3. **Uso de QAW para la priorización**

- **Identificar el problema o la evaluación de arquitectura:** Definir claramente el propósito de la evaluación de arquitectura, que en este caso es evaluar la arquitectura del Sistema de Gestión de Tareas.

- **Generar preguntas clave:** Plantear preguntas relevantes sobre la arquitectura del sistema, como "¿Cómo se asegura la escalabilidad del sistema a medida que aumenta el número de usuarios?", "¿Cuáles son los principales desafíos de seguridad que enfrenta el sistema?" y "¿Cómo se gestionará la integración entre front-end y back-end?".

¿Qué nivel de prioridad asignaría a la usabilidad en este sistema, considerando que los usuarios deben ser capaces de crear, editar, visualizar y eliminar tareas de manera eficiente? (Escala: Baja, Media, Alta) R:// Media

En una escala del 1 al 10, ¿qué importancia daría a la Seguridad de los Datos, teniendo en cuenta que los usuarios almacenarán información sensible sobre sus tareas en el sistema? (1 = No importante, 10 = Muy importante) R:// 10

¿Qué nivel de énfasis otorgaría a la Disponibilidad del sistema, considerando que los usuarios dependerán de él para la gestión diaria de sus tareas y proyectos? (Escala: Baja, Media, Alta) R:// Alta

¿Qué grado de importancia asignaría a la Eficiencia en la ejecución de operaciones clave, como la creación y eliminación de tareas? (Escala: Baja, Media, Alta) R:// Alta

¿Qué nivel de escalabilidad consideraría necesario para que el sistema pueda crecer y adaptarse a las necesidades de los usuarios a medida que aumenta la cantidad de tareas y proyectos? (Escala: Baja, Media, Alta) R:// Alta

¿Cuán crítica sería la Interoperabilidad del sistema con otras aplicaciones y plataformas para la productividad personal, como calendarios y aplicaciones de correo electrónico? (Escala: Baja, Media, Alta) R:// Media

En una escala del 1 al 10, ¿cuánta importancia daría a la Integridad de Datos, considerando que la información de las tareas debe mantenerse precisa y sin corrupción? (1 = No importante, 10 = Muy importante) R:// 10

¿Cuánto énfasis pondría en la Retroalimentación al Usuario, como mensajes de confirmación y notificaciones, para mantener a los usuarios informados sobre el estado de sus tareas? (Escala: Baja, Media, Alta) R:// Alta

¿Qué nivel de Facilidad de Mantenimiento consideraría necesario para asegurar que el sistema pueda actualizarse y mantenerse de manera eficiente en el tiempo? (Escala: Baja, Media, Alta) R:// Media

¿Qué tan importante sería la Personalización de la Interfaz de Usuario, permitiendo a los usuarios adaptar la apariencia y el flujo de trabajo a sus preferencias individuales? (Escala: Baja, Media, Alta) R:// Media

¿Cuánto valor asignaría a la Escalabilidad del sistema para manejar un crecimiento potencial en la cantidad de usuarios que lo utilizan simultáneamente? (Escala: Baja, Media, Alta) R:// Alta

En una escala del 1 al 10, ¿cuánta importancia daría a la Colaboración entre usuarios, cómo compartir tareas o proyectos? (1 = No importante, 10 = Muy importante) R:// 5

evaluación de atributos de calidad.

- **Garantizar el uso de complejidad no elevada:** Este atributo se refiere a la importancia de mantener el código simple y fácil de entender. Ejemplo: Evitar la anidación excesiva de bucles o condicionales y dividir tareas complejas en funciones más pequeñas y manejables poniendo como límite una complejidad de 15.

- **Usar Camelcase:** Utilizar convenciones de nomenclatura como Camelcase para nombres de variables y funciones. Ejemplo: `miVariableLocal` o `calcularTotalFactura`.

- **Aplicar el principio DRY (Don't Repeat Yourself):** Evitar la duplicación de código escribiendo funciones o clases reutilizables para tareas comunes. Ejemplo: Si varias partes del código realizan la misma operación, crear una función que la encapsule y reutilizarla.

- **Mantener cohesión y bajo acoplamiento:** Los módulos o componentes del código deben estar relacionados de manera lógica (cohesión) y no deben depender en exceso unos de otros (bajo acoplamiento). Ejemplo: Evitar que una clase tenga muchas responsabilidades diferentes y que las clases tengan acoplamientos débiles entre sí.

- **Gestionar excepciones y errores de manera adecuada:** Capturar y manejar errores de forma apropiada en lugar de simplemente ignorarlos. Ejemplo: Utilizar bloques `try...catch` para capturar excepciones y proporcionar información útil sobre el error.

- **Optimizar el rendimiento:** Identificar y resolver cuellos de botella de rendimiento en el código para garantizar que la aplicación sea rápida y eficiente. Ejemplo: Utilizar algoritmos más eficientes o mejorar la gestión de recursos(Procesador, almacenamiento en general).

- **Seguir estándares de codificación:** Adherirse a convenciones y estándares de codificación específicos del lenguaje de programación. Ejemplo: Para .NET, seguir las.NET Documentación.

- **Realizar análisis de seguridad estática y dinámica(Se evaluará para desarollo futuro):** Identificar y abordar posibles vulnerabilidades de seguridad en el código mediante herramientas de análisis estático y pruebas de penetración. Ejemplo: Utilizar herramientas como SonarQube o Nessus para escanear el código en busca de problemas de seguridad.

- **Versionamiento y control de cambios:** Se utilizan el sistemas de control de versiones como Git para rastrear y gestionar las modificaciones en el código. Ejemplo: Utilizar ramas para desarrollar nuevas características o solucionar problemas sin afectar la rama principal.

Ejemplo de una entrega basada en el proceso QAW para el sistema de gestión de tareas:

Entrega de Evaluación de Atributos de Calidad para el Sistema de Gestión de Tareas
Fecha de Entrega: 08 de Octubre de 2023
Entregable: Informe de Evaluación de Atributos de Calidad

Resumen Ejecutivo:

La presente entrega consiste en un Informe de Evaluación de Atributos de Calidad para el sistema de gestión de tareas. El objetivo de esta evaluación es identificar, priorizar y comprender los atributos de calidad clave que son relevantes y definir métricas específicas para su medición y evaluación. A continuación, se resumen los hallazgos clave de esta evaluación.

Hallazgos Clave:

Atributos de Calidad Identificados: Se identificaron los siguientes atributos de calidad como relevantes:

- **Usabilidad**
- **Seguridad de Datos**
- **Eficiencia**
- **Disponibilidad**
- **Escalabilidad**
- Actualizaciones frecuentes
- Estabilidad
- Eficiencia
- Rendimiento

Definición de Escenarios de Calidad: Se describieron varios escenarios específicos para evaluar los atributos de calidad. Por ejemplo:

- Escenario de Usabilidad: Un usuario nuevo intenta crear una tarea en menos de 2 minutos.
- Escenario de Seguridad de Datos: Un intento de acceso no autorizado a la base de datos de tareas se detecta y bloquea dentro de 30 segundos.
- Escenario de Eficiencia: Durante una carga de trabajo alta, el sistema debe mantener un tiempo de respuesta promedio de creación de tareas por debajo de 5 segundos.
- Priorización de Atributos de Calidad: Los atributos de calidad se priorizaron de la siguiente manera:
    * Usabilidad: Alta prioridad
    * Seguridad de Datos: Alta prioridad
    * Eficiencia: Media prioridad
    * Disponibilidad: Alta prioridad
    * Escalabilidad: Alta prioridad
    * Actualizaciones frecuentes: Alta prioridad
    * Estabilidad: Media prioridad
    * Eficiencia: Media prioridad
    * Rendimiento: Media prioridad

- Definición de Métricas: Se establecieron métricas específicas para cada atributo de calidad. Por ejemplo, para Usabilidad, se medirá el tiempo promedio de creación de tareas. Para Seguridad de Datos, se medirá la tasa de detección de intentos de acceso no autorizado.

Conclusiones:

Este informe servirá como guía fundamental para el diseño, desarrollo y pruebas, asegurando que se alcancen y mantengan los niveles de calidad deseados por los usuarios y stakeholders.