BUREAU OF INDIAN STANDARDS

PRELIMINARY DRAFT FOR COMMENTS ONLY

(Not to be reproduced without the permission of BIS or used as an Indian Standard)

मसौदा भारतीय मानक

Draft Indian Standard

Software and Systems Engineering - DevOps Principles and Practices

ICS 35.240.70

LITD 14 Software and Systems Engineering	Last Date for Comments: 09 Feb 2025

FOREWORD

This standard defines the set of core principles and practices that are present in any DevOps implementation. These practices enable the application of DevOps in development, operations, and maintenance of software systems.

All software systems have a lifecycle, irrespective of whether it is established formally or not. A lifecycle is a set of distinguishable phases or stages that the software goes through from its conceptualization until it ceases to exist. Software systems typically pass through their lifecycle as the result of tasks being performed by organizations and people, by instantiating a framework of processes and activities for their respective contexts.

DevOps is one such framework of processes and activities for software development and delivery. DevOps is a culture-driven combination of **principles, practices** and **tools** that enable delivery of reliable, secure, resilient, rapidly evolving software systems. It integrates the practices and teams of Development, Quality Assurance (QA), Security, Platform engineering and Operations to enable high performing teams. It requires a strong collaboration and active involvement of business and IT.

The Committee responsible for the formulation of this standard has reviewed the provisions of the international publications listed in Annex E and has decided that these may be used in conjunction with this standard till Indian Standards on these subjects are published.

The composition of the Committee responsible for the formulation of this standard is given in Annex F.

Contents

Introduction

0.1 Overview

All software systems have a lifecycle, irrespective of whether it is established formally or not. A lifecycle is a set of distinguishable phases or stages that the software goes through from its conceptualization until it ceases to exist. Software systems typically pass through their lifecycle as the result of tasks being performed by organizations and people, by instantiating a framework of processes and activities for their respective contexts.

DevOps is one such framework of processes and activities for software development and delivery. DevOps is a culture-driven combination of **principles, practices** and **tools** that enable delivery of reliable, secure, resilient, rapidly evolving software systems. It integrates the practices and teams of Development, Quality Assurance (QA), Security, Platform Engineering and Operations to enable high performing teams. It requires a strong collaboration and active involvement of business and IT.

Organizational topologies enabling DevOps

Organizational topologies that enable the DevOps process are:

a) **Collaborative Teams:**
Business, Development, Quality Assurance, Security, Operations and Platform/Enablement teams working collaboratively to achieve the organizational goals.

b) **Embedded Teams:**
Self-sufficient multi skilled project teams working together to achieve operational stability of the product. The project team has the Product Owner, Development, Quality Assurance, Security and Operations skills within it. These teams shall also have capabilities for infrastructure provisioning.

Different products/product lines/ business units may follow different organizational topologies.

Tools Considerations

Appropriate tools are required at each stage of the software development lifecycle such as backlog management, source code management, static code quality analysis, functional and non-functional testing, build, integration, deployment, and monitoring. Key considerations are listed below:

a) Appropriate tools shall be made available to the team to reduce the manual effort and improve quality. The mundane tasks shall be automated and the team shall be able to choose the tools that help them to continuously improve efficiency and effectiveness of software development lifecycle.

b) Technology may be a limitation to adoption of tools, especially in case of legacy-based applications which require the right architectural intervention and investment to be loosely coupled. This enables teams to take control of deploying changes without bringing down the system.

c) Tools that enable collaboration and provide visibility to development and operations shall be used. For example

   - Development and Operations teams collaborate to fix incidents confidently and push them to production.
   - Service desk staff can view changes and stay updated on the system stability.
   - Developers can view the customer issues in real time and provide permanent fixes.

d) Tools that provide real time visibility into the progress and health of the release and post release, provide insight into the quality of product through automated monitoring.

e) Since there is a plethora of tools available, it is important to align tools to architecture vision and roadmap of the organization.

Culture-led principles and practices are followed to foster a collaborative environment in which the team members work together to achieve a collective goal. They are open and transparent and able to take decisions quickly based on outcomes, rather than being closed and hierarchical. The culture of experimentation, shift left for quality and security processes and the discipline of implementing the practices enable DevOps implementation.

Building of the right practices is enabled by lean philosophies of limiting work in progress, visualizing the work, incorporating feedback from every learning point, observing the application performance and business monitoring to make quick business decisions. Risk based and light weight change management practices enable flow of work without any bottleneck.

Key principles to be adopted are:

- Customer centricity
- Version Control Everything
- Automation First
- Shift-left of Quality and Security
- Observability
- Continuous Improvement
- Collaborative Culture

For details on these principles, refer to the section on DevOps Principles.
One of the key focus of DevOps implementation is one visibility to all stakeholders, preferably in a real-time manner, which is enabled and achieved using a set of metrics (as given in Annex C of this document, as a representative set) and generally implemented through DevOps pipeline.

0.2 Need for the Standard

Standardization of the DevOps framework is essential for the following reasons:

a) **For Vendors** — it provides a list of core practices.

b) **For Acquirers** — it provides a basis to differentiate the maturity of the vendor by assessing these practices.

c) **For Both Parties** — it provides a basis for agreement. Standardization in this area will be very useful for software vendors, software acquirers, software service providers, large delivery and maintenance projects, system integration projects having software components and in general, the entire software community.

0.3 Use of AI in DevOps

AI and Gen AI may be used to enhance the DevOps process and support the DevOps practices by bringing consistency to the repetitive and complex tasks, enabling continuous experimentation and strengthening the overall automation practices. For more details refer Annex D.

Indian Standard

SOFTWARE AND SYSTEMS ENGINEERING – DevOps Principles and Practices

1. Scope

This standard provides principles, practices and maturity model for implementing DevOps framework to deliver secure and quality software and increase the speed to market with frequent/on demand releases. This standard covers the following practices:

a) Continuous Planning
b) Configuration Management
c) Continuous Integration
d) Continuous Testing
e) Continuous Delivery
f) Continuous Deployment
g) Continuous Monitoring
h) Continuous Experimentation

This DevOps standard is applicable to

a) Software development projects including enhancements and operations
b) Product based teams with development, operations, and support
c) Infrastructure projects and support
d) DevOps enablement and Platform teams
e) This standard is applicable to Indian organizations that are aiming to adopt DevOps framework/ practices. Any deviations from this standard will be identified and documented by the organization.
f) The practices defined in this standard are applicable for a single DevOps project, as well as for an organization performing multiple DevOps projects. These practices are applicable throughout the lifecycle of software systems, products and services.

2. Conformance

2.1 Intended Usage

This document is meant for use by DevOps Sponsors, Practitioners, and Stakeholders for Acquirer (or Customer) and Supplier organizations.

This standard is written to be applicable to all types of IT projects, irrespective of the technology and domain/industry vertical.

2.2 Full conformance

Full conformance is achieved by demonstrating that all the requirements of all the practices have been satisfied using the outcomes as evidence.

2.3 Tailored conformance

An organization shall document the deviations expected from the practices with rationale and obtain sign-off before use. Tailored conformance shall be achieved by demonstrating that requirements for the practices along with the documented expected deviations, have been satisfied using the outcomes as evidence.

If certain practice requirements for a level are not feasible due to technical limitations (such as for legacy applications or COTS products) or due to organizational strategy, such tailoring and rationale shall be documented by the organization. Conformance will be established against this tailored scope.

3 Normative References

The standards listed in Annexure E are necessary adjuncts to this standard. They, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

4 Terminology

The terms pertaining to systems and software engineering used in this document shall be referred from the following standards:

- ISO/IEC/IEEE 24765 Systems and software engineering — Vocabulary
- IS/ISO/IEC 2382: 2015 – Information technology - Vocabulary

In addition to the above, the below definitions shall also apply to this standard:

4.1 **Blue-Green deployment**

This is a deployment strategy wherein two identical production servers or nodes or environments are maintained, of which one is exposed to the users (green) while the other is the standby (blue). During production release, products are deployed to the standby environment and a subset of users is routed to the standby. The user base is gradually increased in the standby environment until the entire user base points to the standby. The earlier standby environment now becomes the live production environment (green), while the earlier environment pointing to production becomes the standby environment (blue). This entire process is known as blue-green deployment.

4.2 Canary Release

Referencing how canaries were brought into coal mines to test the air, this is a go-live strategy where a new application version gets released to a small batch of production servers, it is then monitored closely to determine if it runs as intended. If the version is stable, it is rolled out to all the production environments.

4.3 **Containerization**

An operating system (OS) level method of virtualization employed for the deployment and running of distributed applications without having to launch an entire virtual machine for every use.

4.4 Fail Fast

A design strategy characterized by a rapid turnaround, where an attempt fails, is reported on time, feedback is quickly returned, the changes are made, and a new attempt is made.

4.5 **Runbook**

A Runbook contains a set of steps that are executed (preferably automated) to address issues or repetitive tasks in IT operations.

4.6 Smoke Test

Tests which are conducted to verify the stability of the application in the production environment.

5 DevOps PRINCIPLES

DevOps principles are a set of rules or beliefs within which the project teams need to operate. These principles are relevant for all practices and sub-practices. They broadly outline qualitative aspects of implementing the practices to optimize either an architectural, operational, or business value. Together with quantitative levers like key performance indicators and metrics, DevOps principles serve to make informed decisions, measure against targets and objectives, and facilitate continuous learning and improvement.

The following are the key guiding principles for DevOps adoption.

a) Customer Centricity
b) Version Control Everything
c) Automation First
d) Shift-Left of Quality and Security
e) Observability
f) Continuous Improvement
g) Collaborative Culture

5.1 Customer Centricity

Customer Centricity refers to keeping the customer at the center in all planning and execution processes. In a DevOps project, the planning, execution, and continuous improvement activities shall be driven by customer needs and priorities. There shall be a conscious effort to proactively engage with customers, provide an early visibility to product increments, solicit feedback, and improve product features so as to meet customer needs and experience.

From a continuous product evolution perspective, this may also mean reducing deficiencies, defects, and mean time to repair such that customer satisfaction and experience is enhanced and there is a constant focus on customer delight.

Project teams shall define these customer objectives through measurable and outcome driven key success criteria and related key results. These success criteria and key results shall be further translated into quantifiable metrics and monitored either to assess past performance, measure impact of changes or predict future outcomes.

DevOps systems shall provide the right framework to enable both qualitative and quantitative aspects of customer centricity. They shall enable incorporation of customer requirements and feedback, fasten validation and aid in incorporating innovative features that specifically are built to enhance customer experience while delivering with high speed, reliability, and security.

5.2 Version Control Everything

Project teams shall enable storing, recovery and baselining of all artefacts that are important to the software systems, including but not limited to code components, plans, requirements, libraries, build scripts, test automation scripts, infrastructure and application configurations, provisioning and deployment scripts. This shall ensure a single source of truth across distributed teams and enable repeatability, certainty, recoverability, and failure-tolerance. This principle also prevents defects and delays due to configuration management issues and human error, thus improving pipeline flow and minimizing the rework. Anything which is to be stored for future use shall be versioned, baselined and its properties and meta-information stored such that comparisons and trends may be generated.

5.3 Automation First

Project teams shall automate software development lifecycle as much as possible and plan for ***"Automation First"*** approach upfront. The intent of this principle is to explore and maximize automation opportunity towards implementing any DevOps sub-practice. Automation helps in standardizing, scaling, and removing any human bias. A successful automation strategy increases team efficiency and promotes reusability. This makes implementation of a practice person-independent both from availability and expertise perspective. Key DevOps goals such as Reliability, Speed, and Security are also maintained in an automation first approach.

This principle cuts across all automation opportunities involving configuring and execution of DevOps Pipelines such as code builds, testing, infrastructure provisioning, artefacts (image, configuration and settings, libraries, services and executables) deployments, Quality Checks, Runbook automation, Application and Infrastructure monitoring. Automation by design can also be extended to integrating various policies on security.

To achieve better outcomes, appropriate automation levers shall be selected. For example, an automation first approach to testing helps in reducing time and effort, manage effective regression cycles, ensures reusability for multiple teams and environments, and reduces dependence on human skill and availability.

5.4 Shift left of Quality and Security

Coupled with principle of automation, shift-left principle shall be followed to induce speed and efficiency of value chain, by ensuring reliability of codebase/product increment all through the software development lifecycle.

The focus of this principle shall be to ensure that the right software enters the value chain and that the software is built right at each stage of the pipeline. This shall also encompass the security by design approach in software development, whereby the software applications are designed and architected from the beginning to have the security attributes built into the applications. Software applications shall be designed to identify the weaknesses of applications which may lead to security exploitations and the same shall be addressed upfront.

This includes quality and security analysis, scanning and testing. This prevents occurrence of defects and anomalies later in the pipeline. The late detection of defects is costly, and it also stretches the release cycle time by disrupting the flow. Shift-left provides early feedback and averts these risks.

It guides that all review, verification, testing and security controls start early in the software development lifecycle and preferably in the Developer's sandbox itself. This is achieved by integrating all the Quality practices and tools into the pipeline and regularly shifting the lessons learnt identified during the testing or deployment phase to the development phase. This enablement of a quick feedback mechanism shall ensure that hypotheses are tested early, risks are identified beforehand, and viability and user response is gauged quicker. This shall also enable a streamlined implementation of shift-left practices resulting in less delay or effort required subsequently.

5.5 Observability

This principle enables real time visibility and deeper insights into the application and systems throughout the pipeline as well as in the production environment. This shall help the teams to collaborate, understand, triage and fix issues. Building observability in the production environment helps watch the application and infrastructure health and performance, thus enabling proactive event management to prevent possible incidents, threats, and outages. Combined with analyzing the data and additional intervention like alerts, failsafe mechanisms, auto-heal or rollback activities, this principle shall enable quick action and ensure robustness and resilience within the system.

To build an observable system, project teams shall ensure that the code, or infrastructure or network or configurations are sufficiently instrumented with traces and logs and key metrics are configured and insights driving decisions can be calculated at runtime. The main objective of observability is to help understand interdependencies, perform root cause analyses, pre-empt faults before they occur and correlate disjoint, diverse data to get realistic insights. Observability also drives optimized meeting duration, provides data for quick decisions, and increases overall pipeline efficiency and developer productivity.

5.6 Continuous Improvement

Continuous Improvement principle aims at continuous evolution of the product and related practices, to ensure the product features meet customer needs and promote usability, reliability, and efficiency.

Continuous Improvement shall be enabled through focus on changing customer needs, iterative product development, faster and early feedback loops, learning and experimentation, decisions based on insights from monitoring and observability. Most importantly, continuous feedback, or the ability to look at a causality (cause and effect) of a change, performing retrospectives and analyses of what changes or best practices can be standardized and then scaling them up on a regular basis ensures continuous improvement.

Continuous improvement enables the continuous flow of value to end users and improves satisfaction of the customer and employee.

5.7 Collaborative culture

This shall include enabling an environment where different stakeholders collaborate to promote unified team culture working towards one common vision and objectives for implementing DevOps.

Project teams shall ensure that the right kind of understanding, focus and common thinking is developed from amongst the various stakeholders so that they can interoperate and complement each other to solve their customers need or problem statement in the most efficient manner. This could happen through enabling team building practices, product enabled communication platforms, and knowledge management. A collaborative culture also makes teams and individuals appreciate the skills and constraints of others and therefore each team can also maximize their own skills and output. Project teams also practice healthy retrospection, are receptive to feedback, effectively handle problems and incidents, and work meaningfully towards continuous improvement.

6 DevOps PRACTICES

6.1 Continuous Planning

Continuous planning is a flexible approach to refine plans continuously, based on any internal or external trigger instead of annual or bi-annual planning. Examples of internal triggers for plan refinement may be changes in priority, unexpected delay/progress in the project execution and external triggers may be changes in business environment, end-user expectations, geo-political issues etc.

With the degree of changes seen in customer expectations, business environment, technological advancement as well as market competition, continuous planning is key to sustain in such a fast-changing environment.

6.1.1 *Key Objective*

To provide ability to adapt and react in continuously changing environment quicker.

6.1.2 *Sub Practices*

The following shall be the sub-practices for a successful Continuous Planning practice:

a) Software Value stream-based delivery

Software value stream-based delivery shall include all the activities from idea to implementation that need to be carried out to deliver value to the end user. Once the software delivery value stream is defined, it is easy to automate the value adding steps and discard the steps that are non-value-adding.

b) Frequent planning meetings

The planning meetings shall be conducted at a predefined frequency established by the organization or as needed based on internal or external triggers. The continuous planning enables an organization to

i. react to any change quickly
ii. help manage finance optimally
iii. deliver value to end users at the right time

c) Right stakeholders' involvement in the planning meetings

The right stakeholder presence shall be ensured in the planning meeting to prioritize the activities and make the right decision for the organization. The following is an indicative guideline for participants required in the planning meetings, which needs to be created by each organization according to their Organization Structure and Operating Model. Refer Table 1.

Table 1
(c of *Clause* 6.1.2)

Sl No (1)	**Participants/ Stakeholders** (2)	**Mandatory/Optional** (3)
i)	Business Sponsor	Mandatory for project start-up, closure, review and whenever clarification is needed
ii)	IT Sponsor	Mandatory for project start-up, closure, review and whenever clarification is needed
iii)	Product Owner	Mandatory
iv)	Enterprise Architect	Mandatory
v)	Team Facilitator	Mandatory
vi)	Project Team	Mandatory for backlog grooming, refinement, and prioritization and whenever their input is needed
vii)	Other IT teams (e.g. Infra, Security etc.)	As per project requirements

d) Use of appropriate tools

Appropriate tools are selected and used for collaboration and backlog management. Collaboration tool helps in:

- Capturing and sharing ideas and insights within the team
- Collaborating from anywhere
- Improving transparency within the team

e) Backlog Management

Smaller level of granularity in business requirements helps in achieving and measuring success effectively. The business requirement is expressed in a smaller granularity for a specific goal so that it can be developed, deployed, and enabled to deliver value to end user faster and independently. Usually, the business requirements are decomposed into epic, feature, and user stories to ensure the right level of granularity and maintained in the backlog by the product owner.

f) Continuous assessment of relative business priority of both existing and new business ideas

With the frequent technological and ecosystem changes, it is required to examine the requirement and business priority to deliver right values to the right end user. Therefore, regular scrutiny and refinement is part of the continuous planning.

g) Continuous focus on change in customer expectation

In real life scenario, customer behavior and needs are changing rapidly, hence as part of the Continuous Planning, a continuous focus on changing customer expectations shall be ensured. These changes shall be considered during the grooming and prioritization of requirements.

h) Continuous feedback from Continuous Monitoring practice

Continuous monitoring shall ensure that the new changes are working appropriately and any performance, compliance or security threats arising in the production environment are detected immediately. The information gathered from the continuous monitoring activities shall be fed back to the relevant teams to take appropriate action. Any needed changes in the production software or environment shall be added to the product backlog.

i) Ongoing regular communications

Regular communication between various teams and stakeholders is encouraged for better transparency.

6.2 Configuration Management (CM)

Configuration management practice maintains consistency in the assets (e.g. source code, binaries, test plans, etc.) by keeping the functional and physical attributes of system and product aligned with its requirement, design, and other operational measures throughout its lifecycle.

6.2.1 *Key Objective*

Define and control the components of IT systems to improve predictability, repeatability, and accuracy.

6.2.2 *Sub Practices*

The following key sub-practices shall be implemented for a successful Configuration Management system:

a) **Configuration Management Strategy**
The Configuration Management Strategy shall implement configuration management effectively and consistently. The strategy shall include:

- Components to be configured
- Version control mechanism
- Branching and merging policy
- Backup plans
- Rollback Mechanism
- Tools

b) **Document Configuration**
All the required documents created during the software development life cycle are important to maintain and run the software. These documents shall be version controlled and configured.

c) **Code Configuration**
The right configuration of the code helps in ensuring the quality and version of the application code.

d) **Branching Strategy**
The right branching strategy shall ensure the right code at the right place by merging all the relevant changes made by different teams (for example development and hotfix).
Branching strategy shall be formalized based on various factors including but not limited to release cadence, team organization and so on, to consider the persistent branches such as Main, Develop and Temporary branches such as Feature branch, Hotfix, etc.

e) **Artifact Management**

Packages/ Container images with versions may be stored in artifact repositories/ registries for central reference enabling distribution, reuse, security scan of binary artefacts and for future reference as required for example Container images with latest changes

f) Deployment Configuration

Deployment shall involve deploying multiple artefacts including the executable in a desired environment. Deployment configuration shall define the properties of the environment, rules for the deployment and commands to be executed for implementing the right software in the right environment.

It shall be ensured that the tools and systems leveraged for software configuration management have appropriate authorization in place such as role based access control to configurable items such as documents, application code, test scripts, binary files and so on to prevent unauthorized access and privileges.

To ensure better management and security of configuration items,

a) application source code shall not be stored in the same repository as its executable or binary files
b) secrets or sensitive information shall not be stored in the same repository as that of the application code

6.3 Continuous Integration (CI)

Continuous Integration is a development practice that requires developers to integrate code into a shared repository for any change. Each *"*commit to code repository*"* shall be verified by an automated build and unit test execution. The quality gates shall be configured to determine the success of the build. CI encourages development teams to commit code changes frequently for early feedback on code quality and integration issues.

6.3.1 *Key Objective*

To establish a consistent and automated practice to merge code, build and unit test the changes.

6.3.2 *Sub Practices*

The following key sub-practices shall be followed for a successful Continuous Integration practice:

a) Reference architecture for CI/CD pipeline

CI/CD reference architecture shall provide details on what the pipeline achieves, flow diagrams, tools and security steps that enable implementation of CI/CD pipeline.

b) Automated Compile

The code shall be pulled from source code management system, and automatically compiled using predefined mechanism as soon as code is checked in.

c) **Automated Build process**
Build tools shall be used to automate and manage build specific activities to create packages that are delivered.

d) **Process-enforced Code Review with Approval of Reviewer**
A stringent review process shall be put in place. Process-enforced Code Review with Approval of Reviewer (Pull Request / Merge Request / Branch Policies) before the code can be merged into the code base shall be followed.

e) **Dependency Management (such as Library or Node Modules)**
Manage Artifact repository on-premises or in Cloud so that all dependencies and versions are managed at one place. Dependency management shall ensure that the right components (including 3rd party components) are used. This activity may be achieved either by in-house developed tool or with the help of an industry standard tool. Dependency management shall also include scanning of dependent components for potential vulnerabilities and license compliance issues as applicable.

f) **Automated Unit Tests**
Automated Unit Tests shall be implemented for quicker runs of unit tests and provide immediate feedback on the code quality. Code Coverage shall be measured for determining the extent of code covered by automated unit tests.

g) **Automated Code Quality and Security Scan**
The right tool supported by right configurations shall assure code quality through detection of issues early in the software development lifecycle. This shall help to propagate clean code though the DevOps pipeline.

h) **Secret Management backed with Role Based Access Control (RBAC)**
All credentials, certificates, confidential details shall be stored in a system which can be accessed only by authorized users.

i) **Role Based Access**
Only authorized users shall be provided Role Based Access Control (RBAC) to Pipelines and Configuration of Automation Servers.

j) **Fully automated Pipeline**
The pipeline, its configurations and built-in approval gates shall be codified and maintained in code repositories.

k) **Fast Feedback**
The continuous integration pipeline shall be configured in a way to aid faster feedback and corrective actions

6.4 Continuous Testing

Continuous Testing is the practice of testing the software continuously and thereby ensuring that quality software is released for business consumption on a frequent basis or on-demand. At every stage of the software development lifecycle, Continuous Testing significantly reduces the risk of delivering defective software.

The software shall be tested but not limited to Unit Testing, Code Quality Check, Functional Testing, UI Testing, API Testing, Integration Testing, Regression Testing, Performance Testing, Security Testing, Smoke Testing, Breakpoint/ Stress testing, Endurance/Soak testing, Failover testing. Continuous testing happens at every stage of Software Development Life Cycle to ensure that good quality software is released continuously. It shall ensure that every change is tested early, and defects are identified early. This prevents defects in the later stages of the software development lifecycle thereby reducing overall time of software delivery.

6.4.1 *Key Objective*

To test the software across the SDLC and provide rapid feedback to enable quicker delivery of high-quality software.

6.4.2 *Sub Practices*

Continuous Testing requires testing early and often to find and eliminate defects earlier in the software development lifecycle. The following key sub-practices shall be implemented for a successful Continuous Testing practice:

a) **Availability of Test Strategy**
In DevOps journey, the changes are carried out incrementally by the cross-functional team. Therefore, the test strategy comprising of a right test framework, test data management and tools for test and defect management compatible with DevOps shall be created for end-to-end testing activities.

b) **Creating Test Scenarios/Test Cases**
Test scenarios and test cases shall be documented clearly and unambiguously in an ALM (Application Lifecycle Management Tool)/Test Management Tool.

c) **Test Auto Triggering**
Test Cases from the predefined test suites shall be triggered and executed as soon as the latest code is integrated into the existing codebase during the Continuous Integration practice. The entire test execution cycle shall be integrated with DevOps pipeline so that tests can be auto triggered and tracked through the orchestration tool.

d) **Recording of Test Result**

Test Results shall be recorded after execution of each test case with the associated supporting artefacts like Screenshots/Documentation. Any Defects, identified during the test shall be logged against the failed Test Cases.

e) **Integration between the test suite and ALM tool to log defects**
Defects identified during Continuous testing shall become part of the Product Backlog.

f) **On Demand Test Infrastructure**
Test environments may be provisioned using Infrastructure as Code and available on-demand to enable continuous testing.

6.5 Continuous Delivery (CD)

Continuous Delivery is important because this enables incremental release of value to the end customers. Continuous delivery means the Product increment has passed all gates (Continuous Integration and Continuous Testing) and is ready for release. At the end of a sprint, the final and feature-ready application is deployed to the specific environment.

6.5.1 *Key Objective*
To deliver values faster and better to the end customer incrementally for improved customer experience.

6.5.2 *Sub Practices*
The following are the key sub-practices for a successful Continuous Delivery practice:

a) **Standard process to establish CD implementation along with approval gates**
End to end pipeline with required approval gates for Continuous Delivery (CD) shall be enabled to perform series of automated steps to deliver change.

b) **Integration of Infrastructure as Code (IaC) practice in the Pipeline**
Infrastructure provisioning, configuration and orchestration shall be automated through code. IaC scripts shall be kept under version control.

c) **Automated deployment based on environments**
Automated deployment provides the ability to deploy changed software to the required environment without any manual intervention. The data for functional and non-functional testing shall be available on demand.

g) **Management and Governance of Staging Environments**
The entire development cycle and its various testing shall be carried out on different staging environments. Availability of different staging environments to perform development of software and testing shall be ensured.

d) **Secret Management backed with Role Based Access**

All credentials, certificates, confidential details shall be stored in a system which can be accessed only by authorized users.

e) **Role Based Access**

Role Based Access Control (RBAC) to Pipelines and Configuration of Automation Servers shall be ensured so that access is available only to authorized users.

f) **Fully automated Pipeline with built-in approval gates**

The pipeline, its configurations and built-in approval gates shall be codified and maintained in code repositories.

6.6 Continuous Deployment

Continuous deployment is the practice of deploying a product or product component with every addition or update to the product or component to an environment. The target environment may be production staging or production.

6.6.1 *Key Objective*

To minimize cycle time to deliver value with high quality to the end customer.

6.6.2 *Sub Practices*

The following are the key sub-practices for a successful Continuous Deployment practice:

a) **Release Management**

Release management shall start at the beginning of SDLC (Software Development Life Cycle) with release planning to determine the scope of release, timelines, milestones in the release cycle, dependencies between teams and constraints.

Release Go/ No Go decision shall be taken collaboratively by product development, IT operations, business and other stakeholders based on pre-defined release criteria. Release Notes (documentation that contains information about the release such as Release number, Release date and list of impacted components, known bugs and so on) shall be generated in the system. The release approval workflow for stakeholders shall also be enabled in the system.

Deployment is included as part of Release management and effectiveness of deployments contribute to the release efficiency. Additionally, automation of release workflow and timely approval of the same, positively influence the release efficiency.

b) **Automated Deployment**

Automated Deployment shall be implemented and supported through a continuous deployment tool.

It shall be enabled through

- Automated toll gates for tests and verification
- Automating the infrastructure provisioning at deployment
- Orchestrating Application deployment

- Risk-reducing strategies for example, canary releases, blue-green deployments and rolling updates, as documented by the organization
- Automated execution of post-deployment smoke tests

c) **Automated Rollback Process**
The deployment process shall be complemented with a proper and robust rollback process so that in case there is any issue in the updated software, it can be rolled back smoothly with no or minimum disruption.

6.7 Continuous Monitoring

Continuous monitoring is the practice involved in monitoring the performance of elements of the IT estate, i.e. servers, routers, networks, storage, and applications.

6.7.1 *Key Objective*

To reduce business loss and increase efficiency of the IT systems and help IT to take preventive and proactive actions.

6.7.2 *Sub Practices*

The following key sub-practices shall be implemented for a successful Continuous Monitoring practice:

a) **Business Objective for IT**
Business objectives shall be clearly defined. These are key to deciding and designing what and how the monitoring is needed for the organization.

b) **Key Performance Indicators**
A well-defined set of KPIs shall be defined for the IT strategy of the organization. These KPIs shall provide insight into the project teams' productivity, progress of the value delivery and other relevant parameters important for the organization. (Refer Annex C)

c) **Continuous monitoring tools**
Continuous monitoring shall be implemented utilizing the capability of the industry standard monitoring tools (one or more) and integrating these tools to build observability (insights) of IT applications along with underlined infrastructure towards traceability of events and actions for fault localization, early detection, and faster resolution.

d) **Automated Monitoring Dashboard**
Automated dashboards (based on data collection through different monitoring tools) shall be implemented for providing better insight for different parameters and help decision makers to take decisions effectively.

e) **Data collection and Analysis**
A mechanism of alerts shall be set up based on pre-defined events and collect data from various systems. The data collected from monitoring tools shall be analyzed and insights identified for faster corrective, preventive and optimization actions.

f) **Alerts and Event Management**
The alerts and events raised by the monitoring tools shall be correlated. The outliers shall be removed using automated processes.

g) **Data-driven Decision Making**
Data driven inference and decision making shall be implemented by the methods mentioned above so as to help the organization take appropriate action quickly and proactively.

h) **Self-Healing**
A system shall have self-healing capabilities so as to enable recovery from any sort of abnormality and minimize any disruption of the services. The action shall be decided based on the data, alerts, events, and logs collected from IT systems.

i) **Responsiveness to disruptions and emergency**
The organization shall ensure that systems and processes are in place to enable actions towards responsiveness to incidents and disasters, along with early detection and faster recovery ensuring minimum business disruption.

6.8 Continuous Experimentation

Continuous experimentation shall adopt an approach to continuously develop ideas and experiment on IT solutions. These experiments enable continuous learning and improvement based on feedback from these experiments.

6.8.1 *Key Objective*

To improve the services and end user experience by continuously ideating, experimenting, learning and industrializing successful ideas

6.8.2 *Sub Practices*

The following are the key sub-practices for a successful Continuous Experimentation practice:

a) **Research and Learning**
The project teams shall participate in events and seminars as defined and documented by the organization, research and gather inputs on how services can be improved for their customers.

b) **Robust environments for experimentation**
For continuous experimentation, project teams shall be provided with an appropriate and adequate environment which is cost effective and provides ease of experimentation.

c) **Container Management**

Containerization of software shall be followed by the organization. Containers help to deploy any software seamlessly and at scale. They enable portable software which can be independently tested and deployed irrespective of the host environment. The ability of having independent isolated application software in containers and the ease of deploying it across environments, once "built" enables experimentation in isolated containerized software applications.

d) Continuous Compliance

Continuous compliance as a sub-practice shall be followed for continuously monitoring the system and identifying any compliance related issues.

e) Hypothesis Based Design and Development

Hypothesis based development practices shall be followed which is a prototype-based method to iteratively design, develop, test, and validate until user accepts it.

7 DevOps Maturity

DevOps maturity model shall be referenced by organizations to adopt DevOps practices guided by the defined principles. While teams progress in the DevOps maturity, it is desired that the team is aware and follows agile practices and processes. (Refer Annexure A)

7.1 Definitions

7.1.1 *Level 1*

The project team shall have frequent planning meetings with the involvement of right stakeholders. The project team shall use a version control repository to check-in and automatically compile the code.

7.1.2 *Level 2*

The project team shall use collaboration tools for communication. Project team shall use tools to manage the artifacts. The team shall use a defined branching strategy. Build, unit testing and system testing shall be automated.

7.1.3 *Level 3*

The projects shall demonstrate a high degree of automation which ensures predictability and resilience in the IT applications. Project team shall use monitoring tools and automated dashboards to get continuous feedback and make informed decisions. DevOps Pipeline shall be fully automated and release management techniques are used. The team has required environments for experimentation. (Refer Annex B)

7.1.4 *Level 4*

The projects shall exhibit a high degree of resilience by adopting practices like Chaos engineering which provide a high degree of confidence for systems to withstand adverse or turbulent execution environments. Production deployment shall be automated and self-healing of issues shall be in place.

7.2Practices and its Maturity

A particular maturity level is achieved only when all the practice requirements of that level are satisfied. Full conformance would require that all the practice requirements for the respective maturity levels are satisfied. It may be noted that higher maturity levels must satisfy sub-practices of lower maturity level.

7.2.1 *Continuous Planning*

a) Level 1

The team shall have frequent planning and regular ongoing communication while also involving the right stakeholders for the planning meetings.

b) Level 2

Team shall use appropriate collaboration tools for communication, continuously validate the business priority of requirements and checks for granularity of the requirements to be taken for the sprint. The focus shall be on Value Stream based delivery.

c) Level 3

The planning/ALM tools shall be integrated with other delivery tools which brings in traceability in the process.

d) Level 4

Insight from proactive measures such as chaos engineering, ethical hacking, etc. shall be used for planning and hypothesis-based design and development.

7.2.2 *Configuration Management*

a) Level 1

The team creates configuration strategy and shall start leveraging configuration management tools for code and documents.

b) Level 2

Team shall develop branching strategy and use it for software development with multiple teams working together to enhance capability for parallel development. Along with the introduction of branching strategy, teams shall use the right configuration tool to configure build artefacts like binaries.

c) Level 3

The team shall use the capability of configuration management tool to configure container artefacts like container images.

d) Level 4

The team shall establish the DevOps tool chain to scale up the design and development process leveraging intelligence of the configuration management tool. The project team shall use advanced techniques such as containerization for different configuration needs for different environments (e.g. lower environments, hybrid, etc.) for scaling.

7.2.3 *Continuous Integration*

a) Level 1

Automated compilation shall be in place once the code is checked into the repository.

b) Level 2

Vision for CI/CD pipeline shall be documented. The build process and unit tests shall be automated in pipeline. Static code analysis tools shall be in place. The code shall be merged into relevant branch only after the approval of reviewer. Tools shall be used for artifact management.

c) Level 3

Static Analysis Security Testing (SAST) tools shall be used to identify security issues early. Role Based Access Control shall be followed for pipeline configuration and access to run the pipelines. Secret Management tools shall be used to effectively store all credentials, certificates, and confidential details.

d) Level 4

The presence of configured pipelines to optimize the infrastructure while the pipeline is executing shall be ensured.

7.2.4 *Continuous Testing*

a) Level 1

The team shall have a test strategy created for end-to-end testing activities. The team shall document test scenarios, test cases and test results so that audit trails are maintained.

b) Level 2

Test Management Tool shall be used to create and store test scenarios and test cases. Test execution cycle shall be integrated with DevOps pipeline so that tests can be auto triggered and tracked through the orchestration tool. Test cases till integration and system testing shall automatically be executed in the pipeline after the latest code is integrated into the existing codebase and approved for test execution.

c) Level 3

The test management tool shall be integrated with ALM tool so that the critical defects that need code changes can be tracked in product/sprint backlog in ALM tool. Static Analysis Security Testing (SAST) tools shall be used to identify security issues early and software composition analysis for open source and vulnerability scans. Test Infrastructure is available on demand.

d) Level 4

Dynamic Application Security Testing shall be automated. Security tests shall be integrated into the pipeline. Intelligent execution of test cases is based on the code changes.

7.2.5 *Continuous Delivery*

a) Level 1

The deployment into development and test environment shall be automated through scripts or standard tools.

b) Level 2

Infrastructure provisioning shall be automated using scripts. The team shall have deployment integrated with DevOps pipeline for development and system test environment with required approval gates. Secret management backed with role-based access shall be enabled. The configuration of the server shall be automated using DevOps pipeline and performed by designated roles.

c) Level 3

Deployment shall be integrated with DevOps pipeline for non-production environments with required approval gates. Environment provisioning shall be automated using Infrastructure as Code (IaC). Release Management Techniques such as Canary or Blue Green deployments shall also be adopted.

d) Level 4

AI based pipeline execution shall be adopted to ensure optimum infrastructure utilization. The deployment in a non-production environment shall be conducted for the software.

7.2.6 *Continuous Deployment*

a) Level 1

Team Manages and Governs the Staging Environments.

b) Level 2

The steps to release the software shall be automated using scripts and executed manually.

c) Level 3

Deployment shall be integrated with DevOps pipeline for production environments with required approval gates. The rollback process shall be automated and will be triggered through the pipeline in case of issues during deployment.

d) Level 4

One-click deployment using DevOps pipeline shall be enabled.

7.2.7 *Continuous Monitoring*

a) Level 1

Documentation of commands and schedule used for monitoring shall be available.

b) Level 2

The business objective for the IT team shall be clearly articulated. The set of KPIs agreed for monitoring shall be defined and the process of data collection for monitoring the agreed KPIs shall also be defined and articulated. Required monitoring tools shall be procured, installed, and enabled. The practice of responsiveness to disruptions and emergencies shall be inculcated.

c) Level 3

The monitoring tools shall be integrated with the pipeline to provide automatic feedback to the ALM tool. The capability to analyze the gathered data shall be enabled. An AI-driven review of data and decision-making capability shall be enabled. Using the gathered data and derived decision, automated dashboard shall be made available to the stakeholders.

d) Level 4

Based on the feedback and decision derived in level 3, the action to be taken using self-healing capabilities shall be ensured

7.2.8 *Continuous Experimentation*

a) Level 1

Team shall keep itself updated by attending seminars/events related to the upcoming industry trends, as documented by the organization

b) Level 2

The team shall collaborate, do research on further improvement areas, and document them.

c) Level 3

Team shall create Proof of Concepts for experimentation of new trends and techniques to improve quality of the services for end users. Projects may experiment in lower environments to improve resilience, reliability and stability.

d) Level 4

Use of Chaos Engineering practice to induce artificial failure into the system and build confidence in the system’s capability shall be practiced. Tools shall be used to orchestrate and manage containers. Automated monitoring of compliance issues according to the organization’s need shall be done.

Annexure A
DevOps Maturity Model Summary
(**Clause** *7*)

The table summarizes activities for various maturity levels of the practices documented in Section 5. If there is conflict in understanding and interpretation of the practices in the table below with respect to the activities documented in Section 5, the activities in Section 5 shall prevail.

Sl No (1)	Practice (2)	Level 1 (3)	Level 2 (4)	Level 3 (5)	Level 4 (6)
i)	Continuous Planning	a) Frequent planning meetings b)Right stakeholders' involvement	a) Appropriate collaboration tools for communication b) Continuous validation of business priority and requirements c) Granular requirements d) Value Stream based delivery	a) Consider insight from feedback received from monitoring tools. b) Integrated planning/ALM tools with delivery tools	a) Insight from proactive measures such as chaos engineering, ethical hacking, etc. b) Hypothesis based planning and design
ii)	Configuration Management	a) Configuration strategy b) Configuration management tool used for application codes and design documents	a) Scaled branching strategy. b) Right configuration tool is used to configure deployment artefacts.	a) Configuration management tool used for other different artefacts like dockers etc.	a) DevOps tool chain to scale up in hybrid environment. b) Advanced techniques such as containerization for different configuration needs
iii)	Continuous Integration	a) Manual integration of code	a) Automated build process integrated with CI/CD pipeline b) Continuous Integration build as per scheduled trigger	a) Frequent continuous integration and automated unit test b) Continuous integration build is triggered on check in	a) Robust continuous integration check gate along with unit testing, code quality, security analysis and code

					composition analysis
iv)	Continuous Testing	a) Test strategy for end-to-end testing b) Test management tools c) Recording of Test results	a) Test execution cycle integrated with DevOps pipeline b) Test scripts are auto triggered and tracked through the orchestration tool. c) Test cases till QA/SIT region are automatically executed with approval gate	a) Test management tool is integrated with ALM tool b) Defects are tracked as user stories in ALM tool c) On demand test Infrastructure	b) Automated Dynamic Application Security Testing with tools. c) The security tests are integrated into the pipeline. d) Intelligent execution of test cases based on the code changes
v)	Continuous Delivery	a) Automated deployment into development and test environment b) Infrastructure provisioning is done using scripts.	a) Deployment integrated with DevOps pipeline for dev and system test environment b) Secret management backed with role-based access is enabled. c) The configuration of the server is automated using DevOps pipeline	a) Deployment integrated with DevOps pipeline for other non-prod environments b) Infrastructure as Code (IaC). c) Release Management Techniques - Canary or Blue Green deployments	a) AI based pipeline execution b) The deployment in non-prod environment only for the required software.
vi)	Continuous Deployment	a) Team Manages and Governs the Staging Environments.	a) Automated release using scripts with manual intervention	a) The deployment integrated with DevOps pipeline for prod environments b) Automated rollback process	a) One-click deployment
vii)	Continuous Monitoring	a) Documentation of queries and schedule used for monitoring	a) Clear business objective for IT b) Defined and agreed KPIs for monitoring. c) Defined and articulated Data gathering process against the agreed KPIs d) Required monitoring tools are procured, installed and enabled.	a) Integrated monitoring tools with pipeline b) Automatic feedback to the ALM tool c) Capability to analyze gathered data d) AI-driven review of data and decision-making capability e) Automated dashboard	a) Self-healing capabilities

			e) Responsiveness to Disruption and Emergency	f) Alert and Event Management	
	Continuous Experimentation	a) Awareness of industry trends.	a) Collaborate, research and document improvement areas	a) Proof of Concepts created for experimentation b) Culture to encourage experimentation in lower environments	a) Chaos Engineering practice b) Tool to orchestrate and manage containers for CI/CD pipeline c) Automated monitoring of compliance

Annexure B
Automation Categories and Technology Requirement

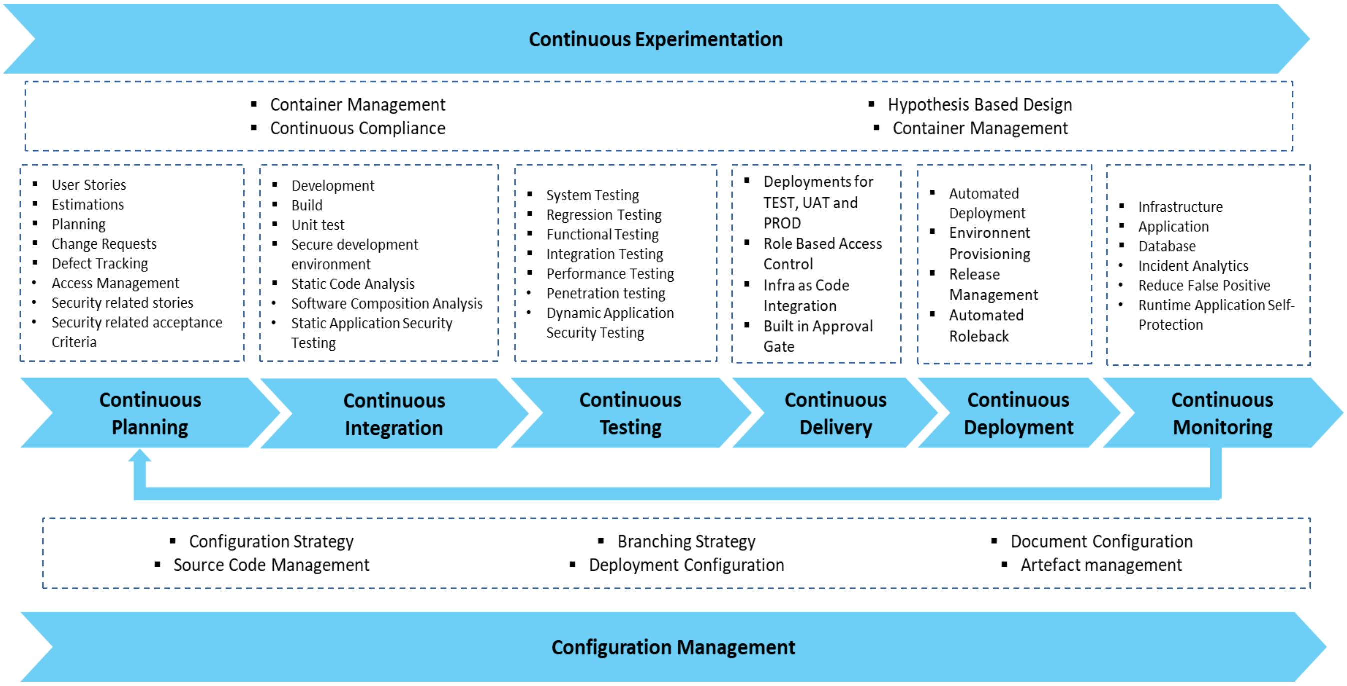
(Informative)

Clause *7.1.3*

Automation in the following practices will benefit the teams using this standard:

a) Continuous Planning
b) Configuration Management
c) Continuous Integration
d) Continuous Testing
e) Continuous Delivery
f) Continuous Deployment
g) Continuous Monitoring
h) Continuous Experimentation

Tools need to be integrated to perform end-to-end activities without any human interventions.



Annexure C
Key Performance Indicators (KPIs)

(Informative)

C of **Clause** *6.7.2*

The following is the sample list of useful KPIs. An organization may select one or more of these KPIs if deemed fit. Organizations will need to identify any additional KPIs relevant to them.

Sl No **(1)**	**KPI** **(2)**	**Description** **(3)**
i)	Technical Debt Ratio	Ratio of cost to fix the code vs build
ii)	Code Commit Per day	Total number of commits divided by number of coding days
iii)	Build Per Day	Total number of builds divided by number of coding days
iv)	Build Success Rate	Percentage of successful build per release
v)	Deployment Frequency	Rate of deployment of code change into production environment
vi)	Throughput	Number of work item delivered in unit time (e.g. day, week etc.)
vii)	Cumulative Flow	Total number of completed task over a known period of time
viii)	Mean Time to Repair	Average time to repair a system
ix)	Average Productivity	Output per unit
x)	Defect Leakage	Number of defects missed during a testing phase
xi)	Availability	It the percentage of system uptime with respect to sum of up and down time
xii)	Latency	Time taken by a packet of data to be transferred over network
xiii)	Resource Usage	Percentage of resource used to process request during peak load
xiv)	Change Failure Rate	Percentage of failed deployment with respect to total number of deployments
xv)	Lead Time for Change	The time between when a code is placed at trunk branch and when it is ready for deployment

Annexure D
Artificial Intelligence (AI) and Generative AI in DevOps
(Informative)

DevOps focusses on automation-first software development to achieve frequent delivery of reliable products. AI and Gen AI support the practices in DevOps by bringing in consistency to the repetitive and complex tasks, enabling continuous experimentation and strengthening the overall automation practices.

AI/ Gen AI use cases encompass the end-to-end of software development lifecycle. They provide contextualized business scenarios for industry domains based on the hypothesis generated from service requirements, market, and technology inputs and so on.

Planning data such as requirements, function-point inputs and other historical data are fed to the AI models to predict software delivery risk, get factors and relevant data for effective planning. User story articulation is done leveraging AI models on requirements.

In the development stage AI/ Gen AI generates code, performs pattern analysis on code to detect code quality and security issues, provide suggestions for fixes and more.

It enables easy debugging and troubleshooting of issues during Continuous Integration and automates platform engineering tasks and workflows (such as, creating and managing containerization clusters). It generates synthetic test data, functional and performance test scenarios, identifies critical functionalities and scenarios for focused testing and so on in the testing stage.

In the Continuous Deployment stage, it provides insights to deployment risks based on analysis of health data of infrastructure and applications from monitoring and observability tools.

IT Service Management (ITSM) platforms may get feed from AI/ Gen AI to orchestrate automated response and resolution workflows for incidents. AI performs pattern analysis on data from disparate sources, triggers alerts and auto-healing actions to provide proactive protection from potential incidents and threats. AI/ Gen AI capabilities drive contextual conversation and user experience and help in predicting service disruptions. AI/ Gen AI drives collaboration and knowledge-sharing. It creates knowledge articles for service management and explainable code with annotations and documentation. It boosts the efficiency of the underlying tasks and workflows in the DevOps pipeline and improves the overall productivity, product quality and speed of delivery.

AI/ Gen AI capabilities and usage are still in the evolving stage and organizations should run proof-of-concepts to contextualize and take judicious decision for leveraging AI/ Gen AI. The AI models should be designed to avoid data bias and should be continuously vetted with the output.

Annexure E

LIST OF REFERRED STANDARDS

Clause *3*

ISO/IEC/IEEE 24765	Systems and software engineering — Vocabulary
IS/ISO/IEC 2382:2015	Information technology - Vocabulary

For dated references, only the edition cited applies.

For undated references, the latest edition of the referenced document (including any amendments) applies.

Annexure F

Composition of Committee