Engineering Standard Operating Procedure (SOP)

Effective Date: January 2026
Version: 1.0
Department: Engineering
Last Updated: January 2026

---

1. Purpose and Scope

This SOP defines standard practices for software engineering, development workflows, code quality standards, and project delivery. It applies to all engineers and development teams and is designed to: - Ensure consistent code quality and architecture - Reduce defects and technical debt - Improve collaboration and knowledge sharing - Accelerate delivery timelines - Maintain operational excellence

---

2. Development Workflow

2.1 Project Initiation

1. **Requirements Gathering:** Product/Project team collects and documents requirements
2. **Design Review:** Architects review and approve design specifications (24 hours)
3. **Task Breakdown:** Lead Engineer breaks down into actionable tasks
4. **Estimation:** Development team estimates effort using planning poker (story points)
5. **Sprint Planning:** Tasks assigned to developers; sprint duration = 2 weeks

2.2 Development Process

Branching Strategy (Git Flow): - `main` branch: Production-ready code (protected branch) - `develop` branch: Integration branch for features - Feature branches: `feature/[task-id]-[feature-name]` - Hotfix branches: `hotfix/[bug-id]-[issue-name]`

Code Commit Requirements: - Commit frequently (multiple times per day) - Meaningful commit messages: `[Task-ID] Brief description of change` - Link commits to task tracking system - No direct commits to `main` or `develop`

2.3 Code Review Process

Mandatory for all pull requests (PRs): - Minimum 2 approvals required before merge - Reviewers assigned based on code ownership - Review completed within 24 hours - Comments resolved before merge - Automated tests must pass

Review Checklist: - [ ] Code follows style guide and naming conventions - [ ] Logic is clear and efficient - [ ] No hardcoded values or sensitive data - [ ] Adequate error handling implemented - [ ] Unit tests cover new functionality - [ ] Documentation updated - [ ] No performance degradation

2.4 Testing Strategy

Unit Testing: - Minimum 80% code coverage for new code - Test framework: Jest (JavaScript), PyTest (Python) - Tests run on every commit (CI/CD pipeline) - Failing tests block merge to `develop`

Integration Testing: - Test API endpoints and integrations - Database interactions verified - External service mocks used in dev environment - Tests run before staging deployment

User Acceptance Testing (UAT): - Conducted in staging environment - Product team and end users validate functionality - Bug fixes deployed within 48 hours - Approval required before production release

Performance Testing: - Load testing conducted for high-traffic features - Threshold: Response time <500ms at peak load - Database query optimization validated - Results documented in deployment checklist

2.5 Deployment Process

Development Environment: - Continuous deployment on every merge to `develop` - Automated tests validate build

Staging Environment: - Manual trigger by QA/Product team - 48-hour UAT window - Rollback plan documented

Production Environment: - Manual deployment by DevOps/Engineering Lead - Deployment window: Tue–Thu, 10:00 AM – 2:00 PM IST - Rollback plan prepared; testing validated - Post-deployment monitoring enabled (2 hours minimum)

---

3. Coding Standards and Best Practices

3.1 Code Style Guidelines

JavaScript/TypeScript: - Use ES6+ syntax (const/let, arrow functions, destructuring) - Linting: ESLint with Airbnb configuration - Formatting: Prettier (2-space indentation) - No console.log in production code; use proper logging

Python: - Follow PEP 8 style guide - Linting: Pylint, Black formatter - Type hints for all functions - Docstrings for public methods

General: - Clear variable naming: `const userEmail` not `const ue` - Comment complex logic; avoid obvious comments - Maximum function length: 50 lines - Maximum file length: 300 lines

3.2 Architecture Principles

- **Modularity:** Separate concerns; reusable components
- **DRY (Don't Repeat Yourself):** Extract common patterns
- **KISS (Keep It Simple):** Prioritize clarity over cleverness
- **SOLID Principles:** Single Responsibility, Open/Closed, Liskov, Interface Segregation, Dependency Inversion
- **Error Handling:** Explicit, informative error messages

3.3 Database Practices

- **Migrations:** Version control for schema changes
- **Naming:** Descriptive table/column names; lowercase with underscores
- **Indexing:** Indexes on frequently queried columns
- **Queries:** Use prepared statements to prevent SQL injection
- **Backups:** Daily automated backups; tested monthly

3.4 Security Best Practices

- **Input Validation:** All user inputs validated and sanitized
- **Authentication:** Secure token management; no hardcoded credentials
- **Authorization:** Role-based access controls (RBAC) enforced
- **Secrets Management:** Use environment variables or secure vaults (never commit secrets)
- **Dependencies:** Regular security updates; vulnerability scanning weekly
- **Logging:** Log errors and security events; no sensitive data in logs

---

4. Documentation Requirements

4.1 Code Documentation

- **README.md:** Project setup, running tests, deployment instructions
- **API Documentation:** Swagger/OpenAPI spec for all endpoints
- **Architecture Decision Records (ADRs):** Rationale for major technical decisions
- **Runbooks:** Troubleshooting guides for common issues
- **Changelog:** Track changes per release

4.2 Documentation Maintenance

- Updated with every feature release
- Code comments updated when logic changes
- Outdated docs removed or archived
- Wiki maintained for knowledge sharing
- Documentation reviewed quarterly

---

5. Incident Management and Debugging

5.1 Bug Severity Levels

- **Critical:** System down, data loss, security breach → Fix within 4 hours
- **High:** Major functionality broken, significant performance degradation → Fix within 1 business day
- **Medium:** Minor feature not working as designed → Fix within 1 week
- **Low:** Cosmetic issues, minor enhancement requests → Fix as capacity allows

5.2 Debugging Process

1. **Reproduce:** Confirm issue in local/staging environment
2. **Isolate:** Identify affected code and root cause
3. **Document:** Create issue in tracking system with reproduction steps
4. **Fix:** Develop fix; review with team
5. **Validate:** Verify fix in multiple environments
6. **Deploy:** Follow deployment process

5.3 Production Incidents

- **On-call rotation:** 24/7 coverage with escalation path
- **Page:** Alert within 5 minutes of detection
- **Response:** Acknowledge within 15 minutes; update status every 30 minutes
- **Resolution:** Target <1 hour for critical; communicate with stakeholders
- **Post-Mortem:** Conducted within 24 hours; action items tracked

---

6. Performance and Optimization

6.1 Performance Monitoring

- Application performance monitoring (APM) enabled in production
- Key metrics tracked: Response time, error rate, throughput, resource utilization
- Alerts configured for anomalies (response time >1s, error rate >1%)

- Weekly performance review meetings

6.2 Optimization Practices

- Database queries optimized; query execution plans reviewed
- Caching strategies implemented (Redis for frequently accessed data)
- Lazy loading and code splitting for frontend
- Asset optimization: Minification, compression, CDN delivery
- Regular profiling to identify bottlenecks

---

7. Team Collaboration and Communication

7.1 Daily Standup

- **Frequency:** Daily, 15 minutes maximum
- **Format:** What did I complete? What's next? Any blockers?
- **Tools:** Jira, Slack, or synchronous meeting
- **Attendance:** All team members (remote-friendly)

7.2 Code Ownership and Accountability

- Lead engineers assigned to critical systems
- On-call responsibilities rotated monthly
- Knowledge sharing through pair programming and code reviews
- Mentoring junior engineers (2+ hours/week)

7.3 Knowledge Sharing

- Monthly tech talks (internal presentations)
- Wiki documentation for common problems
- Slack channels for technical discussions
- Quarterly architectural reviews and planning

---

8. Tools and Infrastructure

Version Control: Git (GitHub/GitLab)
CI/CD: Jenkins, GitHub Actions, or GitLab CI
Monitoring: Datadog, New Relic, or CloudWatch
Issue Tracking: Jira or Linear
Documentation: Confluence or Wiki.js
Communication: Slack, Jira, Email

---

9. Training and Skill Development

- New team members paired with mentors for 4 weeks
- Quarterly skill development plans
- Annual training budget per engineer: 50,000–100,000
- Internal certifications encouraged (AWS, Kubernetes, etc.)

---

10. Compliance and Auditing

- Code audits conducted quarterly
- Security reviews for sensitive systems
- Compliance with company policies and external standards
- Process effectiveness reviewed semi-annually

Engineering Lead: [Name]
DevOps Lead: [Name]
Next Review Date: July 2026
Contact: engineering@company.com