Universidad Nacional de Cajamarca
"Norte de la Universidad Peruana"

INGENIERÍA DE SOFTWARE I

Diseño de software

SABERES PREVIOS

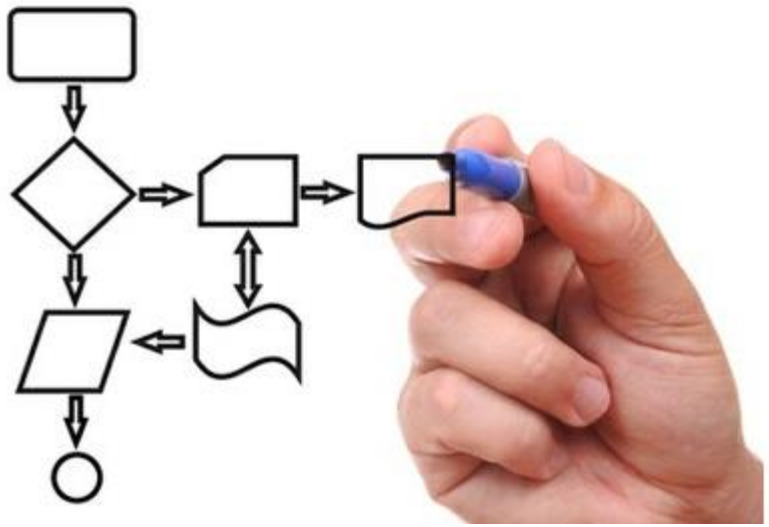
¿Qué entendemos por diseño del software?

¿Cuál es la diferencia entre la arquitectura y el diseño del software?

.

LOGRO

- Al término de la sesión, los estudiantes reconocen los fundamentos del diseño de software.

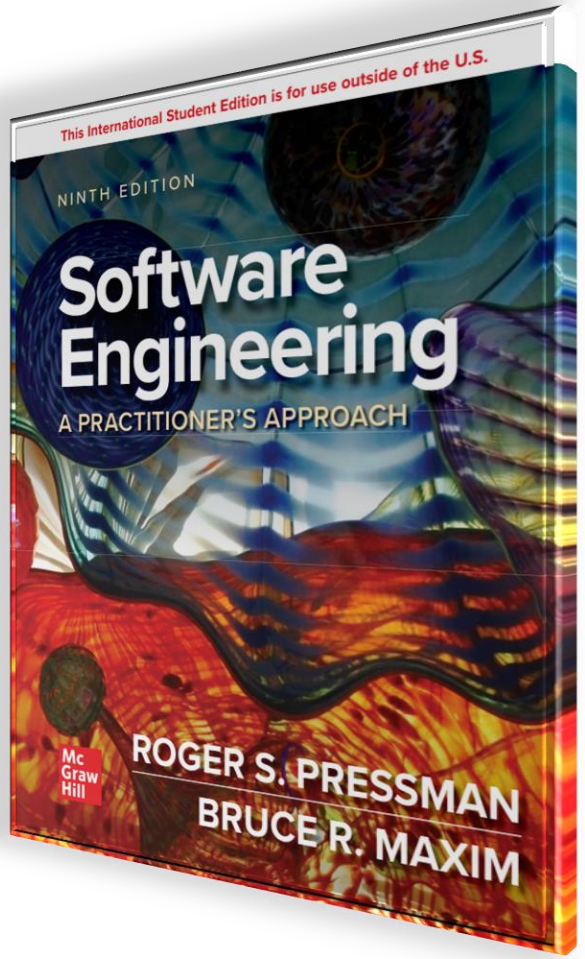


Temario: Diseño de software:

1. Fundamentos del diseño del software
2. Proceso del diseño del software
3. Calidad del diseño del software
4. Registro del diseño del software
5. Estrategias y métodos del diseño del software
6. Análisis y evaluación del diseño del software

Documentos de consulta

SWEBOK®
v4.0



Introducción

- El diseño de software, visto como una actividad del ciclo de vida, es la aplicación de la disciplina de ingeniería de software en la que se analizan los requisitos del software para definir las características externas y la estructura interna del software como base para su construcción.

Descripción de diseño de software

- Una descripción de diseño de software (SDD) documenta el resultado del diseño de software.
- Es una “representación de software creado para facilitar el análisis, la planificación, la implementación y la toma de decisiones.
- La descripción del diseño de software se utiliza como medio para comunicar información de diseño de software y puede **considerarse** como un plano o modelo del sistema”

El diseño del software se lleva a cabo en tres etapas:

Diseño arquitectónico del sistema de software

Diseño de alto nivel o orientado al **exterior** del sistema y sus componentes

Diseño detallado o orientado hacia el **interior**

Universidad Nacional de Cajamarca

"Norte de la Universidad Peruana"

1 FUNDAMENTOS DEL DISEÑO DE SOFTWARE

1.1 Pensamiento de diseño

- El diseño está a nuestro alrededor, en las cosas y organizaciones que han sido creados para satisfacer una necesidad o resolver un problema.
- En un sentido general, el diseño puede verse como una forma de resolución de problemas.
- El pensamiento de diseño comprende dos elementos esenciales: (1) comprender la necesidad o problema y (2) idear una solución.

- Ross, Goodenough e Irvine ofrecen una elaboración del pensamiento de diseño apropiado para el software:
- Este proceso consta de cinco pasos básicos:

(1) cristalizar un propósito u objetivo;

(2) formular un concepto sobre cómo se puede lograr el propósito;

(3) idear un mecanismo que implemente la estructura conceptual;

(4) introducir una notación para expresar las capacidades del mecanismo e invocar su uso;

(5) describir el uso de la notación en un contexto de problema específico para invocar el mecanismo de modo que el se logra el propósito

- Esto es particularmente apropiado porque gran parte del diseño de software consiste en crear el vocabulario necesario para expresar un problema, expresar su solución e implementar esa solución.
- Los pasos enfatizan la naturaleza lingüística de la resolución de problemas de diseño de software.
- Este es un patrón recurrente que vemos en todo el diseño de alto nivel, el diseño detallado y la arquitectura. Por lo tanto, el Diseño de Software es un proceso práctico de transformar un planteamiento de problema en un planteamiento de solución. El diseño de software comparte puntos en común con otros tipos de diseño. El diseño se puede comprender mejor a través de la teoría del diseño

1.2 Contexto del diseño de software

- El diseño de software es una parte importante del proceso de desarrollo de software. Comprender el papel del diseño de software. es ver cómo encaja en el ciclo de vida del desarrollo de software.
- Para comprender ese contexto, es importante comprender las principales características y funciones de los requisitos de software, la construcción de software, las pruebas de software y el mantenimiento de software. El contexto varía según muchos factores, incluido el grado de formalidad y la etapa del ciclo de vida.

- El diseño de software es la transformación de los requisitos, necesidades e inquietudes del cliente y otros en especificaciones de diseño implementables. Sus contextos incluyen los siguientes

- Diseño de interfaz con los requisitos del software: Los requisitos establecen un conjunto de problemas que el diseño del software debe resolver
- Interfaz de diseño con arquitectura de software: en los casos en que se ha establecido una arquitectura, esa arquitectura limita el diseño al capturar aspectos fundamentales del sistema, sus componentes principales y sus interconexiones, interfaces de programación de aplicaciones (API), estilos y patrones que se utilizarán, y principios arquitectónicos que se observarán y aplicarán.

- Diseño de interfaz con construcción de software: El diseño de software debe proporcionar una guía a los implementadores sobre la construcción del sistema.
- Interfaz de diseño con pruebas de software: el diseño de software proporciona una base para una estrategia de prueba general y casos de prueba que garantizan que el diseño se implemente correctamente y funcione según lo previsto.

1.3 Cuestiones clave en el diseño de software

- Se deben abordar muchas cuestiones clave al diseñar software. Algunas son preocupaciones de calidad que todo software debe abordar (**rendimiento, seguridad, confiabilidad, usabilidad**, etc.
- Otra cuestión importante es cómo refinar, organizar, interconectar y empaquetar componentes de software. Estas cuestiones son tan fundamentales que todos los enfoques de diseño las abordan de una forma u otra.

- Por el contrario, otras cuestiones “tienen que ver con algún aspecto del comportamiento del software que no está en el dominio de la aplicación, pero que aborda algunos de los dominios de soporte”. Estas cuestiones, que a menudo atraviesan la funcionalidad del sistema, se denominan aspectos que “tienden a no ser unidades de descomposición funcional del software, sino más bien propiedades que afectan el rendimiento o la semántica de los componentes de manera sistémica”.

1.4 Principios de diseño de software

- Un principio es "una verdad o proposición fundamental que sirve como fundamento de un sistema de creencias o comportamiento o de una cadena de razonamiento". [Diccionario de ingles Oxford]

- Los principios de diseño proporcionan dirección u orientación para tomar decisiones durante el diseño. Algunos principios se originaron durante los primeros días de la ingeniería de software, otros incluso son anteriores a la disciplina, derivada de las mejores prácticas de ingeniería ajenas al software.
- Los principios de diseño de software son nociones clave que proporcionan la base para muchos conceptos, enfoques y métodos de diseño de diferentes software

- Los principios de diseño de software incluyen lo siguiente:
- La abstracción es “una visión de un objeto que se centra en la información relevante para un propósito particular e ignora el resto de la información”. El principio de abstracción ayuda a identificar propiedades esenciales comunes a entidades superficialmente diferentes”.

- Separación de preocupaciones (SoC). Una preocupación de diseño es un “área de interés con respecto a un diseño de software” que es relevante para una o más de sus partes interesadas. Al identificar y separar las preocupaciones, el diseñador puede centrarse en cada preocupación del sistema de forma aislada.

- La modularización (o refinamiento o descomposición) estructura el software grande como si estuviera formado por componentes o unidades más pequeñas. Cada componente tiene un nombre y tiene interfaces bien definidas para sus interacciones con otros componentes. Los componentes más pequeños son más fáciles de entender y, por tanto, de mantener.

- Tradicionalmente, el objetivo es colocar distintas funcionalidades y responsabilidades en diferentes componentes.
- De dice que cada módulo de un sistema debería tener una única responsabilidad.
- Una forma de pensar en la modularización es dividir y conquistar.

- La encapsulación (u ocultación de información) se basa en los principios de abstracción y modularización para que la información no esencial sea menos accesible, lo que permite a los usuarios del módulo centrarse en los elementos esenciales de la interfaz.

- La separación de interfaz e implementación es una aplicación de encapsulación que implica definir un componente especificando sus interfaces públicas, que son conocidas y accesibles para los clientes. aislar el uso de un componente de los detalles de cómo se construye ese componente.

- El acoplamiento se define como “una medida de la interdependencia entre módulos en un programa de computadora”. La mayoría de los métodos de diseño recomiendan que los módulos estén débilmente acoplados.
- La cohesión (o localización) se define como “una medida de la fuerza de asociación de los elementos dentro de un módulo”.
- Cohesión destaca la organización de los componentes de un módulo en función de su relación. La mayoría de los métodos de diseño defienden que los módulos deben maximizar su cohesión/localidad.

- La uniformidad es un principio de coherencia entre los componentes del software: se deben producir soluciones comunes para abordar problemas comunes o recurrentes. Estos incluyen esquemas de nombres, notaciones y sintaxis, interfaces que definen el acceso a servicios y mecanismos, y ordenación de elementos y parámetros. Esto se puede lograr mediante convenciones como reglas, formatos y estilos.

- La integridad (o suficiencia) significa garantizar que un componente de software capture las características importantes de una abstracción y no omita nada. La integridad adopta varias formas, quizás la más importante de las cuales es la integridad del diseño frente a los requisitos: un diseño debe ser suficiente para que los diseñadores demuestren cómo se cumplirán los requisitos y cómo el trabajo posterior satisfará esos requisitos.
- El diseño debe ser completo con respecto a los modos y estados del software

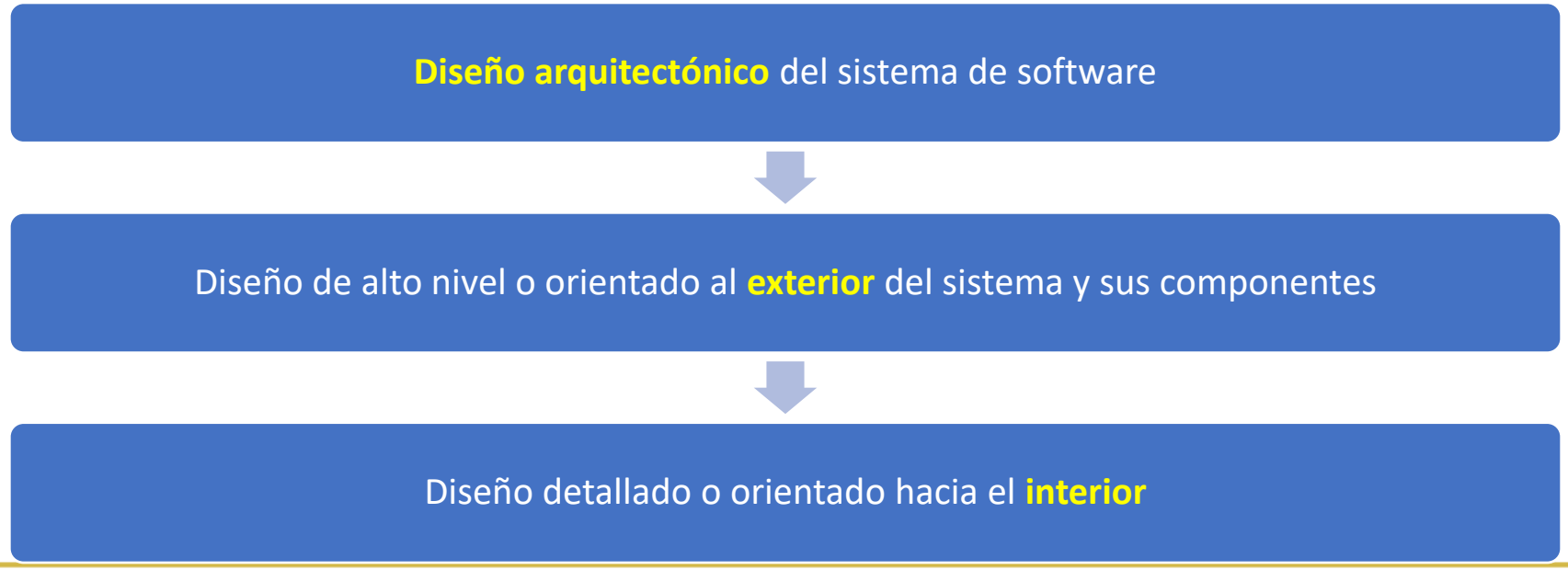
- Confirmabilidad significa que la información necesaria para verificar que el software es correcto, completo y apto para su uso está disponible. Esto es relevante para cualquier software, pero es de particular importancia para el software de alta seguridad, como el software donde la seguridad, existen preocupaciones críticas sobre confiabilidad o seguridad.

- Otros principios de diseño. Recientemente, con la creciente aparición de sistemas autónomos, el uso de aprendizaje automático e inteligencia artificial y, en general, sistemas con impactos sociales cada vez mayores, se han desarrollado enfoques de Diseño Éticamente Alineado para abordar preocupaciones que incluyen los valores humanos universales, la autodeterminación política, y agencia de datos y confiabilidad técnica. Los principios generales del Diseño Éticamente Alineado son los derechos humanos, el bienestar, la agencia de datos, la eficacia, la transparencia, la rendición de cuentas, la conciencia del mal uso y la competencia.

Universidad Nacional de Cajamarca
"Norte de la Universidad Peruana"

2
PROCESOS DE DISEÑO DE SOFTWARE

- El diseño de software generalmente se considera un proceso o actividad de varias etapas. El diseño de software se puede dividir en las siguientes etapas o fases. Cuando sea necesario, distinguimos la fase de la actividad general:



Diseño de alto nivel y diseño detallado

- La etapa de diseño de alto nivel está orientada hacia afuera: desarrolla la estructura y organización de alto nivel del software, identifica sus diversos componentes y cómo ese sistema de software y sus componentes interactúan con el entorno y sus elementos.
- La etapa de diseño detallado es interna: especifica cada componente con suficiente detalle para facilitar su construcción y cumplir con sus obligaciones externas, incluida la forma en que los componentes de software se refinan aún más en módulos y unidades.

- No todas las etapas se encuentran en todos los procesos de software. Sin embargo, cuando está presente, cada etapa crea una obligación para la siguiente etapa con respecto al software que está en desarrollo.
- Aunque los desarrolladores de software generalmente siguen pautas similares para lo que sucede en cada etapa, no existen límites estrictos entre las etapas con respecto a lo que se debe hacer y cuándo.
- Por ejemplo, para muchos sistemas de software, la elección de un algoritmo para clasificar datos se dejará en manos de los programadores, dentro de las limitaciones y la orientación proporcionadas por los requisitos del sistema, su descripción de arquitectura o especificaciones de diseño. Sin embargo, para otro sistema de software, la existencia de un algoritmo adecuado podría ser significativa desde el punto de vista arquitectónico y debe determinarse en una etapa temprana del ciclo de vida.

- Sin ese algoritmo, no hay posibilidad de construir el software para satisfacer sus necesidades
- Algunas reglas generales para cada etapa incluyen las siguientes:
- La etapa de diseño arquitectónico define un modelo computacional, los principales elementos computacionales y el importantes protocolos y relaciones entre ellos. Esta etapa desarrolla estrategias para abordar preocupaciones transversales, como el rendimiento, la confiabilidad, la seguridad y la protección, y la articulación de decisiones transversales, incluidos estilos de todo el sistema (por ejemplo, un estilo Modelo-Vista-Controlador versus un estilo Tuberías y filtros, junto con la justificación de dichas decisiones).

- La etapa de diseño de alto nivel incluye la identificación de los elementos computacionales primarios y las relaciones significativas entre ellos, con un enfoque en la existencia, función e interfaces de cada componente principal. Esa definición debe ser lo suficientemente detallada como para permitir a los diseñadores o programadores de componentes del cliente acceder correcta y eficientemente a las capacidades de cada servidor, sin tener que leer su código.

- La etapa de diseño detallado define la estructura interna de cada módulo, enfocándose en detallar y justificar las elecciones de algoritmos, acceso a datos y representación de datos. Las especificaciones de diseño detalladas deben ser suficientes para permitir a los programadores codificar cada módulo durante la construcción. El código es una representación de la solución que es lo suficientemente detallada y completa como para que un compilador (o intérprete) pueda ejecutarla.

2.1 Diseño de alto nivel

- El diseño de alto nivel especifica la interacción de los componentes principales de un sistema entre sí y con el entorno, incluidos usuarios, dispositivos y otros sistemas. El diseño de alto nivel aborda lo siguiente:
  - Eventos y mensajes externos a los que el sistema debe responder.
  - Eventos y mensajes que el sistema debe producir
  - Especificación de los formatos y protocolos de datos para eventos y mensajes
  - Especificación de las relaciones de orden y temporización entre eventos y mensajes de entrada, y eventos y mensajes de salida.
  - Persistencia de datos (cómo se almacenan y gestionan los datos)

- El diseño de alto nivel se lleva a cabo dentro del ámbito establecido por la arquitectura de software del sistema. Por ejemplo, la señalización y mensajería de eventos utilizará los protocolos y modos de interacción establecidos por la arquitectura. Los formatos y protocolos de datos utilizarán estándares de comunicación y datos especificados por la arquitectura.
- En ausencia de una etapa de diseño de arquitectura explícita, algunas de estas directivas serán establecidas por los requisitos del software o decididas durante el diseño de alto nivel.

2.2 Diseño detallado

- La etapa de diseño detallado se desarrolla dentro de las limitaciones establecidas por el diseño de alto nivel. Especifica las características internas de los componentes principales del sistema, los módulos internos y sus interconexiones con otros módulos, servicios y procesos que proporcionar propiedades informáticas, algoritmos y reglas de acceso a datos y estructuras de datos. Esto incluye lo siguiente:
- Refinamiento de los principales componentes del sistema en módulos o unidades de programa, incluidas oportunidades para utilizar componentes disponibles en el mercado y marcos de aplicaciones

- Asignación de responsabilidades de diseño a módulos y unidades de programa.
- Interacciones entre módulos
- Alcance y visibilidad entre componentes, módulos y unidades de programa
- Modos de componentes, estados de componentes y transiciones entre ellos
- Interdependencias de datos y control
- Organización, empaquetado e implementación de datos.
- Interfaces de usuario
- Algoritmos y estructuras de datos necesarios

Universidad Nacional de Cajamarca

"Norte de la Universidad Peruana"

3 CUALIDADES DEL DISEÑO DEL SOFTWARE

- Los requisitos de software y las directivas de arquitectura tienen como objetivo guiar el software hacia ciertas características o cualidades de diseño.
- Las cualidades de diseño son una subclase importante de preocupaciones.
- Principios de diseño de software es ayudar al software a lograr estas cualidades.
- Entre las características de interés para los diseñadores se encuentran las siguientes:

3.1 Concurrencia

- El diseño para la concurrencia se refiere a cómo se refina el software en unidades concurrentes como procesos, tareas y subprocesos y las consecuencias de esas decisiones con respecto a la eficiencia, la atomicidad, la sincronización y la programación

3.2 Control y manejo de eventos

- El manejo de eventos se ocupa de cómo organizar el flujo de control, así como de cómo manejar eventos reactivos y temporales a través de varios mecanismos, incluida la sincronización, la invocación implícita y las devoluciones de llamadas

3.3 Persistencia de datos

- La persistencia de datos se refiere al almacenamiento y gestión de datos en todo el sistema.

3.4 Distribución de componentes

- La distribución se refiere a cómo se distribuyen los componentes de software en el hardware (incluidas computadoras, redes y otros dispositivos) y cómo esos componentes se comunican mientras cumplen con el rendimiento, la confiabilidad, la escalabilidad, la disponibilidad, la monitorización, la continuidad del negocio y otras expectativas.

3.5 Manejo de errores y excepciones, tolerancia a fallo

- Esta preocupación se refiere a cómo prevenir, evitar, mitigar, tolerar y procesar errores y condiciones excepcionales.

3.6 Integración e Interoperabilidad

- Este problema surge a nivel empresarial o de sistema de sistemas o para cualquier software complejo cuando sistemas o aplicaciones heterogéneos necesitan interfuncionar mediante intercambios de datos o accediendo a los servicios de otros. Dentro de un sistema de software, el problema surge cuando los componentes se diseñan utilizando diferentes marcos, bibliotecas o protocolos

3.7 Garantía, protección y protección

- La alta seguridad abarca una serie de cualidades del software, incluidas las preocupaciones de seguridad y protección, relacionadas con si el software se comporta según lo previsto en situaciones críticas, como frente a peligros.
- El diseño para la seguridad se refiere a cómo prevenir la divulgación, creación, cambio, eliminación o denegación de acceso no autorizado a la información y otros recursos frente a ataques al sistema o violaciones de las políticas del sistema para limitar el daño; proporcionar continuidad del servicio; y ayudar en la reparación y recuperación.
- El diseño para la seguridad se refiere a la gestión del comportamiento del software en circunstancias que podrían provocar daños o pérdida de vidas humanas o daños a la propiedad o al medio ambiente.

3.8 Variabilidad

- La variabilidad se refiere a las variaciones permitidas en el software, como surge, por ejemplo, en líneas de productos de software y familias de sistemas, para acomodar y gestionar múltiples implementaciones diferentes, como para diferentes organizaciones o mercados.

Universidad Nacional de Cajamarca
"Norte de la Universidad Peruana"

4
DISEÑOS DE SOFTWARE DE GRABACIÓN

- El resultado de los procesos de diseño es el conocimiento acumulado y los productos del trabajo que registran ese conocimiento. Los productos de trabajo del diseño de software capturan
- (1) aspectos de los problemas a resolver, utilizando el vocabulario del dominio.
- (2) un vocabulario de solución para resolver los problemas de diseño.
- (3) las principales decisiones que se han tomado; y
- (4) explicaciones del fundamento de cada decisión no trivial.

- Registrar la justificación de las decisiones importantes mejora la mantenibilidad a largo plazo del producto de software cuando se consideran modificaciones o mejoras.
- Estos productos de trabajo, a menudo denominados descripciones de diseño o especificaciones de diseño, pueden tomar la forma de textos, diagramas, modelos y prototipos que comprenden los planos del software que se implementará.

- Un aspecto fundamental del diseño de software es la comunicación sobre el diseño a los clientes, otros diseñadores, implementadores y otras partes interesadas. Esto es cierto ya sea que el software se desarrolle utilizando métodos ágiles, tradicionales o formales. La comunicación variará según el público objetivo, el nivel de detalle que se comunique y la relevancia para las preocupaciones de las partes interesadas.
- Si bien los implementadores son una audiencia importante para las especificaciones de diseño, el personal de pruebas y control de calidad, las autoridades de certificación y los analistas de requisitos también utilizarán las especificaciones de diseño en su trabajo.

- Por lo tanto, las especificaciones de diseño deben tener una audiencia, un tema y un uso previsto claramente definidos
- Los diseñadores pueden analizar y evaluar estos productos de trabajo para determinar si el diseño puede cumplir con los requisitos y limitaciones del software.
- El diseño de software también examina y evalúa soluciones alternativas y compensaciones. Además de utilizarlos como insumos y como punto de partida para la construcción y pruebas, las partes interesadas pueden utilizar los productos del trabajo de diseño para planificar actividades posteriores, como la verificación y validación del sistema.

- A medida que los conceptos de diseño evolucionan, también lo hacen sus representaciones.
- Parte del proceso de diseño implica la creación de vocabularios apropiados para problemas y soluciones.
- Un boceto informal puede ser lo más apropiado para las primeras etapas. Es útil distinguir las especificaciones en proceso (“en funcionamiento”) de los productos de diseño final.
- Los primeros son producidos por el equipo de diseño para el equipo de diseño. este último puede producirse para partes interesadas conocidas o incluso para una audiencia futura desconocida.

- Existen muchas notaciones para representar artefactos de diseño de software.
- El diseño de software a menudo se lleva a cabo utilizando múltiples tipos de notación.
- Dos grandes áreas de preocupación son las estructuras y los comportamientos del software. Algunos se utilizan para describir un diseño.
- organización estructural, otros para representar la estructura del software.
- comportamiento previsto.
- A continuación, se clasifican como notaciones para preocupaciones estructurales y de comportamiento. Ciertas notaciones se utilizan principalmente durante el diseño arquitectónico y otras principalmente durante el diseño detallado. algunos son útiles en todas las etapas del diseño de software, algunas notaciones están estrechamente vinculadas al contexto de métodos de diseño específicos.

- El Lenguaje Unificado de Modelado (UML) es una familia de notaciones ampliamente utilizada que aborda cuestiones estructurales y de comportamiento y se utiliza en todas las etapas de diseño, desde el diseño arquitectónico hasta el diseño detallado.

4.1 Diseño basado en modelos

- A lo largo de la historia de la ingeniería de software, incluida la arquitectura y el diseño, ha habido una evolución de artefactos basados en documentos a artefactos basados en modelos. El diseño basado en modelos (MBD) es un enfoque para registrar diseños donde los modelos juegan un papel importante.

- Esta tendencia refleja las limitaciones de los artefactos basados en documentos y las mayores capacidades de las herramientas automatizadas.
- Los artefactos basados en documentos utilizan lenguaje natural y diagramas informales para transmitir las intenciones de los diseñadores, lo que podría introducir ambigüedad e incompletitud. Incluso cuando los documentos utilizan formatos bien definidos, la información relevante puede estar distribuida entre ellos, lo que dificulta la comprensión y el análisis.
- Con MBD, las herramientas adecuadas pueden recopilar y organizar información relevante para que la utilicen los diseñadores y otras partes interesadas de una forma accesible.

- Las herramientas modernas han acelerado la tendencia de pasar de documentos a artefactos basados en modelos. Las herramientas permiten la animación o simulación de diversos aspectos del software, análisis de escenarios hipotéticos y compensaciones, y creación rápida de prototipos. Las herramientas también facilitan enfoques de prueba e integración continua, trazabilidad mejorada e interactiva y captura y gestión de conocimientos, que son ineficientes o incluso inviables con enfoques basados en documentos.

- El desarrollo impulsado por modelos (MDD) es un paradigma de desarrollo que utiliza modelos como principal componente del proceso de desarrollo artefactos

4.2 Descripciones de diseño estructural

- Los siguientes tipos de notación, la mayoría de los cuales son gráficos, se utilizan para representar los aspectos estructurales de un diseño de software, es decir, se utilizan para describir los componentes principales y cómo están interconectados (vista estática) y la asignación de responsabilidades a cada uno de ellos. componentes y módulos:

- Los diagramas de clases y objetos se utilizan para representar un conjunto de clases y objetos y sus interrelaciones.
- Los diagramas de componentes se utilizan para representar un conjunto de componentes (“parte[s] física[s] y reemplazable(s) de un sistema que [se ajustan] y [proporcionan] la realización de un conjunto de interfaces”) y sus interconexiones.
- Los modelos de componentes evolucionaron desde lenguajes de interconexión de módulos anteriores hasta los sistemas de paquetes de lenguajes de programación como Ada y Java y los sofisticados sistemas de módulos de los sistemas de lenguajes funcionales actuales como

- Las tarjetas de colaborador con responsabilidad de clase (CRC) se utilizan para indicar los nombres de los componentes (clases), sus responsabilidades y los componentes con los que interactúan para cumplir con esas responsabilidades
- Los diagramas de implementación se utilizan para representar un conjunto de nodos físicos y sus interconexiones para modelar los aspectos físicos del software implementado en el hardware.
- Los diagramas de entidad relación (ERD) se utilizan para representar modelos conceptuales, lógicos y físicos de datos almacenados en repositorios de información o como parte de descripciones de interfaz.

- Los lenguajes de descripción de interfaces (IDL) son lenguajes similares a la programación que se utilizan para definir las interfaces (nombres y tipos de operaciones exportadas) de componentes de software Los diagramas de estructura se utilizan para describir la estructura de llamada de los programas (es decir, muestran qué módulos llaman y qué otros módulos llaman y son llamados por).

4.3 Descripciones del diseño conductual

- Las siguientes notaciones y lenguajes, algunos gráficos y otros textuales, se utilizan para describir el comportamiento dinámico de los sistemas de software y sus componentes. Muchas de estas notaciones son útiles principalmente, pero no exclusivamente, durante el diseño detallado
- Además, las descripciones de comportamiento pueden incluir la justificación de las decisiones de diseño

- Los diagramas de actividades se utilizan para mostrar el flujo de un cálculo de na actividad a otra. También pueden representar actividades concurrentes, sus entradas y salidas y oportunidades de concurrencia.
- Los diagramas de comunicación se utilizan para mostrar las interacciones entre un grupo de objetos. el énfasis está en los objetos, sus enlaces y los mensajes que intercambian en esos enlaces.
- Los diagramas de flujo de datos (DFD) se utilizan para mostrar el flujo de datos entre elementos informáticos. Un DFD proporciona “una descripción basada en modelar el flujo de información alrededor de una red de elementos operativos, donde cada elemento hace uso o modifica la información que fluye hacia ese elemento”. Los DFD tienen otros usos, como el análisis de seguridad, ya que identifican posibles rutas de ataque y divulgación de información confidencial

- Las tablas y diagramas de decisión se utilizan para representar combinaciones complejas de condiciones y acciones
- Los diagramas de flujo se utilizan para representar el flujo de control y la secuencia de acciones asociadas.
- Los diagramas de secuencia se utilizan para mostrar las interacciones entre un grupo de objetos, y representan el orden temporal de los mensajes transmitidos entre objetos.
- Los diagramas de estado (transición) y los gráficos de estado se utilizan para mostrar las transiciones de un estado a otro y cómo cambia el comportamiento de un componente según su estado actual y su respuesta a los eventos de entrada.

- Los lenguajes de especificación formal son predominantemente lenguajes textuales basados en nociones básicas de las matemáticas (por ejemplo, tipo, conjunto, secuencia, proposición lógica) para definir de manera rigurosa y abstracta las interfaces y el comportamiento de los componentes de software, a menudo en términos de condiciones previas y posteriores, e invariantes. , verificación de tipos y modelos computacionales

- El pseudocódigo y los lenguajes de diseño de programas (PDL) son notaciones estructuradas similares a lenguajes de programación que se utilizan para describir el comportamiento de procesamiento de un procedimiento, generalmente en la etapa de diseño detallado. El uso de estos lenguajes es menos común hoy en día, pero todavía se encuentra en la documentación de algoritmos.

4.4 Patrones y estilos de diseño

- Descrito de manera sucinta, un patrón es “una solución común a un problema común en un contexto dado”. Los estilos arquitectónicos pueden verse como patrones "en general", que describen soluciones comunes a problemas a nivel de arquitectura que impregnan el software. Los patrones de diseño incluyen lo siguiente:

- Patrones de creación (p. ej., constructor, fábrica, prototipo, singleton)
- Patrones estructurales (p. ej., adaptador, puente, compuesto, decorador, fachada, contrapeso, proxy)
- Patrones de comportamiento (p. ej., comando, intérprete, iterador, mediador, recuerdo, observador, peertopeer, publicación suscripción, estado, estrategia, plantilla, visitante)
- Los patrones y estilos de diseño reflejan modismos que han demostrado ser útiles para resolver problemas de diseño particulares en el pasado. Surgen en todas las etapas del diseño, incluida la arquitectura

4.5 Idiomas especializados y específicos de dominio

- No todas las representaciones de diseño caen fácilmente en la dicotomía estructura/comportamiento. Por ejemplo, el diseño de la interfaz de usuario mezcla el diseño estructural de lo que un usuario podría ver con La lógica de comportamiento de secuenciar pantallas basada en las acciones del usuario. Las preocupaciones especializadas, como la seguridad y la confiabilidad, a menudo tienen sus propias formas de representación que han evolucionado entre los especialistas de esas comunidades

- Una tendencia reciente ha sido la maduración de los lenguajes de dominio específico (DSL) y las herramientas ampliamente disponibles para desarrollarlos En este enfoque, parte del proceso de diseño consiste en codificar conceptos y construcciones de un dominio de aplicación específico para crear un lenguaje informático para ese dominio, de modo que la representación del diseño utilizando estas construcciones conduzca a una implementación animada o ejecutable.
- En este enfoque, los DSL desdibujan las líneas entre los lenguajes de modelado, los lenguajes de diseño y los lenguajes de programación.

- Hay DSL y herramientas de soporte para dominios como simulación, sistemas en tiempo real, reactivos y distribuidos, desarrollo de juegos, interfaces de usuario, desarrollo de pruebas, y herramientas de procesamiento del lenguaje.
- El crecimiento de los DSL se ha visto facilitado por herramientas gramaticales cada vez más potentes que, dada una definición de lenguaje, pueden generar una interfaz gráfica de usuario, correctores de sintaxis, generadores de código, compiladores y enlazadores para el lenguaje especializado

4.6 Justificación del diseño

- Un resultado útil del diseño es conocer y documentar explícitamente las principales decisiones tomadas, junto con una explicación del fundamento de cada decisión. La justificación del diseño captura por qué se tomó una decisión de diseño. Esto incluye previa suposiciones hechas, alternativas consideradas y compensaciones y criterios analizados para seleccionar un enfoque y rechazar otros.

- Aunque las razones de las decisiones probablemente sean obvias para el equipo de diseño actual, pueden ser menos obvias para quienes modifican o mantienen el sistema después de la implementación.
- Registrar la justificación mejora la mantenibilidad a largo plazo del producto de software. Continuar capturando la justificación de los cambios durante el mantenimiento también contribuye a la viabilidad del software.

- También puede resultar útil capturar decisiones rechazadas y los motivos del rechazo. Captar estos fundamentos puede permitir a un equipo revisar una decisión previamente rechazada cuando cambian los supuestos, requisitos o limitaciones.
- La importancia de la justificación es visible, por ejemplo, en los proyectos de software libre y de código abierto (FOSS), que a menudo involucran equipos grandes y distribuidos de desarrolladores con rotación frecuente.

- La justificación del diseño puede capturarse como parte de una descripción del diseño de software o como un artefacto complementario. A menudo, la justificación se captura en el texto, pero también se pueden utilizar otras formas de representación, como gráficos que representan un diseño como una red interconectada de decisiones.

Universidad Nacional de Cajamarca
"Norte de la Universidad Peruana"

5
ESTRATEGIAS Y MÉTODOS DE DISEÑO DE SOFTWAR

- Existen varias estrategias y métodos para estructurar y guiar el proceso de diseño, muchos de ellos evolucionaron a partir de estilos o paradigmas de programación.
- Además de incorporar una o más estrategias generales, la mayoría de los métodos de diseño se centran en hacer que uno o más conceptos de diseño (ya sean objetos, métodos o eventos) destaquen como temas organizadores para el software.
- Luego, estos temas guían a los diseñadores sobre en qué centrarse primero, cómo proceder y cómo estructurar los módulos.

5.1 Estrategias generales

- Algunos ejemplos citados con frecuencia de estrategias generales útiles en el proceso de diseño incluyen estrategias de divide y vencerás y de refinamiento gradual, estrategias de arriba hacia abajo versus estrategias de abajo hacia arriba, estrategias que utilizan heurísticas, patrones y lenguajes de patrones, y enfoques iterativos e incrementales.

5.2 Diseño orientado a funciones (o estructurado)

- Este es uno de los métodos clásicos de diseño de software. Se centra en el refinamiento (o descomposición) para identificar las funciones principales del software, elaborándolas de manera de arriba hacia abajo
- El diseño estructurado a menudo sigue a un análisis estructurado, lo que produce DFD y descripciones de procesos asociados. Varias herramientas permiten la traducción automática de DFD a diseños de alto nivel.

5.3 Diseño centrado en datos

- El diseño centrado en datos comienza a partir de las estructuras de datos que manipula un programa y no de las funciones que realiza.
- El diseñador de software especifica las estructuras de datos de entrada y salida y luego desarrolla unidades de programa que transforman las entradas en salidas.
- Se han propuesto varias heurísticas para abordar casos especiales, como aquellos en los que existe un desajuste entre las estructuras de entrada y salida

5.4 Diseño orientado a objetos

- Se han propuesto numerosos métodos de diseño de software basados en objetos. El campo ha evolucionado desde los primeros diseños orientados a objetos de mediados de la década de 1980 (donde los sustantivos representan objetos, los verbos representan métodos y los adjetivos representan atributos), donde la herencia y el polimorfismo desempeñan papeles clave, hasta el campo deldiseño basado en componentes (CBD), donde se puede definir y acceder a la metainformación (a través de la reflexión, por ejemplo).
- Aunque las raíces de OOD provienen del concepto de abstracción de datos, el diseño impulsado por la responsabilidad se ha propuesto como un principio subyacente alternativo de OOD

5.5 Diseño centrado en el usuario

- El diseño centrado en el usuario es más que un método de diseño, Es un enfoque multidisciplinario que enfatiza una comprensión profunda de los usuarios y sus necesidades como base para diseñar experiencias de usuario.
- en el contexto de su organización y de las tareas a realizar. Implica recopilar los requisitos del usuario, crear un flujo de tareas y decisiones del usuario, crear prototipos o maquetas representativas de las interfaces de usuario y evaluar la solución de diseño frente a los requisitos originales.

5.6 Diseño Basado en Componentes (CBD)

- CBD descompone un sistema de software en uno o más componentes independientes que se comunican sólo en interfaces bien definidas y se ajustan a un modelo de componente estándar para todo el sistema. Un componente de software es una unidad independiente que tiene interfaces y dependencias bien definidas que pueden componerse e implementarse de forma independiente.
- El CDB aborda cuestiones relacionadas con el suministro, el desarrollo y la integración de dichos componentes para mejorar la reutilización. CBD a menudo enfatiza API comunes para todos los componentes y API especializadas para servicios o responsabilidades específicas.

5.7 Diseño basado en eventos

- El diseño impulsado por eventos es un enfoque en el que un sistema o componente invoca sus operaciones en reacción a eventos (indirecto).
- Los mensajes de publicación/suscripción (difusión) se utilizan a menudo como medio para transportar eventos a través de la red a todos los suscriptores interesados. La publicación/suscripción mantiene a los productores y consumidores desacoplados mediante un intermediario de mensajes con canales llamados temas. Esto difiere de la mensajería punto a punto, donde los remitentes y los receptores necesitan conocerse para entregar y recibir un mensaje. Existen diferentes tipos de procesamiento de eventos, es decir, procesamiento de eventos simple, procesamiento de flujo de eventos y procesamiento de eventos complejos. Los sistemas basados en mensajes frecuentemente incorporan remitentes y receptores identificables dentro del diseño.
- Es posible que los sistemas controlados por eventos no identifiquen explícitamente a los remitentes y receptores, en cambio, cada módulo produce eventos mientras escucha cualquier evento que le interese o al que necesite responder.
- El procesamiento asincrónico de eventos y mensajes “anónimos” son buenas estrategias para sistemas escalables.

5.8 Diseño Orientado a Aspectos (AOD)

- AOD es un método mediante el cual se construye software utilizando aspectos para implementar las preocupaciones transversales y las extensiones identificadas en los requisitos del software. AOD evolucionó a partir de prácticas de programación y diseño orientado a objetos. Aunque aún no se ha convertido en un paradigma de diseño o programación generalizado, la perspectiva orientada a aspectos se utiliza con frecuencia en marcos de aplicaciones y bibliotecas de software donde los parámetros del marco o biblioteca se pueden configurar con aspectos.

5.9 Diseño basado en restricciones

- El papel de las restricciones en el proceso de diseño es limitar el tamaño de un espacio de diseño para excluir alternativas inviables o inaceptables.
- Las restricciones aceleran el diseño porque obligan a tomar algunas decisiones tempranas. Las restricciones pueden reflejar límites impuestos al hardware, software, datos, procedimientos operativos, interfaces o cualquier cosa que afecte al software. Luego, el espacio de diseño restringido se puede explorar con métodos de búsqueda o retroceso.
- Los enfoques de diseño basados en restricciones se utilizan en el diseño de interfaces de usuario, juegos y otras aplicaciones. En general, los problemas de satisfacción de restricciones pueden ser NPdifíciles, sin embargo, se pueden utilizar varios tipos de programación basada en restricciones para aproximar o resolver problemas con restricciones.

5.10 Otros métodos

- Existen otros enfoques de diseño (ver Modelos y métodos de ingeniería de software KA). Por ejemplo, los métodos iterativos y adaptativos implementan incrementos de software y reducen el énfasis en los requisitos y el diseño rigurosos del software.
- Los métodos orientados a servicios crean software distribuido utilizando servicios web ejecutados en computadoras distribuidas. Los sistemas de software a menudo se construyen utilizando servicios de diferentes proveedores interconectados con protocolos estándar (por ejemplo, HTTP, HTTPS, SOAP) diseñados para soportar la comunicación de servicios y el intercambio de información de servicios

Universidad Nacional de Cajamarca
"Norte de la Universidad Peruana"

6 ANÁLISIS DE CALIDAD DEL DISEÑO DE SOFTWARE Y EVALUACIÓN

6.1 Revisiones y auditorías de diseño

- Las revisiones de diseño pretenden ser exámenes exhaustivos de un diseño para evaluar inquietudes como el estado o grado de finalización, la cobertura de requisitos, cuestiones abiertas o no resueltas y problemas potenciales. Se puede realizar una revisión del diseño en cualquier etapa del diseño. Las revisiones de diseño pueden ser realizadas por el equipo de diseño, por un tercero independiente u otra parte interesada.
- Una auditoría de diseño se centra más estrictamente en una lista establecida de características (por ejemplo, una auditoría funcional).

6.2 Atributos de Calidad

- Varios atributos contribuyen a la calidad de un diseño de software, incluidas varias “capacidades” (mantenibilidad, portabilidad, capacidad de prueba, usabilidad) y “capacidades” (corrección, solidez).
- Las cualidades son un subconjunto importante de preocupaciones
- Algunas cualidades se pueden observar en tiempo de ejecución (por ejemplo, rendimiento, seguridad, disponibilidad, funcionalidad, usabilidad), otros no pueden (por ejemplo, modificabilidad, portabilidad, reutilización, capacidad de prueba), algunos (por ejemplo, integridad conceptual, corrección, integridad) son intrínsecos al software.

6.3 Técnicas de análisis y evaluación de la calidad

- Varias herramientas y técnicas pueden ayudar a analizar y evaluar la calidad del diseño de software.
- Las revisiones de diseño de software incluyen revisiones informales y rigurosas. técnicas para determinar las cualidades del software basadas en SDD y otros artefactos de diseño, por ejemplo, revisiones de arquitectura, revisiones e inspecciones de diseño, técnicas basadas en escenarios, seguimiento de requisitos.

- Análisis estático: análisis estático (no ejecutable) formal o semiformal que se puede utilizar para evaluar un diseño (por ejemplo, análisis de árbol de fallas o verificación cruzada automatizada). Se puede realizar un análisis de vulnerabilidad del diseño (por ejemplo, análisis estático de debilidades de seguridad) si la seguridad es una preocupación. El análisis de diseño formal utiliza modelos matemáticos que permiten a los diseñadores predecir el comportamiento y validar el rendimiento del software en lugar de tener que depender exclusivamente de las pruebas.

- Cuestiones clave en El análisis de diseño formal se puede utilizar para detectar errores residuales de especificación y diseño (quizás causados por imprecisión, ambigüedad y, a veces, otros tipos de errores).
- Simulación y creación de prototipos: técnicas dinámicas para evaluar un diseño (por ejemplo, simulación de rendimiento o prototipos de viabilidad).

6.4 Medidas y Métricas

- Las medidas se pueden utilizar para evaluar o estimar cuantitativamente varios aspectos de un diseño de software, por ejemplo, tamaño, estructura o calidad. La mayoría de las medidas que se han propuesto se basan en el enfoque utilizado para producir el diseño
- Estas medidas se clasifican en dos grandes categorías:

- Medidas de diseño (estructuradas) basadas en funciones: medidas obtenidas analizando la descomposición funcional, generalmente se representa mediante un gráfico de estructura (o diagrama jerárquico) en el que se pueden calcular varias medidas.

- Medidas de diseño orientadas a objetos: la estructura de diseño normalmente se representa como un diagrama de clases, en el que se pueden calcular varias medidas. También se pueden calcular medidas sobre las propiedades del contenido interno de cada clase

Verificación, Validación y Certificación

- El análisis o evaluación sistemática del diseño juega un papel papel importante en cada una de estas tres áreas:
- verificación: para confirmar que el diseño satisface lo indicado requisitos.
- validación: para establecer que el diseño permitirá que el sistema cumpla con las expectativas de sus partes interesadas, incluidos clientes, usuarios, operadores y mantenedores, certificación: certificación de un tercero de la conformidad del diseño con sus especificaciones generales y su uso previsto.

Referencias

- Bourque, P., & Fairley, R. E. (Eds.). (2023). SWEBOK: Guide to the software engineering body of knowledge (Version 4.0). IEEE Computer Society.
- About SWEBoK:

https://www.computer.org/education/bodies-of-knowledge/software-engineering

GRACIAS

Ingeniería de Sistemas
Universidad Nacional de Cajamarca