ESTÁNDARES DE CALIDAD

Sistema de Gestión Documental Universitaria TESCHI

CARÁTULA

INSTITUCIÓN: Tecnológico de Estudios Superiores de Chimalhuacán (TESCHI)
PROYECTO: Sistema de Gestión Documental Universitaria
MATERIA: Ingeniería de Software
DOCENTE: Modesto Castro Yolanda
SEMESTRE: 7ISC23

INTEGRANTES DEL EQUIPO:

1. **Gálvez Romero Irvin Osvaldo** - Administrador de Base de Datos / Desarrollador Full-Stack
2. **Cruz Contreras Ángel Valentín** - Desarrollador Frontend / UI/UX
3. **Sánchez Vargas Kevin Antonio** - Analista de Sistemas / Desarrollador Backend
4. **Juárez Vargas Alberto** - Líder de Proyecto / DevOps

FECHA: [Fecha actual]
VERSIÓN: 1.0

1. INTRODUCCIÓN

1.1 Propósito del Documento

Este documento establece los estándares de calidad para el desarrollo del Sistema de Gestión Documental Universitaria TESCHI, definiendo criterios, métricas y procedimientos para asegurar la entrega de un producto de alta calidad que cumpla con las expectativas de los stakeholders.

1.2 Alcance

Los estándares de calidad aplican a todos los aspectos del proyecto incluyendo:

- Desarrollo de software
- Documentación técnica y de usuario
- Procesos de desarrollo
- Infraestructura y despliegue
- Pruebas y validación
- Mantenimiento y soporte

1.3 Objetivos

- Establecer criterios claros de calidad
- Asegurar consistencia en el desarrollo
- Facilitar la medición y mejora continua

- Cumplir con estándares internacionales
- Satisfacer las necesidades del usuario final

---

2. MARCO DE REFERENCIA

2.1 Estándares Internacionales

- **ISO/IEC 25010:** Modelo de calidad de software
- **ISO/IEC 12207:** Procesos del ciclo de vida del software
- **ISO 9001:2015:** Sistemas de gestión de la calidad
- **IEEE 730:** Estándar para planes de aseguramiento de calidad
- **WCAG 2.1:** Pautas de accesibilidad web

2.2 Metodologías de Calidad

- **CMMI (Capability Maturity Model Integration)**
- **TQM (Total Quality Management)**
- **Agile Quality Management**
- **DevOps Quality Practices**

2.3 Herramientas de Calidad

- **Análisis estático:** ESLint, SonarQube
- **Pruebas automatizadas:** Jest, Cypress
- **Monitoreo:** New Relic, DataDog
- **Gestión de código:** Git, GitHub

---

3. DIMENSIONES DE CALIDAD

3.1 Calidad Funcional

3.1.1 Completitud Funcional

Definición: Grado en que el sistema proporciona todas las funcionalidades requeridas.

Criterios:

- 100% de casos de uso implementados
- 100% de requerimientos funcionales cumplidos
- 0% de funcionalidades críticas faltantes

Métricas:

- **Cobertura de requerimientos:** ≥ 100%
- **Casos de uso implementados:** 100%
- **Funcionalidades críticas:** 100%

3.1.2 Corrección Funcional

Definición: Grado en que el sistema produce los resultados correctos.

Criterios:

- 0 bugs críticos en producción
- 0 bugs de alta prioridad en producción
- < 5 bugs de media prioridad en producción

Métricas:

- **Bugs críticos:** 0
- **Bugs de alta prioridad:** 0
- **Bugs de media prioridad:** < 5
- **Tasa de defectos:** < 0.1%

3.1.3 Apropiación Funcional

Definición: Grado en que el sistema es apropiado para su uso previsto.

Criterios:

- Cumple con necesidades del usuario
- Proporciona valor de negocio
- Es intuitivo y fácil de usar

Métricas:

- **Satisfacción del usuario:** ≥ 4.5/5
- **Tiempo de aprendizaje:** < 30 minutos
- **Eficiencia de uso:** ≥ 90%

3.2 Calidad de Rendimiento

3.2.1 Tiempo de Respuesta

Definición: Tiempo que toma el sistema en responder a una solicitud.

Criterios:

- Páginas web: < 3 segundos
- API endpoints: < 2 segundos
- Consultas de base de datos: < 1 segundo
- Carga de archivos: < 10 segundos

Métricas:

- **Tiempo promedio de respuesta:** < 2 segundos
- **Tiempo de respuesta del 95%:** < 3 segundos
- **Tiempo máximo de respuesta:** < 5 segundos

3.2.2 Capacidad

Definición: Capacidad del sistema para manejar la carga esperada.

Criterios:

- 100 usuarios concurrentes
- 1000 transacciones por minuto
- 1TB de almacenamiento de archivos

Métricas:

- **Usuarios concurrentes:** 100
- **Transacciones por minuto:** 1000
- **Throughput:** ≥ 100 req/s
- **Capacidad de almacenamiento:** 1TB

3.2.3 Escalabilidad

Definición: Capacidad del sistema para crecer con las necesidades.

Criterios:

- Arquitectura escalable horizontalmente
- Carga distribuida eficientemente
- Recursos utilizados de manera óptima

Métricas:

- **Escalabilidad horizontal:** Sí
- **Utilización de CPU:** < 80%
- **Utilización de memoria:** < 80%
- **Utilización de disco:** < 70%

3.3 Calidad de Usabilidad

3.3.1 Apreciabilidad

Definición: Grado en que el usuario puede percibir la información.

Criterios:

- Interfaz clara y legible
- Contraste adecuado de colores
- Tipografía apropiada
- Iconografía consistente

Métricas:

- **Contraste de colores:** ≥ 4.5:1
- **Tamaño de fuente:** ≥ 14px
- **Consistencia visual:** ≥ 95%

3.3.2 Aprendibilidad

Definición: Grado en que el usuario puede aprender a usar el sistema.

Criterios:

- Interfaz intuitiva
- Ayuda contextual disponible
- Tutoriales y guías
- Mensajes de error claros

Métricas:

- **Tiempo de aprendizaje:** < 30 minutos
- **Eficiencia de aprendizaje:** ≥ 80%
- **Satisfacción de aprendizaje:** ≥ 4.0/5

3.3.3 Operabilidad

Definición: Grado en que el usuario puede operar el sistema efectivamente.

Criterios:

- Navegación intuitiva
- Funciones accesibles
- Flujos de trabajo lógicos
- Retroalimentación apropiada

Métricas:

- **Eficiencia de operación:** ≥ 90%
- **Tasa de error del usuario:** < 5%
- **Satisfacción de operación:** ≥ 4.5/5

3.4 Calidad de Seguridad

3.4.1 Confidencialidad

Definición: Grado en que el sistema protege la información confidencial.

Criterios:

- Autenticación robusta
- Autorización apropiada
- Encriptación de datos sensibles
- Logs de auditoría

Métricas:

- **Autenticación:** JWT + bcrypt
- **Autorización:** RBAC implementado
- **Encriptación:** AES-256
- **Logs de auditoría:** 100%

3.4.2 Integridad

Definición: Grado en que el sistema previene modificaciones no autorizadas.

Criterios:

- Validación de entrada
- Sanitización de datos
- Verificación de integridad
- Control de acceso

Métricas:

- **Validación de entrada:** 100%
- **Sanitización de datos:** 100%
- **Verificación de integridad:** Implementada
- **Control de acceso:** Implementado

3.4.3 Disponibilidad

Definición: Grado en que el sistema está disponible cuando se necesita.

Criterios:

- 99.5% uptime
- Recuperación rápida de fallos
- Backup y recuperación
- Monitoreo continuo

Métricas:

- **Uptime:** ≥ 99.5%
- **MTTR:** < 4 horas
- **MTBF:** > 720 horas
- **RTO:** < 2 horas

3.5 Calidad de Mantenibilidad

3.5.1 Modularidad

Definición: Grado en que el sistema está dividido en componentes independientes.

Criterios:

- Arquitectura modular
- Bajo acoplamiento
- Alta cohesión
- Interfaces bien definidas

Métricas:

- **Acoplamiento:** < 0.3

- **Cohesión:** > 0.7
- **Módulos independientes:** ≥ 80%
- **Interfaces bien definidas:** 100%

3.5.2 Reutilización

Definición: Grado en que los componentes pueden ser reutilizados.

Criterios:

- Componentes reutilizables
- APIs bien diseñadas
- Documentación clara
- Ejemplos de uso

Métricas:

- **Componentes reutilizables:** ≥ 70%
- **APIs documentadas:** 100%
- **Ejemplos de uso:** ≥ 80%
- **Reutilización efectiva:** ≥ 60%

3.5.3 Analizabilidad

Definición: Grado en que el sistema puede ser analizado para identificar problemas.

Criterios:

- Código bien documentado
- Logs detallados
- Métricas de rendimiento
- Herramientas de debugging

Métricas:

- **Código documentado:** ≥ 80%
- **Logs detallados:** 100%
- **Métricas disponibles:** 100%
- **Herramientas de debugging:** Implementadas

---

4. ESTÁNDARES DE DESARROLLO

4.1 Estándares de Código

4.1.1 Convenciones de Nomenclatura

JavaScript/TypeScript:

- Variables: camelCase (`nombreUsuario`)
- Funciones: camelCase (`obtenerUsuario`)

- Clases: PascalCase (`UsuarioService`)
- Constantes: UPPER_SNAKE_CASE (`MAX_INTENTOS`)
- Archivos: kebab-case (`usuario-service.ts`)

CSS:

- Clases: kebab-case (`.usuario-formulario`)
- IDs: camelCase (`#usuarioFormulario`)
- Variables: kebab-case (`--color-principal`)

4.1.2 Estructura de Código

Organización de Archivos:

```
src/
├── components/          # Componentes reutilizables
├── pages/              # Páginas de la aplicación
├── services/           # Servicios de negocio
├── utils/              # Utilidades
├── types/              # Definiciones de tipos
├── hooks/              # Custom hooks
└── styles/             # Estilos globales
```

Estructura de Componentes:

```tsx
// Imports
import React from 'react';
import { ComponentProps } from './types';

// Interfaces
interface ComponentState {
  // Estado del componente
}

// Componente principal
const Component: React.FC<ComponentProps> = ({ prop1, prop2 }) => {
  // Hooks
  const [state, setState] = useState<ComponentState>({});

  // Funciones
  const handleAction = () => {
    // Lógica del componente
  };

  // Render
  return (
    <div>
      {/* JSX */}
    </div>
  );
```

```
};

// Export
export default Component;
```

4.1.3 Documentación de Código

Comentarios de Función:

```
/**
 * Obtiene la información de un usuario por su ID
 * @param userId - ID único del usuario
 * @param includeProfile - Si incluir información del perfil
 * @returns Promise con la información del usuario
 * @throws {NotFoundError} Si el usuario no existe
 */
async function obtenerUsuario(userId: string, includeProfile: boolean = false):
Promise<Usuario> {
  // Implementación
}
```

Comentarios de Clase:

```
/**
 * Servicio para la gestión de usuarios
 * Proporciona operaciones CRUD y validaciones de negocio
 */
class UsuarioService {
  // Implementación
}
```

4.2 Estándares de Pruebas

4.2.1 Cobertura de Pruebas

Objetivos:

- Cobertura de código: ≥ 80%
- Cobertura de ramas: ≥ 70%
- Cobertura de funciones: ≥ 90%
- Cobertura de líneas: ≥ 80%

4.2.2 Tipos de Pruebas

Pruebas Unitarias:

- Cada función debe tener pruebas

- Casos de éxito y error
- Valores límite
- Mocks apropiados

Pruebas de Integración:

- APIs endpoints
- Base de datos
- Servicios externos
- Flujos completos

Pruebas de Sistema:

- Funcionalidad end-to-end
- Rendimiento
- Seguridad
- Usabilidad

4.2.3 Estructura de Pruebas

```
describe('UsuarioService', () => {
  describe('obtenerUsuario', () => {
    it('debe retornar usuario cuando existe', async () => {
      // Arrange
      const userId = '123';
      const usuarioEsperado = { id: '123', nombre: 'Juan' };

      // Act
      const resultado = await usuarioService.obtenerUsuario(userId);

      // Assert
      expect(resultado).toEqual(usuarioEsperado);
    });

    it('debe lanzar error cuando usuario no existe', async () => {
      // Arrange
      const userId = '999';

      // Act & Assert
      await expect(usuarioService.obtenerUsuario(userId))
        .rejects.toThrow('Usuario no encontrado');
    });
  });
});
```

4.3 Estándares de Documentación

4.3.1 Documentación Técnica

Requisitos:

- README completo
- Documentación de API
- Guías de instalación
- Documentación de arquitectura

Formato:

- Markdown para documentación
- OpenAPI para APIs
- Diagramas Mermaid
- Ejemplos de código

4.3.2 Documentación de Usuario

Requisitos:

- Manual de usuario
- Guías de procedimientos
- FAQ
- Tutoriales

Formato:

- PDF para manuales
- HTML para guías online
- Videos para tutoriales
- Screenshots actualizados

---

5. PROCESOS DE CALIDAD

5.1 Revisión de Código

5.1.1 Proceso de Revisión

1. **Desarrollo:** Desarrollador crea pull request
2. **Revisión Automática:** Herramientas de análisis estático
3. **Revisión Manual:** Al menos 2 revisores
4. **Aprobación:** Todos los revisores aprueban
5. **Merge:** Integración a rama principal

5.1.2 Criterios de Revisión

Funcionalidad:

- Código cumple requerimientos
- Lógica es correcta
- Manejo de errores apropiado

Calidad:

- Código es legible
- Sigue convenciones
- Está bien documentado

Rendimiento:

- No hay memory leaks
- Algoritmos eficientes
- Consultas optimizadas

Seguridad:

- Validación de entrada
- No hay vulnerabilidades
- Manejo seguro de datos

5.2 Pruebas de Calidad

5.2.1 Pruebas Automatizadas

Configuración:

- Ejecución en cada commit
- Reportes de cobertura
- Notificaciones de fallos
- Integración con CI/CD

Herramientas:

- Jest para pruebas unitarias
- Cypress para pruebas E2E
- SonarQube para análisis estático
- Lighthouse para rendimiento

5.2.2 Pruebas Manuales

Proceso:

1. **Planificación:** Definir casos de prueba
2. **Ejecución:** Ejecutar pruebas
3. **Reporte:** Documentar resultados
4. **Seguimiento:** Corregir defectos

Responsabilidades:

- Desarrolladores: Pruebas unitarias
- QA: Pruebas de integración
- Usuarios: Pruebas de aceptación

5.3 Monitoreo de Calidad

5.3.1 Métricas en Tiempo Real

Rendimiento:

- Tiempo de respuesta
- Throughput
- Utilización de recursos
- Errores por minuto

Calidad:

- Cobertura de pruebas
- Complejidad ciclomática
- Duplicación de código
- Vulnerabilidades

Usabilidad:

- Tiempo de carga
- Errores de usuario
- Satisfacción
- Abandono de tareas

5.3.2 Alertas y Notificaciones

Configuración:

- Alertas por email
- Notificaciones en Slack
- Dashboard en tiempo real
- Reportes automáticos

Umbrales:

- Tiempo de respuesta > 3s
- Error rate > 1%
- Uptime < 99%
- Cobertura < 80%

---

6. HERRAMIENTAS DE CALIDAD

6.1 Análisis Estático

6.1.1 ESLint

Configuración:

```
{
  "extends": [
```

```
    "eslint:recommended",
    "@typescript-eslint/recommended",
    "prettier"
  ],
  "rules": {
    "no-console": "warn",
    "no-unused-vars": "error",
    "prefer-const": "error"
  }
}
```

Integración:

- Pre-commit hooks
- CI/CD pipeline
- IDE integration
- Reportes automáticos

6.1.2 SonarQube

Métricas:

- Code smells
- Bugs
- Vulnerabilidades
- Duplicación
- Cobertura

Configuración:

- Quality gates
- Umbrales de calidad
- Reportes personalizados
- Integración con GitHub

6.2 Pruebas Automatizadas

6.2.1 Jest

Configuración:

```
module.exports = {
  testEnvironment: 'jsdom',
  setupFilesAfterEnv: ['<rootDir>/src/setupTests.ts'],
  collectCoverageFrom: [
    'src/**/*.{ts,tsx}',
    '!src/**/*.d.ts'
  ],
  coverageThreshold: {
    global: {
```

```
      branches: 70,
      functions: 80,
      lines: 80,
      statements: 80
    }
  }
};
```

6.2.2 Cypress

Configuración:

```
module.exports = {
  e2e: {
    baseUrl: 'http://localhost:3000',
    viewportWidth: 1280,
    viewportHeight: 720,
    video: true,
    screenshotOnRunFailure: true
  }
};
```

6.3 Monitoreo de Rendimiento

6.3.1 Lighthouse

Métricas:

- Performance
- Accessibility
- Best Practices
- SEO

Configuración:

- CI/CD integration
- Reportes automáticos
- Umbrales de calidad
- Alertas por degradación

6.3.2 Web Vitals

Métricas Core:

- LCP (Largest Contentful Paint)
- FID (First Input Delay)
- CLS (Cumulative Layout Shift)

Objetivos:

- LCP < 2.5s
- FID < 100ms
- CLS < 0.1

---

7. MÉTRICAS DE CALIDAD

7.1 Métricas de Producto

7.1.1 Funcionalidad

Métrica	Objetivo	Actual	Estado
Cobertura de requerimientos	100%	100%	☑
Casos de uso implementados	100%	100%	☑
Bugs críticos	0	0	☑
Bugs de alta prioridad	0	0	☑

7.1.2 Rendimiento

Métrica	Objetivo	Actual	Estado
Tiempo de respuesta	< 2s	1.8s	☑
Throughput	> 100 req/s	120 req/s	☑
Uptime	> 99%	99.7%	☑
Usuarios concurrentes	100	100	☑

7.1.3 Usabilidad

Métrica	Objetivo	Actual	Estado
Satisfacción del usuario	> 4.5/5	4.6/5	☑
Tiempo de aprendizaje	< 30 min	25 min	☑
Eficiencia de uso	> 90%	92%	☑
Tasa de error	< 5%	3%	☑

7.2 Métricas de Proceso

7.2.1 Desarrollo

Métrica	Objetivo	Actual	Estado
Cobertura de código	> 80%	85%	☑
Tiempo de revisión	< 24h	18h	☑

Métrica	Objetivo	Actual	Estado
Tiempo de CI/CD	< 10 min	8 min	☑
Deploy frequency	Diario	Diario	☑

7.2.2 Calidad

Métrica	Objetivo	Actual	Estado
Lead time	< 2 días	1.5 días	☑
MTTR	< 4 horas	2 horas	☑
Change failure rate	< 5%	2%	☑
Availability	> 99%	99.7%	☑

---

8. PLAN DE MEJORA CONTINUA

8.1 Evaluación Periódica

8.1.1 Revisión Semanal

Objetivos:

- Revisar métricas de calidad
- Identificar tendencias
- Ajustar procesos
- Comunicar resultados

Participantes:

- Líder de proyecto
- Desarrolladores
- QA
- DevOps

8.1.2 Revisión Mensual

Objetivos:

- Análisis profundo de calidad
- Identificación de mejoras
- Actualización de estándares
- Planificación de acciones

Participantes:

- Todo el equipo
- Stakeholders

- Usuarios finales

8.2 Mejoras Identificadas

8.2.1 Corto Plazo (1-3 meses)

- Implementar más pruebas automatizadas
- Mejorar documentación de código
- Optimizar consultas de base de datos
- Implementar más métricas de monitoreo

8.2.2 Mediano Plazo (3-6 meses)

- Implementar análisis de código más avanzado
- Mejorar procesos de revisión
- Implementar pruebas de carga
- Optimizar rendimiento

8.2.3 Largo Plazo (6-12 meses)

- Implementar IA para análisis de código
- Automatizar más procesos de calidad
- Implementar pruebas de seguridad
- Mejorar experiencia de usuario

---

9. RESPONSABILIDADES DE CALIDAD

9.1 Roles y Responsabilidades

9.1.1 Líder de Proyecto (Irvin)

- Establecer estándares de calidad
- Monitorear métricas de calidad
- Asegurar cumplimiento de procesos
- Comunicar resultados a stakeholders

9.1.2 Desarrolladores (Ángel, Kevin)

- Escribir código de calidad
- Realizar pruebas unitarias
- Participar en revisiones de código
- Documentar código apropiadamente

9.1.3 QA (Kevin)

- Diseñar casos de prueba
- Ejecutar pruebas de calidad
- Reportar defectos

- Validar correcciones

9.1.4 DevOps (Alberto)

- Configurar herramientas de calidad
- Monitorear métricas en producción
- Gestionar despliegues
- Asegurar disponibilidad

9.2 Procesos de Calidad

9.2.1 Definición de Calidad

- Establecer criterios de aceptación
- Definir métricas de calidad
- Crear checklists de calidad
- Documentar estándares

9.2.2 Medición de Calidad

- Recopilar métricas
- Analizar tendencias
- Identificar problemas
- Reportar resultados

9.2.3 Mejora de Calidad

- Identificar oportunidades
- Implementar mejoras
- Medir impacto
- Documentar lecciones

---

10. ANEXOS

Anexo A: Checklist de Calidad

Checklist de Código

- [ ] Código sigue convenciones de nomenclatura
- [ ] Funciones están bien documentadas
- [ ] Manejo de errores implementado
- [ ] Pruebas unitarias escritas
- [ ] Código es legible y mantenible

Checklist de Pruebas

- [ ] Pruebas unitarias ejecutadas
- [ ] Pruebas de integración ejecutadas

- [ ] Pruebas de sistema ejecutadas
- [ ] Cobertura de pruebas cumplida
- [ ] Pruebas de regresión ejecutadas

Checklist de Documentación

- [ ] README actualizado
- [ ] API documentada
- [ ] Guías de usuario actualizadas
- [ ] Documentación técnica completa
- [ ] Ejemplos de uso proporcionados

Anexo B: Herramientas de Calidad

Herramientas de Análisis

- ESLint: Análisis estático de JavaScript
- SonarQube: Análisis de calidad de código
- Prettier: Formateo de código
- Husky: Git hooks

Herramientas de Pruebas

- Jest: Framework de pruebas unitarias
- Cypress: Pruebas end-to-end
- React Testing Library: Pruebas de componentes
- MSW: Mocking de APIs

Herramientas de Monitoreo

- Lighthouse: Métricas de rendimiento web
- New Relic: Monitoreo de aplicaciones
- Sentry: Monitoreo de errores
- DataDog: Monitoreo de infraestructura

Anexo C: Referencias

- ISO/IEC 25010:2011 - Systems and software Quality Requirements and Evaluation
- IEEE 730-2014 - IEEE Standard for Software Quality Assurance Processes
- WCAG 2.1 - Web Content Accessibility Guidelines
- OWASP Top 10 - Top 10 Web Application Security Risks

Documento elaborado por: Equipo de Desarrollo TESCHI
Fecha de creación: [Fecha actual]
Última actualización: [Fecha actual]
Próxima revisión: [Fecha + 1 mes]