Estrategia de Control de Calidad y Pruebas Unitarias

Para garantizar la calidad del software y reducir una deuda técnica se seguirá una estrategia de implementación de pruebas unitarias y validación del código asegurando así una calidad en el software. Se seguirán los siguientes puntos:

- Definición de Pruebas Unitarias, Las pruebas unitarias son pruebas automatizadas que verifican el correcto funcionamiento de unidades individuales de código (por ejemplo, métodos o clases) de forma aislada. Estas pruebas deben ser rápidas, repetibles y fáciles de mantener.

- Herramientas y Frameworks, Se utilizarán las siguientes:

    - JUnit: Framework de pruebas unitarias para Java.
    - Mockito: Biblioteca para crear objetos simulados (mocks) y aislar las unidades de código bajo prueba.
    - JaCoCo para medir la cobertura de las pruebas.

- Estrategia de Implementación:

    - Identificación de unidades a probar. Los módulos clave son enfocarse en los módulos críticos del sistema, como la lógica de negocio, la gestión de proyectos y la integración con IA. En cuanto a los métodos públicos, probar todos los métodos públicos de las clases, ya que son los puntos de entrada al sistema.
    - Creación de Casos de Prueba.
      En casos normales hay que verificar el comportamiento esperado del código con entradas válidas. Casos Extremos se prueban entradas límite, como valores mínimos, máximos o nulos. En casos de error hay que asegurar que el código maneja correctamente situaciones excepcionales.
    - Uso de Mocks y Stubs.
      - Aislamiento de Dependencias: Utilizar Mockito para simular dependencias externas (por ejemplo, servicios de IA o bases de datos) y probar las unidades de código de forma aislada.
      -Comportamiento Simulado: Definir el comportamiento esperado de los mocks para evitar fallos debido a dependencias externas.
    - Automatización de Pruebas. Integrar las pruebas unitarias en un pipeline de CI (por ejemplo, con GitHub Actions) para ejecutarlas automáticamente en cada cambio de código con integración Continua (CI). Asegurar que las pruebas unitarias se ejecuten en menos de unos segundos para no retrasar el desarrollo.
    - Cobertura de Código. Se debe establecer un objetivo mínimo de cobertura de código (por ejemplo, 80%) y monitorearlo con herramientas como JaCoCo. También hay que cubrir la identificación de brechas, usar informes de cobertura para identificar áreas del código que no están siendo probadas.

Beneficios de las Pruebas Unitarias

1) Reducción de la Deuda Técnica:

Detección temprana de errores: Las pruebas unitarias permiten identificar y corregir errores en etapas tempranas del desarrollo, evitando que se acumulen problemas técnicos.
Código más limpio: Al escribir pruebas, los desarrolladores se ven obligados a escribir código modular y desacoplado, lo que facilita su mantenimiento.

2) Garantía de Calidad:

Validación continua: Las pruebas unitarias aseguran que el código funciona correctamente después de cada cambio, reduciendo el riesgo de introducir regresiones.
Documentación viva: Las pruebas unitarias sirven como documentación del comportamiento esperado del código, lo que facilita su comprensión y mantenimiento.

3) Facilidad de Refactorización:

Seguridad al cambiar código: Con una suite de pruebas unitarias robusta, los desarrolladores pueden refactorizar el código con confianza, sabiendo que las pruebas detectarán cualquier comportamiento inesperado.

Conclusión

La implementación de pruebas unitarias es de vital importancia para la garantización de un software de calidad. Al seguir una estrategia bien definida, utilizando herramientas como JUnit y Mockito, y automatizando las pruebas en un pipeline de CI, se puede asegurar que el código sea robusto, mantenible y libre de errores.