜在威胁，制定相应的质量和安全改进措施，确保系统的安全性和稳定性；

c）标准化流程，建立统一的质量与安全标准，确保两者在项目中的一致性和协调性，避免因标准不一导致的协同障碍；

d）全面风险覆盖，同时关注功能质量和安全性，降低系统漏洞和功能缺陷的风险，确保产品的全面可靠性；

e）提升产品信任度，高质量和高安全性的产品更能赢得用户信任，提升市场竞争力和用户满意度；

f）减少返工成本，通过早期发现安全问题并通过质量管理流程解决，减少后期修复成本，避免因安全问题导致的返工和交付延期。

最佳实践包括以下方面：

a）集成安全需求：在需求定义阶段就纳入安全需求，确保安全性被充分考虑和集成到系统设计中，避免后期因安全问题导致的返工；

b）联合审查：质量管理和安全管理团队共同参与需求评审和测试计划制定，确保功能和安全需求同时得到验证和满足；

c）共享工具与资源：使用统一的工具进行质量和安全检测，实现工具资源的共享和协同，提升检测效率和准确性。

12.9 技术运营管理与需求管理的协同

在需求全生命周期管理中，将技术运营要求（可维护性/稳定性/可观测性）结构化植入需求阶段，以更有效提升系统的可维护性，保障系统稳定运行。在内容协同上，遵循以下原则：

a）监管合规先行：满足监管要求作为需求基线条目；

b）风险防御左移：运维团队在需求评审阶段介入，识别容灾/性能/安全设计缺陷；

c）闭环度量驱动：通过生产事件反向追溯需求缺陷，持续优化标准。

最佳实践包括以下方面：

a）需求定义与结构化拆解（业务+运维双视角）：在传统用户故事模板中增加“技术运营约束”字段，强制描述可维护性要求：配置管理规则（环境变量加密存储）、依赖库版本锁定机制等；可观测性要求：关键业务指标埋点、日志格式、告警阈值等；稳定性要求：容灾方案、重试策略、资源限流；

b）反馈闭环与持续优化：运维事件溯源机制，生产环境故障单关联原始需求标识，分类事件根因，标注事件类型（需求缺失/实现偏差）；运营指标驱动优化，建立核心指标看板（错误率/延迟/资源成本），当指标持续劣化时（如错误率>阈值），自动创建优化需求工单；模板迭代机制，按阶段进行生产故障根因分析总结，转化为新增需求检查项。

12.10 技术运营管理与安全管理的协同

技术运营管理与安全管理的协同通过安全运维和自动化安全响应等实现，确保系统在运行阶段持续保持高安全性和稳定性。这种协同不仅提升了系统的防护能力，还确保了系统符合安全和合规要求，具体表现在以下方面：

a）安全运维，运维团队在日常运维中实施和监控安全措施（如防火墙规则等），通过 DAST、漏洞扫描、渗透测试、资产测绘和运行时应用自我保护等自动化安全工具持续监控和保护运行状态的系统，确保系统持续安全运行；

b）自动化安全响应运维，通过自动化工具监控安全事件，并自动触发响应措施，如自动封禁可疑网络地址、重启受影响的服务等；

c）权限管理与审计，运维管理系统权限，通过安全管理工具进行审计，确保访问控制和合规性，防止未经授权的访问和操作；

d）实时安全监控运维，通过安全工具实时监控系统，快速响应潜在威胁，保障系统的持续安全性；

e）提升系统防护能力，运维与安全管理的紧密协作增强了系统的整体防护能力，确保系统在面对各种安全威胁时具备较高的防护水平；

f）合规性保障，通过协同管理权限和安全事件，确保系统符合安全和合规要求，降低合规风险，满足行业和法律法规的标准。

最佳实践包括以下方面：

a）统一监控平台：运维和安全团队共享监控平台，集中管理和分析安全事件，提升监控效率和响应速度；

b）自动化响应流程：定义并实施自动化的安全事件响应流程，利用基础设施代码化工具自动执行响应措施，提升响应速度和效率；

c）定期安全审计：运维与安全团队定期进行系统安全审计，发现并修复安全漏洞，确保系统持

续符合安全和合规要求，提升系统的整体安全性和可靠性。