Soluciones-examen-de-ejemplo.pdf

Anónimo

Ingeniería del Software

3º Grado en Ingeniería Informática

Escuela Politécnica Superior
Universidad Autónoma de Madrid

AMRO
ESTUDIANTES

GRANADA | MÁLAGA

LA RESIDENCIA DE ESTUDIANTES PERFECTA PARA TI

Tú Decides | Estancias Flexibles

PLAN AMIGO
120€
Cheque Amazon

Tour virtual

amroestudiantes.es

Amro Estudiantes, la residencia universitaria con todo listo para que vivas una época inolvidable, hagas nuevos amigos, crees recuerdos inolvidables y puedas concentrarte en tus estudios sin ningún tipo de preocupación y teniendo acceso a todas las comodidades de primer nivel.

Escuela Politécnica Superior, Ingeniería Informática

Examen Ingeniería del Software II - Parte I

Ejercicio 1 *(4 puntos; 0,5 por pregunta)*

Contesta de forma breve (máximo 5 líneas por respuesta) a las siguientes cuestiones:

a) En el entorno de la Ingeniería del Software, ¿se deberían utilizar metodologías durante el mantenimiento? ¿Por qué?

Sí, porque principalmente en los mantenimientos de tipo planificado se exige una consecución específica de fases, actividades y productos bajo una gestión y control específicos, lo que hace necesario el uso de una metodología para la sistematización de las actividades que dan soporte al ciclo de vida del mantenimiento del software.

b) ¿Por qué surge el concepto de agilidad o metodología ágil dentro de la Ingeniería del Software?

Principalmente para hacer frente al cambio continuo en los requisitos y a la necesidad de obtener, de forma ágil, productos intermedios que puedan ser validados por el usuario, frente a las metodologías clásicas demasiado pesadas y con fases largas en el tiempo. También, como medio para aplicar otras características modernas en la creación de software, como la sostenibilidad.

c) ¿Qué implica, dentro del dominio de la Ingeniería del Software, que el software, a diferencia de otros productos, es desarrollado y no fabricado?

Básicamente que el software es desarrollado, no manufacturado o fabricado con medios mecánicos. Es decir, para el desarrollo del software se necesitan distintos procedimientos, técnicas, herramientas y métodos, desde su concepción inicial hasta obtener el producto final resultante.

d) Indica las semejanzas y diferencia entre las Pruebas de Verificación y las Pruebas de Validación

SEMEJANZAS	DIFERENCIAS
Ambas están orientadas al aseguramiento de la calidad, y tratan de encontrar errores y deficiencias en el producto en distintos momentos del ciclo de vida.	Las Pruebas de Validación se llevan a cabo durante todo el ciclo de vida, mientras que las Pruebas de Verificación se llevan a cabo en el entorno de programación (pruebas unitarias) o en el entorno de desarrollo, después de la codificación.

e) En un caso general, ¿cómo eliminarías un alto acoplamiento entre dos módulos?
El acoplamiento nos indica el grado de dependencia entre módulos. Un alto acoplamiento no es deseable, debido al efecto ola que produce y que incide en la propagación de errores. Para eliminar el acoplamiento, lo mejor es rediseñar los módulos para que no compartan información, uniéndolos en uno si es posible.

f) ¿Qué tipo de actividades permiten el aseguramiento de la usabilidad y la accesibilidad durante el desarrollo de software?

Principalmente: el estudio del usuario, su contexto y sus tareas, el desarrollo y validación a partir de prototipos evolutivos, uso de estándares y normas ergonómicas, y una evaluación acorde que permita comprobar los objetivos de usabilidad y accesibilidad establecidos.

g) ¿En qué casos se debe entregar el manual técnico al cliente y en qué casos se lo debe quedar la empresa desarrolladora?

Sólo se debe entregar cuando el mantenimiento lo vaya a llevar el mismo cliente u otra empresa, o cuando así lo estipule en el contrato.

h) ¿Qué elementos definen un modelo de ciclo de vida?

Las fases, sus relaciones y los criterios para la transición de fases.

Ejercicio 2 *(1 punto)*

Contesta a las siguientes cuestiones:

a) Supón que trabajas en una empresa como analista, y para un proyecto en el que has previsto dificultades de entendimiento con el cliente en la fase de análisis has decidido realizar una maqueta. Los clientes te dicen que la maqueta es justo lo que ellos querían, te animan a que sigas trabajando en ella para que se la entregues cuanto antes ya que, según ellos, añadiéndole un par de características (unos cálculos numéricos y comunicaciones con TCP/IP) el producto estaría terminado. ¿Cómo reaccionarías ante esta situación?

Las maquetas son sólo una interfaz con la apariencia de la aplicación, y nunca evolucionan hacia el producto final. Se han realizado de manera rápida, prestando poca atención al diseño, en un lenguaje de programación de alto nivel, de gran potencia gráfica pero probablemente poco adecuado para el desarrollo de una aplicación (y más de las características descritas en la pregunta). La maqueta se debería reimplementarse en lenguaje de programación adecuado, utilizando las técnicas de diseño, codificación, etc. adecuadas.

b) Identifica brevemente las semejanzas y diferencias entre Acoplamiento/Cohesión y los Principios de diseño.

Acoplamiento/Cohesión – Principios de diseño	Ambos grupos de conceptos se aplican para obtener un diseño de alta calidad.	Acoplamiento y cohesión son métricas.

Ejercicio 3 *(2 puntos; 0.5 por apartado)*

Sean dos grafos de flujo G_1 y G_2 que detallan la representación del flujo de instrucciones de dos módulos M_1 y M_2, respectivamente. Al calcular la complejidad ciclomática de estos dos grafos de flujo, se obtiene que $V(G_1)=20$ y $V(G_2)=5$. Responde a las siguientes preguntas:

a) ¿Cuál es el número de nodos predicado de G_1?
b) ¿Cuántas regiones cerradas comprende el grafo de flujo G_2?
c) En base a los valores calculados para $V(G_1)$ y $V(G_2)$, ¿qué recomendaciones de diseño se pueden replantear para los módulos M_1 y M_2?
d) ¿Consideras que la complejidad ciclomática puede ser independiente del lenguaje de programación utilizado? Razona tu respuesta.

Solución:

a) El número de nodos predicado de G_1 es 19, ya que $V(G_1)$=Nº de Nodos Predicado + 1, luego Nº de Nodos Predicado = $V(G_1)$ - 1 => 19.
b) El número de regiones cerradas de G_2 es 4, ya que $V(G_2)$=Nº de Regiones Cerradas + 1, luego Nº de Regiones Cerradas = $V(G_2)$ - 1 => 4.
c) La complejidad ciclomática, que es una métrica del software, no debe ser idealmente mayor de 10. En el caso de M_1, se recomienda un rediseño para disminuir la complejidad ciclomática (y aumentar la cohesión). En el caso de M_2, no es necesario.
d) Sí, la complejidad ciclomática es una métrica independiente del lenguaje de programación utilizado, ya que se basa en la complejidad lógica del código, es decir, en las estructuras de control del código, que son las que hacen aumentar realmente su complejidad ciclomática y su mantenibilidad / facilidad de prueba, y esto es común para todos los lenguajes en general

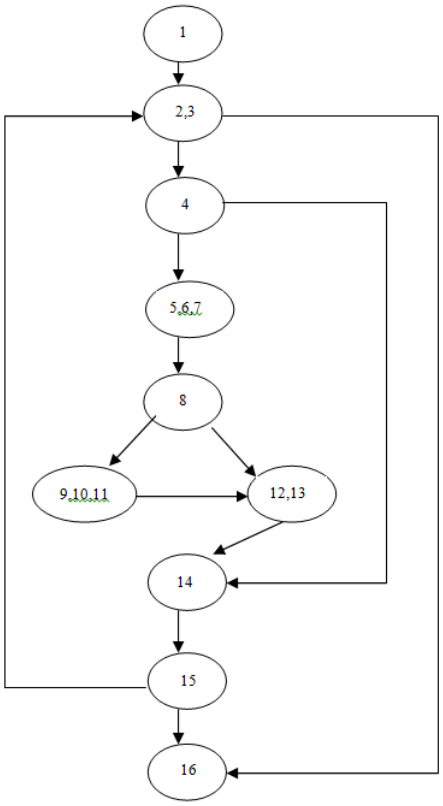
Ejercicio 4 *(3 puntos; 1,5 por apartado)*

Dado el siguiente código para la impresión de datos de juegos:

```
1  Begin
2    repeat
3        LeerDatosJuegoUsuario(Datos);
4        if not eof() then
5            Puntuacion=CalculaPuntuacion(Datos.JuegosJugados, Datos.Puntos);
6            RegistrarPuntuacion(Datos.Usuario, Puntuacion);
7            Bonificacion=0;
8            if  Datos.NivelJuego>4  and  Datos.JuegosJugados>=3
```

```
 9            then
10                Bonificacion=CalculaBonus(Datos.JuegosJugados, Datos.Bonus);
11                RegistrarBonificacion(Datos.Usuario, Datos.NivelJuego);
12            endif
13            ImprimePuntuacion(Datos.Usuario, Puntuacion, Bonificacion);
14        endif
15    until eof();
16  End
```

a) Representa el grafo de flujo para realizar pruebas de caja blanca y calcula la complejidad ciclomática.



Complejidad ciclomática = Aristas – Nodos + 2 = 13 – 10 + 2 = 5

b) Supongamos que la interfaz del módulo es la que se especifica a continuación. Con el objetivo de realizar pruebas de caja negra, ¿cuáles serían las clases de equivalencia? Represéntalas en una tabla. Diseña, además, 5 casos de prueba de caja negra.

```
FUNCTION ValidaUsuario (Usuario: String): Integer;

{Esta función se utiliza para validar el identificador de un usuario.

Los parámetros de la función son los siguientes:

* Usuario: Representa el identificador del usuario en la plataforma. Todos
```

los nombres de usuarios deben ser de 10 caracteres. El primer caracter debe ser obligatoriamente una letra. Los ocho siguientes pueden ser cualquier combinación de letras y espacios en blanco (no pudiendo incluir signos de puntuacion) y el último caracter debe ser obligatoriamente un signo "#".

* Parámetro de retorno de la función (Codigo_Error): Puede ser cualquiera de los siguientes valores indicando el exito o fracaso de la función:

 0 = Exito imprimiendo la lista de juegos.
-1 = Identificador demasiado largo (si el identificador es > 10 caracteres)
-2 = Elementos no permitidos dentro del identificador (si hay signos de puntuacion).
-3 = Identificador incorrecto.
}

Solución:

Atributo	Clases Válidas	Clases Inválidas
Usuario	10 caracteres 1er carácter = letra 8 siguientes = letras, espacios en blanco, pero no signos de puntuación Último carácter = #	< 10 caracteres > 10 caracteres 1er carácter no letra 8 siguientes algún signo de puntuación Último carácter no #
Codigo_Error	0, -1, -2, -3	Cualquier otro valor

Nº Caso de Prueba	Usuario	Codigo_Error
1	Luismanue#	0
2	Luismanuel#	-1
3	Luis-manu#	-2
4	Luismanuel	-3
5	Luisma	-3

Examen Ingeniería del Software II
Parte II

Ejercicio 1 *(2,4 puntos; 0,4 por pregunta).*

Contesta brevemente a las siguientes cuestiones:

a) En algún tipo de mantenimiento, los cambios pueden empezar a realizarse en cualquier fase del ciclo de vida de desarrollo.

Sí, en el correctivo, porque puede corregir directamente el código o el diseño, etc. sin pasar por las fases anteriores. Puedes comenzar desde cualquier fase.

b) Desde el punto de vista temporal de un proyecto informático, ¿hasta cuándo hay que realizar inspecciones?

Hasta el final del proyecto, cuando se retira el producto.

c) ¿Consiste la ingeniería inversa en cambiar el código de la aplicación modificando su forma pero no su funcionalidad?

No, eso es la reingeniería. La ingeniería inversa consiste en obtener un nivel de abstracción superior.

d) ¿La ingeniería inversa y la reingeniería son dos procesos relacionados con la evolución del software?
Sí, son procesos de mantenimiento estructural que se realiza debido, entre otras cosas, a la evolución del software.

e) ¿Cuándo se definen las principales líneas base de un proyecto?

En la fase de planificación

f) ¿Crees que influyen en el coste de mantenimiento las técnicas de GCS empleadas?

Sí, está íntimamente ligado, ya que cada técnica tiene su planificación y recursos asociados, y por tanto el coste cambia dependiendo de cada técnica a aplicar. Es algo que debe estimarse y planificarse adecuadamente en fases iniciales al proyecto.

Ejercicio 2 *(2 puntos; 0.2 puntos por cada pregunta)*

a) ¿Qué medida o medidas de calidad aplicarías en cada caso para comprobar la actividad que se expone? ***(1,2 puntos; 0.2 puntos por cada pregunta)***

PRODUCTO	**MEDIDA DE CALIDAD**
Un conjunto de casos de prueba	Revisiones (Inspecciones)
Un informe de cambio	Revisiones
Un cambio realizado sobre el producto	Medidas dinámicas, Revisiones
Validación del producto final	Medidas dinámicas
Plan de mantenimiento	Revisión, Auditoria y Medidas constructivas
El proceso de mantenimiento	Auditoría

b) Completa con las medidas de calidad a aplicar más adecuadas (aunque no de forma exclusiva) para cada tipo de actividad de un proyecto software. ***(0,8 puntos)***

	Actividades técnicas	**Actividades de gestión**
Medidas de calidad	• Medidas dinámicas • Inspecciones • Walkthroughs • Medidas constructivas técnicas	• Auditorías • Medidas constructivas organizativas • Medidas constructivas humanas

Ejercicio 3 *(1 punto)*

A continuación se enumeran una serie de decisiones que se toman en algún momento durante el desarrollo y mantenimiento de un proyecto software. Identifica, al lado de cada decisión, qué rol o roles toma cada una de estas decisiones, es decir, toman la decisión de llevar a cabo la actividad. Los posibles roles son: Cliente (C), Desarrolladores (D), Jefe de proyecto (JP) y Dirección empresa desarrolladora (ED).

1. Contratación de un servicio de mantenimiento (C),(ED)
2. Realización de reingeniería (JP),(ED)
3. Selección de los casos de pruebas más adecuados (D),(JP)
4. Realización y seguimiento de un plan de calidad (JP),(C)
5. Utilización de herramientas específicas de gestión de configuraciones (JP),(ED)

Ejercicio 4 *(1,6 puntos)*

De los productos entregados al usuario a lo largo del ciclo de vida de mantenimiento de un proyecto software, especifica cuáles son líneas base y cuáles no. Utiliza la tabla dada a continuación

Productos	**Línea Base**	
	SI	**NO**
Producto final	SI	
Manual de usuario	SI	
(Contrato)		
Actas de reuniones		NO
Informes de seguimiento		NO
		NO

Ejercicio 5 *(1,2 punto)*

Enumera al menos 6 características que debería tener todo sistema software que se considere usable.

- Facilidad de uso
- Facilidad de aprendizaje
- Flexibilidad
- Robustez
- Predictibilidad
- Consistencia
- Buena recuperabilidad
- Buen tiempo de respuesta
- Adecuación a las tareas
- Baja carga cognitiva

Ejercicio 6 *(1,8 puntos; 0,3 por pregunta)*

Determina cuáles de los siguientes casos o afirmaciones son verdaderos o falsos. En caso de que alguno sea falso, justifica por qué y redáctalo para que resulte verdadero.

a) La mantenibilidad es un atributo de calidad del software consistente en un grupo de actividades a considerar durante la fase de mantenimiento. (F -> En todo el CV)

b) La usabilidad de una aplicación se ve afectada solamente por lo bien o mal diseñada que esté la interfaz de usuario (F -> Se ve afectada tanto por aspectos funcionales como no funcionales)

c) Cuando un sistema software en mantenimiento falla muy a menudo y cada vez se utiliza menos la mejor opción, a priori, es continuar con el mantenimiento tal cual, porque aunque es costoso mantener la aplicación se va a dejar de usar en breve. (V)

d) Una empresa tiene en producción una aplicación informática que falla muy frecuentemente. Además, presenta dos aspectos que incrementa su ineficacia: por una parte, la empresa ha ampliado su rango de actividades, por lo que es necesario añadir un gran número de funciones y tareas nuevas a la aplicación y, por otra, se ha decidido cambiar todos los ordenadores de la empresa al sistema operativo OSX Lion, por lo que la aplicación se tiene que portar a dicho entorno. En este caso, la mejor opción es aplicar reingeniería para reducir costes y aumentar la productividad. (F -> Compensa mejor un nuevo desarrollo, en vez de reingeniería)

e) Durante la fase de mantenimiento, el coste de arreglar un error cometido en la fase de análisis, es menor que el coste de arreglar un error cometido en la fase de codificación. (F -> El coste es mayor)

f) La mejor opción para la eficacia del proceso de mantenimiento de un proyecto ya instalado en una empresa cliente es llevarlo a cabo de manera no planificada, es decir, bajo petición (F-> El mantenimiento preventivo siempre debe ser planificado)