

INGENIERÍA DEL SOFTWARE (2º Curso) Cód. 53210 SISTEMAS 54208 GESTIÓN				
	1ª Semana			
			Nacional	

Todas las preguntas de este ejercicio son eliminatorias en el sentido de que debe obtener una nota mínima en cada una de ellas. En cada pregunta teórica, que se valora con 2'5 puntos, la nota mínima es 1 punto; en la segunda parte (ejercicio de teoría aplicada que se valora con 5 puntos) la nota mínima que debe obtener es de 2 puntos.

SI ESTE EJERCICIO NO SE HA IMPRESO EN HOJA DE LECTURA ÓPTICA, SOLICITE UNA AL TRIBUNAL.

¡ATENCIÓN! PONGA SUS DATOS EN LA HOJA DE LECTURA ÓPTICA QUE DEBERÁ ENTREGAR JUNTO CON EL RESTO DE LAS RESPUESTAS.

Conteste a las preguntas teóricas, en cualquier orden, en hojas diferentes a las que utilice para la contestación de la segunda parte. En cada parte, la cantidad MÁXIMA de papel (de examen, timbrado) que puede emplear ESTÁ LIMITADA al equivalente a DOS (2) HOJAS de tamaño A4 (210 x 297 mm)

PRIMERA PARTE. PREGUNTAS TEÓRICAS (2'5 PUNTOS CADA UNA)

1. Dos aspectos clave que el equipo de analistas debe desarrollar en la fase de especificación de requisitos son: determinar con rapidez las necesidades reales del cliente y proponer unas soluciones funcionales que las satisfagan con eficacia. Piense y explique dos métodos que permitan hacer un seguimiento de estos dos aspectos (por ejemplo con medidas de algún parámetro del proceso en la fase de análisis).

Solución

Para el primer aspecto, determinar con rapidez y agilidad las necesidades del cliente, una medida podría ser el número de cambios en los requisitos que se identifica durante la fase de análisis y elicitación. Un número elevado podría indicar un problema en la comunicación con el cliente o un estudio pobre del dominio. En cuanto a las propuestas de soluciones que satisfagan las necesidades del cliente, se podría contemplar revisando cuantos cambios se producen en los requisitos o en el modelo, en el resto de las fases del desarrollo. Un valor elevado indicaría una comprensión pobre del sistema que se desarrolla, es decir, un análisis incompleto o defectuoso que obligaría a frecuentes 'vueltas atrás' y replanteamientos.

Nota: Esta pregunta se ha calificado con el siguiente algoritmo:

La calificación es el máximo entre uno (1) y la puntuación obtenida en la corrección '*normal*' de la respuesta (puntuación máxima de 2'5).

2. En que medida incorpora o facilita, el Diseño Orientado a Objetos, la aplicación de los conceptos de '*abstracción*', '*modularidad*', '*ocultación*', '*herencia*' y '*polimorfismo*'.

Solución

- A. En cuanto a la abstracción, el concepto de clase nos permite manejar información como elementos individuales abstrayéndonos de sus posible composición interna y atendiendo sólo a su comportamiento.
- B. La modularidad: los módulos de nuestro diseño básicamente son las clases. Se relacionan entre ellas por medio del modelo de clases y el de relaciones de uso.
- C. Ocultación, propiedad por la cual conseguimos que nuestros módulos oculten sus estructura interna, también se consigue utilizando Clases, pues, su implementación se realiza dentro de la propia clase y es totalmente transparente cuando se utilizan éstas en la aplicación.

- D. Herencia, es un mecanismo propio del diseño Orientado a Objetos, que nos permite, de entrada, reutilizar código.
- E. Polimorfismo. La herencia, a parte de reutilizar código, tiene como principal característica el permitir polimorfismo: de anulación o diferido.

SEGUNDA PARTE. PREGUNTA DE TEORÍA APLICADA (MÁXIMO 5 PUNTOS)

3. Se ha codificado en Modula-2 el siguiente subprograma que distingue, por el tamaño de sus lados si un triángulo es isósceles (dos lados iguales), equilátero (todos los lados iguales) o escaleno (ningún lado igual).

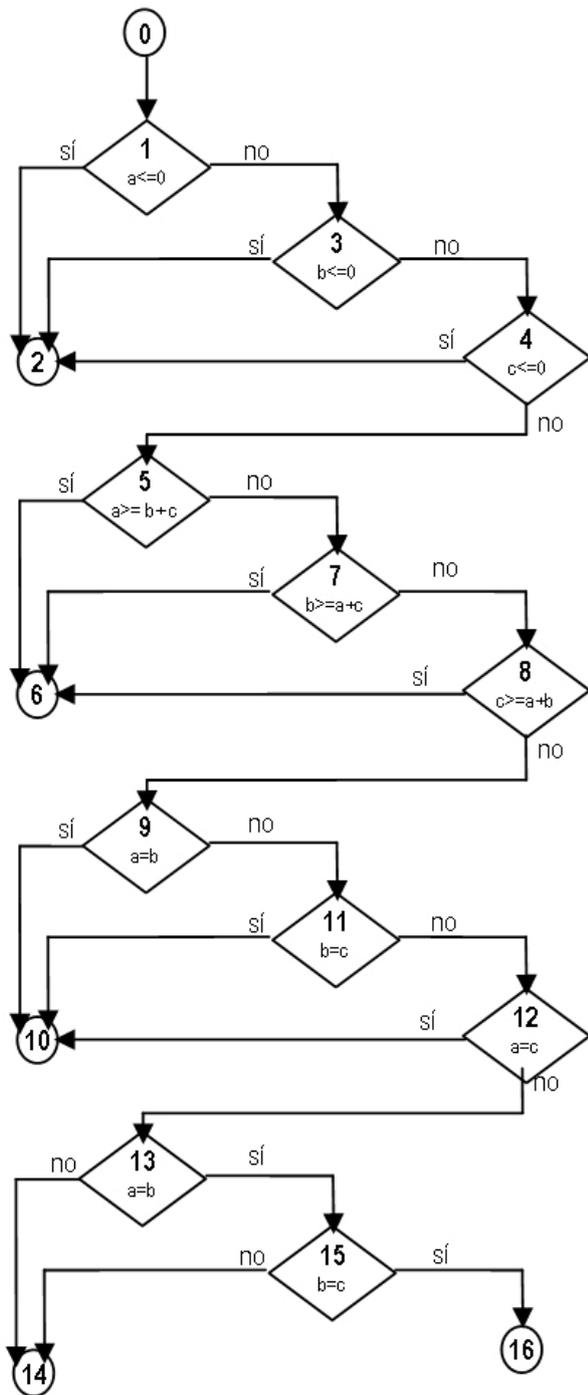
```
PROCEDURE TipoDeTriangulo(a, b, c: INTEGER);
BEGIN
  IF (a<=0) OR (b<=0) OR (c<=0) THEN
    WriteString("Error: alguno de los parámetros no es un número natural");
  ELSIF (a>=b+c) OR (b>=a+c) OR (c>=a+b) THEN
    WriteString("Error: no es posible cerrar el triángulo");
  ELSIF (a=b) OR (b=c) OR (a=c) THEN
    WriteString("El triángulo es isósceles");
  ELSIF (a=b) AND (b=c) THEN
    WriteString("El triángulo es equilátero");
  ELSE
    WriteString("El triángulo es escaleno");
  END;
END TipoDeTriangulo;
```

Verifique el subprograma con pruebas de caja transparente, realizando el cubrimiento lógico correspondiente.

Solución

Para la verificación del subprograma se elaborará un conjunto de casos de prueba que consigan que se transite por todos los posibles caminos de ejecución y que pongan en juego todos los elementos del código.

En primer lugar, el código del subprograma se transformará en el siguiente diagrama de flujo, donde cada rombo representa un predicado lógico simple:



2 Parámetros erróneos: valores no naturales

6 Parámetros erróneos: imposible cerrar el triángulo

10 Triángulo isósceles

14 Triángulo escaleno

16 Triángulo equilátero

A continuación, se calculará el nº de caminos básicos para recorrer todas las líneas de flujo del diagrama al menos una vez:

Nº de predicados = 11

Nº máximo de caminos = Nº de predicados + 1 = 12

A continuación se determinarán los caminos y los casos de prueba que forzarán su recorrido.

Como indica la siguiente figura, es imposible escribir un juego de prueba para los caminos 0-1-3-4-5-7-8-9-12-13-15-14 y 0-1-3-4-5-7-8-9-12-13-15-16. Es decir, **el subprograma es incorrecto** por que contiene dos caminos que no pueden recorrerse. Concretamente, la imposibilidad de transitar por el camino 0-1-3-4-5-7-8-9-12-13-15-16 hace que el subprograma clasifique erróneamente los triángulos equiláteros como isósceles.

	Camino	Resultado esperado	Juego de prueba
1	0-1-2	Parámetros erróneos: valores no naturales	a=0, b=1, c=1
2	0-1-3-2		a=1, b=-3, c=1
3	0-1-3-4-2		a=3, b=1, c=0
4	0-1-3-4-5-6	Parámetros erróneos: imposible cerrar el triángulo	a=5, b=1, c=1
5	0-1-3-4-5-7-6		a=2, b=4, c=2
6	0-1-3-4-5-7-8-6		a=3, b=1, c=5
7	0-1-3-4-5-7-8-9-10	Triángulo isósceles	a=2, b=2, c=1
8	0-1-3-4-5-7-8-9-11-10		a=2, b=3, c=3
9	0-1-3-4-5-7-8-9-12-10		a=4, b=3, c=4
10	0-1-3-4-5-7-8-9-12-13-14	Triángulo escaleno	a=3, b=4, c=5
11	0-1-3-4-5-7-8-9-12-13-15-14		Es imposible crear un juego de prueba
12	0-1-3-4-5-7-8-9-12-13-15-16		

INGENIERÍA DEL SOFTWARE (2º Curso) Cód. 53210 SISTEMAS 54208 GESTIÓN			
	Original		
		Unión Europea	

Todas las preguntas de este ejercicio son eliminatorias en el sentido de que debe obtener una nota mínima en cada una de ellas. En cada pregunta teórica, que se valora con 2'5 puntos, la nota mínima es 1 punto; en la segunda parte (ejercicio de teoría aplicada que se valora con 5 puntos) la nota mínima que debe obtener es de 2 puntos.

SI ESTE EJERCICIO NO SE HA IMPRESO EN HOJA DE LECTURA ÓPTICA, SOLICITE UNA AL TRIBUNAL.

¡ATENCIÓN! PONGA SUS DATOS EN LA HOJA DE LECTURA ÓPTICA QUE DEBERÁ ENTREGAR JUNTO CON EL RESTO DE LAS RESPUESTAS.

Conteste a las preguntas teóricas, en cualquier orden, en hojas diferentes a las que utilice para la contestación de la segunda parte. En cada parte, la cantidad MÁXIMA de papel (de examen, timbrado) que puede emplear ESTÁ LIMITADA al equivalente a DOS (2) HOJAS de tamaño A4 (210 x 297 mm)

PRIMERA PARTE. PREGUNTAS TEÓRICAS (2'5 PUNTOS CADA UNA)

1. Acaba de incorporarse a la empresa 'Victoria & Niagara Software' como director/a de software. Dicha empresa lleva muchos años desarrollando software de gestión contable para pequeñas empresas y utilizando el ciclo de vida en cascada con éxito aceptable. Sin embargo, según su experiencia, usted piensa que el modelo con prototipo rápido es una forma bastante mejor para desarrollar software. Escriba un informe, dirigido al vicepresidente de desarrollo de software, explicando por qué cree que la organización debería cambiar al uso del modelo con prototipo rápido. Recuerde que a los vicepresidentes no les agradan los informes de más de una página.

Solución

La ventaja del ciclo de vida en cascada es que establece un estilo de trabajo disciplinado y está dirigido por la documentación que se va generando. Sin embargo, no garantiza que el producto entregado sea el que necesita el cliente, porque la línea de fabricación se 'separa' del contacto con el cliente a partir de la fase de análisis. Con el modelo con prototipo rápido, sin embargo, el cliente ve inmediatamente si sus necesidades se reflejan, o no, en el prototipo y esto aumenta la confianza en la garantía de que el producto final responda a sus necesidades. Por otro lado, el hecho de que la organización tenga experiencia en un dominio concreto, hace que la creación de un prototipo sea casi inmediata. O dicho de otra manera, la experiencia de la organización permite crear un esquema o patrón general, aplicable al dominio, del que se puede obtener rápidamente (con una parametrización adecuada) el prototipo rápido necesario para cada desarrollo particular.

2. ¿Cuáles son las distintas estrategias de integración de los módulos de un producto software? Explíquelas brevemente.

Solución

(Pág. 285 y siguientes).

SEGUNDA PARTE. PREGUNTA DE TEORÍA APLICADA (MÁXIMO 5 PUNTOS)

3. Una clínica dental quiere informatizar su sistema de información. De ella recibimos el siguiente texto escrito:

“Deseamos almacenar registros de los pacientes, en los que figuren sus datos personales (nombre, dirección, teléfono, antecedentes médicos, etc.), las dolencias detectadas en sus visitas a nuestra clínica y los tratamientos realizados.

Disponemos de un equipo médico encargado de realizar los diagnósticos y llevar a cabo los tratamientos a los pacientes.

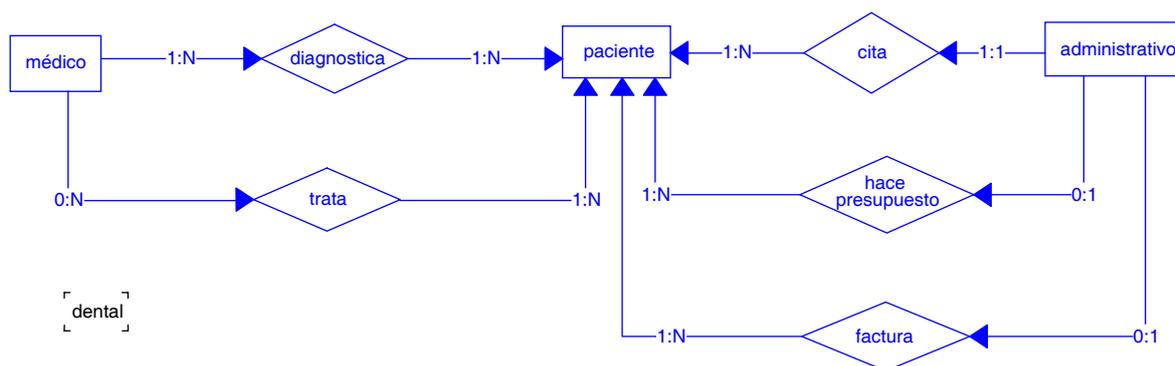
El personal administrativo se encarga de citar a los pacientes, realizar presupuestos de los tratamientos recomendados por los médicos y facturar los tratamientos realizados a los pacientes”

Se pide:

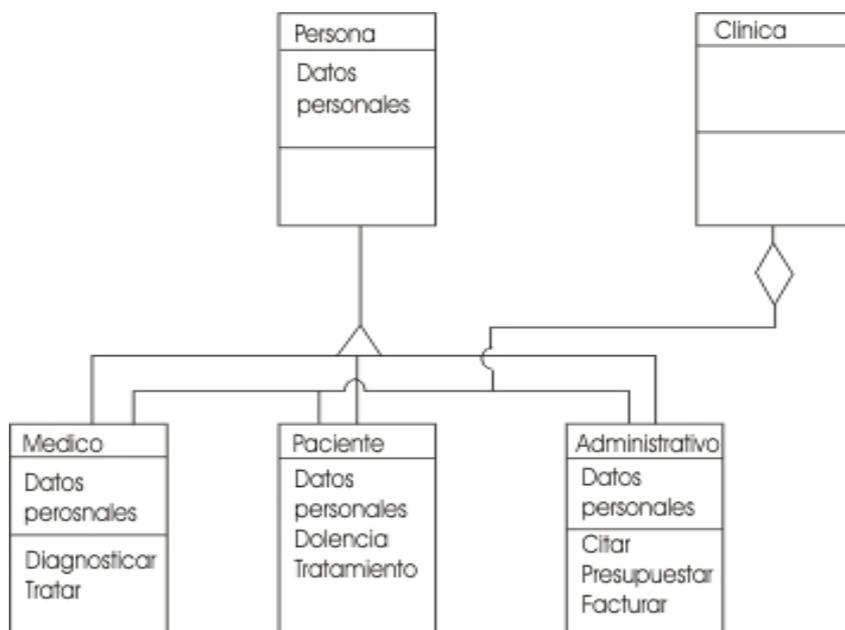
- Realice un análisis de los datos que va a manejar el sistema mediante el modelo entidad-relación.
- Establezca la estructura modular del sistema mediante la técnica de diseño orientado a objetos.

Solución

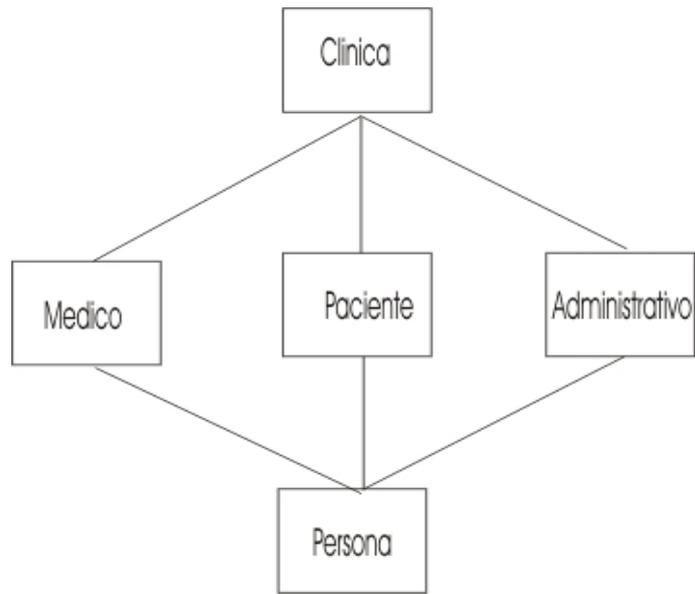
Diagrama entidad-relación:



Modelo de objetos:



Para obtener la estructura modular asignamos un módulo a cada clase de objetos. Las relaciones de herencia se traducen a relaciones de uso y las subclases utilizan el código de su superclase:



INGENIERÍA DEL SOFTWARE (2º Curso) Cód. 53210 SISTEMAS 54208 GESTIÓN			
	Original		
			C. Penitenciarios

Todas las preguntas de este ejercicio son eliminatorias en el sentido de que debe obtener una nota mínima en cada una de ellas. En cada pregunta teórica, que se valora con 2'5 puntos, la nota mínima es 1 punto; en la segunda parte (ejercicio de teoría aplicada que se valora con 5 puntos) la nota mínima que debe obtener es de 2 puntos.

SI ESTE EJERCICIO NO SE HA IMPRESO EN HOJA DE LECTURA ÓPTICA, SOLICITE UNA AL TRIBUNAL.

¡ATENCIÓN! PONGA SUS DATOS EN LA HOJA DE LECTURA ÓPTICA QUE DEBERÁ ENTREGAR JUNTO CON EL RESTO DE LAS RESPUESTAS.

Conteste a las preguntas teóricas, en cualquier orden, en hojas diferentes a las que utilice para la contestación de la segunda parte. En cada parte, la cantidad MÁXIMA de papel (de examen, timbrado) que puede emplear ESTÁ LIMITADA al equivalente a DOS (2) HOJAS de tamaño A4 (210 x 297 mm)

PRIMERA PARTE. PREGUNTAS TEÓRICAS (2'5 PUNTOS CADA UNA)

1. Razone qué criterios deben guiar la elección de un modelo de ciclo de vida.

Solución

Algunos de los criterios que deben guiar la selección de un modelo de ciclo de vida son los siguientes:

- Volatilidad de los requisitos. ¿Qué prestaciones ofrecen el modelo para reaccionar ante cambios en los requisitos?
- Incertidumbre asociada al proyecto debido a la falta de experiencia en proyectos similares, la dificultad de comprender el requisito, el uso de tecnologías novedosas...
- Facilidad que el modelo da a los gestores para controlar el progreso de los sistemas.
- Rapidez con que el usuario dispone de parte o de todo el sistema.

El modelo de ciclo de vida en cascada es el más antiguo y el más ampliamente utilizado. Debido a su simplicidad, el control del progreso de los sistemas es muy sencillo. Además, no carga el ciclo de vida con actividades que ralentizan la entrega del sistema total. Sin embargo, dada la dificultad de volver atrás, no responde eficazmente a cambios en los requisitos ni maneja de manera apropiada la incertidumbre.

El modelo evolutivo corrige la necesidad de una secuencia no lineal de pasos de desarrollo, gestiona eficazmente la volatilidad de los requisitos y la incertidumbre, y permite la rápida entrega parcial del sistema ("por fascículos").

Respecto al modelo evolutivo, en el modelo en espiral:

- Existe un reconocimiento explícito de las diferentes alternativas para alcanzar los objetivos de un proyecto.
- La identificación de riesgos asociados con cada una de las alternativas y las diferentes maneras de resolverlos son el centro del modelo. Con el modelo evolutivo es habitual dejar las partes más difíciles para el final y empezar con las más fáciles y de menor riesgo, obteniendo así la ilusión de un gran avance.

- La división de los proyectos en ciclos, cada uno con un acuerdo final de cada ciclo, implica que existe un acuerdo para los cambios que hay que realizar o para terminar el proyecto, en función de lo que se ha aprendido desde el inicio del proyecto.

Respecto al modelo de ciclo de vida en cascada, en los modelos evolutivos y en espiral se complica el control del progreso de los sistemas y se introducen nuevas actividades que pueden ralentizar la entrega total de sistemas sencillos.

2. Describa las estructuras de programa recomendadas por la metodología de *programación estructurada*. Explique la equivalencia existente, empleada en la *metodología de Jackson* (Diseño Dirigido por los Datos), entre las estructuras de programa y las estructuras de datos.

Solución

(Pág, 128, 162, 244 y 253).

Las estructuras de programa recomendadas por la metodología de programación estructurada son la **secuencia**, la **selección** y la **iteración**. Cualquier lenguaje de programación imperativo proporciona sentencias para facilitar la programación estructurada:

- secuencia: escritura consecutiva de sentencias
- selección: construcciones *if – then – else, case –of, etc.*
- Iteración: construcciones *while – do, repeat – until, for – to – do, loop y exit.*

La *metodología de Jackson* es un procedimiento de diseño sistemático de software a partir de las estructuras de los datos de entrada y salida del sistema. La equivalencia que existe entre las estructuras de programa anteriores y las estructuras de datos es la siguiente:

- secuencia – tupla (estructura de datos compuesta de elementos heterogéneos; la mayoría de los lenguajes disponen de la estructura registro o *record* para su implementación).
- selección – unión (también se usa el tipo registro para definir el esquema unión, que es una agrupación de elementos posibles de tipos diferentes; en el caso de Pascal y Modula-2 correspondería al registro con campos variantes).
- iteración – formación (colección de elementos del mismo tipo; la estructura que proporcionan los lenguajes se denomina formación, vector o *array*).

SEGUNDA PARTE. PREGUNTA DE TEORÍA APLICADA (MÁXIMO 5 PUNTOS)

3. Actualmente, las bicicletas de montaña están compuestas, entre otras piezas, por un manillar, un sistema de cambio, un sillín, una horquilla de suspensión, dos pedales, una cadena y un cuadro. La cadena está formada por un conjunto de eslabones y el cuadro, dependiendo de si la bicicleta es de “doble suspensión” o no, dispone de un amortiguador.

En la asignatura se estudian dos notaciones para modelar datos, que en ocasiones pueden considerarse equivalentes: los diccionarios de datos y los diagramas entidad relación. **Modele el enunciado anterior utilizando ambas notaciones. En el diccionario de datos omita el campo “Utilidad” y las descripciones de los componentes de la bicicleta (límitese a cumplimentar los campos “Nombre” y “Estructura”).**

Solución

a) Diccionario de datos:

Nombre: Bicicleta de montaña

Estructura: Manillar + Cambio + Sillín + Horquilla de suspensión + {Pedal}² + Cadena + [Cuadro rígido | Cuadro con suspensión]

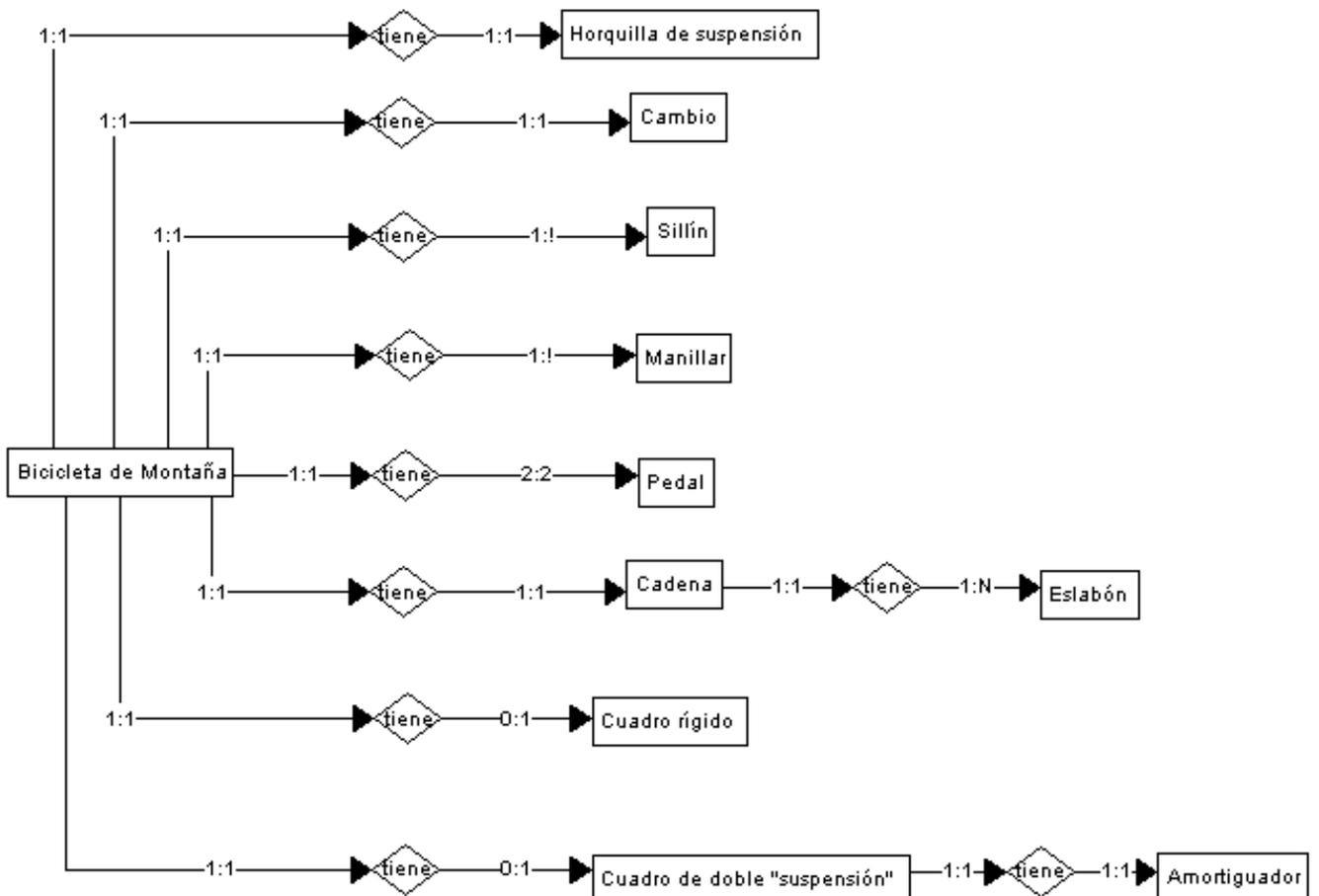
Nombre: Cadena

Estructura: {Eslabón}

Nombre: Cuadro con suspensión

Estructura: Amortiguador

b) Diagrama entidad relación:



INGENIERÍA DEL SOFTWARE (2º Curso) Cód. 53210 SISTEMAS 54208 GESTIÓN			
	Septiembre 2007 Original		
			Nacional y UE

ESTE EJERCICIO NO ES DE TEST. NO TIENE RETORNO TELEMÁTICO.

NECESITO EL EJERCICIO MANUSCRITO POR EL ALUMNO PARA PODER CALIFICAR.

SI ESTE EJERCICIO NO SE HA IMPRESO EN HOJA DE LECTURA ÓPTICA, SOLICITE UNA AL TRIBUNAL.

Todas las preguntas de este ejercicio son eliminatorias en el sentido de que debe obtener una nota mínima en cada una de ellas. En cada pregunta teórica, que se valora con 2'5 puntos, la nota mínima es 1 punto; en la segunda parte (ejercicio de teoría aplicada que se valora con 5 puntos) la nota mínima que debe obtener es de 2 puntos.

¡ATENCIÓN! PONGA SUS DATOS EN LA HOJA DE LECTURA ÓPTICA QUE DEBERÁ ENTREGAR JUNTO CON EL RESTO DE LAS RESPUESTAS.

Conteste a las preguntas teóricas, en cualquier orden, en hojas diferentes a las que utilice para la contestación de la segunda parte. En cada parte, la cantidad MÁXIMA de papel (de examen, timbrado) que puede emplear ESTÁ LIMITADA al equivalente a DOS (2) HOJAS de tamaño A4 (210 x 297 mm)

PRIMERA PARTE. PREGUNTAS TEÓRICAS (2'5 PUNTOS CADA UNA)

1. Frente a la fabricación 'intuitiva' o 'artesanal' de un producto software ¿qué ventajas tiene el uso de las técnicas de ingeniería del producto software, vistas en la asignatura (en cuanto al uso de un ciclo de vida, el análisis, el diseño, la codificación, cómo se hacen las pruebas, la integración, etc.), respecto al producto final obtenido y el proceso de su desarrollo?

Solución

- A. Ciclo de vida. Disponer de un modelo de referencia para el comportamiento del proceso de desarrollo que permite la visibilidad, el seguimiento y control de las actividades implicadas en la fabricación del producto.
- B. El Análisis. Permite alcanzar la comprensión de las necesidades del cliente, la situación, el objetivo, el comportamiento y la funcionalidad de lo que tendrá que ser la aplicación cuando se construya. El conjunto de actividades de esta fase establecen la definición del producto. Por tanto, la otra ventaja, además de la comprensión, es que se establecen, de manera explícita, un conjunto de especificaciones, más o menos formales, **imprescindibles** para el diseño, la codificación, validación y aceptación del producto.
- C. En el diseño se define un modelo teórico del funcionamiento del sistema. El análisis de estas especificaciones de funcionamiento permite deducir: defectos en la funcionalidad del sistema o de alguno de sus componentes; el grado de independencia entre módulos, su comprensibilidad y transportabilidad y, por tanto, se pueden estimar los posibles problemas de mantenibilidad, reutilización y codificación. El refinamiento del diseño reduce los problemas anteriores y facilita la verificación, la integración y las pruebas.
- D. En la codificación se realiza la construcción 'física' del producto. Cuando se hace teniendo en cuenta las pautas de la Ingeniería del Producto Software y basándose en la realización correcta de las fases anteriores, se garantizan mejores resultados en cuanto a los plazos de entrega, la mayoría de los aspectos de calidad del producto; detección, control y eliminación de defectos, facilidad de integración y reutilización futura del código.

- E. Un buen programa de pruebas permite detectar el máximo de defectos de parte del producto o del sistema completo. Esta detección, sobre todo si es precoz, permite disminuir los plazos de entrega, facilita la eliminación de defectos (calidad) y ahorra mucho trabajo. Las pruebas se pueden y deben realizar durante todas las fases del ciclo de vida. De esta manera se posibilita la visibilidad y el control del desarrollo del producto y se permite que los replanteamientos sean tempranos y eficaces. En cuanto a las pruebas del código, del funcionamiento de los componentes y del sistema; la identificación de los defectos y de su origen depende, en gran medida, de la calidad del diseño realizado y de que la ejecución de la codificación haya sido ordenada y controlada, bien documentada y se hayan realizado las *‘buenas prácticas’* recomendadas. El proceso de corrección, cambios y mantenimiento se facilitará tanto más cuanto se incorporen estas técnicas en el desarrollo.
- F. Integración y pruebas del sistema. Las ventajas mencionadas anteriormente para el diseño, la codificación y pruebas modulares, se extienden y aplican a la facilidad y éxito de la integración; tanto en el caso de grandes aplicaciones como en las de pequeño tamaño que requieran una integración mínima. En cualquier caso, los problemas que se puedan detectar serán de origen más fácilmente identificable con un diseño claro, una descomposición modular altamente independiente, un código *‘limpio’* y bien documentado que si este es farragoso o si el diseño tiene dependencias fuertes.
- G. La gestión de la configuración y, en concreto, el establecimiento y mantenimiento de la línea base, permite que todas las actividades del desarrollo se realicen en direcciones que son compatibles entre sí, aunando los esfuerzos en el sentido de la finalización del producto. Las especificaciones de la línea base se van estableciendo paulatinamente a lo largo de todo el ciclo de vida, en una sucesión de distintas líneas base de las que sólo una está vigente (congelada) en cada instante. Como las especificaciones de la línea base garantizan que los trabajos que se realicen son compatibles entre sí y con lo anteriormente desarrollado, se evita la dispersión de esfuerzos y el trabajo que supone la adecuación de una parte del desarrollo para que sea compatible con el resto.
2. Resuma en qué consiste el principio de ocultación y qué ventajas se derivan de su aplicación.

Solución

El principio de ocultación y sus ventajas se resumen en las páginas 113 y 114 de J. A. Cerrada Somolinos, et al. Introducción a la Ingeniería del Software. Ed. Ramón Areces, 2000. A continuación, se da una explicación complementaria a dicho resumen:

¿Qué es el principio de ocultación? Es un principio que promueve esconder todos los detalles que sean irrelevantes para utilizar un determinado artefacto.

Los lenguajes de programación ofrecen diversos medios para seguir este principio. Por ejemplo, los detalles de implementación de los subprogramas suelen ocultarse mediante cabeceras, que muestran exclusivamente cuales son los parámetros de entrada y salida de los subprogramas.

¿Qué ventajas se derivan de su aplicación? El principio de ocultación es “de propósito general”, ya que su aplicación va más allá del mero desarrollo de software. Por ejemplo, la electrónica de los equipos de música se esconde dentro de una caja que ofrece una escueta interfaz con los botones ON, OFF, PLAY... ¿Qué ocurriría si no fuera así y para escuchar un disco tuviéramos que configurar cables, transistores...?

1. La utilización del equipo exigiría años de estudio para conocer la electrónica del equipo.
2. Si el equipo se estropea y adquirimos un nuevo modelo, habría que volver a invertir tiempo en aprender qué cables, transistores... hay que configurar. Es decir, al estar “acoplados” a la tecnología del equipo, si ésta cambia, nos vemos forzados a cambiar nosotros también.
3. La prueba de los equipos se encarecería, ya que, además de expertos en acústica, forzosamente deberán participar expertos en electrónica.

Centrándonos en el desarrollo de software, el principio de ocultación facilita:

1. La reutilización de componentes desarrollados por terceros.

2. El mantenimiento del código que reutiliza componentes desarrollados por terceros. Mientras los componentes mantengan su interfaz, la evolución de los componentes no impondrá cambios en el código que los reutiliza.
3. La realización de pruebas de caja negra.

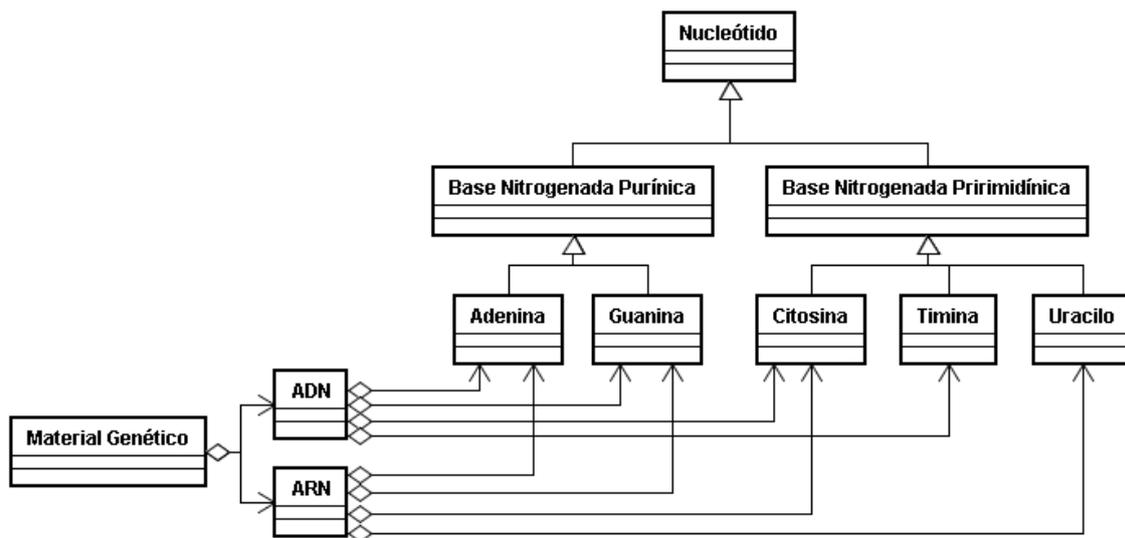
SEGUNDA PARTE. PREGUNTA DE TEORÍA APLICADA (MÁXIMO 5 PUNTOS)

3. Los principales componentes del material genético de los organismos son el ADN (Ácido DesoxirriboNucleico) y el ARN (Ácido RiboNucleico). El ADN está formado por cuatro tipos de nucleótidos, dos con bases nitrogenadas purínicas: la adenina y la guanina, y dos con bases nitrogenadas pirimidínicas: la citosina y la timina. El ARN está compuesto por los mismos tipos de nucleótidos, salvo que en lugar de timina contiene uracilo.

Modele el enunciado anterior mediante un Diagrama de Objetos.

Solución

Del análisis del enunciado se desprenden las relaciones de agregación y composición: **componentes** del 'material genético'; 'ADN' y 'ARN' **formado por...** Por otro lado, cualquiera de los compuestos que forman el materia genético **son** 'bases nitrogenadas', de un tipo u otro, y **son** 'nucleótidos'; de donde se deducen las relaciones de herencia.



INGENIERÍA DEL SOFTWARE (2º Curso) Cód. 53210 SISTEMAS 54208 GESTIÓN			
	Sept - 2007 Reserva		
			Nacional y UE

ESTE EJERCICIO NO ES DE TEST. NO TIENE RETORNO TELEMÁTICO.

NECESITO EL EJERCICIO MANUSCRITO POR EL ALUMNO PARA PODER CALIFICAR.

SI ESTE EJERCICIO NO SE HA IMPRESO EN HOJA DE LECTURA ÓPTICA, SOLICITE UNA AL TRIBUNAL.

Todas las preguntas de este ejercicio son eliminatorias en el sentido de que debe obtener una nota mínima en cada una de ellas. En cada pregunta teórica, que se valora con 2'5 puntos, la nota mínima es 1 punto; en la segunda parte (ejercicio de teoría aplicada que se valora con 5 puntos) la nota mínima que debe obtener es de 2 puntos.

¡ATENCIÓN! PONGA SUS DATOS EN LA HOJA DE LECTURA ÓPTICA QUE DEBERÁ ENTREGAR JUNTO CON EL RESTO DE LAS RESPUESTAS.

Conteste a las preguntas teóricas, en cualquier orden, en hojas diferentes a las que utilice para la contestación de la segunda parte. En cada parte, la cantidad MÁXIMA de papel (de examen, timbrado) que puede emplear ESTÁ LIMITADA al equivalente a DOS (2) HOJAS de tamaño A4 (210 x 297 mm)

PRIMERA PARTE. PREGUNTAS TEÓRICAS (2'5 PUNTOS CADA UNA)

1. Resuma gráficamente los modelos de ciclo de vida en V, con prototipos evolutivos y en espiral.

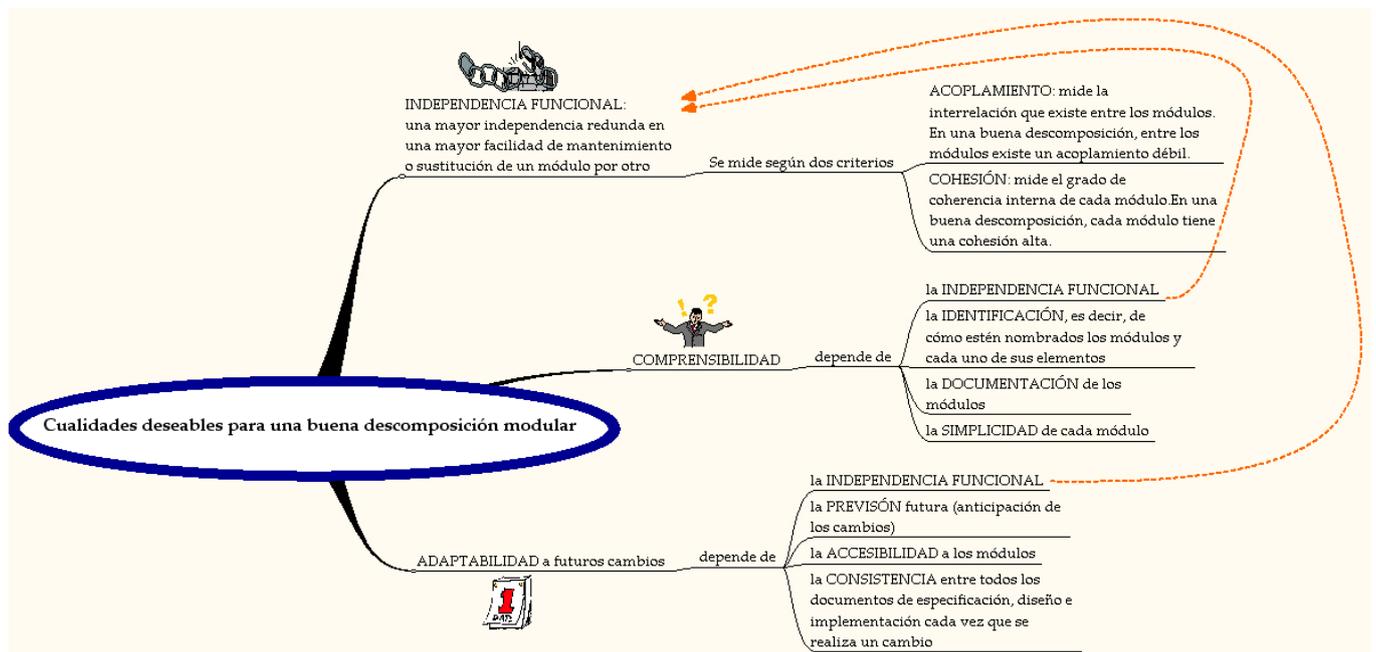
Solución

Figuras 1.3, 1.6 y 1.7 de J. A. Cerrada Somolinos, et al. *Introducción a la Ingeniería del Software*. Ed. Ramón Areces, 2000.

2. Resuma qué cualidades debe poseer una buena descomposición modular.

Solución

Las cualidades que debe poseer una buena descomposición modular se indican en el apartado 4.1 de J. A. Cerrada Somolinos, et al. *Introducción a la Ingeniería del Software*. Ed. Ramón Areces, 2000. A continuación, se incluye un esquema que resume dichas cualidades.



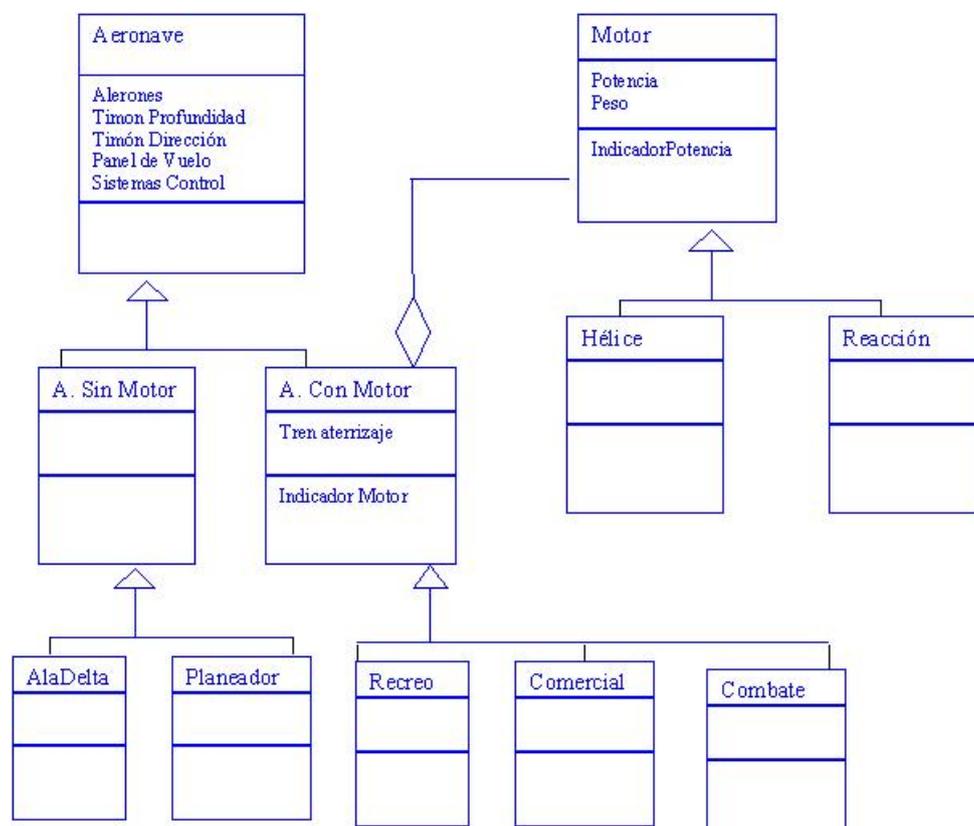
SEGUNDA PARTE. PREGUNTA DE TEORÍA APLICADA (MÁXIMO 5 PUNTOS)

3. En un juego de simulación aérea, el usuario puede elegir entre distintos tipos de aeronaves: ala-delta, planeador, ultraligero, avioneta de recreo, avión de combate, etc. Todas las aeronaves se rigen por el mismo principio básico de navegación aérea, sin embargo, cada uno ofrece además su propio comportamiento dependiendo si se trata de una aeronave con motor o sin él, si éste es de hélice o a reacción o, incluso, del número de motores de que dispone.

Diseñe un modelo de objetos que represente los distintos modelos de aeronave. Utilice herencia siempre que pueda y justifique sus pasos en la construcción del diseño. Intente añadir funcionalidad a las clases en forma de algunos métodos que se le ocurran.

Solución

Para poder identificar las clases que tendrá nuestro diseño, emplearemos las directivas de Abbot.



Dado que las características básicas de vuelo son comunes a todos los aviones, y atendiendo a criterios de abstracción y coherencia, parece natural diseñar una clase padre llamada *Aeronave* que recoja estas características. A partir de ella irán heredando las distintas clases de aeronaves que dispongamos. Antes, sin embargo, hacemos la distinción entre aeronaves de motor y sin motor ya que su comportamiento e instrumentación serán distintas; estas dos clases serán las *A. Sin Motor* y *A. Con Motor* desde las cuales heredarán todas las distintas clases de aeronaves que dispondrá nuestro juego.

Por otro lado, dado que el motor juega una parte importante en la navegación de las aeronaves, se ha diseñado una clase *Motor* que recogerá la funcionalidad básica que tendrán los dos tipos de motores: los de hélice y los de reacción.

INGENIERÍA DEL SOFTWARE (2º Curso) Cód. 53210 SISTEMAS 54208 GESTIÓN			
	Sept - 2007 Original		
			C. Penitenciarios

ESTE EJERCICIO NO ES DE TEST. NO TIENE RETORNO TELEMÁTICO.

NECESITO EL EJERCICIO MANUSCRITO POR EL ALUMNO PARA PODER CALIFICAR.

SI ESTE EJERCICIO NO SE HA IMPRESO EN HOJA DE LECTURA ÓPTICA, SOLICITE UNA AL TRIBUNAL.

Todas las preguntas de este ejercicio son eliminatorias en el sentido de que debe obtener una nota mínima en cada una de ellas. En cada pregunta teórica, que se valora con 2'5 puntos, la nota mínima es 1 punto; en la segunda parte (ejercicio de teoría aplicada que se valora con 5 puntos) la nota mínima que debe obtener es de 2 puntos.

¡ATENCIÓN! PONGA SUS DATOS EN LA HOJA DE LECTURA ÓPTICA QUE DEBERÁ ENTREGAR JUNTO CON EL RESTO DE LAS RESPUESTAS.

Conteste a las preguntas teóricas, en cualquier orden, en hojas diferentes a las que utilice para la contestación de la segunda parte. En cada parte, la cantidad MÁXIMA de papel (de examen, timbrado) que puede emplear ESTÁ LIMITADA al equivalente a DOS (2) HOJAS de tamaño A4 (210 x 297 mm)

PRIMERA PARTE. PREGUNTAS TEÓRICAS (2'5 PUNTOS CADA UNA)

- En el texto base de la asignatura, al presentar los distintos modelos de ciclo de vida, se ilustran los conceptos de '*verificación*' y '*validación*' solamente al hablar del ciclo de vida en V (ciclo de vida que es idéntico al de cascada –clásico como él- y, lo único que aporta es el ámbito en el que se desarrollan las actividades de cada fase).
 - Diferencie estos dos conceptos entre sí.
 - Razone si la verificación y la validación son actividades sólo del ciclo de vida en V o, por el contrario, dichas actividades tienen que aparecer en cualquier modelo de ciclo de vida.

Solución

- Verificación es la comprobación de que cada parte, módulo, componente o elemento codificado funciona y se comporta correctamente según lo especificado en el diseño.
Por el contrario, la validación es la comprobación de que el sistema, ya integrado, cumple las expectativas del cliente que se hayan reflejado en la documentación del análisis. Dicho de otra manera, en la validación se comprueba que las especificaciones que se han acordado con el cliente y que se han reflejado en la documentación del análisis, se ejecutan satisfactoriamente.
- Por el propio contenido de los dos conceptos: comprobar que cada parte del código se comporta como se esperaba y constatar que se cumplen las necesidades acordadas con el cliente; estas actividades forman parte, necesariamente, de una u otra fase de todos los ciclos de vida. Es inaceptable que no se compruebe si el código funciona o si el producto desarrollado es el que quería el cliente.

2. Explique brevemente las dos estrategias de pruebas de módulos estudiadas en la asignatura.

Solución

Para evitar el caos que supondría el tener que hacer una prueba global única a todo el sistema, antes de la integración, se deberán hacer pruebas a cada módulo de diseño conforme avance la codificación.

Existen dos estrategias fundamentales de prueba de módulos o unidades:

- Pruebas de caja negra
- Pruebas de caja transparente

Las **pruebas de caja negra** se basan exclusivamente en la comprobación de las especificaciones de entrada y salida del módulo. Son pruebas que ignoran la estructura interna del módulo, ya que solo lo 'ven' como una caja negra donde a una determinada entrada deberá corresponder una salida perfectamente determinada.

El objetivo de las pruebas no es 'probar' de forma exhaustiva todas las distintas posibles entradas, sino descubrir errores o incorrecciones del módulo. Por eso, la elección de un buen conjunto de casos de prueba, conseguirá detectar el máximo número de errores con el menor número de pruebas. Estos casos de prueba se pueden escoger mediante distintos métodos, entre ellos podemos destacar:

- Partición de clases
- Análisis de valores medios
- Comparación de versiones
- Empleando la intuición y experiencia

Las **pruebas de caja transparente** tienen como objetivo 'determinar' la salida de un módulo ante una entrada pero teniendo en cuenta la estructura interna del módulo, esto es, se trata de conseguir que la ejecución del módulo transite por todos los posibles caminos de ejecución y ponga en juego todos los elementos del código. Para conseguir casos de pruebas eficaces podemos aplicar las siguientes estrategias:

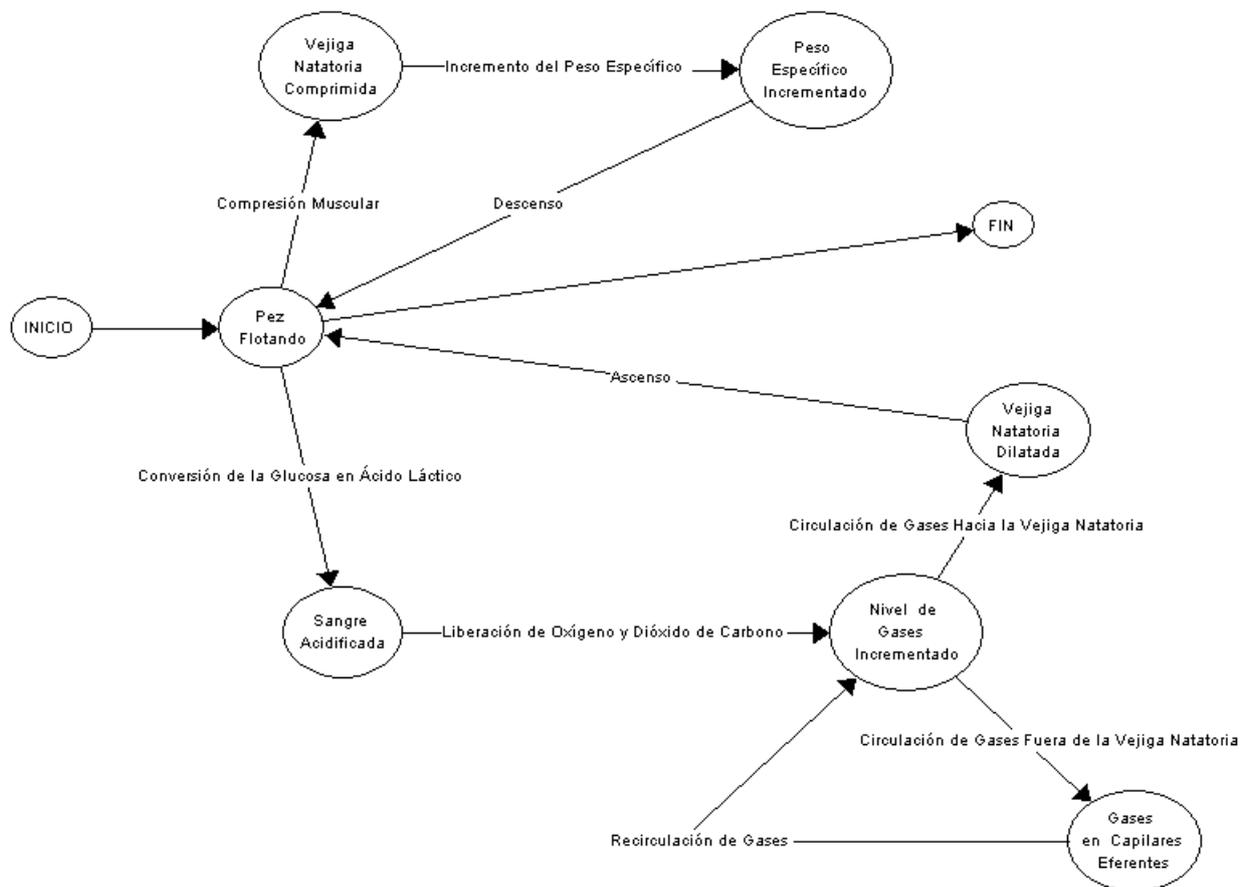
- Cubrimiento lógico
- Pruebas de bucle
- Empleo de la intuición

SEGUNDA PARTE. PREGUNTA DE TEORÍA APLICADA (MÁXIMO 5 PUNTOS)

3. La vejiga natatoria es un órgano de flotación que poseen muchos peces. Se trata de una bolsa de paredes flexibles, llena de gas, situada dorsalmente por debajo de la columna vertebral y por encima del tubo digestivo. Controla la flotabilidad mediante un complejo sistema de intercambio gaseoso con la sangre, y permite al pez ascender o descender en el agua. Cuando expulsa gases por compresión muscular, aumenta el peso específico, facilitando el descenso en el agua. Si se llena de gases, favorece el ascenso hacia la superficie. El proceso de secreción gaseosa se basa en la acidificación de la sangre producida al convertirse la glucosa en ácido láctico. Con ello se libera en la sangre oxígeno de la oxihemoglobina y dióxido de carbono del bicarbonato sódico, que se difunden en la vejiga natatoria. Los gases que no llegan inmediatamente hasta la vejiga natatoria pasan a los capilares eferentes y circulan de nuevo.

Modele el movimiento ascendente y descendente de los peces, enunciado en el párrafo anterior, mediante un Diagrama de Transición de Estados.

Solución



INGENIERÍA DEL SOFTWARE (2º Curso) Cód. 53210 SISTEMAS 54208 GESTIÓN			
	Sept - 2007 Original		
		Soto del Real	

ESTE EJERCICIO NO ES DE TEST. NO TIENE RETORNO TELEMÁTICO.

NECESITO EL EJERCICIO MANUSCRITO POR EL ALUMNO PARA PODER CALIFICAR.

SI ESTE EJERCICIO NO SE HA IMPRESO EN HOJA DE LECTURA ÓPTICA, SOLICITE UNA AL TRIBUNAL.

Todas las preguntas de este ejercicio son eliminatorias en el sentido de que debe obtener una nota mínima en cada una de ellas. En cada pregunta teórica, que se valora con 2'5 puntos, la nota mínima es 1 punto; en la segunda parte (ejercicio de teoría aplicada que se valora con 5 puntos) la nota mínima que debe obtener es de 2 puntos.

¡ATENCIÓN! PONGA SUS DATOS EN LA HOJA DE LECTURA ÓPTICA QUE DEBERÁ ENTREGAR JUNTO CON EL RESTO DE LAS RESPUESTAS.

Conteste a las preguntas teóricas, en cualquier orden, en hojas diferentes a las que utilice para la contestación de la segunda parte. En cada parte, la cantidad MÁXIMA de papel (de examen, timbrado) que puede emplear ESTÁ LIMITADA al equivalente a DOS (2) HOJAS de tamaño A4 (210 x 297 mm)

PRIMERA PARTE. PREGUNTAS TEÓRICAS (2'5 PUNTOS CADA UNA)

1. Describa cinco factores de calidad del software.

Solución

Páginas 27 y 28 de J. A. Cerrada Somolinos, et al. Introducción a la Ingeniería del Software. Ed. Ramón Areces, 2000.

2. ¿Para qué sirven las pruebas de sistema?

Solución

El principal objetivo de las pruebas es conseguir que el programa funcione incorrectamente, esto es, las pruebas deben ser capaces de encontrar posibles errores de software.

En particular se llaman pruebas de sistema a aquellas que se realizan sobre todo el sistema software una vez terminado la integración del mismo. Es este escenario, es cuando se realizan pruebas para evaluar criterios de calidad. Se pueden agrupar en las siguientes clases según el objetivo perseguido:

- Pruebas de recuperación.
- Pruebas de seguridad.
- Pruebas de resistencia.
- Pruebas de rendimiento.

Por último, para ver que el sistema es realmente útil al usuario final, se realizan las pruebas alfa y beta.

Véase la sección 5.9 del libro de referencia (pág. 289 y 290).

SEGUNDA PARTE. PREGUNTA DE TEORÍA APLICADA (MÁXIMO 5 PUNTOS)

3. Una compañía editorial está dividida en 4 departamentos: edición, maquetación, fabricación y administración. Los trabajadores se dividen entre: directivos, técnicos y operarios. Un proyecto editorial implica a todos los departamentos y tiene un directivo como responsable. Los demás empleados están adscritos, al menos, a un proyecto. El jefe de un departamento es un directivo.

Modele el sistema anterior utilizando orientación a objetos.

Solución

