¿Qué debemos considerar al diseñar software de calidad?

Linda María Fernández Olvera
UPIICSA-IPN
Ingeniería de Diseño
lfernandezo1500@alumno.ipn.mx

Abstract – Este ensayo plantea una respuesta a las consideraciones necesarias para diseñar software de calidad, diferentes elementos que se toman en cuenta para un correcto diseño además de repasar modelos de diseño.

Índice de Términos – diseño, modelo, estética, interfaz, calidad, arquitectura, abstracción.

I. INTRODUCCIÓN

Los principios de diseño establecen una filosofía que guía el trabajo de diseño que debe ser ejecutado. El diseño de software agrupa principios, conceptos y prácticas que tienen como objetivo desarrollar un sistema o producto de calidad a partir de un modelo o representación que tenga resistencia, funcionalidad y estética, para conseguirlo es necesario practicar la diversificación y luego la convergencia.

La creatividad es un factor importante en el diseño de software, ya que, a diferencia del modelo de requerimientos, el modelo de diseño proporción detalles sobre arquitectura del software, estructura de datos, interfaces y componentes que se necesitan para implementar el sistema. El diseño es el lugar en el que se establece la calidad del software.

A continuación, en el presente ensayó describiré algunos conceptos del diseño de software, así como los pasos del proceso para obtener un diseño de calidad que satisfaga las necesidades del cliente y cumpla con la usabilidad y experiencia de usuario deseada.

II. DESARROLLO

La diversificación es la adquisición de un repertorio de alternativas, materia prima del diseño como lo son los componentes, soluciones con los componentes y conocimiento. Una vez reunidos estos conjuntos de información se seleccionan los elementos que cumplen con los requerimientos definidos por la ingeniería y por el modelo de análisis, se evalúan las alternativas que pueden ser rechazadas o no y se converge en una configuración particular de componentes y con ello la creación del producto final.

La diversificación y la convergencia combinan la intuición y criterio del ingeniero y con base en su experiencia un conjunto de principios guía la forma en que evoluciona el modelo, el conjunto de criterios que evalúan la calidad y el proceso iterativo que conduce a una representación del diseño definido.

El diseño de software evoluciona constantemente con nuevos metodología es por eso por lo que carecen de profundidad, flexibilidad y naturaleza cuantitativa, no obstante, existen métodos para diseñar software con criterios para el diseño de calidad y notación de diseño. Por lo que es importante identificar y estudiar los conceptos y principios fundamentales aplicables.

A. Diseño en el contexto de la ingeniería de software

El diseño del software comienza una vez que se han analizado y modelado los requerimientos, es la última acción de la ingeniería de software dentro de la actividad de modelado y prepara la etapa de construcción.

El trabajo de diseño es alimentado por el modelo de requerimientos, manifestado por elementos basados en el escenario, en la clase, orientados al flujo, y del comportamiento.

El objetivo principal del diseño de software es la calidad, sin diseño se corre el riesgo de obtener un sistema inestable, que falle cuando se hagan cambios pequeños, o uno que sea difícil de someter a prueba, o en el que no sea posible evaluar la calidad hasta que sea demasiado tarde en el proceso de software, cuando no queda mucho tiempo y ya se ha gastado mucho dinero.

B. El proceso de Diseño

Se trata de un proceso iterativo por medio del cual se traducen los requerimientos en un “plano” para construir el software. El diseño se representa en un nivel alto de abstracción, en el que se rastrea directamente el objetivo específico del sistema y los requerimientos más detallados de datos,

funcionamiento y comportamiento. A medida que tienen

lugar las iteraciones del diseño, las mejoras posteriores conducen a niveles menores de abstracción.

A través del proceso de diseño se evalúa la calidad de éste de acuerdo con la serie de revisiones técnicas que deben cumplir con tres características importantes:

- Debe implementar todos los requerimientos explícitos contenidos en el modelo de requerimientos y dar cabida a todos los requerimientos implícitos.
- Debe ser una guía legible y comprensible para quienes generan el código y para los que lo prueban y dan el apoyo posterior.
- Debe proporcionar el panorama completo del software, y abordar los dominios de los datos, las funciones y el comportamiento desde el punto de vista de la implementación.

Lineamientos de Calidad

A fin de evaluar la calidad de una representación del diseño, se debe establecer los criterios técnicos de un buen diseño, estos lineamientos se cumplen con la aplicación de los principios de diseño y son los siguientes:

1. Debe tener una arquitectura que 1) se haya creado con el empleo de estilos o patrones arquitectónicos reconocibles, 2) esté compuesta de componentes con buenas de diseño (éstas se analizan más adelante, en este capítulo), y 3) se implementen en forma evolutiva, de modo que faciliten la implementación y las pruebas.
2. Debe ser modular, es decir, el software debe estar dividido de manera lógica en elementos o subsistemas.
3. Debe contener distintas representaciones de datos, arquitectura, interfaces y componentes.
4. Debe conducir a estructuras de datos apropiadas para las clases que se van a implementar y que surjan de patrones reconocibles de datos.
5. Debe llevar a componentes que tengan características funcionales independientes.
6. Debe conducir a interfaces que reduzcan la complejidad de las conexiones entre los componentes y el ambiente externo.
7. Debe obtenerse con el empleo de un método repetible motivado por la información obtenida durante el análisis de los requerimientos del software.
8. Debe representarse con una notación que comunique con eficacia su significado.

Atributos de calidad

Hewlett-Packard desarrolló un conjunto de atributos de la calidad del software a los que se dio el acrónimo FURPS: funcionalidad, usabilidad, confiabilidad, rendimiento y mantenibilidad, que representan los objetivos del software que se desarrolla.

La funcionalidad se califica según el conjunto de características y capacidades del programa, la generalidad de las funciones que se entregan y la seguridad general del sistema.

La usabilidad se evalúa tomando en cuenta factores humanos, la estética general, la consistencia y la documentación.

La confiabilidad se evalúa con la medición de la frecuencia y gravedad de las fallas, la exactitud de los resultados que salen, el tiempo medio para que ocurra una falla (TMPF), la capacidad de recuperación ante ésta y lo predecible del programa.

• El rendimiento se mide con base en la velocidad de procesamiento, el tiempo de respuesta, el uso de recursos, el conjunto y la eficiencia.

• La mantenibilidad combina la capacidad del programa para ser ampliable, adaptable y servicial, además que pueda probarse, ser compatible y configurable.

C. Evolución del diseño de software

Se trata de un proceso continuo, al principio se concentraban en criterios para el desarrollo de programas modulares y en métodos para mejorar estructuras de software con un enfoque de arriba abajo.

En la industria del software se aplican varios métodos de diseño, todos estos métodos tienen algunas características en común:
1) un mecanismo para traducir el modelo de requerimientos en una representación del diseño.
2) una notación para representar las componentes funcionales y sus interfaces.
3) una heurística para mejorar y hacer particiones.
4) lineamientos para evaluar la calidad.

Sin importar el método de diseño que se utilice, debe aplicarse un conjunto de conceptos básicos al diseño en el nivel de datos, arquitectura, interfaz y componente.

D. Conceptos de diseño

Abstracción

Cuando se considera una solución modular para cualquier problema, es posible plantear muchos niveles de abstracción. En el más elevado se enuncia una solución en términos gruesos con el uso del lenguaje del ambiente del problema. En niveles más bajos de abstracción se da la descripción más detallada de la solución. Cuando se desarrollan niveles de abstracción distintos, se trabaja para crear abstracciones tanto de procedimiento como de datos. Una abstracción de

procedimiento es una secuencia de instrucciones que tienen una función específica y limitada.

Una abstracción de datos es un conjunto de éstos con nombre que describe a un objeto de datos.

Arquitectura

Es la estructura de organización de los componentes de un programa (módulos), la forma en la que éstos interactúan y la estructura de datos que utilizan. Sin embargo, en un sentido más amplio, los componentes se generalizan para que representen los elementos de un sistema grande y sus interacciones.

Es necesario especificar un conjunto de propiedades como parte del diseño de la arquitectura:

Propiedades estructurales. Este aspecto de la representación del diseño arquitectónico define los componentes de un sistema (módulos, objetos, filtros, etc.) y la manera en la que están agrupados e interactúan unos con otros.

Propiedades extra funcionales. La descripción del diseño arquitectónico debe abordar la forma en la que la arquitectura del diseño satisface los requerimientos de desempeño, capacidad, confiabilidad, seguridad y adaptabilidad, así como otras características del sistema.

Familias de sistemas relacionados. El diseño arquitectónico debe basarse en patrones repetibles que es común encontrar en el diseño de familias de sistemas similares. En esencia, el diseño debe tener la capacidad de reutilizar bloques de construcción arquitectónica.

Los modelos estructurales representan la arquitectura como un conjunto organizado de componentes del programa. Los modelos de marco aumentan el nivel de abstracción del diseño, al tratar de identificar patrones de diseño arquitectónico repetibles que se encuentran en tipos similares de aplicaciones. Los modelos dinámicos abordan los aspectos estructurales de la arquitectura del programa e indican cómo cambia la estructura o la configuración del sistema en función de eventos externos. Los modelos del proceso se centran en el diseño del negocio o proceso técnico al que debe dar acomodo el sistema. Por último, los modelos funcionales se usan para representar la jerarquía funcional de un sistema.

Patrones

Describe una estructura de diseño que resuelve un problema particular del diseño dentro de un contexto específico y entre "fuerzas" que afectan la manera en la que se aplica y en la que se utiliza dicho patrón.

El objetivo de cada patrón de diseño es proporcionar una descripción que permita a un diseñador determinar

1) si el patrón es aplicable al trabajo en cuestión.
2) si puede volverse a usar (con lo que se ahorra tiempo de diseño).
3) si sirve como guía para desarrollar un patrón distinto en funciones o estructura.

División de problemas

Es un concepto de diseño que sugiere que cualquier problema complejo puede manejarse con más facilidad si se subdivide en elementos susceptibles de resolverse u optimizarse de manera independiente.

Un problema es una característica o comportamiento que se especifica en el modelo de los requerimientos para el software. Al separar un problema en sus piezas más pequeñas y por ello más manejables, se requiere menos esfuerzo y tiempo para resolverlo.

Modularidad

Es la manifestación más común de la división de problemas. El software se divide en componentes con nombres distintos y abordables por separado, en ocasiones llamados módulos, que se integran para satisfacer los requerimientos del problema.

Ocultamiento de información

El principio del ocultamiento de información sugiere que los módulos se "caractericen por decisiones de diseño que se oculten de las demás". Entonces, deben especificarse y diseñarse módulos, de forma que la información contenida en un módulo sea inaccesible para los que no necesiten de ella.

Independencia Funcional

Es resultado directo de la separación de problemas y de los conceptos de abstracción y ocultamiento de información. Se logra desarrollando módulos con funciones "miopes" que tengan "aversión" a la interacción excesiva con otros módulos.

La independencia se evalúa con el uso de dos criterios cualitativos: la cohesión y el acoplamiento. La cohesión es un indicador de la fortaleza relativa funcional de un módulo. El acoplamiento lo es de la independencia relativa entre módulos. La cohesión es una extensión natural del concepto de ocultamiento de información.

Refinamiento

Es una estrategia de diseño, se comienza con un enunciado de la función (o descripción de la información), definida en un nivel de abstracción alto. Un programa se elabora por medio del refinamiento sucesivo de los detalles del procedimiento. Se desarrolla una jerarquía con la descomposición de un

enunciado macroscópico de la función (abstracción del procedimiento) en forma escalonada hasta llegar a los comandos del lenguaje de programación.

La abstracción y el refinamiento son conceptos complementarios. La primera permite especificar internamente el procedimiento y los datos. El refinamiento ayuda a revelar estos detalles a medida que avanza el diseño.

Rediseño

Es técnica de reorganización que simplifica el diseño (o código) de un componente sin cambiar su función o comportamiento.

Cuando se rediseña el software, se examina el diseño existente en busca de redundancias, elementos de diseño no utilizados, algoritmos ineficientes o innecesarios, estructuras de datos mal construidas o inapropiadas y cualquier otra falla del diseño que pueda corregirse para obtener un diseño mejor.

E. Conceptos de diseño orientado a objetos

Clases de diseño

Pueden desarrollarse cinco tipos diferentes de clases de diseño, cada una de las cuales representa una capa distinta de la arquitectura del diseño:

- Clases de usuario de la interfaz. Definen todas las abstracciones necesarias para la interacción humano-computadora (IHC).
- Clases del dominio de negocios. Es frecuente que sean refinamientos de las clases de análisis definidas antes
- Clases de proceso. Implantan abstracciones de negocios de bajo nivel que se requieren para administrar por completo las clases de dominio de negocios.
- Clases persistentes. Representan almacenamientos de datos (por ejemplo, una base de datos) que persistirán más allá de la ejecución del software.
- Clases de sistemas. Implantan las funciones de administración y control del software que permiten que el sistema opere y se comunique dentro de su ambiente de computación y con el mundo exterior.

Características de las clases

Completa y suficiente. Una clase de diseño debe ser el encapsulado total de todos los atributos y métodos que sea razonable y que existan para la clase.

Primitivismo. Los métodos asociados con una clase de diseño deben centrarse en el cumplimiento de un servicio para la clase.

Cohesión. La clase tiene un conjunto pequeño y centrado de responsabilidades; para implementarlas emplea atributos y métodos de objetivo único.

Acoplamiento. es necesario que las clases de diseño colaboren una con otra. Sin embargo, la colaboración debe mantenerse en un mínimo aceptable.

F. Modelo del diseño

El modelo del diseño puede ser percibido en dos dimensiones distintas, la dimensión del proceso indica la evolución del modelo del diseño conforme se ejecutan las tareas de éste como parte del proceso del software y la dimensión de la abstracción representa el nivel de detalle a medida que cada elemento del modelo de análisis se transforma en un equivalente de diseño y luego se mejora en forma iterativa.

G. Elementos del diseño de datos

El diseño de datos (arquitectura de datos) crea un modelo de datos o información que se representa en un nivel de abstracción elevado.

Este modelo de los datos se refina después en forma progresiva hacia representaciones más específicas de la implementación que puedan ser procesadas por el sistema basado en computadora.

H. Elementos del diseño arquitectónico

Por lo general, el elemento de diseño arquitectónico se ilustra como un conjunto de sistemas interconectados, con frecuencia obtenidos de paquetes de análisis dentro del modelo de requerimientos. Cada subsistema puede tener su propia arquitectura.

El modelo arquitectónico proviene de tres fuentes:

1. Información sobre el dominio de la aplicación del software que se va a elaborar.
2. Los elementos específicos del modelo de requerimientos, tales como diagramas de flujo de datos o clases de análisis, sus relaciones y colaboraciones para el problema en cuestión.
3. La disponibilidad de estilos arquitectónicos y sus patrones.
4.

I. Elementos del diseño de la interfaz

Hay tres elementos importantes del diseño de la interfaz:

1. La interfaz de usuario (IU).
2. Las interfaces externas que tienen que ver con otros sistemas, dispositivos, redes y otros productores o consumidores de información.

3. Interfaces internas que involucran a los distintos componentes del diseño.

Permiten que el software se comunique externamente y permita la comunicación y colaboración internas entre los componentes que constituyen la arquitectura del software.

El diseño de la IU (usabilidad) es una acción principal de la ingeniería de software. Incorpora elementos estéticos, elementos ergonómicos y elementos técnicos.

J. Elementos del diseño en el nivel de los componentes

Este diseño define estructuras de datos para todos los objetos de datos locales y detalles algorítmicos para todo el procesamiento que tiene lugar dentro de un componente, así como la interfaz que permite el acceso a todas las operaciones de los componentes.

K. Elementos del diseño del despliegue

Indican la forma en la que se acomodarán la funcionalidad del software y los subsistemas dentro del ambiente físico de la computación que lo apoyará.

III. CONCLUSIÓN

El objetivo del diseño del software es aplicar un conjunto de principios, conceptos y prácticas que llevan al desarrollo de un sistema o producto de alta calidad. El diseño crea un modelo de software que implantará correctamente todos los requerimientos del usuario y satisfacción para quienes lo utilicen.

Los conceptos de diseño dan a entender la abstracción como mecanismo para crear componentes reutilizables de software, en la importancia de la arquitectura como forma de entender mejor la estructura general de un sistema, en los beneficios de la ingeniería basada en patrones como técnica e diseño de software con capacidad comprobada, en el valor de la separación de problemas y de la modularidad eficaz como forma de elaborar software más entendible, más fácil de probar y de recibir mantenimiento.

También en las consecuencias de ocultar información como mecanismo para reducir la propagación de los efectos colaterales cuando hay errores, en el efecto de la independencia funcional como criterio para construir módulos eficaces, en el uso del refinamiento como mecanismo de diseño, en una consideración de los aspectos que interfieren con los requerimientos del sistema, en la aplicación del rediseño para optimizar el diseño obtenido y en la importancia de las clases orientadas a objetos y de las características relacionadas con ellos.

REFERENCES

[1] Roger S. Pressman, "*Ingeniería del Software, un enfoque práctico*" 7a ed.Connecticut University, New York: McGraw-Hill, 2010, pp. 183 - 203.