GUIA DE PREGUNTAS 5

Material "CONCEPTOS Y PRINCIPIOS DE DISEÑO. Ingeniería del Software (Capítulo 13) de Roger S. Pressman (5ta Edición)"

Contenido

1. Enuncie y defina en que consiste cada tipo de diseño.

- **Diseño De Datos:** transforma el modelo del dominio de información que se crea durante el análisis en las estructuras de datos que se necesitarán para implementar el software. Los objetos de datos y las relaciones definidas en el diagrama relación entidad y el contenido de datos detallado que se representa en el diccionario de datos proporcionan la base de la actividad del diseño de datos. Es posible que parte del diseño de datos tenga lugar junto con el diseño de la arquitectura del software. A medida que se van diseñando cada uno de los componentes del software, van apareciendo más detalles de diseño
- **Diseño Arquitectónico:** define la relación entre los elementos estructurales principales del software, los patrones de diseño que se pueden utilizar para lograr los requisitos que se han definido para el sistema, y las restricciones que afectan a la manera en que se pueden aplicar los patrones de diseño arquitectónicos La representación del diseño arquitectónico -el marco de trabajo de un sistema basado en computadora-puede derivarse de la especificación del sistema, del modelo de análisis y de la interacción del subsistema definido dentro del modelo de análisis.
- **Diseño De La Interfaz:** describe la manera de comunicarse el software dentro de sí mismo, con sistemas que interoperan dentro de él y con las personas que lo utilizan. Una interfaz implica un flujo de información (por ejemplo, datos control) y un tipo específico de comportamiento. Por tanto, los diagramas de flujo de control y de datos proporcionan gran parte de la información que se requiere para el diseño de la interfaz
- **Diseño A Nivel De Componentes:** transforma los elementos estructurales de la arquitectura del software en una descripción procedimental de los componentes del software. La información que se obtiene de EP, EC y de DTE sirve como base para el diseño de los componentes

2. Fundamente la importancia del diseño del software

La importancia del diseño del software se puede describir con una sola palabra -calidad-.

El diseño es la única forma de convertir exactamente los requisitos de un cliente en un producto o sistema de software finalizado. El diseño del software sirve como fundamento para todos los pasos siguientes del soporte del software y de la ingeniería del software.

3. Justifique la concepción iterativa del proceso de diseño de software

El diseño del software es un proceso iterativo mediante el cual los requisitos se traducen

en un plano para construir el software. Inicialmente, el plano representa una visión holística

del software

4. Enuncie y describa las características que sirven como guía para la evaluación de un buen diseño de software

- el diseño deberá implementar todos los requisitos explícitos del modelo de análisis, y deberán ajustarse a todos los requisitos implícitos que desea el cliente

- el diseño deberá ser una guía legible y comprensible para aquellos que generan código y para aquellos que comprueban consecuentemente, dan soporte al software
- el diseño deberá proporcionar una imagen completa del software, enfrentándose a los dominios de comportamiento, funcionales y de datos desde una perspectiva de implementación

5. Enuncie las directrices para evaluar la calidad de una representación de diseño

1. Un diseño deberá presentar una estructura arquitectónica que
    a. se haya creado mediante patrones de diseño reconocibles
    b. este formada por componentes que exhiban características de buen diseño
    c. que puedan implementarse de manera evolutiva facilitando la implementación y comprobación

2. Un diseño deberá ser modular; esto es, el software deberá dividirse lógicamente en elementos que realicen funciones y subfunciones específicas.

3. Un diseño deberá contener distintas representaciones de datos, arquitectura, interfaces y componentes (módulos).

4. Un diseño deberá conducir a estructuras de datos adecuadas para los objetos que se van implementar y que procedan de patrones de datos reconocibles
5. Un diseño deberá conducir a componente que presenten características funcionales independientes
6. Un diseño deberá conducir a interfaces que reduzcan la complejidad de las conexiones entre los módulos y con el entorno externo.
7. Un diseño deberá derivarse mediante un método repetitivo y controlado por la información obtenida durante el análisis de los requisitos de software.

6. Justifique porque el diseño de software es tanto un proceso como un modelo

- **El Proceso De Diseño**: es una secuencia de pasos que hacen posible que el diseñador describa todos los aspectos del software que se va construir.
- **El Modelo De Diseño:** es el equivalente a los planes de un arquitecto para una casa. Comienza representando la totalidad de todo lo que se va a construir (por ejemplo, una representación en tres dimensiones de la casa) y refina lentamente lo que va a proporcionar la guía para construir cada detalle (por ejemplo, el diseño de fontanería).

7. Enuncie y describa los principios básicos de diseño del software

- Un buen diseñador deberá tener en cuenta enfoques alternativos, juzgando todos los que se basan en los requisitos del problema.
- El diseño deberá poderse rastrear hasta el modelo de análisis.
- El diseño no deberá inventar nada que ya esté inventado. Los sistemas se construyen utilizando un conjunto de patrones de diseño, muchos de los cuales probablemente ya se han encontrado antes.
- El diseño deberá «minimizar la distancia intelectual» entre el software y el problema como si de la vida real se tratara
- El diseño deberá presentar uniformidad e integración. Un diseño es uniforme si parece que fue una persona la que lo desarrolló por completo.
- El diseño deberá estructurarse para admitir cambios. Los conceptos de diseño estudiados en la sección siguiente hacen posible un diseño que logra este principio.
- El diseño deberá estructurarse para degradarse poco a poco, incluso cuando se enfrenta con datos, sucesos o condiciones de operación aberrantes.
- El diseño no es escribir código y escribir código no es diseñar. Incluso cuando se crean diseños procedimentales para componentes de programas, el nivel de abstracción del modelo de diseño es mayor que el código fuente.
- El diseño deberá evaluarse en función de la calidad mientras se va creando, no después de terminarlo.
- El diseño deberá revisar separa minimizarlos errores conceptuales (semánticos).A veces existe la tendencia de centrarse en cuando se revisa el diseño, olvidándose del bosque por culpa de los árboles.

8. Defina abstracción procedimental, abstracción de datos y abstracción de control

La noción psicológica de «abstracción» permite concentrarse en un problema a algún nivel de generalización sin tener en consideración los datos irrelevantes de bajo nivel; la utilización de la

abstracción también permite trabajar con conceptos y términos que son familiares en el entorno del problema sin tener que transformarlos en una estructura no familiar...

- **Abstracción Procedimental:** tal es una secuencia nombrada de instrucciones que tiene una función específica y limitada. Un ejemplo de abstracción procedimental sería la palabra para una puerta. «Abrir» implica una secuencia larga de pasos procedimentales (llegar a la puerta; alcanzar y agarrar el pomo de la puerta; girar el pomo y tirar de la puerta; separarse al mover la puerta, etc.)

- **Abstracción De Datos:** es una colección nombrada de datos que describe un objeto de datos. En el contexto de la abstracción procedimental abrir, podemos definir una abstracción de datos llamada puerta. a a un conjunto de atributos que describen esta puerta (por ejemplo, tipo de puerta, dirección de apertura, mecanismo de apertura, peso, dimensiones)

- **Abstracción De Control:** implica un mecanismo de control de programa sin especificar los datos internos. Un ejemplo de abstracción de control es el semáforo de sincronización que se utiliza para coordinar las actividades en un sistema operativo.

9. Defina Refinamiento paso a paso

El refinamiento paso a paso es una estrategia de diseño. El desarrollo de un programa se realiza refinando sucesivamente los niveles de detalle procedimentales. Una jerarquía se desarrolla descomponiendo una sentencia macroscópica de función.

10. Defina modularidad

La arquitectura de expresa el modularidad; es decir, el software se divide en componentes nombrados y abordados por separado, llamados frecuentemente módulos, que se integran para satisfacer los requisitos del problema.

11. Fundamente porque subdividir un problema de software indefinidamente no minimiza el esfuerzo de desarrollo.

Sin embargo, a medida que aumenta el número de módulos, también crece el esfuerzo (coste)asociado con la integración de módulos. Estas conducen también a la curva total del coste o esfuerzo que se muestra en la figura. Existe un número M de módulos que daría como resultado un coste mínimo de desarrollo, aunque no tenemos la sofisticación necesaria para predecir con seguridad.

12. Defina capacidad de descomposición modular, capacidad de empleo de componentes modulares capacidad de comprensión modular, continuidad modular y protección modular

- **Capacidad de descomposición modular.** Si un método de diseño proporciona un mecanismo sistemático para descomponer el problema en subproblemas, reducirá la

complejidad de todo el problema, consiguiendo de esta manera una solución modular efectiva.

- **Capacidad de empleo de componentes modular:** Si un método de diseño permite ensamblar los componentes de diseño (reusables) existentes en un sistema nuevo, producirá una solución modular que no inventa nada ya inventado.
- **Capacidad de comprensión modular:** Si un módulo se puede comprender como una unidad Autónoma (sin referencias a otros módulos) será más fácil de construir y de cambiar.
- **Continuidad modular**: Si pequeños cambios en los requisitos del sistema provocan cambios en los módulos individuales, en vez de cambios generalizados en el sistema, se minimizará el impacto de los efectos secundarios de los cambios.
- **Protección modular**. Si dentro de un módulo se produce una condición aberrante y sus efectos se limitan a ese módulo, se minimizará el impacto de los efectos secundarios inducidos por los errores.

13. Defina arquitectura del software

En su forma más simple, la arquitectura es la estructura jerárquica de los componentes del programa (módulos), la manera en que los componentes interactúan y la estructura de datos que van a utilizar los componentes

14. Enuncie propiedades que deben especificarse como parte del diseño arquitectónico

- **Propiedades estructurales:** Este aspecto de la representación del diseño arquitectónico define los componentes de un sistema (por ejemplo, módulos, objetos, filtros) y la manera en que esos componentes se empaquetan e interactúan unos con otros. Por ejemplo, los objetos se empaquetan para encapsular tanto los datos como el procesamiento que manipula los datos e interactúan mediante la invocación de métodos
- **Propiedades extra-funcionales:** La descripción del diseño arquitectónico deberá ocuparse de cómo la arquitectura de diseño consigue los requisitos para el rendimiento, capacidad, fiabilidad,seguridad,capacidad de adaptación y otras características del sistema.

15. Defina [a] jerarquía de control, [b] arquitecturas de llamada y de retorno [c] niveles de control (profundidad y anchura), [d] ámbito de control, [e] grado de entrada, [f] grado de salida, [g] módulo superior y [h] módulo subordinado

A**)** **Jerarquía De Control** denominada también estructura de programa, representa la organización de los componentes de programa (módulos)e implica una jerarquía de control. No representa los aspectos procedimentales del software, ni se puede aplicar necesariamente a todos los estilos arquitectónicos.

B) **Arquitecturas De Llamada** Para representar la jerarquía control de aquellos estilos arquitectónicos que se avienen a la representación se utiliza un conjunto de notaciones diferentes. El diagrama más común es el de forma de árbol.

C) **Niveles de control la profundidad y la anchura** proporcionan una indicación de la cantidad de niveles de control y el ámbito de control global, respectivamente

D) **El ámbito de control** del módulo se compone de todos los módulos subordinados y superiores al módulo.

E) **El grado de entrada** indica la cantidad de módulos que controlan directamente un módulo dado.

F) **El grado de salida** es una medida del número de módulos que se controlan directamente con otro módulo

G) **Módulo superior** se dice que un módulo que controla otro módulo superior a el

H) **módulo subordinado** se dice que un módulo controlado por otro módulo es subordinado del controlador

16. Defina visibilidad y conectividad

- **Visibilidad** indica el conjunto de componentes de programa que un componente dado puede invocar o utilizar como datos, incluso cuando se lleva a cabo indirectamente. Por ejemplo, un módulo en un sistema orientado a objetos puede acceder al amplio abanico de objetos de datos que haya heredado, ahora bien, solo utiliza una pequeña cantidad de estos objetos de datos. Todos los objetos son visibles para el módulo
- **Conectividad** indica el conjunto de componentes que un componente dado invoca o utiliza directamente como datos. Por ejemplo, un módulo que hace directamente que otro módulo empiece la ejecución está conectado a el

17. Defina división estructural (división horizontal y división vertical a factorización)

División horizontal de la arquitectura proporciona diferentes ventajas: proporciona software más fácil de probar conduce a un software más fácil de mantener propaga menos efectos secundarios proporciona software más fácil de ampliar.

División vertical sugiere que dentro de la estructura de programa el control y el trabajo se distribuyan de manera descendente. Los módulos del nivel superior deberán llevar a cabo funciones de control y no realizarán mucho trabajo de procesamiento. Los módulos que residen en la parte inferior de la estructura deberán ser los trabajadores, aquellos que realizan todas las tareas de entrada, proceso y salida.

18. Enuncie ventajas y desventajas de la división estructural horizontal

La división horizontal suele hacer que los datos pasen a través de interfaces de módulos que puedan complicar el control global del flujo del programa. En las estructuras con división vertical son menos susceptibles a los efectos secundarios cuando se producen cambios y por tanto se podrán mantener mejor

19. Justifique la preferencia por la división estructural vertical

La división vertical frecuentemente llamada sugiere que dentro de la estructura de programa el control (toma de decisiones) y el trabajo se distribuyan de manera descendente. Los módulos del nivel superior deberán llevar a cabo funciones de control y no realizarán mucho trabajo de procesamiento.

20. Defina estructura de datos, elemento escalar, vector, espacio n-dimensional, lista

Estructura de datos es una representación de la relación lógica entre elementos individuales de datos. Cómo la estructura de la información afectará invariablemente al diseño procedimental final, la estructura de datos es tan importante como la estructura de programa para la representación de la arquitectura del software.

Elemento escalar representa un solo elemento de información que puede ser tratado por un identificador, es decir, se puede lograr acceso especificando una sola dirección en memoria. El tamaño y formato de un elemento escalar puede variar dentro de los límites que dicta el lenguaje de programación

Vector son las estructuras de datos más comunes y abren la puerta a la indexación variable de la información.

Espacio n-dimensional Cuando el vector secuencia1 se amplía a dos, tres y por último a un número arbitrario de dimensiones, se crea un espacio n-dimensional. El espacio n-dimensional más común es la matriz bidimensional. En muchos lenguajes de programación, un espacio n-dimensional se llama array.

Lista enlazada es una estructura de datos que organiza elementos escalares no contiguos, vectores o espacios de manera que les permita ser procesados como una lista.

21. Defina procedimiento de software

Procedimiento de software se centra en el procesamiento de cada módulo individualmente. El procedimiento debe proporcionar una especificación precisa de procesamiento, incluyendo la secuencia de sucesos, los puntos de decisión exactos, las operaciones repetitivas e incluso la estructura/organización de datos

22. Defina ocultación de información

El principio de ocultación de información sugiere que los módulos se caracterizan por las decisiones de diseño que (cada uno) oculta al otro. En otras palabras, los módulos deberán especificarse y diseñarse de manera que la información (procedimiento y datos) que está dentro de un módulo sea inaccesible a otros módulos que no necesiten esa información

23. Defina independencia funcional

El concepto de independencia funcional es la suma del modularidad y de los conceptos de abstracción y ocultación de información. En referencias obligadas sobre el diseño del software

24. Defina cohesión

La cohesión es una extensión natural del concepto de ocultación de información descrito en la Sección. Un módulo cohesivo lleva a cabo una sola tarea dentro de un procedimiento de software

25. Defina modulo incidentalmente, cohesivo lógicamente cohesivo, temporalmente cohesivo

Modulo incidentalmente es un módulo que lleva a cabo un conjunto de tareas que se relacionan con otras débilmente, si es que tienen algo que ver

Lógicamente cohesivo Un módulo que realiza tareas relacionadas lógicamente (por ejemplo, un módulo que produce todas las salidas independientemente del tipo)

Temporalmente cohesivo cuando un módulo contiene tareas que están relacionadas entre sí por el hecho de que todas deben ser ejecutadas en el mismo intervalo de tiempo.

26. Defina acoplamiento y ejemplifique tipos de acoplamiento

El acoplamiento es una medida de interconexión entre módulos dentro de una estructura de software. El acoplamiento depende de la complejidad de interconexión entre los módulos, el punto donde se realiza una entrada o referencia a un módulo, y los datos que pasan a través de la interfaz.

Acoplamiento de control es muy común en la mayoría de los diseños de software y se muestra en la en donde un indicador de control (una variable que controla las decisiones en un módulo superior o subordinado) se pasa entre los módulos d y e.

Acoplamiento externo es esencial, pero deberá limitarse a unos pocos módulos en una estructura. También aparece un acoplamiento alto cuando varios módulos hacen referencia a un área global de datos.

Acoplamiento común Los módulos c, g y k acceden a elementos de datos en un área de datos global (por ejemplo, un archivo de disco o un área de memoria totalmente accesible). El módulo c inicializa el elemento.

Acoplamiento de contenido, se da cuando un módulo hace uso de datos o de información de control mantenidos dentro de los límites de otro módulo. En segundo lugar, el acoplamiento de contenido ocurre cuando se realizan bifurcaciones a mitad de módulo. Este modo de acoplamiento puede y deberá evitarse.

27. Enuncie y describa heurísticas de diseño

Evaluar la primera iteración de la estructura de programa para reducir al acoplamiento y mejorar la cohesión. Un módulo explosionado se convierte en dos módulos o más en la estructura final de programa. Un módulo implosionado es el resultado de combinar el proceso implicado en dos o más módulos.

28. Defina especificación de diseño y enuncie que elementos contiene

La Especificación del diseño aborda diferentes aspectos del modelo de diseño y se completa a medida que el diseñador refina su propia representación del software.

En primer lugar, se describe el ámbito global del esfuerzo realizado en el diseño. La mayor parte de la información que se presenta aquí se deriva de la Especificación del sistema y del modelo de análisis.

Especificación del diseño contiene una referencia cruzada de requisitos. El propósito de esta referencia cruzada (normalmente representada como una matriz simple) es:

(1) establecer que todos los requisitos se satisfagan mediante el diseño del software, y

(2) indicar cuales son los componentes críticos para la implementación de requisitos específicos.