Forma de controlar y costear la calidad del sistema de información

Introducción

Controlar y costear la calidad de un sistema de información es crucial para asegurar que el sistema sea eficaz, eficiente y cumpla con los requisitos del usuario.

Es por ello que deben considerarse varios aspectos.

Definir requisitos claros

Requisitos Funcionales y No Funcionales: Asegurarse de tener una lista detallada de lo que el sistema debe hacer (funcional) y de las características de rendimiento y calidad (no funcional) como la seguridad, escalabilidad, y usabilidad.

Documentación: Mantener la documentación actualizada y clara para todos los stakeholders.

Desarrollar un Plan de Calidad

Estándares y Metodologías: Utilizar estándares de calidad, que proporcionan directrices para asegurar la calidad.

Planificación de Pruebas: Definir un plan de pruebas exhaustivo que incluya pruebas unitarias, integración, sistema y aceptación.

Implementar Control de Calidad

Revisión de Código: Realizar revisiones de código regulares para identificar problemas y mejorar la calidad.

Auditorías de Calidad: Realizar auditorías periódicas del sistema para asegurar que cumple con los estándares de calidad establecidos.

Utilizar Herramientas de Gestión de Calidad

- Herramientas de Pruebas: Usar herramientas como Selenium, JUnit o TestComplete para automatizar y facilitar las pruebas.
- Herramientas de Monitoreo: Implementar herramientas de monitoreo como Nagios o Prometheus para seguir el rendimiento y la disponibilidad del sistema.

Gestión de Riesgos

- Identificación y Evaluación de Riesgos: Identificar los riesgos potenciales que puedan afectar la calidad y establece estrategias para mitigarlos.
- Planes de Contingencia: Desarrollar planes de contingencia para manejar problemas cuando ocurran.

Capacitación y Formación Continua

- Entrenamiento del Equipo: Asegurarse de que el equipo de desarrollo esté capacitado en las mejores prácticas de desarrollo y control de calidad.
- Actualización Continua: Mantener al equipo actualizado con las últimas tecnologías y metodologías.

Costeo de la Calidad

Costos de Prevención: Inversiones en capacitación, herramientas y procesos para prevenir defectos (por ejemplo, inversión en herramientas de pruebas y revisión de código).

Costos de Evaluación: Costos asociados con la prueba y auditoría del sistema.

Costos de Fallos Internos: Costos asociados con defectos encontrados antes de la entrega al usuario (por ejemplo, tiempo perdido en corrección).

Costos de Fallos Externos: Costos asociados con defectos encontrados por los usuarios después de la implementación (por ejemplo, soporte técnico y correcciones).

Retroalimentación Continua

- Feedback de Usuarios: Recoge y analiza el feedback de los usuarios para identificar áreas de mejora y problemas no detectados anteriormente.

- Mejora Continua: Utiliza la retroalimentación para implementar mejoras continuas en el sistema y en los procesos de desarrollo.

Análisis de Métricas

- Métricas de Calidad: Medir indicadores clave como la tasa de defectos, la cobertura de pruebas, y el tiempo de respuesta.

- Análisis de Costos: Realizar un análisis regular de los costos asociados con la calidad para ajustar presupuestos y recursos.

Métricas de software

Métricas de calidad del código

Métrica	Descripción
Cobertura de Pruebas	Proporción del código que ha sido ejecutado durante las pruebas. Ejemplo: Si 80% del código ha sido probado, la cobertura es del 80%.
Complejidad Ciclomática:	Medida de la complejidad del código, basada en el número de caminos independientes a través del código. Ejemplo: Un método con una complejidad ciclomática de 10 sugiere 10 caminos lógicos independientes.
Defectos por KLOC (Mil Líneas de Código):	Número de defectos encontrados en cada 1,000 líneas de código. Ejemplo: Si se encuentran 5 defectos en 10,000 líneas de código, el número de defectos por KLOC es 0.5.
Número de Errores por Función:	Promedio de errores encontrados por cada función del software. Ejemplo: Un sistema con 100 funciones y 200 errores tiene un promedio de 2 errores por función.

Métricas de Desempeño

Métrica	Descripción
Tiempo de Respuesta	Tiempo que tarda el sistema en responder a una solicitud del usuario. Ejemplo: Si una solicitud toma 2 segundos en procesarse, el tiempo de respuesta es de 2 segundos.
Tiempo de Carga	Tiempo que tarda una página o una aplicación en cargar completamente. Ejemplo: Una página web que carga en 3 segundos tiene un tiempo de carga de 3 segundos
Uso de Recursos	Cantidad de recursos del sistema utilizados, como CPU, memoria o almacenamiento. Ejemplo: Una aplicación que usa el 70% de la CPU durante su ejecución

Métricas de Productividad

Métrica	Descripción
Líneas de Código por Unidad de Tiempo:	Cantidad de líneas de código escritas por unidad de tiempo (por ejemplo, por semana). Ejemplo: Un desarrollador que escribe 200 líneas de código por semana.
Tasa de Defectos:	Número de defectos detectados durante un período de desarrollo. Ejemplo: Un proyecto que detecta 50 defectos en una semana.
Tiempo de Desarrollo:	Tiempo total invertido en desarrollar una característica o módulo. Ejemplo: Si una característica se desarrolla en 40 horas, el tiempo de desarrollo es de 40 horas.

Métricas de Mantenimiento

Métrica	Descripción
Tiempo de Resolución de Defectos:	Tiempo promedio necesario para resolver un defecto una vez que se ha reportado. Ejemplo: Si se tarda un promedio de 5 días en solucionar defectos, el tiempo de resolución es de 5 días.
Frecuencia de Cambios:	Número de cambios realizados en el código durante un período de tiempo. Ejemplo: Un equipo que realiza 30 cambios en una semana.
Costo de Mantenimiento:	Costo asociado con la corrección de defectos y la mejora del software. Ejemplo: Si se gastan $10,000 en corrección de errores en un mes, el costo de mantenimiento es de $10,000.

Métricas de Satisfacción del Usuario

Métrica	Descripción
Tasa de Error del Usuario:	Número de errores cometidos por los usuarios al interactuar con el software. Ejemplo: Si los usuarios cometen 15 errores en una sesión, la tasa de error es 15.
NPS (Net Promoter Score):	Métrica que mide la disposición de los usuarios a recomendar el software. Ejemplo: Un NPS de 50 indica un alto nivel de satisfacción del usuario.
Tiempo de Uso:	Cantidad de tiempo que los usuarios pasan utilizando el software. Ejemplo: Si los usuarios pasan un promedio de 30 minutos al día en la aplicación, el tiempo de uso es de 30 minutos.

Ingeniería y Sistemas
UNIVERSIDAD FRANCISCO GAVIDIA

Métricas de Gestión de Proyectos

Métrica	Descripción
Cumplimiento de Plazos:	Porcentaje de tareas o hitos completados dentro del plazo establecido. Ejemplo: Si se completaron 90 de 100 tareas a tiempo, el cumplimiento de plazos es del 90%.
Desviación de Costos:	Diferencia entre el presupuesto planeado y el costo real. Ejemplo: Si el presupuesto era $50,000 y el costo real es $55,000, la desviación de costos es de $5,000.
Porcentaje de Cumplimiento de Requisitos:	Proporción de requisitos que se cumplen en el software final. Ejemplo: Si 45 de 50 requisitos se cumplen, el porcentaje de cumplimiento es del 90%.

Ejemplo

Imagine un sistema con varios módulos

- Clientes.cs
- Pedidos.cs
- Facturacion.cs
- Reportes.cs

Ejemplo

- Supongamos que usamos **Visual Studio Code Metrics** o **SonarQube** para calcular:

Archivo	LDC (líneas de código)	Complejidad Ciclomática	Métodos	Comentarios (%)	Duplicación (%)
Clientes.cs	450	12	15	25%	2%
Pedidos.cs	700	25	20	10%	8%
Facturacion.cs	1200	35	30	5%	15%
Reportes.cs	300	8	10	30%	1%

Análisis paso a paso

Tamaño (LDC y métodos)

Facturacion.cs es el módulo más grande (1200 LDC, 30 métodos).

Esto sugiere que está sobrecargado y puede dividirse en clases más pequeñas.

Complejidad ciclomática

Valores recomendados: menor de 10 es sencillo, entre 10–20 moderado, más de 20 complejo.

Pedidos.cs (25) y Facturacion.cs (35) son riesgosos → más difíciles de probar y mantener.

Comentarios

Facturacion.cs solo tiene 5% de comentarios → mala documentación.

Reportes.cs está mejor documentado (30%).

Duplicación

Facturacion.cs tiene 15% de duplicación de código → oportunidad de refactorización.

Código duplicado genera más errores y retrabajo.

Interpretación y acciones

- **Refactorizar Facturacion.cs:**

Dividir en submódulos (ej. GeneradorFactura, ValidadorImpuestos, GestorPagos).

Reducir duplicación aplicando patrones de diseño (Strategy, Factory).

- **Mejorar Pedidos.cs:**

Reducir complejidad dividiendo métodos largos en funciones más pequeñas.

Aumentar comentarios para mayor mantenibilidad.

- **Buenas prácticas a reforzar:**

Mantener módulos de tamaño moderado (≈300–600 LDC).

Buscar complejidad ≤ 15 por clase.

Mantener duplicación < 5%.

EJERCICIO EN CSHARP

- Maintainability Index (Índice de Mantenibilidad)

Escala de 0 a 100 → más alto = mejor.

Verde (≥ 20) = aceptable

Amarillo (10–19) = riesgo

Rojo (< 10) = crítico

- Cyclomatic Complexity (Complejidad ciclomática)

< 10 = bajo riesgo

10–20 = medio

20 = alto

- Class Coupling (Acoplamiento de clases)

Cuántas dependencias tiene una clase.

Alto → difícil de mantener.

- Lines of Code (LOC)

El tamaño del método/clase.