|  |
| --- |
| **CALIFICACION**  **OBTENIDA**  **EVALUACION PARCIAL ESTRUCTURADA 1 (EPE1)**  **Segundo Semestre** |

**Información General**

|  |  |  |  |
| --- | --- | --- | --- |
| Carrera | Ingeniería en  Informática. | N° Identificador de carrera | 800 |
| Asignatura | TALLER DE INTEGRACIÓN | Código asignatura |  |
| Docente | Hugo Gutiérrez | Sección | 01 |

**Información del estudiante**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Marco Antonio González Pérez | RUN | 19.782.697-5 |

**Información del proceso evaluativo**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Puntaje total |  | | Puntaje obtenido |  |
| Puntaje de aprobación |  | | Tiempo máximo |  |
| Exigencia de escala | 60% | | Ponderación |  |
| Fecha de rendición |  | | Fecha |  |
| Resultado de aprendizaje | Analiza conceptos de la ingeniería de software, su génesis y evolución, para determinar el contexto donde debe desenvolverse, reconociendo las diferencias del entorno. | | | |
| Agentes evaluadores |  | % de ponderación | |  |
| Metodología de aprendizaje aplicada |  | Tipo de instrumento de evaluación | | rúbrica |
|  | | | | |

## 

INSTRUCCIONES GENERALES

### **Evaluación: Documentos Generados en el Ciclo de Vida del Software**

#### **Pregunta 1: Etapa de Análisis de Requisitos**

La etapa de análisis de requisitos es crítica en el ciclo de vida del software, ya que define las expectativas y necesidades del cliente.

1. **Documentos entregables en esta etapa**:
   * ¿Qué es un **Documento de Requerimientos del Software (SRS)** y qué información debe contener?

El SRS es el contrato técnico entre desarrolladores y stakeholders. Debe contener:

* + - Sección 1: Introducción y Propósito

- Objetivo del sistema

- Audiencia objetivo

- Referencias y glosario de términos técnicos

* + - Sección 2: Descripción General

- Perspectiva del producto en el ecosistema existente

- Funciones principales (no más de 10 puntos clave)

- Características de usuarios (personas, roles, permisos)

- Restricciones operativas (tecnológicas, regulatorias, presupuestarias)

* + - Sección 3: Requisitos Específicos

- **Funcionales**: "El sistema DEBE permitir..." con criterios de aceptación medibles

- **No funcionales**: Rendimiento (ej: "Respuesta < 2 segundos"), Seguridad (ej: "Cifrado AES-256"), Usabilidad (ej: "3 clics máximo para cualquier función")

- **Requisitos de interfaz**: APIs, integraciones, formatos de datos

* + **Carta Gantt**: Explica cómo se utiliza la Carta Gantt en esta fase y cómo se planifica el proyecto en términos de tiempo.

La Carta Gantt aquí no es solo planificación temporal, sino una herramienta de **gestión de dependencias de requisitos**:

* + - **Fase 1**: Levantamiento inicial (1-2 semanas)
      * Entrevistas con stakeholders clave
      * Análisis de sistemas legacy
      * Benchmarking competitivo
    - **Fase 2**: Refinamiento y validación (2-3 semanas)
      * Workshops colaborativos
      * Prototipado de interfaces críticas
      * Validación técnica de viabilidad
    - **Fase 3**: Documentación y aprobación (1 semana)
      * Elaboración SRS final
      * Revisión legal/compliance
      * Sign-off formal
  + ¿Cuál es la importancia de los **alcances del proyecto** en esta etapa? ¿Cómo influyen en el resto del ciclo de vida del software?

El alcance actúa como **firewall de scope creep (cuando el alcance de un proyecto crece gradualmente más allá de lo que se planificó)**. Su importancia radica en:

* + - **Define límites claros**: "Lo que SÍ incluye vs lo que NO incluye"
    - **Establece criterios de éxito**: Métricas cuantificables (ej: "Procesar 10,000 transacciones/hora")
    - **Impacto en ciclo de vida**: Un alcance mal definido genera:
      * Estimaciones erróneas en diseño
      * Retrabajos en implementación
      * Pruebas incompletas
      * Mantenimiento costoso

1. **Actividad práctica**:
   * Describe cómo organizarías un **workshop de toma de requisitos** con el cliente y qué documentos serían entregados al final de esta fase.

**Estructura del Workshop (Metodología Design Thinking + Agile)**

**Pre-workshop** (1 semana antes):

* + - Envío de cuestionario pre-estructura
    - Análisis preliminar del negocio
    - Preparación de plantillas de trabajo

**Día del Workshop** (8 horas):

* + - **Hora 1-2**: Sesión de alineación (visión, objetivos, restricciones)
    - **Hora 3-4**: Mapeo de procesos AS-IS vs TO-BE
    - **Hora 5-6**: Brainstorming de funcionalidades (técnica MoSCoW)
    - **Hora 7-8**: Priorización y definición de MVP

**Documentos Entregables post-workshop:**

* + 1. **Acta de Workshop**: Decisiones tomadas, participantes, compromisos
    2. **Backlog Priorizado**: User stories con puntos de historia
    3. **Matriz de Trazabilidad**: Requisito → Justificación de negocio
    4. **Roadmap Visual**: Releases planificados con features principales

#### **Pregunta 2: Etapa de Diseño**

La etapa de diseño es fundamental para la arquitectura y la estructura del software.

1. **Documentos entregables en esta etapa**:
   * **Especificaciones de Diseño**: Explica qué debe incluir un informe de diseño y cómo se documenta el diseño arquitectónico y de componentes.

Un informe de diseño efectivo debe seguir la **arquitectura en capas**:

**Diseño Arquitectónico:**

* + - **Vista Lógica**: Componentes y sus responsabilidades
    - **Vista de Desarrollo**: Organización del código (módulos, paquetes)
    - **Vista de Proceso**: Flujos de datos y control
    - **Vista Física**: Infraestructura y deployment
    - **Vista de Escenarios**: Casos de uso arquitectónicamente significativos

**Diseño de Componentes:**

* + - Interfaces públicas de cada módulo
    - Patrones de diseño aplicados (Strategy, Observer, Factory, etc.)
    - Manejo de errores y excepciones
    - Estrategias de logging y monitoreo
  + **Diagrama UML**: Menciona los diferentes tipos de diagramas UML (por ejemplo, diagrama de clases, de secuencia, de casos de uso) y explica su propósito y cuándo se utilizan.

**Diagrama de Casos de Uso**:

* + - **Cuándo**: Validación inicial con stakeholders
    - **Propósito**: Comunicar funcionalidades en lenguaje de negocio

**Diagrama de Clases**:

* + - **Cuándo**: Diseño detallado de la estructura
    - **Propósito**: Blueprint del código orientado a objetos

**Diagrama de Secuencia**:

* + - **Cuándo**: Procesos complejos con múltiples actores
    - **Propósito**: Modelar interacciones temporales críticas

**Diagrama de Componentes**:

* + - **Cuándo**: Arquitecturas modulares/microservicios
    - **Propósito**: Visualizar dependencias y interfaces

**Diagrama de Despliegue**:

* + - **Cuándo**: Sistemas distribuidos
    - **Propósito**: Mapear componentes a infraestructura
  + **Prototipos**: Explica el papel de los prototipos en el diseño. ¿Cómo puede un prototipo ayudar a validar el diseño antes de pasar a la implementación?

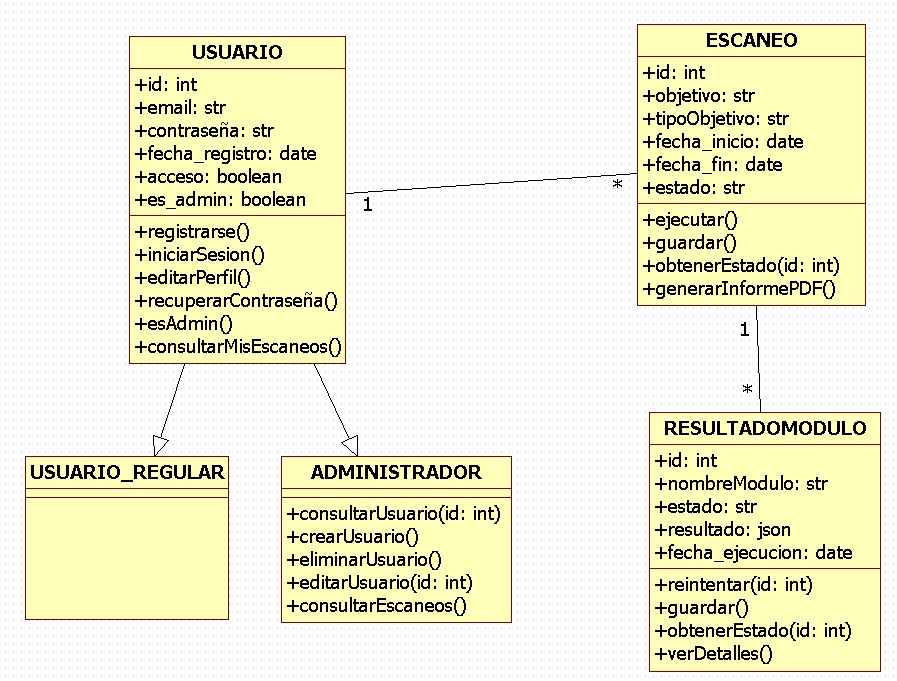
**Tipos de Prototipos por Propósito:**

* + - **Prototipos de Papel**: Validación rápida de flujos UX
    - **Prototipos Interactivos**: Pruebas de usabilidad con usuarios reales
    - **Prototipos Técnicos**: Proof of concept de algoritmos complejos
    - **Prototipos de Arquitectura**: Validación de patrones y rendimiento

**Valor en Validación:**

* + - **Feedback temprano**: Correcciones 10x más baratas que en implementación
    - **Comunicación efectiva**: Lenguaje visual universal
    - **Reducción de riesgo**: Identificación temprana de problemas técnicos

1. **Actividad práctica**:
   * Crea un **diagrama de clases** para una aplicación de gestión de tareas e incluye las clases, atributos y relaciones básicas entre ellas.

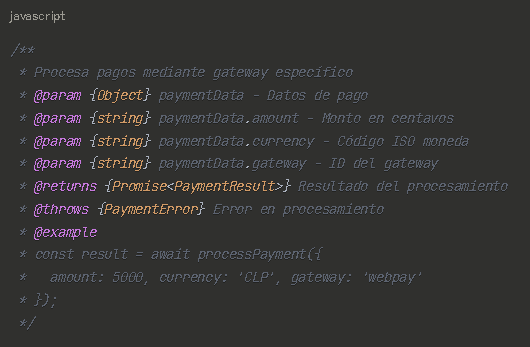


#### **Pregunta 3: Etapa de Implementación o Desarrollo**

La implementación es la fase en la que se lleva el diseño a la realidad mediante código y desarrollo.

1. **Documentos entregables en esta etapa**:
   * **Código Fuente**: Explica qué documentación debe acompañar al código fuente, como comentarios y guías de estilo de código.

Estándares de Comentarios:



**Guías de Estilo Específicas:**

* + - **Naming conventions**: camelCase para variables, PascalCase para clases
    - **Estructura de archivos**: Separación por features, no por tipo
    - **Manejo de errores**: Estrategia fail-fast con logging estructurado
    - **Performance**: Patrones de lazy loading y memorización
  + **Plan de Pruebas Unitarias**: ¿Cómo se organiza el plan de pruebas unitarias y cuál es su propósito?

El plan de pruebas unitarias se organiza siguiendo la estructura del código fuente, con una cobertura mínima del 80% en funciones críticas. Su propósito es crear una red de seguridad que permita refactorizar con confianza y detectar regresiones automáticamente.

La organización práctica incluye:

* + - Tests de happy path para escenarios exitosos
    - Edge cases para valores límite y condiciones especiales
    - Error handling para validar manejo de excepciones
    - Tests de integración entre módulos relacionados

Cada test valida una sola responsabilidad y sirve como documentación viva del comportamiento esperado de cada función.

* + **Documentación Técnica**: Describe qué documentos técnicos deben entregarse durante la fase de implementación para facilitar el mantenimiento y la escalabilidad.

Durante la implementación se generan documentos técnicos esenciales para mantenimiento y escalabilidad. El README técnico permite configurar el entorno local rápidamente, mientras la documentación de APIs especifica endpoints, formatos y códigos de error con ejemplos.

Los Architecture Decision Records (ADRs) documentan decisiones técnicas importantes, explicando contexto y alternativas consideradas. Esto previene que futuros desarrolladores repitan análisis o contradigan la arquitectura establecida.

1. **Actividad práctica**:
   * Describe cómo organizarías una revisión de código en equipo y qué tipo de documentos estarías generando durante esta fase.

La revisión se estructura como proceso colaborativo que inicia con auto-revisión del autor usando checklist técnico. El pull request incluye descripción clara, justificación y evidencia de testing.

El equipo evalúa cuatro aspectos: corrección técnica, adherencia a estándares, cumplimiento arquitectónico y mantenibilidad futura.

**Documentos generados durante esta fase:**

* + - Templates estandarizados de pull request
    - Logs de comentarios y decisiones técnicas
    - Métricas automáticas de calidad de código
    - Notas de transferencia de conocimiento para lógica compleja

La clave está en mantener revisiones constructivas donde cada comentario agregue valor al código y al conocimiento del equipo, transformando el code review en herramienta de mejora continua.

#### **Pregunta 4: Etapa de Pruebas**

La etapa de pruebas asegura que el software cumpla con los requisitos y funcione correctamente.

1. **Documentos entregables en esta etapa**:
   * **Plan de Pruebas**: Explica qué debe contener un plan de pruebas y cómo se determina qué pruebas realizar.

El plan de pruebas define la estrategia completa de validación del software. Debe contener el alcance de pruebas especificando qué funcionalidades se probarán y cuáles quedan fuera, los criterios de entrada y salida que determinan cuándo iniciar y finalizar las pruebas, y la estimación de recursos necesarios incluyendo tiempo y personal.

Para determinar qué pruebas realizar se aplica análisis de riesgo, priorizando funcionalidades críticas del negocio, puntos de integración complejos y áreas con mayor probabilidad de fallos. También se consideran los requisitos no funcionales como rendimiento y seguridad.

* + **Informe de Resultados de Pruebas**: ¿Qué información debe incluir un informe de pruebas? ¿Cómo se documentan los defectos encontrados durante las pruebas?

El informe documenta el estado final de calidad del software mediante un resumen ejecutivo con porcentajes de casos exitosos y fallidos, análisis detallado por módulo y severidad de defectos encontrados.

Los defectos se documentan con identificador único, descripción clara del problema, pasos para reproducir, severidad (crítica, alta, media, baja) y prioridad de corrección. Cada defecto incluye evidencia como capturas de pantalla o logs, facilitando su resolución por el equipo de desarrollo.

* + **Reporte de Cobertura de Pruebas**: ¿Por qué es importante medir la cobertura de pruebas y qué métricas se deben considerar?

La cobertura mide qué porcentaje del código ha sido ejecutado durante las pruebas, siendo importante porque identifica áreas no validadas que podrían contener defectos ocultos.

Las métricas clave incluyen:

* + - Cobertura de líneas (mínimo 80% en funciones críticas)
    - Cobertura de ramas (validación de todas las decisiones condicionales)
    - Cobertura funcional (porcentaje de requisitos probados)
    - Cobertura de casos de uso (escenarios de negocio validados)

1. **Actividad práctica**:
   * Crea un **plan de pruebas básico** para una función de inicio de sesión de usuario, especificando los casos de prueba, entradas, resultados esperados y tipo de prueba.

**CP001 - Login Exitoso:**

* + - Entrada: Usuario válido ([admin@test.com](mailto:admin@test.com)), contraseña correcta (Test123!)
    - Resultado esperado: Redirección al dashboard, token de sesión generado
    - Tipo: Funcional positiva

**CP002 - Credenciales Incorrectas:**

* + - Entrada: Usuario válido, contraseña incorrecta (wrongpass)
    - Resultado esperado: Mensaje "Credenciales inválidas", permanecer en login
    - Tipo: Funcional negativa

**CP003 - Usuario Inexistente:**

* + - Entrada: Usuario no registrado ([fake@test.com](mailto:fake@test.com)), cualquier contraseña
    - Resultado esperado: Mensaje genérico de error, sin revelar si usuario existe
    - Tipo: Seguridad

**CP004 - Campos Vacíos:**

* + - Entrada: Email vacío o contraseña vacía
    - Resultado esperado: Validación client-side, botón login deshabilitado
    - Tipo: Validación de entrada

**CP005 - Bloqueo por Intentos:**

* + - Entrada: 5 intentos fallidos consecutivos
    - Resultado esperado: Cuenta bloqueada temporalmente (15 minutos)
    - Tipo: Seguridad

#### **Pregunta 5: Etapa de Despliegue e Implementación**

La etapa de despliegue es cuando el software se entrega al cliente o se pone en producción.

1. **Documentos entregables en esta etapa**:
   * **Plan de Despliegue**: Explica qué debe incluir un plan de despliegue y cómo se realiza la instalación en el entorno de producción.

El plan especifica la estrategia completa de puesta en producción, incluyendo cronograma detallado con ventanas de mantenimiento, procedimientos de rollback en caso de fallos y matriz de responsabilidades del equipo.

La instalación en producción sigue una checklist que incluye backup completo del sistema actual, validación de requisitos de infraestructura, ejecución de scripts de migración de datos y verificación de conectividad con sistemas externos. Se define además el plan de comunicación para usuarios finales sobre el cambio.

* + **Guía de Usuario**: ¿Qué debe incluir una guía de usuario para que sea útil para el cliente final?

Una guía efectiva debe ser práctica y orientada a tareas reales. Incluye introducción breve al sistema, flujos de trabajo paso a paso con capturas de pantalla actuales, sección de preguntas frecuentes basadas en consultas reales de usuarios y glosario de términos técnicos en lenguaje simple.

La clave está en organizar el contenido por roles de usuario y tareas frecuentes, no por funcionalidades técnicas, facilitando que cada usuario encuentre rápidamente la información relevante para su trabajo diario.

* + **Documentación de Soporte y Mantenimiento**: ¿Qué documentos son necesarios para el soporte técnico post-despliegue y cómo pueden ayudar a la solución de problemas?

Esta documentación garantiza continuidad operacional post-despliegue. Incluye procedimientos de troubleshooting con síntomas, causas probables y soluciones, información de arquitectura técnica para diagnosticar problemas complejos y contactos escalados para diferentes tipos de incidentes.

Los documentos esenciales son matriz de escalamiento con tiempos de respuesta por severidad, procedimientos de backup y recuperación, y playbooks para incidentes comunes que permitan resolución rápida sin necesidad de desarrolladores.

1. **Actividad práctica**:
   * Redacta un **informe de lanzamiento** que describa el proceso de despliegue de una nueva funcionalidad en el sistema.

**Informe de Lanzamiento - Nueva Funcionalidad de Reportes Automáticos**

El despliegue se ejecutó exitosamente el 15 de septiembre entre las 02:00 y 04:30 hrs, cumpliendo la ventana de mantenimiento programada. Se implementó la nueva funcionalidad de generación automática de reportes mensuales que permite a los usuarios configurar reportes personalizados con envío automático por email.

Actividades realizadas: Backup completo de base de datos, migración de esquema con nuevas tablas de configuración, despliegue de código en servidores de aplicación y configuración de jobs automatizados en el scheduler del sistema.

Validaciones post-despliegue: Verificación de funcionalidad core existente sin regresiones, pruebas de la nueva característica con datos de prueba y confirmación de envío automático de reportes configurados. No se registraron incidentes durante el proceso.

Impacto en usuarios: 500 usuarios activos pueden ahora configurar hasta 5 reportes automáticos cada uno. Se envió comunicación previa y se habilitó soporte extendido durante las primeras 48 horas post-lanzamiento.

#### **Pregunta 6: Etapa de Mantenimiento y Evolución del Software**

El mantenimiento y evolución del software es crucial para mantener el sistema en funcionamiento y mejorar sus características.

1. **Documentos entregables en esta etapa**:
   * **Informe de Mantenimiento Correctivo**: Explica cómo se documentan los errores y las soluciones que se implementan durante el mantenimiento correctivo.

Los errores se documentan con formato estandardizado que incluye identificación del problema (ticket ID, fecha, severidad), análisis de causa raíz explicando por qué ocurrió el error y descripción técnica de la solución implementada con código modificado.

Cada informe incluye impacto en usuarios afectados, tiempo de resolución y medidas preventivas para evitar recurrencia. Esta documentación sirve como base de conocimiento para problemas futuros similares y permite identificar patrones de errores recurrentes.

* + **Informe de Mantenimiento Evolutivo**: ¿Qué documentos se generan cuando se agregan nuevas funcionalidades al software?

Cuando se agregan nuevas funcionalidades se generan documentos que incluyen justificación de negocio para el cambio, especificación técnica de la implementación y análisis de impacto en funcionalidades existentes.

Los documentos clave son especificación de requisitos para la nueva funcionalidad, documentación de cambios en APIs o interfaces existentes, plan de migración si se requieren cambios en datos y actualización de documentación de usuario para las nuevas características.

* + **Registros de Actualizaciones y Versiones**: ¿Por qué es importante tener registros de las versiones del software, y qué información debe contener?

El control de versiones es crítico para trazabilidad y rollback. Cada versión debe documentar changelog detallado con nuevas características, errores corregidos y cambios que afecten compatibilidad hacia atrás.

La información esencial incluye número de versión siguiendo semantic versioning (mayor.menor.patch), fecha de lanzamiento, lista de cambios categorizados y dependencias o requisitos actualizados. Esto permite a desarrolladores y usuarios entender exactamente qué cambió entre versiones.

1. **Actividad práctica**:
   * Crea un **registro de actualización** para una versión de software que incluye corrección de errores y nuevas funcionalidades.

**Registro de Actualización - Versión 2.3.1**

**Fecha de Lanzamiento:** 20 de septiembre de 2025 **Tipo de Actualización:** Minor release con correcciones críticas

**Nuevas Funcionalidades:**

* + - Implementación de autenticación de dos factores (2FA) opcional para usuarios
    - Dashboard personalizable con widgets configurables por rol
    - Exportación de datos en formato Excel con filtros avanzados

**Correcciones de Errores:**

* + - Solucionado problema de timeout en reportes con más de 10,000 registros
    - Corregida pérdida de sesión al usar navegador en modo privado
    - Reparado cálculo incorrecto de totales en módulo de facturación

**Mejoras de Rendimiento:**

* + - Optimización de consultas en módulo de búsqueda (50% más rápido)
    - Implementación de caché en dashboard principal
    - Compresión de imágenes automática en upload de documentos

**Notas de Migración:** Esta versión requiere ejecución de script de migración de base de datos. Tiempo estimado de actualización: 30 minutos. Compatible con versiones 2.2.x, no compatible con versiones anteriores a 2.0.0.