DOMUMENTACION DISEÑO DEL SOFWARE

APRENDIZES:
KAREN JOHANA PALACIOS
JUANCARLOS MONROY
OYIUSO ANDERZO NEUTO GONZALEZ

INSTRUCTORES:

DORIS GONZALEZ
JESUS ARIEL GONZALEZ

ANALISI Y DESARROLLO DEL SOFTWARE

SENA
NEIVA-HUILA

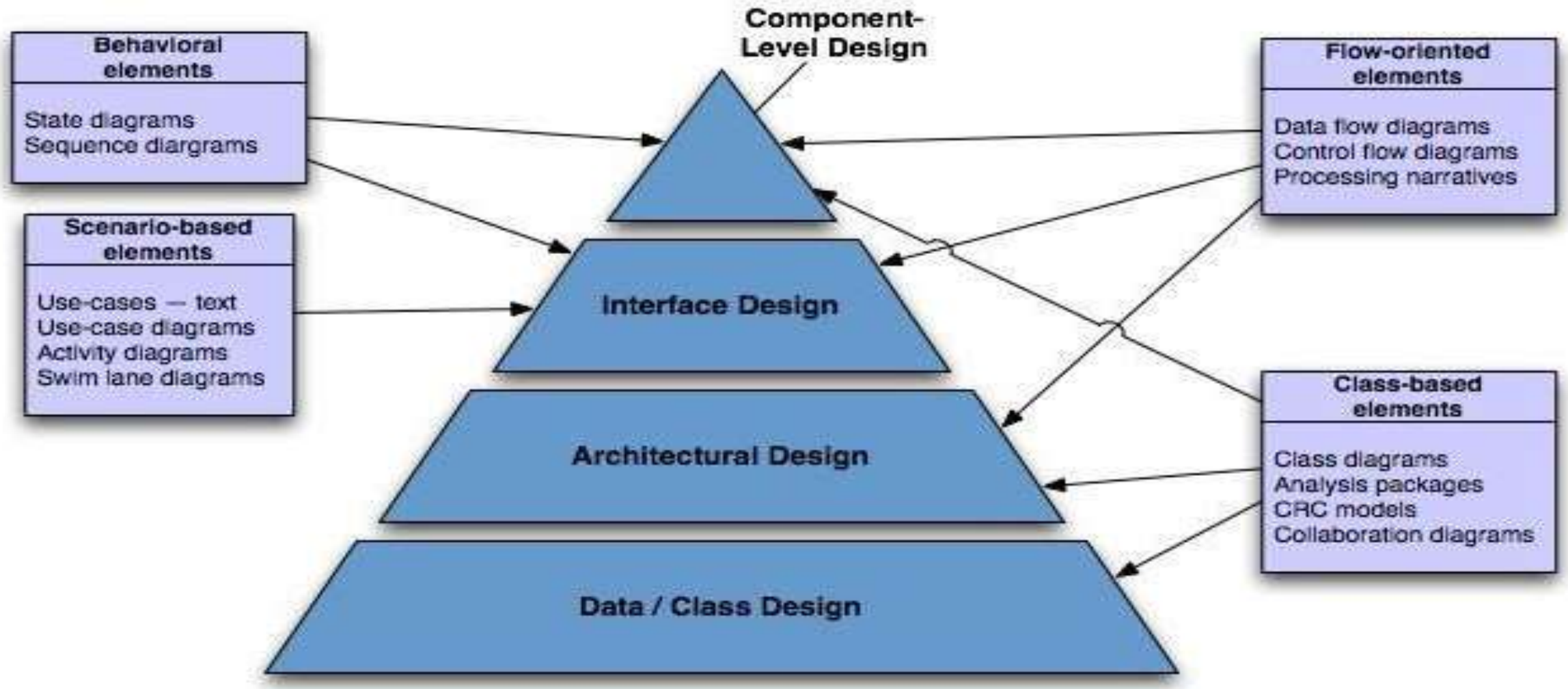
Diseño del software Foto Mago

- Cubre el conjunto de principios, conceptos y prácticas que conducen al desarrollo de un sistema o producto de alta calidad.
- El objetivo del diseño es producir un modelo o representación que represente:
- Firmeza: el programa no debe tener ningún error que inhiba sus funciones.
- Mercancía – adecuada para su uso previsto.
- Delicia - placentero de usar
- El modelo de diseño proporciona detalles sobre las estructuras de datos de software, la arquitectura, las interfaces y los componentes que son necesarios para implementar el sistema.

El modelo de diseño de software Foto Mago esta basado en 4 diseños:

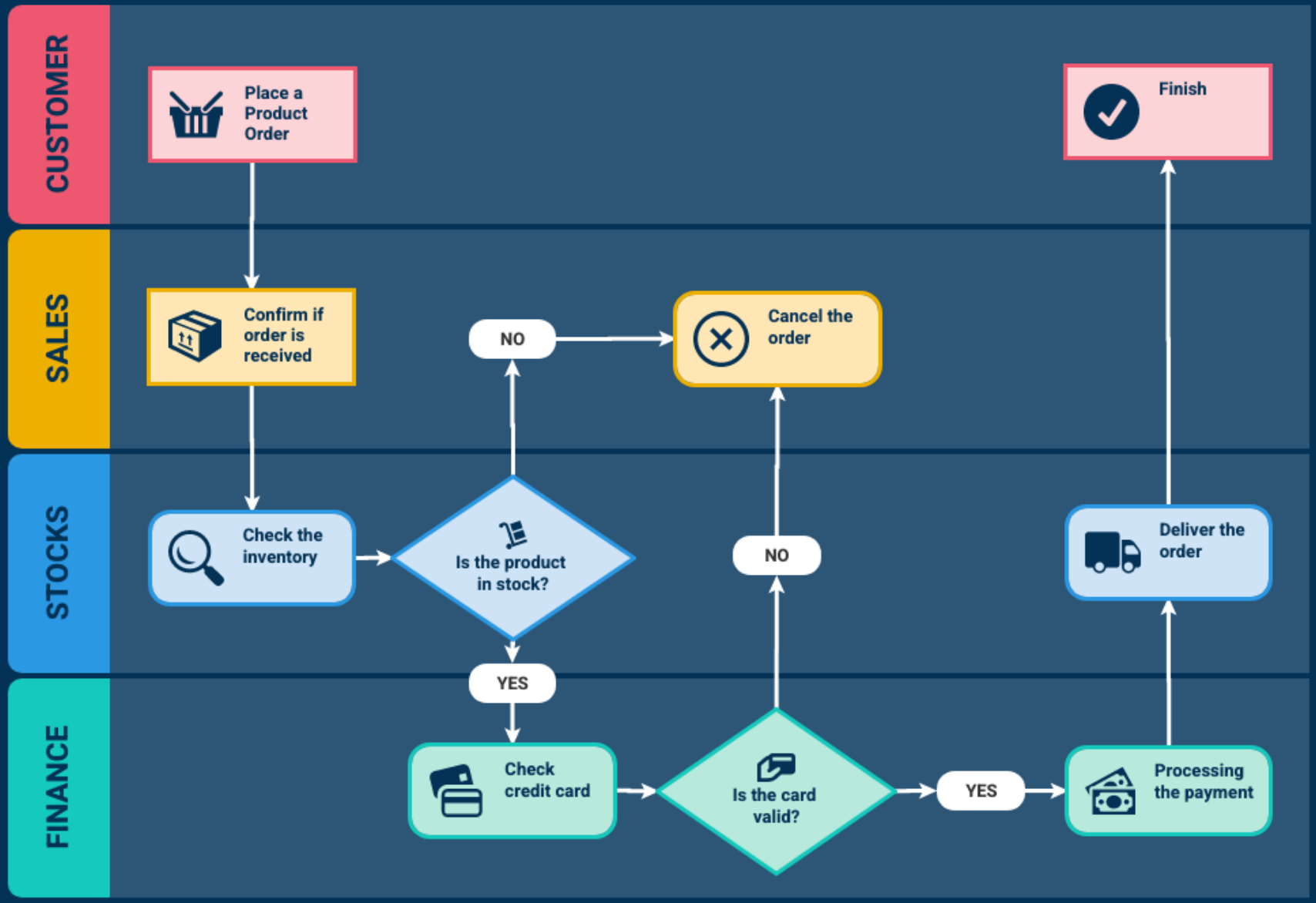
- Diseño de datos/clases
- Diseño Arquitectónico
- Diseño de interfaz
- Diseño de componentes

Diagrama de diseño



Sales Process Swimlane Flowchart

ZOOM RABBIT



- Diseño de datos/clases: creado mediante la transformación de los elementos basados en clases del modelo de análisis en clases y estructuras de datos necesarias para implementar el software

- Diseño arquitectónico: define las relaciones entre los principales elementos estructurales del software, se deriva de los elementos basados en clases y los elementos orientados al flujo del modelo de análisis.

- Diseño de interfaz: describe cómo los elementos de software, los elementos de hardware y los usuarios finales se comunican entre sí, se deriva de los elementos basados en escenarios del modelo de análisis, los elementos orientados al flujo y los elementos de comportamiento.

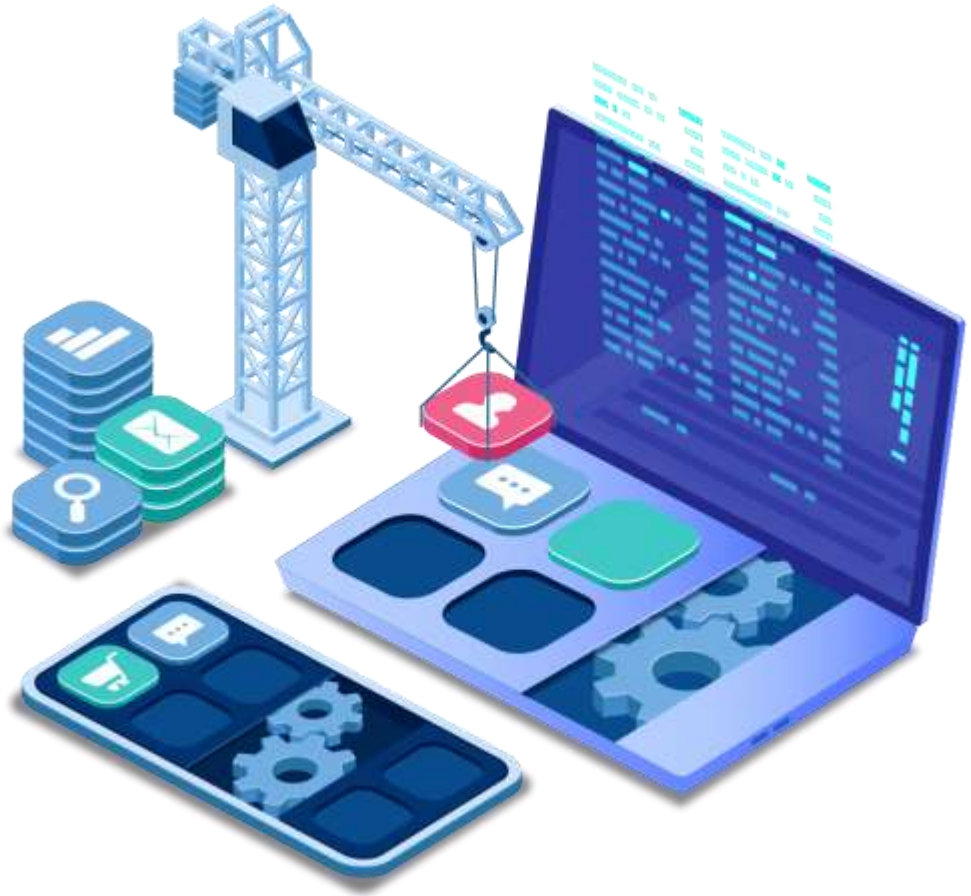
- Diseño a nivel de componente: creado mediante la transformación de los elementos estructurales definidos por la arquitectura de software en una descripción procedimental de los componentes de software utilizando información obtenida del modelo de análisis de elementos basados en clases, elementos orientados al flujo y elementos de comportamiento.

¿Por qué el diseño es tan importante para Foto mago?

- Es un lugar donde se fomenta la calidad y creatividad de nuestros desarrolladores.
- Nos proporciona una representación de software que se puede evaluar por su calidad.
- La única forma en que puede traducir con precisión los requisitos de un cliente en un producto de software terminado.
- Sirve como base para todas las actividades de ingeniería de software.
- Sin diseño difícil de evaluar:
- Riesgo:
- Prueba
- Calidad

Proceso de diseño y calidad del diseño

- El diseño S/w es un proceso iterativo a través del cual los requisitos se traducen en un "plano" para construir el s/w.
- A medida que se produce la iteración del diseño, el refinamiento posterior conduce a la representación del diseño en niveles mucho más bajos de abstracción.



Objetivo del proceso de diseño

- El diseño debe implementar todos los requisitos explícitos contenidos en el modelo de análisis, y debe acomodar todos los requisitos implícitos deseados por el cliente.
- El diseño debe ser una guía legible y comprensible para aquellos que generan código y para aquellos que prueban y posteriormente soportan el software.
- El diseño debe proporcionar una imagen completa del software, abordando los dominios de datos, funcionales y de comportamiento desde una perspectiva de implementación.

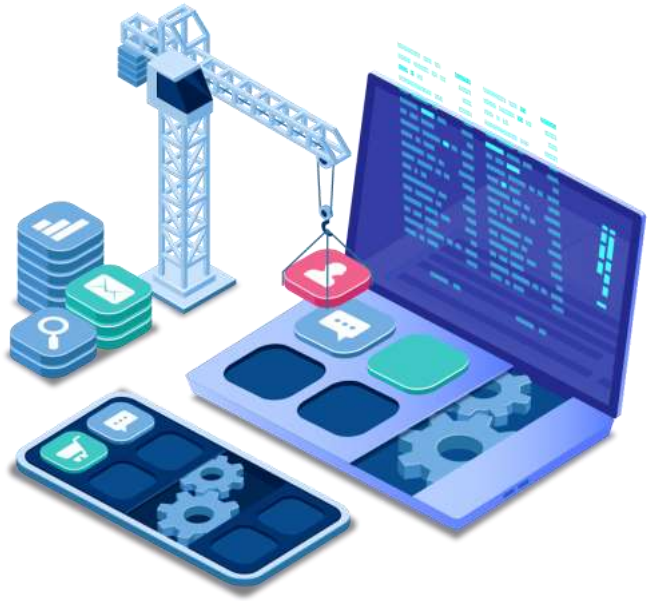
Directrices de calidad

Características del buen diseño

- Un diseño debe exhibir una arquitectura que:
  - Tal como se ha creado utilizando estilos o patrones arquitectónicos reconocibles.
  - se compone de componentes que exhiben buenas características de diseño y
  - se puede implementar de manera evolutive.
- Un diseño debe ser modular; es decir, el software debe estar particionado lógicamente en elementos o subsistemas
- Un diseño debe contener representaciones distintas de datos, arquitectura, interfaces y componentes.
- Un diseño debe conducir a estructuras de datos que sean apropiadas para las clases que se implementarán y que se extraigan de patrones de datos reconocibles.
- Un diseño debe conducir a componentes que exhiban características funcionales independientes.

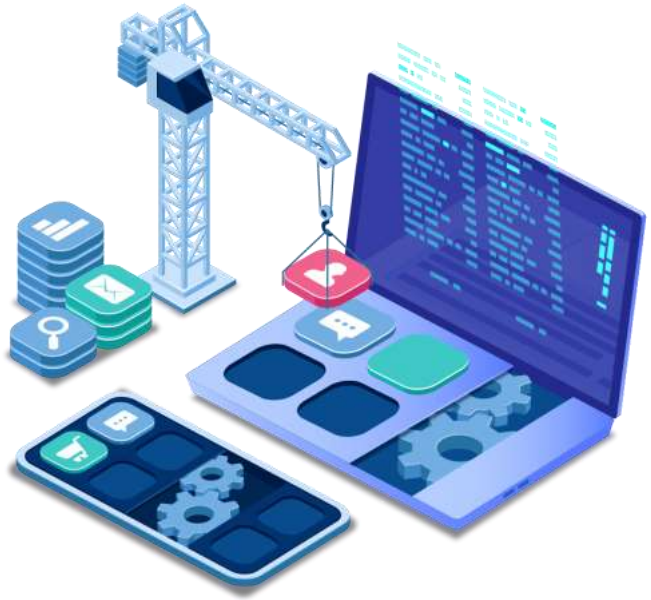
Directrices de calidad (contd.)

- Un diseño debe conducir a interfaces que reduzcan la complejidad de las conexiones entre componentes y con el entorno externo.
- Un diseño debe derivarse utilizando un método repetible que se basa en la información obtenida durante el análisis de los requisitos de software.
- Un diseño debe representarse utilizando una notación que comunique efectivamente su significado.

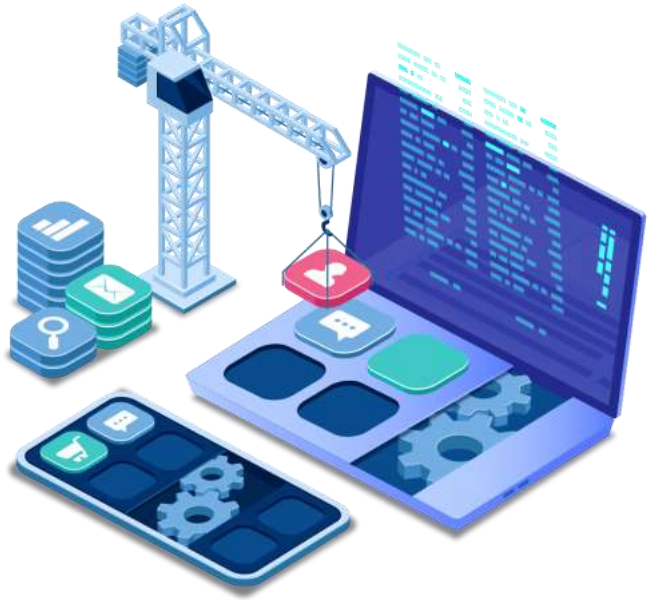


Principios de diseño

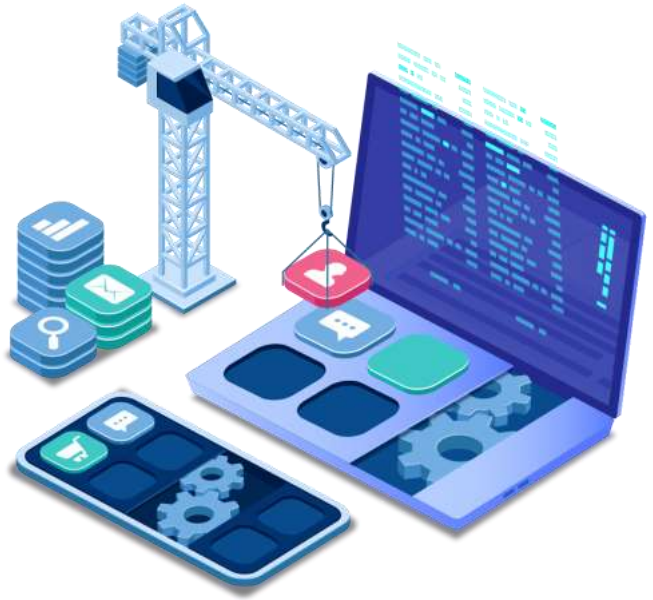
- El diseño S/W es tanto un proceso como un modelo.
- Proceso de diseño: secuencia de pasos que permiten al diseñador describir todos los aspectos del software que se va a construir.
- Modelo de diseño: creado para software proporciona una variedad de vistas diferentes del software de la computadora.



- *El proceso de diseño no debe sufrir de "visión de túnel". - El diseñador debe considerar enfoques alternativos.*
- *El diseño debe ser trazable hasta el modelo de análisis: un solo elemento del modelo de diseño a menudo se basa en múltiples requisitos, es necesario tener un medio para rastrear cómo se han satisfecho los requisitos del modelo de diseño.*
- *El diseño no debe reinventar la rueda: el uso ya existe un patrón de diseño porque el tiempo es corto y los recursos son limitados.*
- *El diseño debe "minimizar la distancia intelectual" entre el software y el problema tal como existe en el mundo real. – el diseño debe ser autoexplicativo*

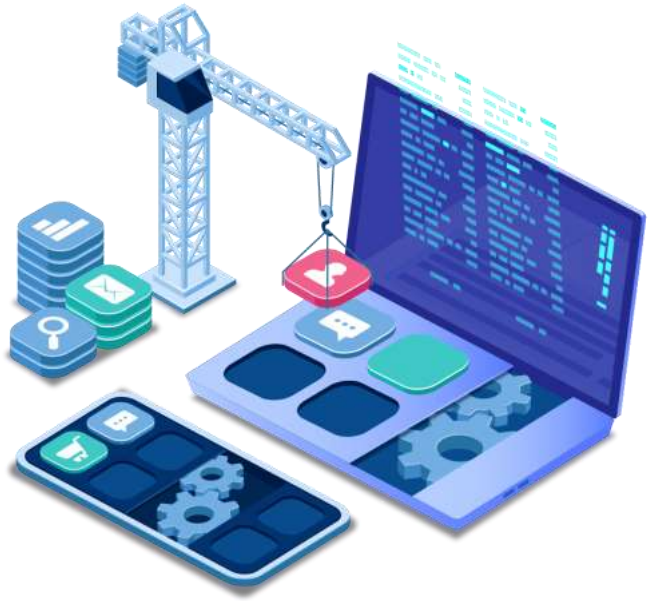


- *El diseño debe exhibir uniformidad e integración: antes de que comience el trabajo de diseño, se deben definir reglas de estilos y formato para un equipo de diseño.*
- *El diseño debe estar estructurado para adaptarse al cambio*
- *El diseño debe estructurarse para degradarse suavemente, incluso cuando se encuentren datos, eventos o condiciones de operación inusuales.*
- *El diseño no es codificación, la codificación no es diseño.*
- *El diseño debe ser evaluado por su calidad a medida que se está creando, no después del hecho.*
- *El diseño debe revisarse para minimizar los errores conceptuales (semánticos).*



Conceptos de diseño

- Los conceptos de diseño proporcionan el marco necesario para "poner la cosa en el camino correcto".
  - Abstracción
  - Refinamiento
  - Modularidad
  - Arquitectura
  - Jerarquía de control
  - Particionamiento estructural
  - Estructura de datos
  - Procedimiento de software
  - Ocultación de información

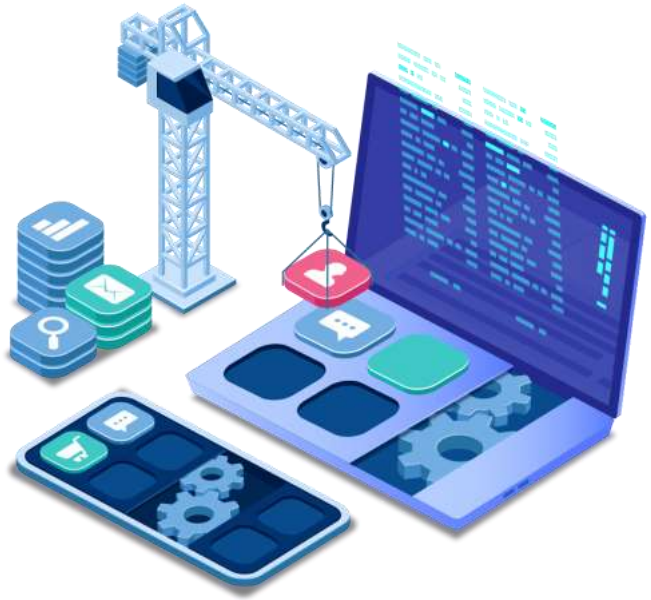


Abstracción

- En el más alto nivel de abstracción, una solución se expresa en términos amplios.
- En el nivel más bajo de abstracción, se proporciona una descripción más detallada de la solución.
- Dos tipos de abstracción:
- Abstracción procedimental: Secuencia de instrucciones que tienen una función específica y limitada.
  - Ej. Abrir una puerta
- abrir implica una larga secuencia de actividades (por ejemplo, caminar hacia la puerta, agarrar la perilla, girar la perilla y tirar de la puerta, etc.).
- Abstracción de datos: recopilación de datos que describen un objeto de datos.
  - Ej. Abre una puerta. – puerta es objeto de datos.
- La abstracción de datos para la puerta abarcaría un conjunto de atributos que describen la puerta. (Por ejemplo, tipo de puerta, dirección de giro, mecanismo de apertura, etc.)

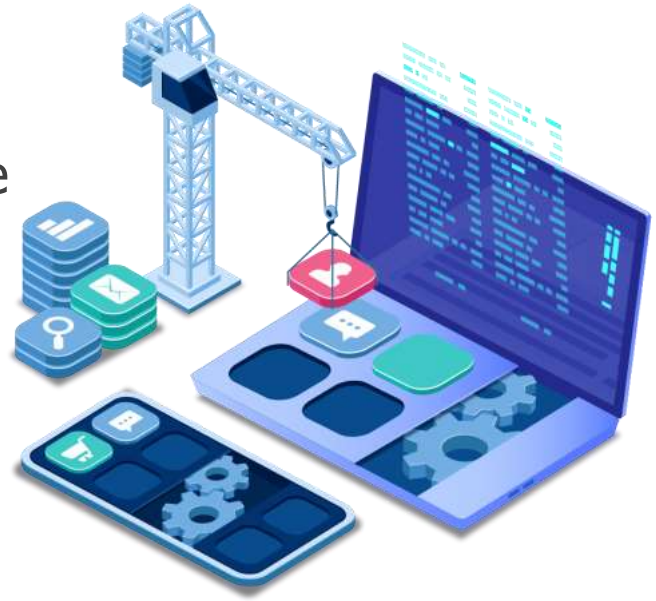
Refinamiento

- El refinamiento es en realidad un proceso de elaboración.
- comience con una declaración de función (o descripción de la información) que se define en un alto nivel de abstracción.
- Es decir, la declaración describe la función o la información conceptualmente, pero no proporciona información sobre el funcionamiento interno de la función o la estructura interna de la información.
- El refinamiento hace que el diseñador elabore la declaración original, proporcionando más y más detalles a medida que se produce cada refinamiento (elaboración) sucesivo.
- La abstracción y el refinamiento son conceptos complementarios. La abstracción permite a un diseñador especificar procedimientos y datos y, sin embargo, suprimir detalles de bajo nivel.
- El refinamiento ayuda al diseñador a exponer detalles de bajo nivel a medida que avanza el diseño.

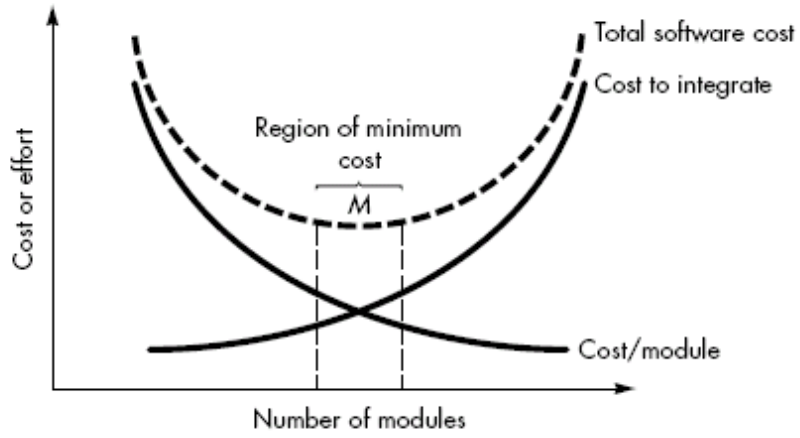


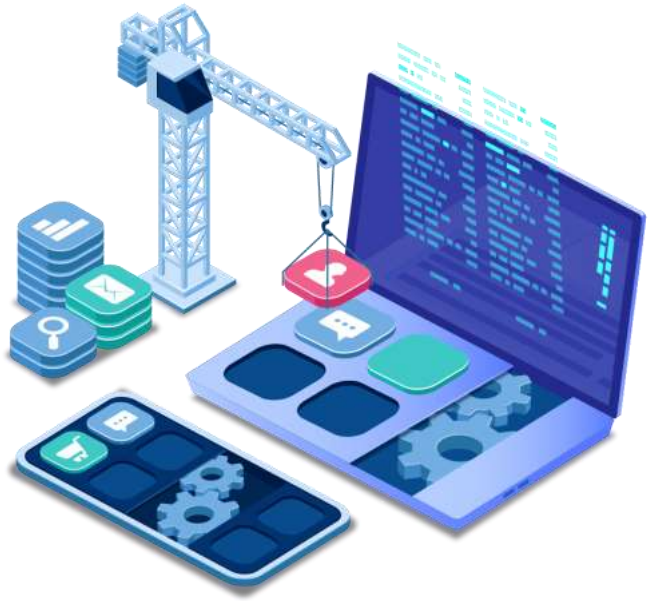
Modularidad

- La arquitectura y el patrón de diseño encarnan la modularidad.
- El software se divide en componentes con nombre y direccionables por separado, a veces llamados módulos, que se integran para satisfacer los requisitos del problema.
- La modularidad es el atributo único del software que permite que un programa sea intelectualmente manejable
- Conduce a una estrategia de "divide y vencerás". – es más fácil resolver un problema complejo cuando se rompe en un pedazo manejable.
- Consulte la fig. que establecen que el esfuerzo (costo) para desarrollar un módulo de software individual disminuye si aumenta el número total de módulos.
- Sin embargo como el no. de los módulos crece, el esfuerzo (costo) asociado con la integración de los módulos también crece.



Modularidad y coste del software

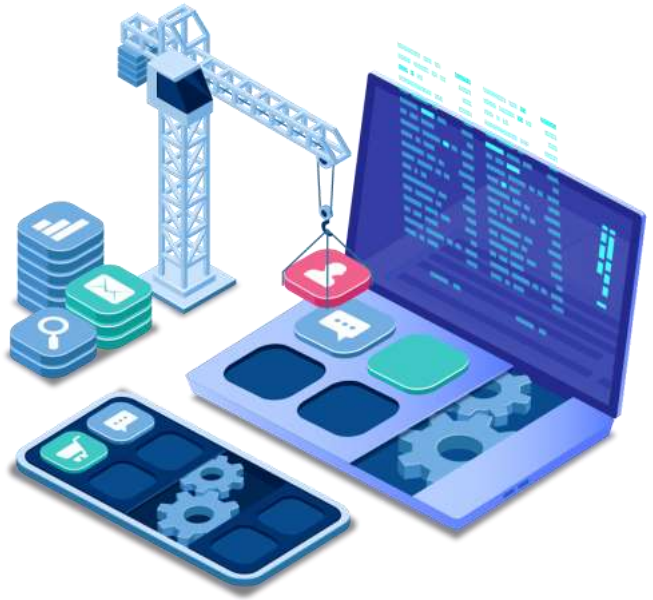




- Se debe evitar la falta de modalidad y el exceso de modalidad. Pero, ¿cómo conocemos la vecindad de M?
- Modularizamos un diseño para que el desarrollo se pueda planificar más fácilmente.
- Los incrementos de software se pueden definir y entregar.
- Los cambios se pueden acomodar más fácilmente.
- Las pruebas y la depuración se pueden realizar de manera más eficiente y se puede mantener a largo plazo sin efectos secundarios graves.

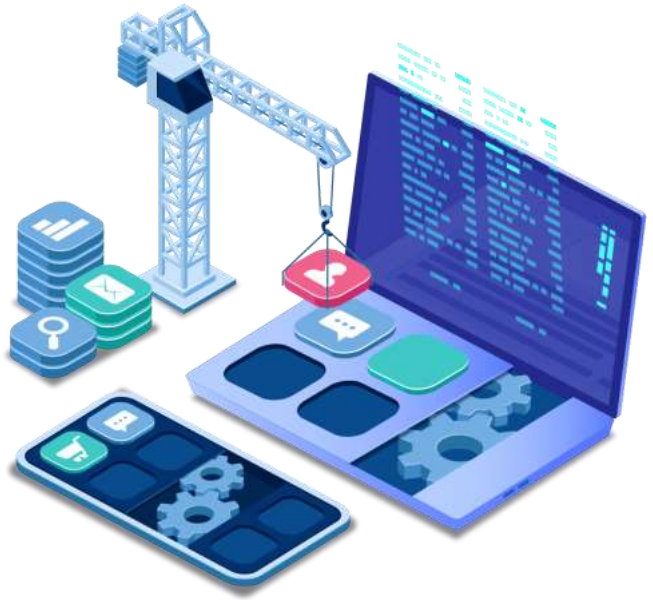
Arquitectura

- La arquitectura de software sugiere " la estructura general del software y las formas en que esa estructura proporciona integridad conceptual para un sistema.
- No. de diferentes modelos se puede utilizar para representar la arquitectura.
  - Modelo estructural: representa la arquitectura como una colección organizada de componentes
  - Modelo de marco: aumente el nivel de abstracción del diseño mediante la identificación de un marco de diseño arquitectónico repetible.
  - Modelo dinámico: aborda el comportamiento de la arquitectura del programa e indica cómo la configuración del sistema o la estructura puede cambiar como una función.
  - Modelo de proceso: concéntrese en el diseño del proceso comercial o técnico que el sistema debe acomodar.
  - Modelos funcionales: se utilizan para representar la jerarquía funcional de un sistema.



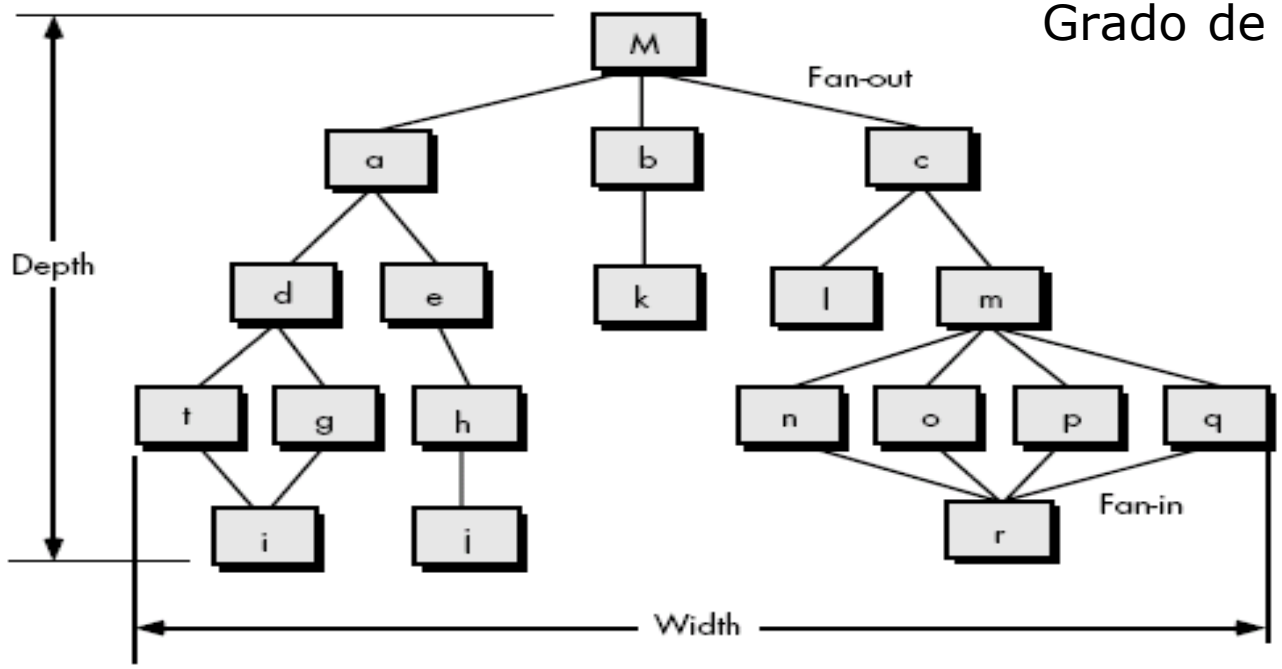
Jerarquía de control

- Jerarquía de control, representa la organización de los componentes del programa (módulos) e implica una jerarquía de control.
- Se utilizan diferentes notaciones para representar la jerarquía de control para aquellos estilos arquitectónicos que son susceptibles de esta representación.
- El más común es el diagrama en forma de árbol que representa el control jerárquico para las arquitecturas de llamada y retorno.



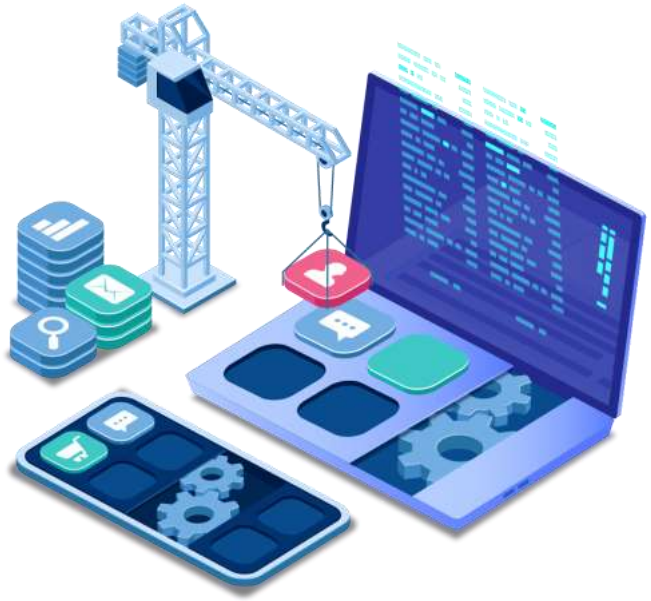
Jerarquía de control

Profundidad



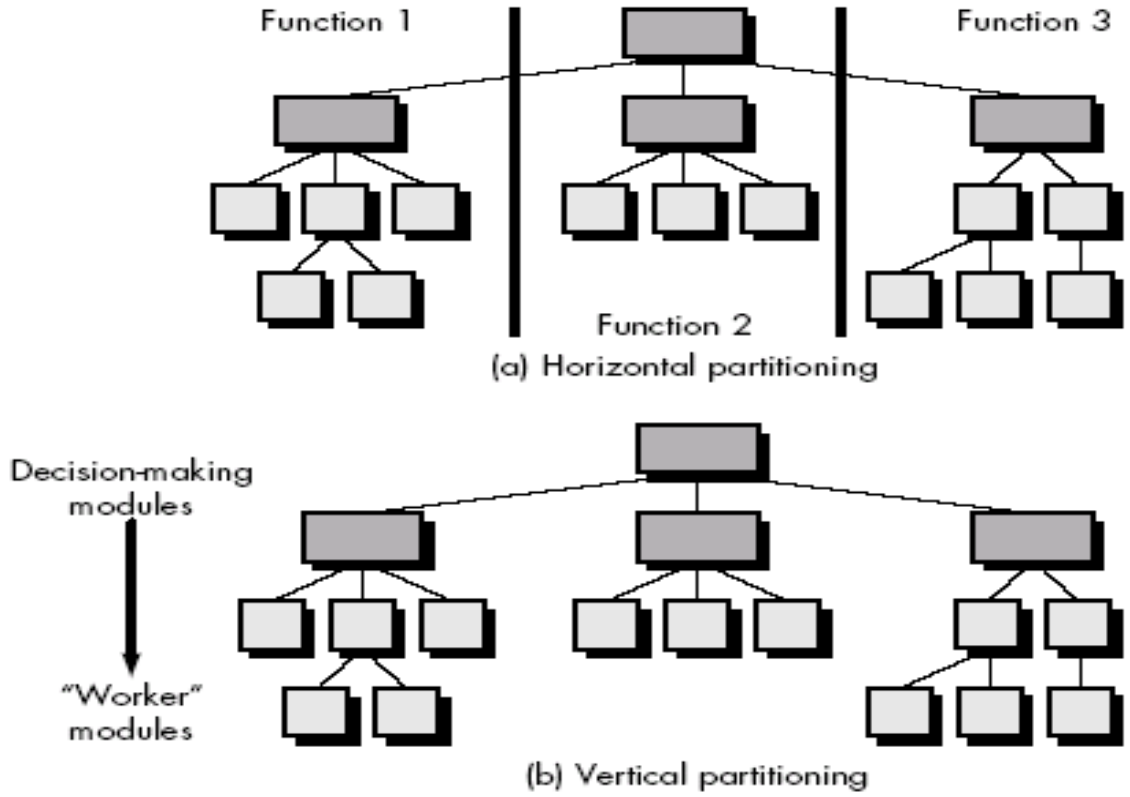
Grado de salida

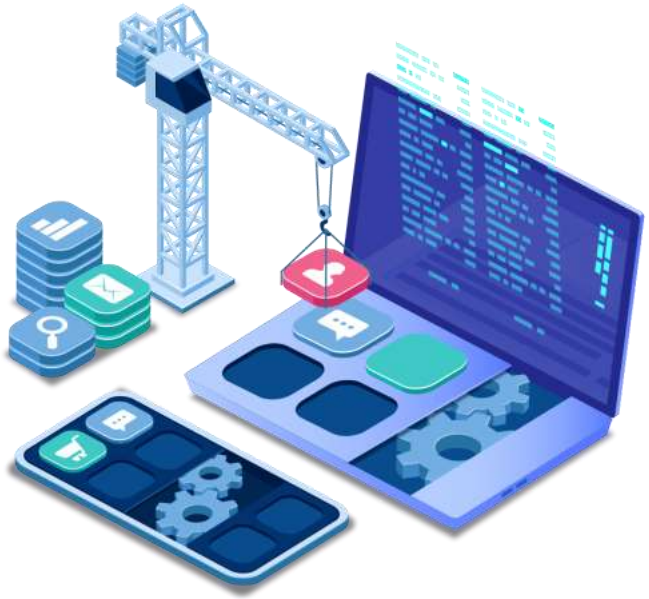
Anchura



- En la fig. la profundidad y el ancho proporcionan una indicación del número de niveles de control y el alcance general del control.
- Fan-out es una medida del número de módulos que son controlados directamente por otro módulo. Fan-in indica cuántos módulos controlan directamente un módulo determinado.
- Se dice que un módulo que controla otro módulo es superior a él, y a la inversa, se dice que un módulo controlado por otro está subordinado al controlador.

Particionamiento estructural

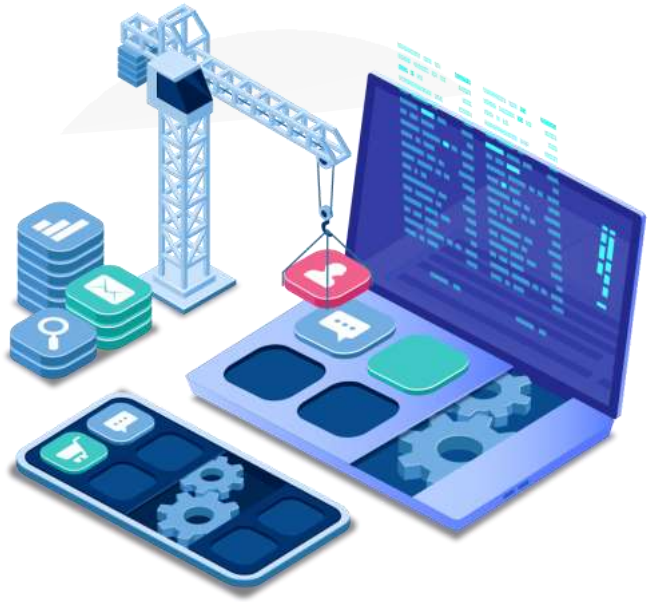




Particionamiento de estructuras

- Dos tipos de particionamiento de estructuras:
  - Particionamiento horizontal
  - Particionamiento vertical
- La partición horizontal define ramas separadas de la jerarquía modular para cada función principal del programa.
- Los módulos de control, representados en un tono más oscuro, se utilizan para coordinar la comunicación y la ejecución entre sus funciones.
- La partición horizontal define tres particiones: entrada, transformación de datos (a menudo llamada procesamiento) y salida.
- Benefits of Horizontal partitioning:
  - software que es más fácil de probar
  - software que es más fácil de mantener
  - propagación de menos efectos secundarios
  - software que es más fácil de extender.

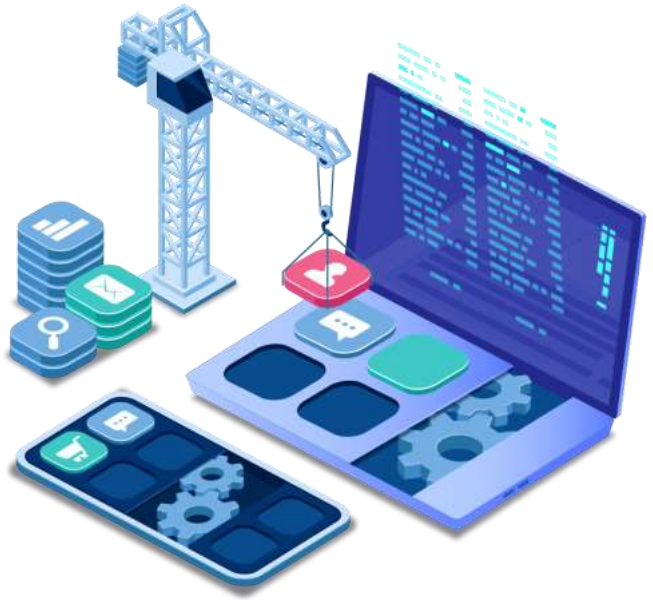
- En el lado negativo (inconveniente), la partición horizontal a menudo hace que se pasen más datos a través de las interfaces del módulo y puede complicar el control general del flujo del programa.
- La partición vertical, a menudo llamada factorización, sugiere que el control (toma de decisiones) y el trabajo deben distribuirse de arriba hacia abajo en la estructura del programa.
- Los módulos de nivel superior deben realizar funciones de control y hacer poco trabajo de procesamiento real.
- Los módulos que residen bajo en la estructura deben ser los trabajadores, realizando todas las tareas de entrada, cómputo y salida.



- Un cambio en un módulo de control (alto en la estructura) tendrá una mayor probabilidad de propagar efectos secundarios a módulos que están subordinados a él.
- Un cambio en un módulo de trabajo, dado su bajo nivel en la estructura, es menos probable que cause la propagación de efectos secundarios.
- Por esta razón, las estructuras divididas verticalmente tienen menos probabilidades de ser susceptibles a los efectos secundarios cuando se realizan cambios y, por lo tanto, serán más fáciles de mantener.
-

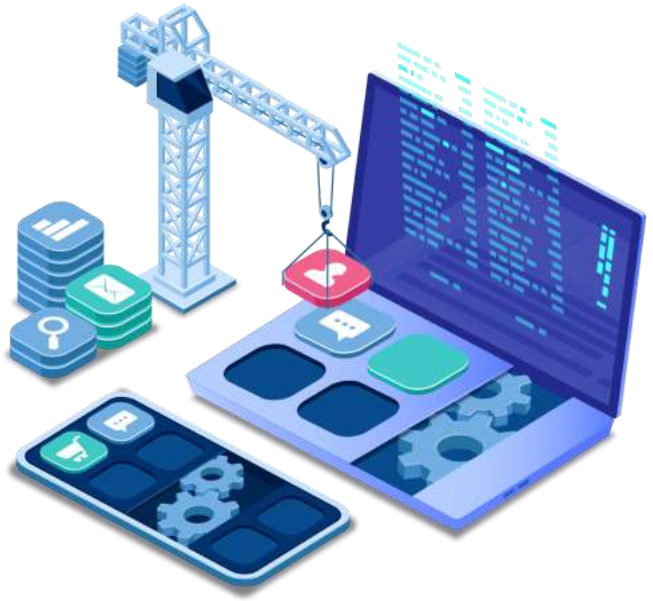
Estructura de datos

- La estructura de datos es una representación de la relación lógica entre los elementos individuales de los datos.
- Un elemento escalar es la más simple de todas las estructuras de datos. Representa un único elemento de información que puede ser abordado por un identificador.
- Cuando los elementos escalares se organizan como una lista o grupo contiguo, se forma un vector secuencial.
- Cuando el vector secuencial se extiende a dos, tres y, en última instancia, un número arbitrario de dimensiones, se crea un espacio n-dimensional. El espacio n-dimensional más común es la matriz bidimensional
- Una lista vinculada es una estructura de datos que organiza elementos escalares, vectores o espacios contiguos de una manera (llamada nodos) que permite procesarlos como una lista.
- Una estructura de datos jerárquica se implementa utilizando listas multienlazadas que contienen elementos escalares, vectores y, posiblemente, espacios n-dimensionales.



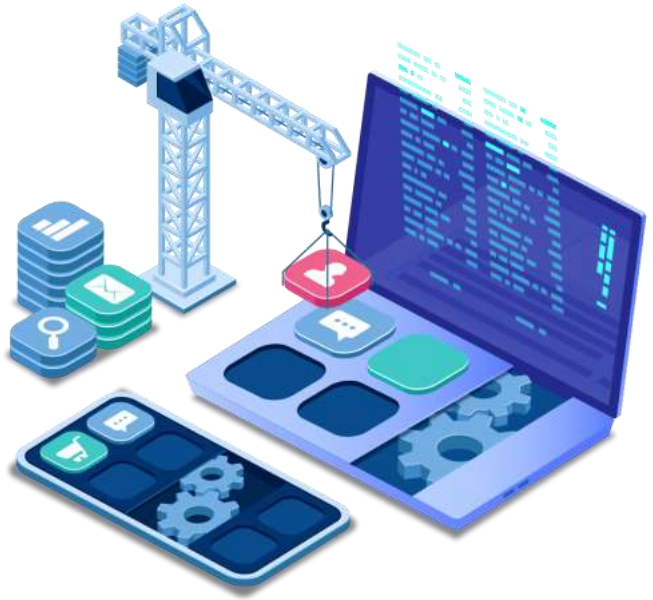
Procedimiento de software

- El procedimiento de software se centra en los detalles de procesamiento de cada módulo individualmente.
- El procedimiento debe proporcionar una especificación precisa del procesamiento, incluida la secuencia de eventos, los puntos de decisión exactos, las operaciones repetitivas e incluso la organización y estructura de los datos.



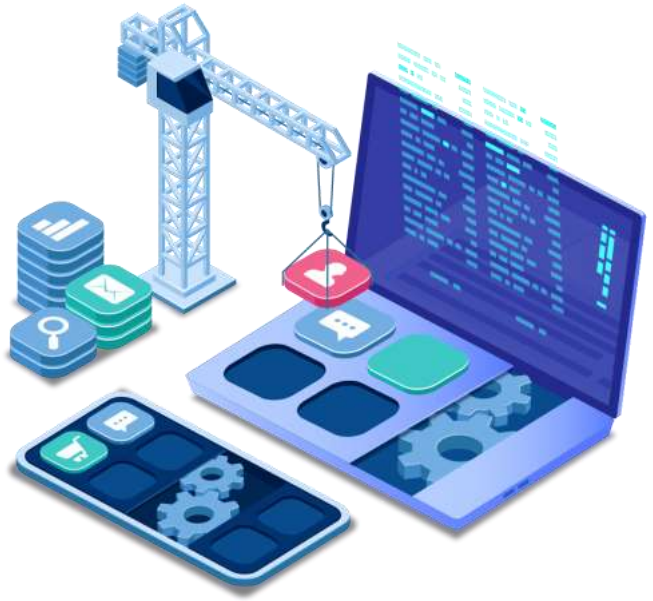
Ocultación de información

- El principio de ocultación de información sugiere que los módulos se "caractericen por decisiones de diseño que (cada uno) se oculta de todos los demás módulos".
- En otras palabras, los módulos deben especificarse y diseñarse de modo que la información (algoritmo y datos) contenida en un módulo sea inaccesible para otros módulos que no tienen necesidad de dicha información.
- La intención de la ocultación de información es ocultar los detalles de la estructura de datos y el procesamiento de procedimientos detrás de una interfaz de módulo.
- Brinda beneficios cuando se requieren modificaciones durante las pruebas y el mantenimiento porque los datos y el procedimiento se ocultan de otras partes del software, los errores involuntarios introducidos durante la modificación son menores.



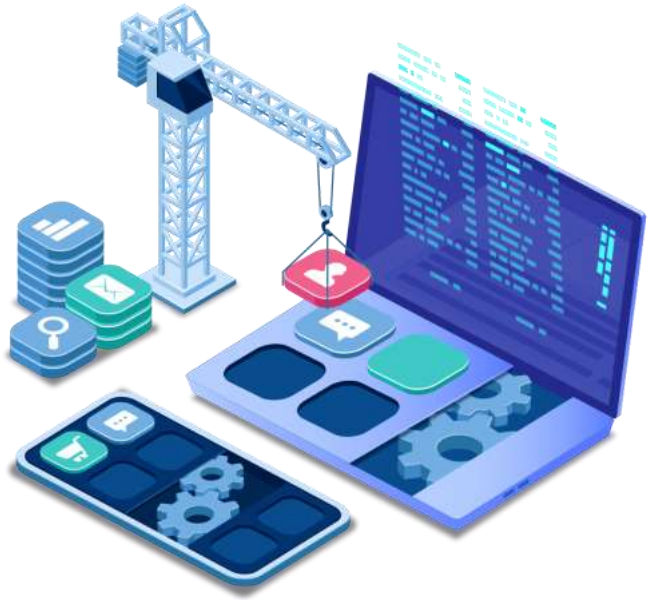
DISEÑO MODULAR EFECTIVO

- El diseño modular efectivo consta de tres cosas:
  - Independencia funcional.
  - Cohesión.
  - Acoplamiento.



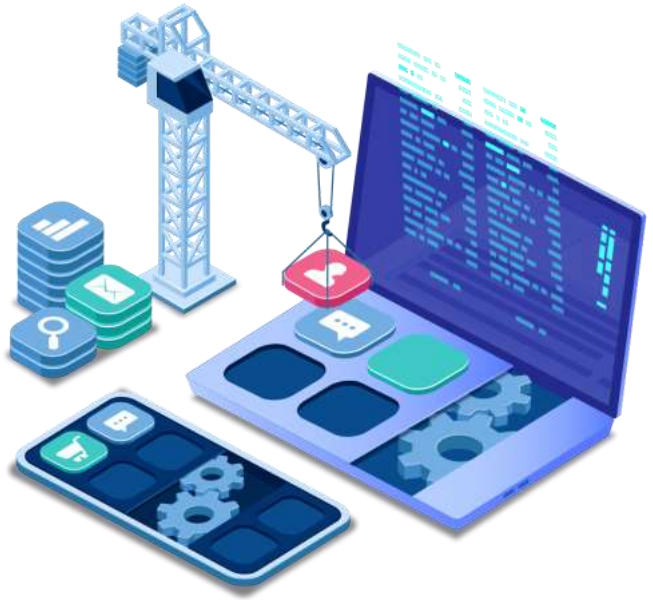
Independencia funcional

- La independencia funcional se logra mediante el desarrollo de módulos con función "única" y una "aversión" a la interacción excesiva con otros módulos.
- En otras palabras, cada módulo aborda una subfunción específica de requisitos y tiene una interfaz simple cuando se ve desde otras partes de la estructura del programa.
- La independencia es importante –
  - Más fácil de desarrollar
  - Más fácil de probar y mantener
  - Se reduce la propagación de errores
  - Módulo reutilizable.



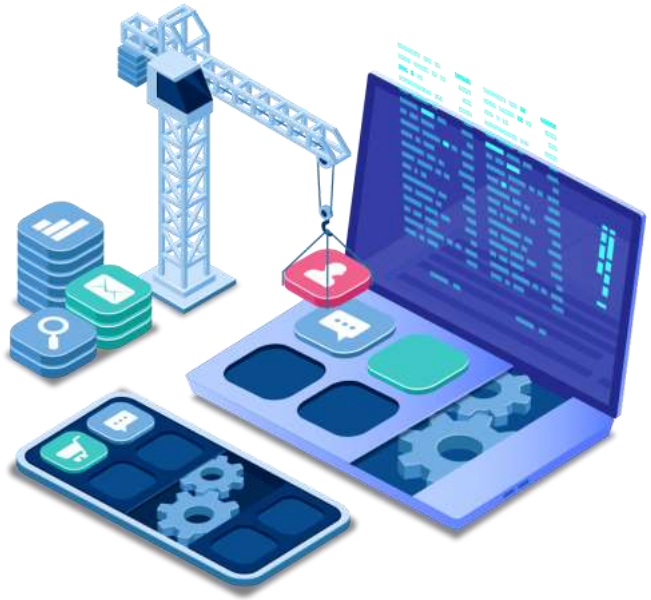
Independencia funcional

- En resumen, la independencia funcional es una clave para un buen diseño, y el diseño es la clave para la calidad del software.
- Para medir la independencia, se tienen dos criterios cualitativos: cohesión y acoplamiento
- La cohesión es una medida de la fuerza funcional relativa de un módulo.
- El acoplamiento es una medida de la interdependencia relativa entre los módulos.



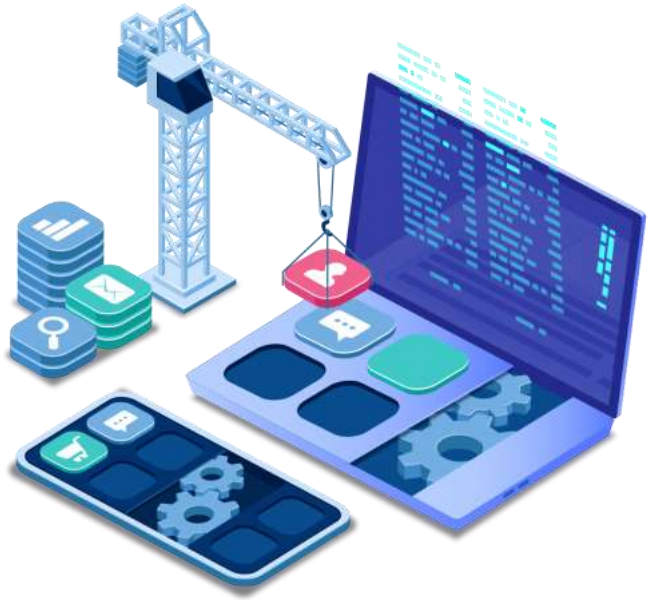
Cohesión

- La cohesión es una extensión natural del concepto de ocultación de información
- Un módulo cohesivo realiza una sola tarea dentro de un procedimiento de software, lo que requiere poca interacción con los procedimientos que se realizan en otras partes de un programa.
- Simplemente diga, un módulo cohesivo debería (idealmente) hacer una sola cosa.
- Siempre nos esforzamos por lograr una alta cohesión, aunque el rango medio del espectro es a menudo aceptable.
- La cohesión de gama baja es mucho "peor" que la de gama media, que es casi tan "buena" como la cohesión de gama alta.
- Por lo tanto, el diseñador debe evitar los bajos niveles de cohesión cuando se diseñan los módulos.



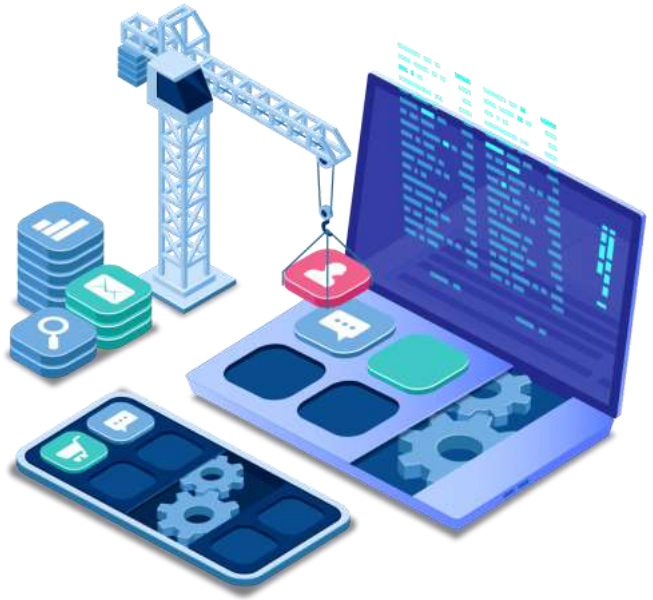
Cohesion

- Cuando los elementos de procesamiento de un módulo están relacionados y deben ejecutarse en un orden específico, existe cohesión procesal.
- Cuando todos los elementos de procesamiento se concentran en un área de una estructura de datos, la cohesión comunicacional está presente.
- La alta cohesión se caracteriza por un módulo que realiza una tarea de procedimiento distinta.



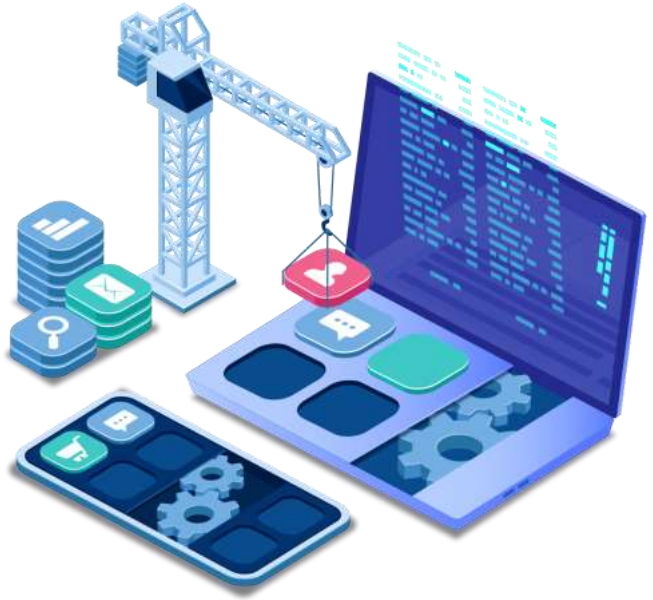
Tipos de cohesión

- Un módulo que realiza tareas que están relacionadas lógicamente es lógicamente cohesivo.
- Cuando un módulo contiene tareas que están relacionadas por el hecho de que todas deben ejecutarse con el mismo lapso de tiempo, el módulo exhibe cohesión temporal.
- En el extremo inferior del espectro, un módulo que realiza un conjunto de tareas que se relacionan entre sí de manera vaga, llamadas casualmente cohesivas..

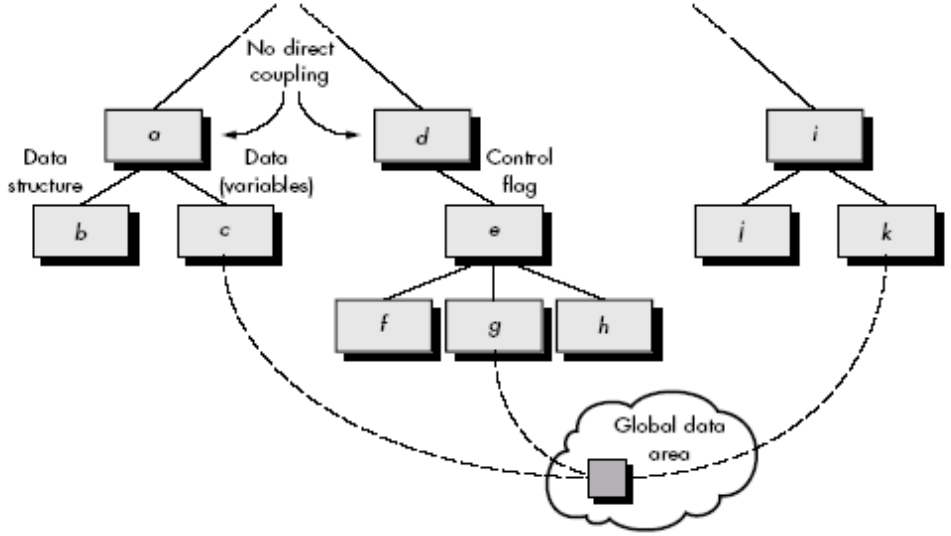


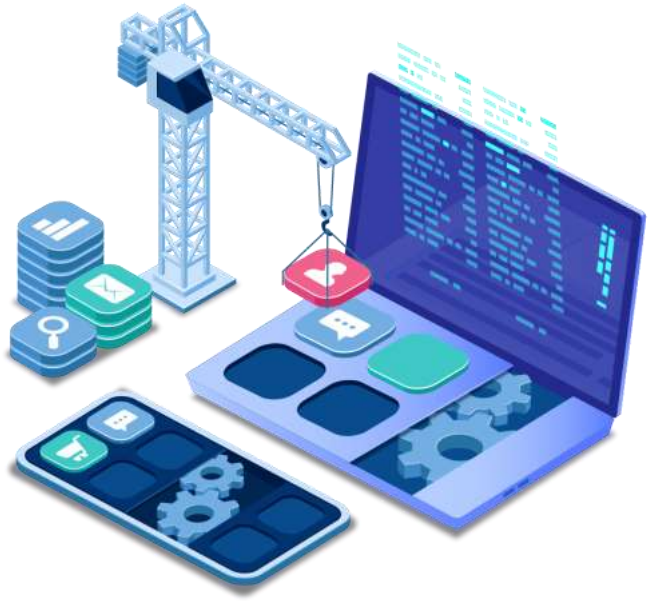
Acoplamiento

- El acoplamiento depende de la complejidad de la interfaz entre módulos, el punto en el que se realiza la entrada o referencia a un módulo y los datos que pasan a través de la interfaz.
- En el diseño de software, nos esforzamos por lograr el menor acoplamiento posible. La conectividad simple entre los módulos da como resultado un software que es más fácil de entender y menos propenso a un "efecto dominó" causado cuando los errores ocurren en una ubicación y se propagan a través de un sistema.
- Ocurre debido a las decisiones de diseño tomadas cuando se desarrolló la estructura.



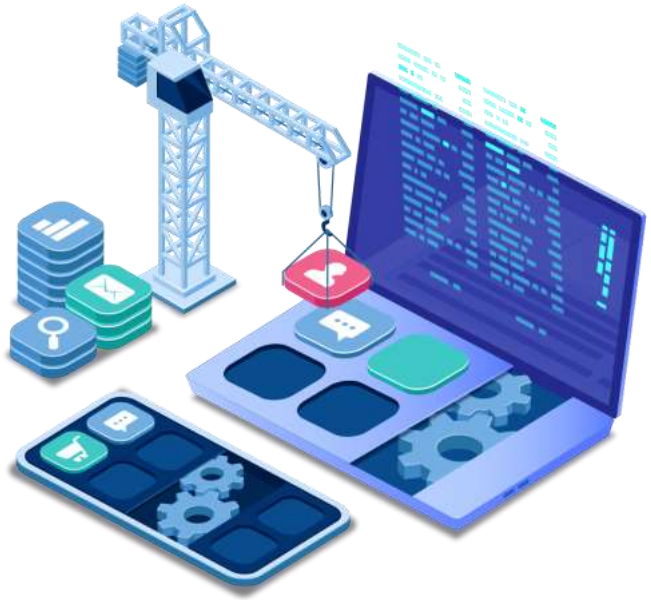
Acoplamiento





Acoplamiento

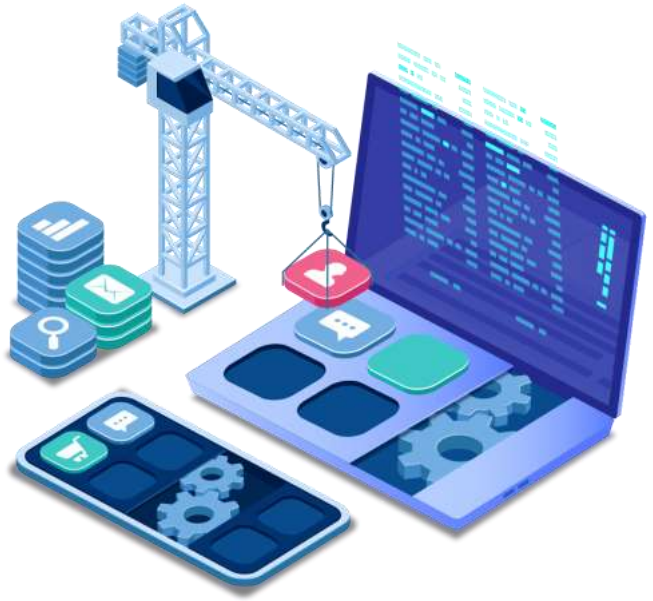
- El acoplamiento se caracteriza por el paso del control entre módulos.
- "Indicador de control" (una variable que controla las decisiones en un módulo subordinado o superordinado) se pasa entre los módulos d y e (llamado acoplamiento de control).
- Los niveles relativamente altos de acoplamiento ocurren cuando los módulos se comunican con software externo.
- El acoplamiento externo es esencial, pero debe limitarse a un pequeño número de módulos con una estructura.



- El acoplamiento alto también ocurre cuando varios módulos hacen referencia a un área de datos global.
- Acoplamiento común, no. De los módulos tienen acceso a un elemento de datos en un área de datos global
- Por lo tanto, no significa que "el uso de datos globales sea malo". Significa que un diseñador de software debe encargarse de esto.

Evolución del diseño de software

- Los primeros trabajos de diseño se concentraron en criterios para el desarrollo de programas modulares y métodos para refinar estructuras de software de arriba hacia abajo.
- Trabajos posteriores propusieron métodos para la traducción del flujo o estructura de datos en una definición de diseño.
- Hoy en día, el énfasis en el diseño de software ha estado en la arquitectura de software y los patrones de diseño que se pueden utilizar para implementar arquitecturas de software.

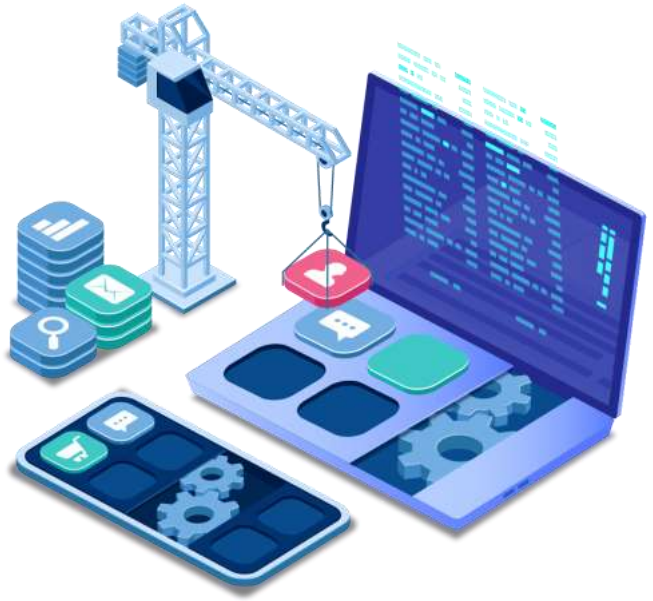


Las características son comunes a todos los métodos de diseño

- Un mecanismo para la traducción del modelo de análisis en una representación de diseño,
- Notación para representar componentes funcionales y sus interfaces.
- Heurística para refinamiento y particionamiento
- Directrices para la evaluación de la calidad.

Atributos de calidad de diseño

- Acrónimo FURPS –
  - Functionality
  - Usability
  - Reliability
  - Performance
  - Supportability



- Functionality – se evalúa evaluando el conjunto de características y las capacidades del programa.
  - Funciones que se entregan y seguridad del sistema en general.
- Usability – evaluado teniendo en cuenta los factores humanos, la consistencia y la documentación.
- Reliability – evaluado por
- medir la frecuencia y la gravedad de la falla.
  - Precisión de los resultados de salida.
  - Capacidad para recuperarse de fallas y previsibilidad del programa.
- Performance - medido por la velocidad de procesamiento, el tiempo de respuesta, el consumo de recursos, la eficiencia.
- Supportability – combina la capacidad de ampliar el programa (extensibilidad), adaptabilidad y capacidad de servicio.