CALIDAD DE CODIGO

ARQUITECTURA DE SOFTWARE

PABLO ENRIQUE QUINTERO, JUAN DIEGO ROMERO, KAMILT BEJARANO DIAZ

CONTENIDO

DEFINICION

La calidad de código es el conjunto de propiedades internas del código fuente que permiten que el software sea mantenible, comprensible, seguro, eficiente y fácil de evolucionar.

CONCEPTO PRINCIPAL

La Calidad de Código no es solo la ausencia de errores simples, sino que constituye una parte central de la arquitectura del software. Se define como el conjunto de propiedades internas del código fuente que aseguran que el software sea mantenible, comprensible, seguro, eficiente y fácil de evolucionar.

OBJETIVOS

Objetivo 1:

Asegurar la sostenibilidad y fiabilidad del sistema, haciendo que el código existente sea fácil de modificar y actualizar

Objetivo 2:

Detección temprana y la mitigación de debilidades críticas, enfocándose en fortalecer la seguridad del programa, minimizando vulnerabilidades, y en optimizar el rendimiento mediante la eficiencia algorítmica y el uso correcto de recursos

ESTANDARES

- **ISO/IEC 25010**: Establece un marco de referencia para la calidad de código dividido en 8 áreas.
- **ISO 9001**: Requisitos genéricos y aplicables para cualquier organización.
- **ISO 10005 (2018)**: Plan de calidad para todo el ciclo de vida.
- **ISO 33000**: Seguimiento a la evolución del sistema para identificar puntos de mejora.
- **ISO 12207**: Estándar para el seguimiento de la evolución de los procesos durante el ciclo de vida.
- **IEEE 730 (2002)**: Define que es software de calidad y establece un plan para asegurar la calidad del software.
- **ISO 5055**: Mide debilidades críticas del sistema, enfocandose en seguridad, confianza, rendimiento y mantenibilidad.

IEE730 (2002)

- **Propósito y alcance del software.**

El estándar exige describir qué se quiere asegurar, para qué y en qué contexto.

- **Referencias normativas.**

Debe listar documentos, estándares y políticas internas que aplican al proceso.

- **Definiciones y acrónimos.**

Para garantizar lenguaje común en el proyecto.

- **Organización del aseguramiento de calidad.**

Define roles, responsabilidades y autoridades del equipo de SQA.

- **Documentación del software.**

Especifica qué documentos deben generarse, revisarse y controlarse (requisitos, diseño, pruebas, manuales, etc.).

- **Normas, prácticas y convenciones**.

Literalmente establece que el plan debe definir los estándares de codificación, diseño, pruebas, revisiones y cualquier directriz obligatoria para el desarrollo.

- **Revisiones y auditorías.**

Indica que deben incluirse tipos de revisiones (peer reviews, inspecciones), criterios de entrada y salida, responsables y artefactos a evaluar.

- **Pruebas del software.**

Debe describir la estrategia general de pruebas, niveles, criterios y documentación requerida.

- **Gestión de problemas y reporte de errores.**

El estándar ordena definir cómo se registran, clasifican, rastrean y cierran los defectos.

- **Control de cambios.**

Indica cómo se gestionan solicitudes de cambio, aprobaciones y seguimiento.

- **Herramientas, técnicas y metodologías.**

IEEE 730 requiere listar las herramientas usadas para análisis, pruebas, seguimiento, revisiones, etc.

- **Control de registros de calidad.**

Cómo se almacenan, protegen y mantienen evidencias (logs, reportes, métricas).

- **Medidas y métricas.**

El estándar obliga a definir qué métricas de calidad se usarán y cómo se medirán.

- **SQA durante la adquisición y suministro.**

Si aplica, debe incluir procesos de calidad para proveedores y componentes externos.

ISO/IEC 25010

Mantenibilidad

- Asegurar que el código existente sea fácil de modificar y actualizar
- Estructura del código
- Complejidad ciclomática
- Duplicación
- Claridad de diseño
- Principios SOLID
- Patrones Arquitectonicos

Seguridad

- Minimizar las posibles vulnerabilidades del programa para disminuir el riesgo de ataques.
- Validación de entradas
- Manejo de excepciones
- Gestión de secretos
- Dependencia de librerías con vulnerabilidades

ISO/IEC 25010

Fiabilidad

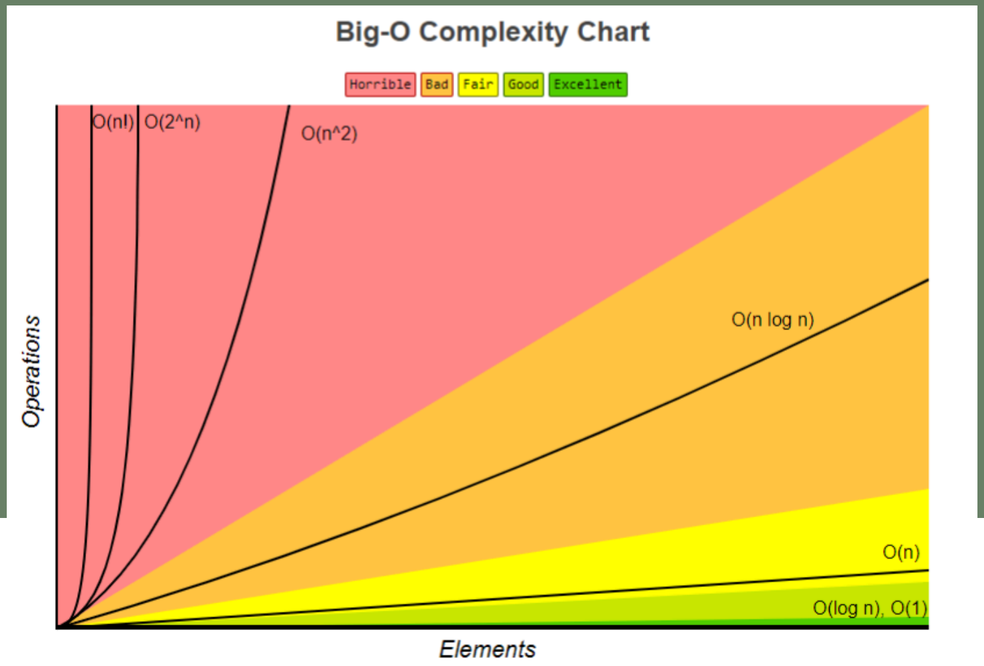
- Asegurar que el sistema entero sea capaz de ejecutar sus tareas sin errores, y sea capaz de manejar errores de entrada o salida sin bloquearse.
- manejo correcto de errores
- robustez del código
- tests automatizados
- mecanismos de fallbacks

Eficiencia

- Minimizar todo lo posible el uso de recursos, ya sea tiempo o memoria.
- Eficiencia algorítmica
- Estructura de los datos
- Paralelismo
- Uso de memoria

COMPLEJIDAD ALGORÍTMICA

Tener en cuenta los atributos de calidad a la hora de escoger algoritmos



POLITICAS Y BUENAS PRÁCTICAS

Legibilidad:

Hacer código que sea fácil de leer y entender.

Estratégias:

- **Gestión de deuda técnica:**
  - Evitar la acomulación de tareas, malas prácticas o falta de documentación y correción del código para evitar que el costo de mantenimiento incremente a largo plazo.
- **Código autodocumentado:**
  - Documentar el código de forma clara y exprésiva, de tal modo que no necesite explicaciones adicionales o genere malinterpretaciones
- **Refactorización constante:**
  - Reescribir y organizar el código periodicamente para asegurar una estructura adecuada y preparada para la adaptación de nuevos requisistos futuros.

PRINCIPIOS DE DISEÑO BUENAS PRÁCTICAS

Principios SOLID

Mejoran la mantenibilidad, extensibilidad y claridad del código orientado a objetos.

- S – Single Responsibility: cada clase debe tener una única responsabilidad clara.
- O – Open/Closed: el código debe estar abierto para extensión, cerrado para modificación.
- L – Liskov Substitution: las subclases deben comportarse como sus superclases sin romper el sistema.
- I – Interface Segregation: interfaces pequeñas y específicas; evitar interfaces gigantes.
- D – Dependency Inversion: depender de abstracciones, no de implementaciones concretas.

Principios complementarios

- KISS (Keep It Simple, Stupid): las soluciones simples suelen ser las más robustas. Evita complejidad innecesaria.
- DRY (Don’t Repeat Yourself): no duplicar lógica; la duplicación aumenta errores y mantenimiento.
- YAGNI (You Ain’t Gonna Need It): no implementar funciones “por si acaso”; construir solo lo que aporta valor hoy.

HERRAMIENTAS

Análisis estático - SonarQube

También llamados Linters, estas herramientas realizan un análisis continuo del código mientras el desarrollador trabaja sin necesidad de compilación, enfocándose en calidad general, errores y vulnerabilidades.

CI/CD Pipeline - Qodana

Estas herramientas se integran en la pipeline de CI/CD del repositorio de elección para realizar las tareas de análisis cada vez que se realiza un PR a la rama principal

EJEMPLOS PRACTICOS

SonarQube

EJEMPLOS PRACTICOS

Qodana

LECCIONES APRENDIDAS

- La calidad de código va más allá de evitar errores simples: forma parte central de la arquitectura del software.
- Herramientas como SonarQube y Qodana nos permitieron detectar problemas reales, entender deuda técnica y aplicar refactorizaciones necesarias.
- Estándares como ISO/IEC 25010 ayudan a conectar la calidad interna del código con atributos clave del producto final.
- La automatización en CI/CD mostró que el análisis continuo es fundamental para mantener sistemas sostenibles y seguros.

CONCLUSIONES

- La calidad de código es esencial para crear software mantenible, seguro y confiable.
- Un buen diseño técnico reduce costos futuros y facilita la evolución del producto.
- El uso de análisis estático y pipelines automatizados mejora la detección temprana de errores y fortalece la arquitectura.

la calidad no es un resultado accidental, sino una capacidad que se construye con disciplina, buenas prácticas y automatización.

MUCHAS GRACIAS

ARQUITECTURA DE SOFTWARE

PABLO ENRIQUE QUINTERO, JUAN DIEGO ROMERO, KAMILT BEJARANO DIAZ

REFERENCIAS

- https://stackoverflow.com/questions/2307283/what-does-olog-n-mean-exactly
- https://es.wikipedia.org/wiki/Eficiencia_algor%C3%ADtmica
- https://iso25000.com/index.php/normas-iso-25000/iso-25010
- Franca, Joyce & Soares, Michel. (2015). SOAQM: Quality model for SOA applications based on ISO 25010. ICEIS 2015 - 17th International Conference on Enterprise Information Systems, Proceedings. 2. 10.5220/0005369100600070.
- https://innevo.com/blog/mejores-practicas-desarrollo-software
- https://icariatechnology.com/la-calidad-del-software-y-sus-estandares-mas-importantes/
- https://www.jetbrains.com/help/qodana/github.html