GUÍA PRÁCTICA DE DISEÑO DE SOFTWARE

Para estudiantes de 6to nivel de la Carrera de Ingeniería de Software

Unidad 3: Conceptos de diseño, principios, estándares y lineamientos de calidad

Fecha: 01/13/2026

Documento base: U3_Conceptos_Diseno_Principios_Estandares_Lineamientos

1. Propósito y alcance de la guía

Esta guia traduce los conceptos clave de diseño de software sus principios y estandares, en una ruta de trabajo aplicable para proyectos de 6to nivel cubriendo arquitectura y diseño detallado. Se incluye la lista de verificacion ISO/IEC 25010 para asegurar la calidad del producto de forma tecnica y profesional.

4. Principios prácticos de diseño

Principio

¿Puedo entender/cambiar una parte sin romper las demás?

Si, Como estamos usando arquitectura en MVC, todo está dividido por módulos independientes. Esto implica, por ejemplo, que podemos cambiar la base de datos por una diferente sin necesidad de modificar la GUI, ya que esto se realizaría a través de interfaces definidas.

¿Cada módulo hace una cosa clara? ¿Depende lo mínimo de otros?

Si, como el diseño sigue el patron MVC, se asegura de que la lógica de negocio solo sea manejada por el controlador mientras el modelo como las clases “Producto o Pedido) se encargan de los datos únicamente. Esto deriva en un bajo acoplamiento usando servicios especializados, así se encapsula el calculo de mezclas personalizadas.

¿Oculto detalles internos y expongo solo contratos?

Si, debido a la metodología usada, el sistema oculta atributos privados y solo permite interactuar con ellos a través de métodos públicos previamente definidos en el diagrama de clases. Esto con el fin de asegurar la integridad de los datos.

¿UI, lógica de negocio y persistencia están separadas?

Si, siguiendo las indicaciones del IEEE 1016, la capa de vista no conoce las reglas de negocio como el calculo de stock, por otro lado el modelo no conoce como se gestionan los colores en pantalla o el manejo de la interfaz ya que el controlador actúa como mediador.

¿Reutilizo componentes y patrones conocidos cuando conviene?

Si, como implementamos el patron “observer” para mejorar la seguridad y tener seguimiento. En lugar de escribir una función de “logs” en cada clase, el sistema notifica automáticamente a los observadores cuando ocurren cambios críticos en productos o pedidos.

¿Puedo probar funciones y componentes de forma aislada (unit tests)?

Si, gracias al enfoque modular, las funciones importantes pueden probarse de forma independiente y no se necesita cargar toda la base de datos o la interfaz, lo que garantiza es que los resultados mostrados al cliente sean precisos.

¿Valido entradas, gestiono permisos y registro auditoría?

Si, el diseño planteado tiene en cuanta validaciones tanto en el controlador como en las vista antes de que se almacenen los datos, adicional, se refuerza la seguridad gracias al registro de autoría automático controlado por el patron Observer, cada movimiento en el inventario de Kairos Mix deja una huella digital en logs.

ANEXO A. Checklist ISO/IEC 25010 (calidad del producto)

Característica:

Adecuación Funcional:

¿Están implementados todos los casos de uso acordados?

Cumple

Se cubren las historias de usuario mas las pruebas de aceptacion

¿Los resultados son correctos para entradas válidas y límites?

Incompleto.

Faltan pruebas para validar si los ingresos de productos o mezclas fueron adecuados.

¿Las funciones ayudan a lograr objetivos del usuario sin pasos extra?

Cumple.

El flujo CRUD es directo para crear, editar o buscar pedidos sin dar vueltas.

Eficiencia de desempeño:

¿Responde dentro del tiempo objetivo bajo carga esperada?

A medias.

Falta medicion con JMeter para definir los SLOs de respuesta.

¿Uso de CPU/RAM/BD es razonable y monitoreable?

Incompleto.

No hay metricas todavia, toca meter logging y mas monitoreo.

¿Soporta el volumen esperado (N registros, concurrencia)?

A medias.

Se deben probar las paginas para manejar volumenes altos de datos en el inventario.

Usabilidad

¿El usuario entiende si el sistema le sirve?

Cumple.

La interfaz tiene etiquetas claras como Producto o Mezclas para que no se pierdan.

¿Un usuario nuevo aprende a usarlo rápidamente?

Listo.

El flujo es el estandar, aunque la ayuda contextual sea muy basica.

¿Se puede operar con facilidad (pocos clics, navegación clara)?

Incompleto.

Faltan atajos de teclado o una busqueda mas avanzada.

¿Previene errores (validación, confirmaciones)?

A medias.

La validacion es basica, pero falta pedir confirmacion al eliminar registros.

¿Diseño consistente y legible?

Cumple.

Los componentes son uniformes y la tipografia se lee bien.

Fiabilidad

¿Falla poco en condiciones normales?

Incompleto.

No hay datos suficientes todavia, faltan pruebas automatizadas.

¿Maneja fallos (BD caída, timeouts) sin colapsar?

A medias.

El manejo de errores HTTP es parcial, falta meter un circuit breaker.

¿Se recupera (backup/restore) ante fallos?

Pendiente.

No existe ningun plan de backup o migracion de base de datos.

Seguridad:

¿Controla acceso a datos (authz/authn)?

Incompleto.

El login es basico, falta implementar control de roles (RBAC).

¿Registra acciones para evidenciar autoría?

Pendiente.

No hay logs de auditoria cuando se crean las mezclas personalizadas.

¿Se puede rastrear quién hizo qué?

Falta.

Toca agregar auditoria para los procesos de CREATE, UPDATE y DELETE.

¿Identidades verificadas de forma robusta?

Incompleto.

No se han definido politicas de contraseña ni MFA.

Mantenibilidad:

¿Módulos separados (capas, paquetes) y dependencias controladas?

Cumple.

La arquitectura por capas MVC separa bien la vista del controlador.

¿Componentes reutilizables (validadores, DTOs, utilidades)?

A medias.

Se reutiliza poco, falta crear una libreria comun para utilidades.

¿Es fácil diagnosticar defectos (logs, trazas, claridad)?

Incompleto.

Los logs son minimos y no hay correlacion en las solicitudes.

¿Hay pruebas unitarias/integración y CI?

Pendiente.

Todavia no se han automatizado las pruebas en el flujo.

¿Puede ejecutarse en distintos entornos sin cambios mayores?

A medias.

Se usan variables de entorno, pero falta configurar un contenedor.

Referencias (APA 7)

1. ISO. (2011). ISO/IEC 25010:2011 Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models.
2. ISO. (2006). ISO/IEC 16085:2006 Systems and software engineering—Life cycle processes—Risk management.
3. ISO/IEC/IEEE. (2011). ISO/IEC/IEEE 42010:2011 Systems and software engineering—Architecture description.
4. Kendall, K. E., & Kendall, J. E. (Ediciones recientes). Systems Analysis and Design. Pearson.