ACTIVIDAD 1: Análisis y Selección de Principios de Diseño

Introducción

El diseño de software eficiente exige la aplicación de principios sólidos que permitan construir soluciones escalables, mantenibles y reutilizables. A lo largo de este documento, se presentan los principios S.O.L.I.D., DRY, KISS y YAGNI, explicando de forma clara cómo se integrarán en la arquitectura de una plataforma inteligente de gestión de proyectos respaldada por inteligencia artificial.

Aplicación de los Principios de Diseño

S.O.L.I.D. abarca cinco principios fundamentales que promueven una estructura de software coherente y flexible:

En primer lugar, el Principio de Responsabilidad Única (SRP) establece que cada componente o clase debe asumir una única responsabilidad. En la práctica, esto implica que los módulos que predicen retrasos en los proyectos y los que administran tareas se mantengan separados, de modo que cada uno se concentre en su función específica.

El Principio Abierto/Cerrado (OCP) sugiere que el software debe permitir extensiones sin requerir cambios en el código existente. Para lograrlo, se usarán interfaces y clases base que faciliten la incorporación de nuevos algoritmos de IA, preservando la estructura actual.

El Principio de Sustitución de Liskov (LSP) garantiza que las subclases puedan reemplazar a sus clases base sin alterar la funcionalidad. Por ejemplo, si se contemplan distintos tipos de proyectos (ágiles, tradicionales), cada derivación respetará los contratos definidos por la clase principal.

El Principio de Segregación de Interfaces (ISP) defiende la creación de interfaces concretas y específicas, evitando la sobrecarga de métodos inútiles. Si una interfaz gestiona usuarios, no debería incluir funciones de reportes, ya que estos aspectos pertenecen a otro ámbito.

Por último, el Principio de Inversión de Dependencias (DIP) propone que los componentes de mayor nivel no dependan de implementaciones concretas, sino de abstracciones. A través de la inyección de dependencias, se reduce el acoplamiento entre módulos, facilitando posibles cambios internos.

En cuanto a DRY (Don’t Repeat Yourself), se evitará la duplicación de código mediante la creación de funciones y servicios compartidos. Si el sistema necesita un algoritmo de recomendación de tareas, este se centralizará para su reutilización y no se replicará en varios lugares.

El principio KISS (Keep It Simple, Stupid) enfatiza la simplicidad y la claridad. Se priorizarán diseños que resulten fáciles de entender y extender, minimizando la complejidad innecesaria en la interfaz y en los modelos de datos.

Por su parte, YAGNI (You Ain’t Gonna Need It) nos anima a desarrollar únicamente lo imprescindible en las primeras etapas del producto. Así, no se desplegarán características avanzadas hasta que exista una clara necesidad, favoreciendo la eficiencia en el desarrollo.

Conclusión

La adopción de estos principios permitirá forjar una base sólida para la plataforma, beneficiándose de un código más ordenado, fácil de mantener y con menos deuda técnica. El enfoque modular, las interfaces diseñadas con ISP, la separación clara de responsabilidades bajo SRP y la flexibilidad de extensiones contempladas por OCP garantizarán que el proyecto pueda crecer sin complicaciones excesivas. Con la aplicación de DRY, KISS y YAGNI, se busca un desarrollo centrado en resolver necesidades reales, sin caer en sobreesfuerzos o duplicaciones innecesarias. En conjunto, estos pilares de diseño contribuyen a crear una solución sostenible y escalable para un producto de gestión de proyectos impulsado por IA.