SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) PROCEDURE

Document Reference: QMS-DEV-PRO-001

Version: 1.0

Classification: RESTRICTED - INTERNAL ENGINEERING ONLY

Role	Name & Signature	Position	Date
Prepared by			
Reviewed by			
Approved by			

TABLE OF CONTENTS

1. AMENDMENT RECORD

This procedure is reviewed at least annually or following significant changes to development practices, tooling, or compliance requirements. All changes are version-controlled and subject to the same approval process as the original document.

Amendment Date	Issue No.	Rev No.	Page No.	Subject of Change	Revised By	Approved By

2. GENERAL

2.1 Scope

This procedure governs all software development activities within the organization. It applies to new product development, iterative feature delivery, maintenance work, and platform operations. The procedure is technology-agnostic and is designed to scale with team size and organizational maturity.

Applicable development contexts include, but are not limited to:

- Cloud infrastructure solutions (IaaS, PaaS, CaaS, SaaS)
- Platform and DevOps engineering
- API development, integration, and microservices
- Mobile and web application development
- Data engineering, analytics, and ML/AI platforms
- Fintech and regulated-industry software
- Telecommunications platforms and network software
- Enterprise security and identity management systems
- Internal tooling, developer experience (DevEx) platforms
- Open-source and community-contribution projects

Teams may adapt specific tooling and cadences to their context while adhering to the phase gates, quality controls, and compliance requirements defined in this procedure.

2.2 Purpose

- Establish a standardized, repeatable approach to software delivery
- Ensure consistency and predictability in releasing high-quality software
- Align with ISO 9001:2015 (Quality) and ISO 27001:2022 (Information Security) requirements
- Embed security-by-design and shift-left testing into every phase
- Support engineering teams of any size; from 2-person startups to enterprise divisions
- Enable measurable continuous improvement through DORA metrics and other industry benchmarks
- Foster a culture of knowledge sharing, blameless retrospectives, and psychological safety

2.3 Guiding Principles

This procedure is built on the following engineering principles:

- Shift Left: Quality, security, and performance considerations are introduced at the earliest possible phase, not as afterthoughts.
- Automation First: Core workflows - including builds, testing, deployments, and verification checks, must be executed through automated pipeline gates to minimize manual intervention.
- Branching and Mainline Merging Strategy: Engineering teams shall utilize short-lived feature branches with frequent integration to minimize merge complexity and mitigate deployment risk

- Everything as Code: Infrastructure, configuration, pipelines, and policy are version-controlled and peer-reviewed.
- Observability by Default: All services ship with logging, metrics, and tracing instrumentation from day one.
- Psychological Safety: Teams are empowered to raise issues, experiment, and learn from failures without blame.

2.4 Responsibility for Implementation

The Engineering Lead (or equivalent senior technical owner) is responsible for implementing and maintaining this procedure in collaboration with the Quality and Security functions. Specifically, this role is accountable for:

- Ensuring the procedure is understood and followed by all contributors
- Providing tooling, training, and environment access required to comply with the procedure
- Monitoring KPIs and driving corrective actions when targets are missed
- Coordinating with Security, Compliance, and Product functions
- Scheduling and leading procedure reviews at least annually

2.5 References

- ISO 9001:2015 - Quality Management Systems (QMS)
- ISO 27001:2022 - Information Security Management Systems (ISMS)
- NIST SP 800-218 (SSDF) - Secure Software Development Framework
- OWASP SAMM - Software Assurance Maturity Model
- DORA State of DevOps Report - Metrics and benchmarks
- SLSA Framework - Supply-chain Levels for Software Artifacts
- Change Management Procedure (QMS-CHG-PRO-001)
- Configuration Management Procedure (QMS-CFG-PRO-001)
- Risk Management Procedure (QMS-RSK-PRO-001)
- Incident Response Procedure (QMS-INC-PRO-001)

2.6 Roles and Responsibilities

Role	Key Responsibilities
Engineering / Platform Lead	Technical strategy, architecture decisions, code review oversight, procedure ownership, DORA metric review
Product Manager / Owner	Requirements prioritization, stakeholder alignment, acceptance criteria definition, roadmap governance
Software Engineers	Design, implementation, unit and integration testing, code review participation, documentation

Senior / Staff Engineers	Architecture review, technical mentorship, cross-team design decisions, ADR authorship
QA / Test Engineers	Test strategy, test plan authorship, defect lifecycle management, release sign-off
Security Engineer	Threat modelling, SAST/DAST tooling, vulnerability triage, security acceptance criteria
DevOps / Platform Engineer	CI/CD pipeline design and operation, environment management, IaC, deployment coordination
Data Engineer / Analyst	Data pipeline design, data quality assurance, schema governance (where applicable)
Project / Delivery Manager	Sprint facilitation, impediment removal, timeline and risk management, stakeholder reporting
System Architect	Solution architecture, non-functional requirements, integration design, technology selection

3. PROCESS DETAILS

3.1 Objectives

- Deliver software that is reliable, secure, and observable from day one
- Maintain a deployment frequency and lead time that enables rapid response to business needs
- Achieve zero critical security vulnerabilities in production through preventative controls
- Ensure all software is auditable and traceable from requirement through to release
- Build and maintain engineering capability through documentation, mentorship, and knowledge sharing
- Foster a continuous improvement culture anchored in data, not intuition

3.2 Key Performance Indicators (KPIs)

KPIs are aligned with the DORA (DevOps Research and Assessment) four key metrics as the primary framework for measuring software delivery performance, supplemented with quality, security, and operational metrics. Targets reflect DORA Elite/High performance profiles where applicable.

KPI	Description	Target	Standard / Source
Deployment Frequency	How often the team deploys to production	On-demand (multiple/day for elite teams)	DORA
Lead Time for Changes	Commit to production	< 1 hour (elite) / < 1 day (high)	DORA

Change Failure Rate	% of deployments causing incidents	< 5% (elite) / < 10% (high)	DORA
Mean Time to Recovery (MTTR)	Time to restore service after incident	< 1 hour (elite) / < 1 day (high)	DORA
Code Coverage	% of code covered by automated tests	> 80% (unit) / > 60% (integration)	Internal
Critical Defect Escape Rate	P0/P1 bugs reaching production	< 1% P0 / < 2 P1 per sprint	Internal
Technical Debt Ratio	Debt vs total development time	< 10%	Internal
Security Vulnerabilities (Critical/High)	Open CVSS 7+ findings	100% (Critical/High), SLAs maintained for Medium/Low	ISO 27001
SAST/DAST Coverage	% of repos with active scanning	100%	NIST SSDF
Compliance Audit Pass Rate	Internal + external audits	Conformance maintained (Zero Major Non-Conformances)	ISO 9001/27001
Sprint Velocity Variance	Consistency of delivery	< ±15% over rolling 3 sprints	Agile
Incident Response Time	Alert to acknowledged	< 15 min (P0) / < 1 hr (P1)	SRE
Pipeline Success Rate	CI pipeline green rate	> 95%	DevOps
Documentation Completeness	Docs updated per release	100% of APIs and services	Internal

3.3 Resources

The following table reflects a comprehensive reference toolchain. Teams are expected to select appropriate tools from each category based on their technology stack, scale, and maturity. The use of open-source alternatives is explicitly supported.

Category	Tools / Platforms
Source Control	Git (GitHub, GitLab, Bitbucket, Azure DevOps)
IDE	VS Code, JetBrains Suite, Neovim (team preference)
CI/CD	GitHub Actions, GitLab CI, Jenkins, CircleCI, Tekton
Containerisation	Docker, Podman, Buildah
Container Orchestration	Kubernetes, OpenShift, Nomad

Cloud Platforms	AWS, GCP, Azure, OpenStack, VMware vSphere
IaC	Terraform, Pulumi, Ansible, Crossplane
Service Mesh	Istio, Linkerd, Consul Connect
Unit Testing	JUnit 5, PyTest, Jest, Go Test, RSpec
Integration / Contract Testing	Postman/Newman, Pact, WireMock, SoapUI
E2E / UI Testing	Playwright, Cypress, Selenium
Performance Testing	k6, Gatling, JMeter, Locust
SAST	SonarQube / SonarCloud, Semgrep, CodeQL
DAST / Dependency Scanning	OWASP ZAP, Trivy, Snyk, Grype
Secret Scanning	GitLeaks, TruffleHog, GitHub Secret Scanning
Monitoring / Alerting	Prometheus, Grafana, Datadog, New Relic, PagerDuty
Logging	ELK Stack, Loki + Grafana, Splunk
Distributed Tracing	Jaeger, Zipkin, OpenTelemetry
Artefact Registry	Harbor, JFrog Artifactory, ECR, GAR
API Gateway	Kong, AWS API Gateway, NGINX, Apigee
Feature Flags	LaunchDarkly, Unleash, Flagsmith
Project Tracking	Jira, Linear, GitHub Issues, Azure Boards

3.4 Process Phases

The SDLC is structured into six phases. Each phase has defined entry criteria, activities, quality gates, and outputs. Phases may overlap in agile / iterative delivery models but must all be completed before a release candidate is promoted to production.

1. Requirements & Discovery

Activities	• Stakeholder workshops and user story mapping • Business requirements document (BRD) / product requirements document (PRD) • Functional and non-functional requirements (NFRs) elicitation • Security requirements analysis (threat modelling kickoff, data classification) • Regulatory / compliance scoping (GDPR, PCI-DSS, SOC 2, etc.) • Feasibility and dependency analysis • Definition of Done (DoD) and acceptance criteria definition

	• Initial risk register update
Quality Gate	Requirements sign-off by Product, Engineering Lead, and Security
Outputs	PRD, NFR register, threat model draft, risk register entry, DoD

2. Design & Architecture

Activities	• High-level and low-level system architecture design • Architecture Decision Records (ADRs) for key technical choices • Database schema design and data flow diagrams • API contract design (OpenAPI / AsyncAPI specs) • Security architecture review: authentication, authorisation, encryption • Infrastructure design and capacity planning • Observability design: metrics, logs, traces, alerting runbooks • Interface and UX design (where applicable) • Design review / RFC process with senior engineers
Quality Gate	Architecture review board (or equivalent) sign-off; security architecture approved
Outputs	Architecture diagrams, ADRs, API specs, data flow diagrams, infra design doc

3. Development

Activities	• Sprint planning with capacity-aware story pointing • Enforce a mainline/trunk-based branching model where feature branches must remain short-lived (maximum lifespan of 48 hours) • Test-driven development (TDD) or behaviour-driven development (BDD) encouraged • Unit test implementation alongside feature code • Mandatory peer code review requiring a minimum of two approved pull requests prior to merging into the primary codebase or production trunk • SAST and linting in pre-commit hooks and CI pipeline • Dependency vulnerability scanning on every build • Secret scanning, no credentials in source code • Incremental commits with clear, conventional commit messages • Technical documentation updated in the same PR as code • Feature flags for in-progress or experimental features
Quality Gate	PR approval (≥ 2 reviewers), CI pipeline green, SAST no new Critical/High findings
Outputs	Feature code, unit tests, updated docs, passing CI build, signed artefact

4. Testing & Validation

Activities	• Integration testing across service boundaries • Contract testing for API consumers and providers • End-to-end (E2E) test suite execution • Performance / load testing against NFR baselines • Security testing: DAST, penetration testing (for major releases), dependency audit • Accessibility testing (WCAG 2.1 AA minimum for user-facing products) • Chaos / resilience engineering (for platform-critical services) • User Acceptance Testing (UAT) with product and business stakeholders • Regression test suite execution • Defect triage, severity classification, and resolution tracking • Test coverage validation against defined thresholds
Quality Gate	Zero open Critical/High defects; UAT sign-off; performance within NFR tolerances; security test pass
Outputs	Test execution reports, defect log, performance benchmarks, UAT sign-off, release candidate

5. Deployment & Release

Activities	• Release readiness review (runbook, rollback plan, comms plan) • Deployment pipeline promotion: dev - staging - production • Blue/green, canary, or feature-flag-gated rollout strategy defined • Environment configuration validation (infrastructure as code diff review) • Observability validation: dashboards, alerts, and SLOs configured pre-launch • Backup and DR configuration verified • Change Advisory Board (CAB) notification or approval per change management procedure • Post-deployment smoke tests and synthetic monitoring verification • Documentation updated: release notes, runbooks, architecture docs • Stakeholder communication, release broadcast sent
Quality Gate	Deployment runbook reviewed; rollback plan tested; change ticket approved; smoke tests pass
Outputs	Production deployment, release notes, updated runbooks, observability dashboard

6. Operations & Maintenance

Activities	• Continuous SLO/SLA monitoring with alerting and on-call rotation • Incident management per incident response procedure, blameless post-mortems for P0/P1 • Security patch management, critical CVEs patched within SLA • Dependency updates on a regular cycle (automated where possible)

	• Capacity planning reviews, quarterly or event-driven • Technical debt grooming, backlog items tracked and prioritised • Feature request intake and prioritisation with product • User feedback analysis and experience improvement cycles • Periodic architecture reviews for long-lived services • Documentation maintenance, drift detection and correction
Quality Gate	SLO compliance reviewed monthly; patch SLAs met; technical debt ratio below threshold
Outputs	Incident reports, post-mortems, patch records, dependency updates, capacity reports

3.5 Process Map

The end-to-end process flow is provided as a separate diagram (see Process Flow Diagram). The map illustrates the six phases, decision gates, feedback loops, and integration points with Change Management, Incident Response, and Risk Management procedures.

SDLC Process Flow Diagram

3.6 Retained Documentation

Document Type	Purpose	Retention Period	Storage Location
Requirements Specification	Define scope, user stories, NFRs, acceptance criteria	Project lifecycle + 3 years	Document Management System
Architecture Decision Records (ADRs)	Record key design decisions and rationale	Product lifetime	Version Control / Wiki
Technical Design Documents	System architecture, data flows, API contracts	Product lifetime	Document Management System
Threat Model	Security analysis per feature or service	Updated each major release; retained 3 years	Secure Document Repository
Test Plans and Results	QA strategy, execution records, coverage reports	2 years	Test Management System
Security Assessment Reports (SAST/DAST/PenTest)	Vulnerability findings and remediation status	3 years	Secure Document Repository
Code Review Records	PR history, approvals, comments	Lifetime of codebase	Version Control System

Deployment Records	Release notes, deployment logs, rollback records	Product lifetime + 2 years	CI/CD System / ITSM
Change Requests	Formal change tracking	3 years	Change Management System
Incident Reports and Post-Mortems	Incident timelines, root cause, action items	5 years	Incident Management System
Runbooks and Operational Playbooks	Step-by-step operational procedures	Current version always; history 3 years	Wiki / Version Control
SLO / SLA Reports	Service level performance data	2 years	Monitoring Platform / Reports

4. RISKS AND OPPORTUNITIES

4.1 Risk Register

Risks are assessed using a Likelihood × Impact matrix and reviewed quarterly or following significant process, tooling, or organisational changes. Risk owners are responsible for implementing and verifying mitigation strategies.

Category	Risk Description	Mitigation Strategy
Technical	Accumulation of technical debt reducing velocity	Regular debt grooming sessions; debt ratio tracked as KPI; refactoring budgeted per sprint
Technical	Integration failures between services or third-party APIs	Contract testing with Pact or equivalent; staging environment mirroring production topology
Technical	Performance degradation under load	NFR-based performance testing in CI; SLO-based alerting in production
Technical	Pipeline instability reducing deployment confidence	Pipeline health tracked; flaky test quarantine process; canary deployments
Security	Introduction of critical vulnerabilities via dependencies	Automated SCA scanning (Trivy/Snyk) on every build; dependency update automation (Renovate/Dependabot)
Security	Secrets or credentials committed to version control	Pre-commit secret scanning hooks; gated CI checks; secret rotation procedures
Security	Insufficient security testing for regulated workloads	Threat model per feature; mandatory DAST for external-facing services; annual penetration testing
Compliance	Non-conformance during ISO 9001/27001 audits	Internal audit schedule; evidence collection automated where possible; procedure compliance reviews quarterly

Operational	Key person dependency / knowledge concentration	Pair programming culture; documented runbooks; on-call rotation; redundant expertise
Operational	Scope creep causing sprint overload	Strict change control; sprint commitment protected; backlog refinement discipline
Business	Rapid requirement changes destabilising delivery	Agile backlog management; change impact assessed before mid-sprint inclusion; stakeholder alignment ceremonies
Business	Technology obsolescence increasing maintenance burden	Technology radar process; annual dependency and platform reviews; upgrade path documented

4.2 Opportunities

Area	Opportunity	Action Plan	Owner
Developer Experience	Improving CI/CD speed and reliability	Pipeline profiling; parallelisation; caching strategies; DORA metrics review	Engineering Lead
Automation	Automated compliance evidence collection	Integrate audit tooling into CI pipeline; generate SBOM and policy reports automatically	DevOps / Security
Platform Engineering	Self-service developer platforms (IDP)	Backstage or equivalent; golden paths; automated environment provisioning	Platform Eng Lead
Quality	Mutation testing for test quality validation	PiTest (Java), Mutmut (Python), Stryker (JS) - validate test effectiveness beyond coverage %	QA Lead
Security	Shifting security further left with ASPM tooling	Application Security Posture Management tooling integrated into developer IDEs and PRs	Security Eng
AI-Assisted Development	LLM-assisted code generation and review	GitHub Copilot, Codeium, or equivalent; guardrails for generated code; security review policy	Engineering Lead
Knowledge Management	Structured engineering documentation	Architecture wiki; decision log; onboarding runbooks; loom walkthrough library	Tech Lead
Observability Maturity	Full OpenTelemetry adoption	Standardise on OTEL SDK; correlate traces, logs, metrics; eliminate vendor lock-in	Platform Eng
Team Development	Structured career development paths	Engineering levelling framework; internal tech talks; conference sponsorship; certification support	Engineering Manager

Customer Value	Continuous deployment enabling rapid feedback loops	Feature flags + A/B testing infrastructure; product analytics integration; customer telemetry	Product + Engineering

5. CONTINUOUS IMPROVEMENT

5.1 Improvement Cycle

Continuous improvement is embedded at three cadences:

- Sprint Retrospective (every 1-2 weeks): Team-level process, tooling, and collaboration improvements. Actions tracked in backlog.
- Quarterly Engineering Review: DORA metrics analysis, KPI trending, risk register review, tooling evaluation, capacity planning.
- Annual Procedure Review: Full review of this procedure, compliance alignment, architectural direction, team structure.

5.2 Blameless Post-Mortems

All P0 and P1 production incidents trigger a blameless post-mortem within 5 business days. The output is a structured report covering:

- Incident timeline and contributing factors
- Root cause analysis (5 Whys or equivalent)
- Impact assessment: customers, revenue, SLO burn
- Action items with owners and due dates
- What went well / what to improve

Post-mortem reports are stored in the Incident Management System and shared across the engineering organisation to drive systemic learning.

5.3 Engineering Maturity Model

Teams are encouraged to assess their maturity against the following simplified model and identify targeted improvement areas:

Dimension	Level 1 - Initial	Level 2 - Developing	Level 3 - Defined	Level 4 - Elite
Deployment	Manual, infrequent	Automated to staging	Automated to prod, weekly	On-demand, multiple/day
Testing	Local manual verification, non-standardized test suites	Unit tests in CI	Full suite, gates in CI	Mutation testing, chaos eng.
Security	Periodic reviews	SAST in CI	DAST + SAST, threat models	Full ASPM, shift-left everywhere

Observability	Basic logging	Metrics + alerts	Logs + metrics + traces	Full OTEL, SLO-based alerting
Documentation	Localized repository documentation (README-driven)	READMEs per repo	ADRs, runbooks, API docs	Living docs, auto-generated
Incident Mgmt	Per-incident manual remediation	Runbooks exist	Post-mortems, on-call rota	Automated post-incident root-cause analysis, proactive simulation

6. DOCUMENT CONTROL

This document is the property of the Engineering Operations department. Distribution, copying, or disclosure to unauthorized personnel without prior written authorization is strictly prohibited.

This procedure will be reviewed and updated whenever any of the following occur:

- Significant changes to the development toolchain or methodology
- Regulatory or compliance requirement changes
- Findings from internal or external audits
- Post-mortem action items that require procedural change
- Annual review cycle (at minimum)

The latest version of this document is always the authoritative version. Previous versions are archived in the document management system.