

# Soluciones del examen de Ingeniería del Software II

7 Junio 2002

## **Ejercicio 1** (1 punto)

El grupo de soporte de gestión de base de datos de la empresa GESTOR. S.L. está formado por cuatro programadores cuyo trabajo consiste fundamentalmente en realizar mejoras funcionales y reparar errores y defectos. Su máquina de desarrollo compartido tiene una copia en disco del código fuente y objeto, que sirve como configuración de referencia. Además, también existe en disco un área de fichero para cada programador que sirve de espacio de trabajo privado. Todo el mundo tiene permiso de lectura para la configuración de referencia, incluyendo la posibilidad de copiar cualquier módulo a su espacio privado. Una vez en su espacio de trabajo, los módulos pueden editarse. Los módulos editados se compilan en su espacio de trabajo privado y se integran para probarse con los módulos de la configuración de referencia.

Luis es el único programador que escribe en la configuración de referencia y es el encargado de la biblioteca del proyecto. Aunque no es director de proyecto, es el programador con más experiencia y ha trabajado con el sistema durante cinco años.

Un programador que ha realizado un cambio y quiere copiarlo en la configuración de referencia se lo da a Luis, quien verifica su corrección y examina los casos de prueba que se le han realizado. Si Luis aprueba el cambio, inserta el módulo revisado en la configuración de referencia.

Teniendo en cuenta este procedimiento de gestión de configuraciones, ¿cuáles son sus ventajas y cuáles sus desventajas?

### VENTAJAS:

- Al haber espacios de trabajo privados cada programador trabaja sin interferir a los demás -> No existen problemas de compartición de datos.
- No existe problema de actualización simultánea de datos porque lo hace Luis.
- Es un grupo pequeño, por lo que Luis puede mantener informados a todos.
- Existe revisión de cambios antes de introducirlos en la configuración de referencia.

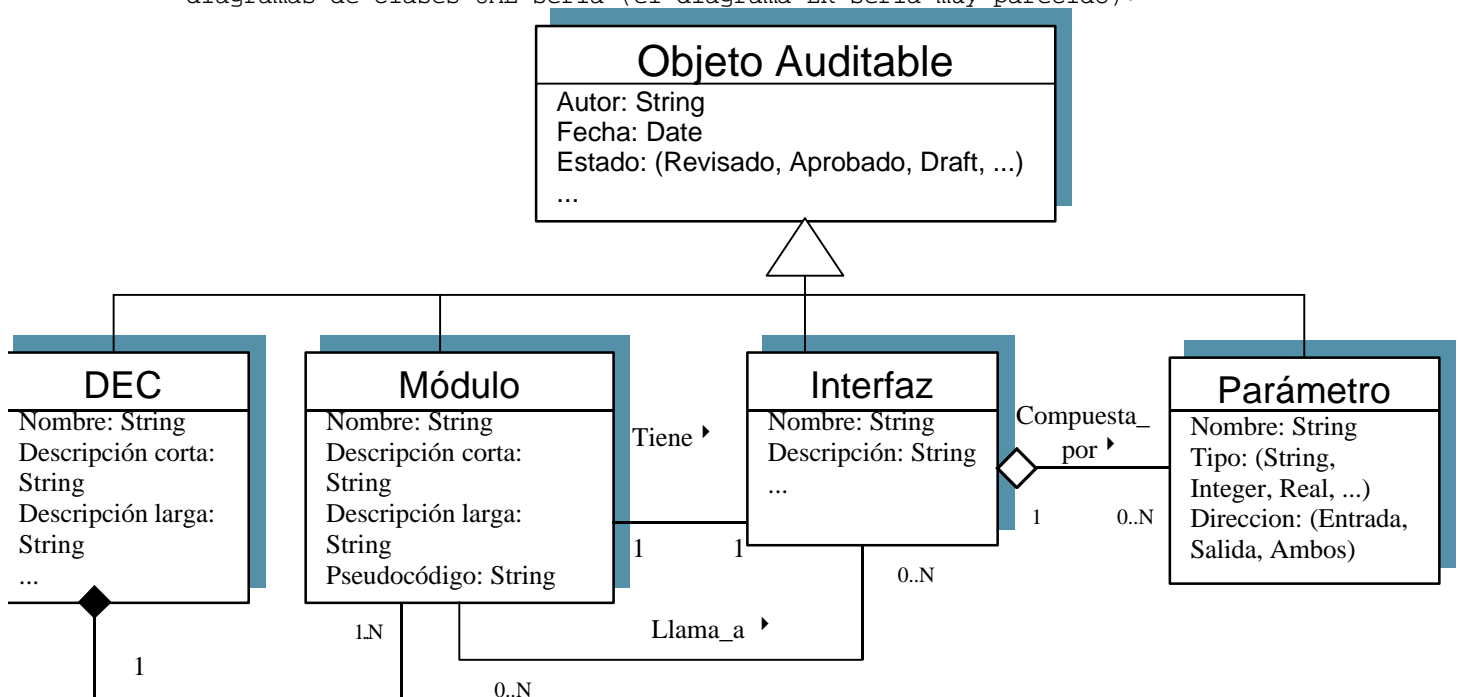
### DESVENTAJAS:

- Dependencia de una persona (Luis).
- Procedimiento manual, sin herramientas (siempre es más propenso a errores).

**Ejercicio 2** (1'5 puntos, *una cara máximo*)

¿Qué información incluiría el meta-modelo (la estructura del repositorio) de una herramienta CASE que, entre otros formalismos, incluyera soporte para Diagramas de Estructura de Cuadros (DECs)? Haz un esquema del mismo usando la notación que consideres oportuna. ¿Qué tipo de validaciones podría realizar una herramienta CASE con la información del meta-modelo?

El metamodelo incluye información sobre la sintaxis de un determinado formalismo; en este caso del formalismo DEC. Es necesario incluir datos básicos (entidades y sus atributos), asociaciones entre las entidades, restricciones (locales y globales) que deban cumplir las entidades y además debería incluir información de auditoría (quién creo la entidad, cuándo, quién la modificó, etc.). Los meta-modelos se describen mediante formalismos tipo Entidad-Relación o diagramas de clases UML. En caso necesario se complementan con un lenguaje de restricciones tipo OCL. Un posible esquema en forma de diagramas de clases UML sería (el diagrama ER sería muy parecido):



Habría una restricción local en los módulos de tal forma que sólo se pueden conectar módulos pertenecientes al mismo DEC. También podríamos añadir la restricción de que debe existir un único módulo que no es llamado por ningún otro (que sería el módulo principal del sistema), el resto debe ser llamado por al menos un módulo. Esta sería una restricción global. Es por este módulo principal que necesitamos poner la relación "Llama\_a" como [0..N] (en la parte de la clase "Interfaz"). El resto de las interfaces recibe al menos [1..N] llamadas.

En general las herramientas CASE pueden hacer dos tipos de validaciones:

- **Horizontal**: entre distintos modelos del mismo sistema. En este caso, si la herramienta soportara también DFDs, se podría validar que ambos modelos sean consistentes, al menos en cuanto a datos que maneje la aplicación: el contenido de los flujos de datos del DFD deben haber ido a parar a las interfaces de los módulos del DEC.
- **Vertical**: entre los distintos niveles de abstracción del modelo (por ejemplo, en el caso del DFD que esté equilibrado). En el caso del DEC no es aplicable.

Además la herramienta asegura que el modelo que construimos es sintácticamente correcto: en nuestro caso que sólo hay un módulo que no es llamado (la raíz de la jerarquía) y que para el resto de los módulos, hay algún módulo que lo llama. Esto es, que se verifican las restricciones que hemos especificado.

**Ejercicio 3** (3 puntos)

Contesta brevemente a las siguientes cuestiones:

a) ¿Qué puede aportar un entorno CASE en relación con la reutilización? ¿Qué elementos le faltan típicamente?

- Aporta: Repositorio, Herramienta de documentación, Herramienta de control de calidad.
- Le suele faltar facilidades de búsqueda y clasificación de componentes por diversos criterios.

b) ¿Cuáles son las estrategias aceptables para la gestión de riesgos? ¿Por qué?

- Sólo la proactiva (y no la reactiva) ya que es mucho mejor adelantarse a los problemas que puedan surgir en el proyecto que esperar a que los problemas se presenten, ya que solucionarlos entonces sería demasiado tarde.

c) En el contexto del desarrollo de componentes reutilizables, ¿Qué relación hay entre la generalidad de un componente, su reusabilidad, su complejidad, su adaptabilidad y su coste de desarrollo?

Cuanto más general es un componente hay más posibilidades de reutilizarlo (mayor reusabilidad), pero es más complejo y más difícil de adaptar, y más costoso de desarrollar.

d) Un código mal estructurado y sin comentarios, pero que funciona bien, está comenzando a ser muy costoso de mantener. ¿Qué aplicarías para resolver esta situación, reestructuración o ingeniería inversa?

Las dos son posibles:

- Si tengo diseño y sólo voy a trabajar con el código → reestructuración.
- Si no tengo el diseño y lo necesito → Ingeniería Inversa y a partir del diseño, obtengo el código bien estructurado por ingeniería directa.

e) Al aplicar ingeniería inversa en un sistema software, ¿qué relación existe habitualmente entre la completitud de la información y el nivel de abstracción de la misma?

Normalmente la completitud de la información extraída decrece a medida que el nivel de abstracción es más alto.

f) En un proyecto software, ¿las revisiones se llevan a cabo sólo durante las primeras fases?

No, pueden tener lugar durante todo el ciclo de vida.

g) ¿En qué puntos del ciclo de vida de un proyecto software se pueden llevar a cabo actividades para mejorar la mantenibilidad del software?

A lo largo de todo el ciclo de vida.

- h) En el contexto de la reutilización, ¿Qué similitudes y diferencias hay en la documentación necesaria para un componente en los enfoques de caja negra y caja blanca ?

En los dos casos: Necesito documentación para poder buscar el componente (e.d. clasificarlo) y para saber qué funcionalidad implementa (e.d. evaluarlos).

Si uso caja negra: necesito centrarme en las interfaces y servicios que provee.

Si uso caja blanca: Además necesito información sobre cómo se ha implementado el componente, porque lo voy a tener que modificar y adaptar.

- i) Una vez finalizado el desarrollo de un proyecto informático, ¿es necesario seguir realizando inspecciones?

Sí, es una medida de calidad que se puede aplicar siempre que se desarrolle o se cambie el software y durante el mantenimiento se desarrolla y se cambia el software.

- j) ¿Se puede afirmar que el director de un proyecto software es el encargado de realizar las actividades financieras del proyecto?

No, el director de proyecto es el responsable final, pero las actividades financieras las puede realizar él personalmente o no.

#### **Ejercicio 4** (1,5 puntos)

Explica brevemente las diferencias entre los siguientes conceptos (máximo 10 líneas cada uno):

- a) Gestión de proyectos SW y Gestión de proyectos de otras ingenierías.

El software es inmaterial y es más difícil por tanto conocer el progreso del proyecto. Hay un alto nivel de abstracción unido a una baja estandarización de métodos y procesos. Los productos SW son únicos, se suelen hacer a medida; las técnicas de reutilización son mucho más inmaduras que por ejemplo en ingeniería electrónica. No hay un entendimiento claro del proceso de construcción del software, que es inherentemente no determinista.

- b) Reestructuración de código y Reingeniería.

La reestructuración de código no modifica el diseño de arquitectura, si no que mejora la calidad del código de módulos individuales. La reingeniería suele incluir un proceso de ingeniería inversa para obtener el diseño, se reestructura el diseño y se obtiene el código por ingeniería directa. Es decir, hay un cambio en el nivel de abstracción anterior.

- c) Línea Base e Hito.

Las línea base son un conjunto de elementos que han sido revisados y aprobados, y se han introducido en el repositorio, a partir de este momento los cambios tienen que hacerse de manera formal. Son un elemento fundamental en la gestión de

configuraciones. Los hitos son puntos en el tiempo en que algo tiene que haberse revisado y aprobado, y son un elemento fundamental para la planificación.

d) **Reutilización basada en Frameworks y basada en patrones de diseño.**

Los frameworks son un conjunto de clases en los que se tienen que rellenar las partes que faltan (subclasificando e implementando métodos abstractos, implementando métodos callback. Etc). Ejemplos típicos son los frameworks para interfaces de usuario, tipo MFC o Java Swing. Los patrones de diseño por el contrario son a menor escala y la idea es tener un catálogo de soluciones estándar a problemas comunes, independientes de la implementación.

e) **Herramientas CASE y Meta-CASE.**

Las herramientas CASE no permiten modificar la estructura del repositorio (del meta-modelo), las Meta-CASE sí, son por tanto más flexibles, ya que permiten definir nuevos formalismos o modificar los ya existentes.

**Ejercicio 5** (1'5 puntos)

Comenta los ciclos de vida de desarrollo de proyectos software para reutilizar y de proyectos software con reutilización identificando las diferencias entre ambos.

CON (Encontrar, Evaluar, Adaptar):

- Hacer estimación y plan de proyecto
- Análisis de requisitos
- Identificar qué requisitos pueden satisfacer los componentes reusables
- Buscar los componentes en el repositorio
- Evaluar y comprender los componentes
- Diseño
- Codificación
- Adaptar los componentes
- Pruebas
- Documentación
- Entrega

PARA (Analizar, Clasificar, Generalizar, Construir):

- Hacer estimación y plan de proyecto teniendo en cuenta el esfuerzo extra para la realización de componentes generales
- Identificar potenciales reutilizadores
- Recoger sus requisitos
- Análisis de requisitos
- Análisis de variabilidad, generalidad, coste/beneficio
- Identificar requisitos posibles de ser generalizables y los que no
- Diseño solución inicial
- Evaluación de diseño, teniendo en cuenta la generalidad
- Aprobar diseño
- Codificación

- Diseño de casos de prueba teniendo en cuenta casos genéricos que correspondan a más de un usuario
- Pruebas exhaustivas
- Documentación del sistema y de los componentes construidos
- Introducción de los componentes en el repositorio
- Entrega del sistema

**Ejercicio 6** (1'5 punto)

Indica las medidas (actividades) de calidad más apropiadas para conseguir los siguientes objetivos del aseguramiento de la calidad del software:

Objetivos de calidad	Medida
Verificación del código	Inspección Walkthrough
Validación del código	Medidas dinámicas
Confirmación del cumplimiento de requisitos de proceso	Auditoría
Confirmación del cumplimiento de requisitos de producto	Auditoría Revisiones Medidas dinámicas
Evaluación de documentación	Revisiones