UAI

INGENIERÍA DE SOFTWARE

2do Parcial

Teórico y Práctico

Alumno: Fecha: Tema 1

Comisión:

Localización: Práctica: Teoría: Nota:

Tema	Puntos Evaluados	Puntos Obtenidos	%	OBS
Arquitectura en capas	5			
Patrones de diseño	5			
TOTAL				

Práctica		%	
Ejercicio			

Notas: Las preguntas en las que se seleccionen opciones se deberá optar solo por una de las posibilidades.
Las preguntas que solicitan justificación serán consideradas válidas si poseen la misma correctamente.
Las preguntas de múltiples posibilidades y verdadero / falso restan 0.50 puntos en caso de estar mal contestadas. En las preguntas verdadero / falso se debe tachar la opción incorrecta. El ponderador numérico de cada pregunta está asignado entre paréntesis al finalizar la misma.
Temas para evaluar: Métricas, Gestión del riesgo y patrones de diseño. Unidades 3, 4 y 5.
Requisitos para aprobar: Para que el parcial esté aprobado el alumno deberá tener correcto: 60% de cada unidad evaluada en la teoría y resuelto el ejercicio práctico.
Recomendaciones:
a) Lea todo el parcial antes de comenzar a responder.
b) Desarrolle una redacción clara y precisa contestando lo que la pregunta requiere.
c) Observe la ortografía ya que la misma es parte del parcial.

NOTA: RESPETAR LAS BUENAS PRÁCTICAS QUE PROPONE LA ORIENTACIÓN A OBJETOS y PATRONES DE DISEÑO

En nuestra empresa hemos guardado y recuperado las bitácoras de nuestros sistemas en bases de datos relacionales y de archivos durante años. Para ello un diseñador de antaño ha creado una interfaz llamada **ILogger** en la cual se han definidos los siguientes métodos: Store(Log) y GetAll() : List<Log>. Tenemos algunas clases concretas como **SqlLogger** y **FileLogger** que implementan dicha interfaz. (La clase **Log** tiene dos atributos: string message y enum Severity)

Un nuevo cliente nos ha pedido que es sus desarrollos utilicemos una clase que ya posee para estos fines llamada **Logger** (Recibe en el constructor un parámetro de tipo enum: File o SQL y otro parámetro de tipo string con el path/connString), la misma posee los métodos Write (string) y ReadAll(): string[] (El parámetro string sería el equivalente a nuestro string message de la clase **Log** y el cliente no desea determinar severidades para sus bitácoras, ya que toda la gestión es manejada con cadenas formateadas para tal fin).

A pesar de que utilizaremos su clase para guardar las bitácoras, queremos seguir utilizando la interfaz **ILogger** para no apartarnos de nuestro estándar.

1) ¿Qué diseño le propondríamos a nuestro cliente para satisfacer el requerimiento? **Desarrolle el nuevo modelo UML.** Implemente la solución como una capa de servicios transversal. Haga sus pruebas con un proyecto de consola como cliente. (30)

2) Implemente en código C# la solución propuesta generando el modelo de base de datos (SQL) y formato de archivo que usted crea conveniente para realizar las bitácoras. (10)

3) Toda la información de rutas y cadenas de conexión deben ser parametrizados desde el archivo app.config y cargados al iniciar la aplicación en una clase llamada ApplicationSettings. (20)

4) ¿Cómo podríamos evitar que los clientes instancien las clases concretas SqlLogger y FileLogger? Implemente. (20)

5) En una nueva reunión el cliente nos comenta que desea agregar la funcionalidad de que al detectar en un mensaje la palabra “CriticalError”, se envíe un email a la casilla soporteNivel1@email.com. y si se detecta la palabra “FatalError” se envíe una copia a soporteNivel2@email.com ¿Cuál sería la mejor manera de plantear esa mejora tratando de modificar lo mínimo posible la clase **Logger** de nuestro cliente? Implemente su solución (No hace falta implementar el envío de emails). (20)