**Biblia de Código Limpio - Buenas Prácticas de Desarrollo**

Versión: 2025.1 | Licencia: MIT

Una colección destilada y opinada de prácticas que mantienen las bases de código del mundo real legibles, refactorizables y amigables para el equipo. Desde convenciones de nomenclatura hasta etiqueta de revisión, desde SOLID hasta higiene de CI—esta es la lista de verificación que desearíamos haber tenido desde el primer día.

**Principios Fundamentales**

**La Regla del Boy Scout**

"Deja el código mejor de como lo encontraste." Cada cambio, sin importar lo pequeño, debe mejorar la calidad del código.

**La Regla del Tres**

Cuando escribes el mismo código por tercera vez, es hora de abstraer y refactorizar.

**El Principio de Responsabilidad Única**

Una función, clase o módulo debe tener una, y solo una, razón para cambiar.

**Estándares de Calidad de Código**

**Convenciones de Nomenclatura**

**Variables y Funciones**

* Usa nombres que revelen la intención: **diasDesdeCreacion** no **ddc**
* Evita desinformación: **listaCuentas** solo si es realmente una Lista
* Haz distinciones significativas: **obtenerUsuariosActivos()** no **obtenerUsuarios2()**

**Clases y Tipos**

* Las clases deben ser sustantivos: **Cliente**, **AnalizadorDirecciones**
* Los métodos deben ser verbos: **guardar()**, **calcularTotal()**
* Las variables booleanas deben sonar como verdadero/falso: **esValido**, **tienePermiso**

**Nombres de Archivos y Directorios**

* Usa kebab-case para archivos: **servicio-usuario.js**
* Usa nombres claros y descriptivos: **procesador-pagos.ts** no **utilidades.ts**
* Agrupa archivos relacionados: **componentes/**, **servicios/**, **modelos/**

**Reglas de Diseño de Funciones**

**Pequeñas y Enfocadas**

* Las funciones deben tener 20 líneas o menos
* Haz una cosa y hazla bien
* Máximo de 3 parámetros
* Sin efectos secundarios a menos que estén explícitamente nombrados

**Lista de Verificación de Calidad de Pruebas**

* Las pruebas son independientes y aisladas
* Cada prueba valida un comportamiento
* Las pruebas usan descripciones significativas
* Sin lógica condicional en las pruebas
* Las pruebas son rápidas (<100ms cada una)
* Cobertura para caminos felices y casos extremos
* Mockea dependencias externas apropiadamente

**Estrategia de Pirámide de Pruebas**

* **Pruebas Unitarias**: 70% - Rápidas, prueban componentes individuales
* **Pruebas de Integración**: 20% - Prueban interacciones entre componentes
* **Pruebas E2E**: 10% - Solo journeys críticos de usuario

**Pautas de Revisión de Código**

**Qué Buscar**

**Funcionalidad**

* ¿El código resuelve el problema correctamente?
* ¿Se manejan apropiadamente los casos extremos?
* ¿El manejo de errores es comprehensivo?

**Calidad de Código**

* ¿El código es legible y mantenible?
* ¿Hay malos olores de código o anti-patrones?
* ¿La complejidad es apropiada para el problema?

**Pruebas**

* ¿Hay suficientes pruebas para los cambios?
* ¿Las pruebas cubren tanto casos de éxito como de falla?
* ¿Las pruebas son claras y bien estructuradas?

**Etiqueta de Revisión**

**Para Revisores**

* Proporciona retroalimentación constructiva y accionable
* Explica el "por qué" detrás de las sugerencias
* Reconoce buen trabajo y patrones
* Revisa dentro de 24 horas después de la asignación

**Para Autores**

* Mantén los PR pequeños y enfocados (<400 líneas)
* Escribe descripciones claras e instrucciones de prueba
* Estate abierto a comentarios y sugerencias
* Resuelve todos los comentarios antes de mergear

**Complejidad de Algoritmos**

* Prefiere O(n log n) sobre O(n²) para datasets grandes
* Usa estructuras de datos apropiadas (Set para tests de membresía)
* Cachea operaciones costosas cuando sea apropiado

**DevOps y CI/CD**

**Estándares de Pipeline**

**Pipeline de Build Debe Incluir**

* Linting y formateo de código
* Ejecución de pruebas unitarias
* Escaneo de vulnerabilidades de seguridad
* Creación de artefactos de build
* Construcción de imágenes Docker (si aplica)

**Puertas de Calidad**

* Cobertura de pruebas >= 80%
* Cero vulnerabilidades de seguridad críticas
* Todas las pruebas pasando
* Aprobación de revisión de código requerida

**Estrategia de Entornos**

**Principios de Aplicación Twelve-Factor**

1. Codebase: Un codebase rastreado en control de versiones
2. Dependencies: Declara y aísla dependencias explícitamente
3. Config: Almacena config en variables de entorno
4. Backing services: Trata como recursos adjuntos
5. Build, release, run: Separa estrictamente etapas de build y run

**Monitoreo y Observabilidad**

**Métricas Esenciales**

* Rendimiento de aplicación (tiempos de respuesta, tasas de error)
* Métricas de negocio (tasas de conversión, actividad de usuario)
* Métricas de infraestructura (CPU, memoria, uso de disco)
* Métricas personalizadas específicas de tu dominio

**Estándares de Colaboración en Equipo**

**Protocolos de Comunicación**

**Daily Standups**

* En qué trabajé ayer
* En qué estoy trabajando hoy
* Algún bloqueo o necesidad de ayuda

**Compartición de Conocimiento**

**Documentación de Código**

* Documentación de API para interfaces públicas
* Registros de decisiones de arquitectura
* Archivos README para setup y deployment
* Comentarios de código solo para el "por qué", no el "qué"

**Pautas de Programación en Pareja**

* Roles conductor/navegador claramente definidos
* Cambio regular de roles
* Enfocado en transferencia de conocimiento, no solo en completar

**Métricas y Puertas de Calidad**

**Métricas de Calidad de Código**

* **Complejidad Ciclomática**: <= 10 por función
* **Índice de Mantenibilidad**: >= 85
* **Duplicación de Código**: <= 5%
* **Cobertura de Pruebas**: >= 80% para caminos críticos

**Métricas de Rendimiento del Equipo**

* **Tiempo de Ciclo**: < 24 horas para features pequeñas
* **Tiempo de Lead**: < 1 semana para features medianas
* **Frecuencia de Deployment**: Diario a semanal
* **Tasa de Fallo de Cambios**: < 5%

**Gestión de Deuda Técnica**

* Evaluación regular de deuda y priorización
* Tiempo dedicado para refactorización (regla del 20%)
* Seguimiento de deuda en herramientas de gestión de proyectos

**Mejora Continua**

**Prácticas de Retrospectiva**

* Retrospectivas regulares del equipo (quincenal)
* Enfocado en mejora de procesos, no en culpar
* Elementos accionables con dueño claro
* Seguimiento de elementos de acción previos

**Cultura de Aprendizaje**

* Tiempo dedicado de aprendizaje (10-20%)
* Charlas técnicas y talleres regulares
* Asistencia a conferencias y compartición
* Contribuciones a open source alentadas

**Licencia**

Licencia MIT - ver archivo LICENSE para detalles.

**Agradecimientos**

Principios inspirados por:

* "Código Limpio" de Robert C. Martin
* "El Programador Pragmático" de Andrew Hunt y David Thomas
* "Domain-Driven Design" de Eric Evans
* Mejores prácticas de la industria de Google, Microsoft, Amazon y Netflix