GOBIERNO DEL
ESTADO DE MÉXICO

TESOEM
TECNOLÓGICO DE ESTUDIOS SUPERIORES
ORIENTE DEL ESTADO DE MÉXICO

CUADERNILLO DE:

FUNDAMENTOS DE DESARROLLO DE SISTEMAS.

GOBIERNO DEL
ESTADO DE MÉXICO

TESOEM
TECNOLÓGICO DE ESTUDIOS SUPERIORES
ORIENTE DEL ESTADO DE MÉXICO

TECNOLOGICO DE ESTUDIOS SUPERIORES DEL ORIENTE DEL ESTADO DE MEXICO

INGENIERIA EN SISTEMAS COMPUTACIONALES

ELABORACIÓN DE
CUADERNILLO DE APUNTES:
FUNDAMENTOS DE DESARROLLO DE SISTEMAS

ELABORADO POR:

ING. ELIAS SILVERIO MARTINEZ

LOS REYES LA PAZ ESTADO DE MEXICO **2011**

TABLA DE CONTENIDO

LISTA DE FIGURA

INTRODUCCIÓN

Objetivo:

- Conoce los elementos básicos y un panorama general para el análisis, diseño, implantación y gestión de software.

En la actualidad la mayoría de los sistemas incluyen **computadoras** y muchos no podrían vivir sin ellas, pero sin duda muchos sistemas existen antes de que las mismas se inventaran; algunos continúan por completo sin computarizar y otros la contienen como componente.

Figura 1. Diseño de Sistemas.

La labor primaria es **analizar o estudiar** un sistema para determinar su esencia, su comportamiento requerido, independientemente de la tecnología utilizada para implantar el sistema.

"El **diseño** es un proceso creativo, y creo firmemente que cada uno de nosotros abordamos dicho proceso creativo de forma particular. No hay una **"fórmula"** para **diseñar software.**

Solo aplicando los **modelos o paradigmas** correctos de desarrollo de software más utilizados y completos, nos podremos apoyar para la realización de software.

UNIDAD 1
CONCEPTOS INTRODUCTORIOS

OBJETIVO: El estudiante identificará los diferentes tipos de sistemas de software que existen y comprenderá las fases del ciclo de vida de un proyecto de software.

Se conoce como **software** al equipamiento lógico o soporte lógico de una computadora digital; comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos, que son llamados **hardware.**

Los **componentes lógicos** incluyen, entre muchos otros, las aplicaciones informáticas; tales como el procesador de textos, que permite al usuario realizar todas las tareas concernientes a la edición de textos; el software de sistema, tal como el sistema operativo, que, básicamente, permite al resto de los programas funcionar adecuadamente, facilitando también la interacción entre los componentes físicos y el resto de las aplicaciones, y proporcionando una interfaz para el usuario.

Sistema es un conjunto organizado de cosas o partes interactuantes e interdependientes, que se relacionan formando un todo unitario y complejo. Cabe aclarar que las cosas o partes que componen al sistema, no se refieren al campo físico (objetos), sino más bien al funcional. De este modo las cosas o partes pasan a ser funciones básicas realizadas por el sistema. Podemos enumerarlas en: **entradas, procesos y salidas**.

1.1 Introducción a los sistemas

Sistema de información es un conjunto de elementos que interactúan entre sí con el fin de realizar cuatro actividades básicas: entrada, almacenamiento, procesamiento y salida de información.

1.1.1 Descripción general

Sistema es un todo organizado y complejo, Los límites o fronteras entre el sistema y su ambiente admiten cierta arbitrariedad.

Un sistema es un conjunto de unidades recíprocamente relacionadas, De ahí se deduce el concepto: globalismo (o totalidad).

Globalismo o totalidad: Un cambio en una de las unidades del sistema, con probabilidad producirá cambios en las otras. El efecto total se presenta como un ajuste a todo el sistema. Hay una relación de causa/efecto. De estos cambios y ajustes, se derivan dos fenómenos: **entropía y homeostasis**.

Entropía: Es la tendencia de los sistemas a desgastarse, a desintegrarse, para el relajamiento de los estándares y un aumento de la aleatoriedad. La entropía aumenta con el correr del tiempo. Si aumenta la información disminuye la entropía, pues la información es la base de la configuración y del orden.

Figura 2. Entropía.

Homeostasis: Es el equilibrio dinámico entre las partes del sistema. Los sistemas tienen una tendencia a adaptarse con el fin de alcanzar un equilibrio interno frente a los cambios externos del entorno. Una organización podrá ser entendida como un sistema o subsistema o un súper sistema, dependiendo del enfoque.

1.1.2 Tipos de Sistemas

La estructura interna determina el **comportamiento de los sistemas**, y así podemos establecer una tipología de la estructura de los sistemas atendiendo al comportamiento que nos muestran.

Esto es especialmente útil ya que nos permite avanzar en nuestro análisis en una dirección perfectamente conocida, ya que buscaremos aquella estructura-tipo que nos provoca el comportamiento observado.

Un **sistema es estable** cuando se halla formado o dominado por un bucle negativo.

Un sistema **es inestable** cuando el bucle es **positivo**, es decir, cuando en el bucle dominante haya un número impar de relaciones negativas, tendremos un bucle negativo, y el sistema será estable.

1.1.3 Clasificación de los Sistemas

La **clasificación de un sistema** al igual que el análisis de los aspectos del mismo es un proceso relativo; depende del individuo que lo hace, del objetivo que se persigue y de las circunstancias particulares en las cuales se desarrolla.

Los sistemas se clasifican:

SEGÚN SU RELACION CON EL MEDIO AMBIENTE

- ✓ **Abiertos:** Sistemas que intercambian materia, energía o información con el ambiente.

  Ejemplos: célula, ser humano, ciudad, perro, televisor, familia, estación de radio.

- ✓ **Cerrado:** Sistemas que no intercambian materia, energía o información con el ambiente.

  Ejemplos: universo, reloj desechable, llanta de carro.

SEGÚN SU NATURALEZA

- ✓ **Concretos:** Sistema físico o tangible.

  Ejemplos: Equipos de sonidos, pájaro, guitarra, elefante.

- ✓ **Abstractos:** Sistemas simbólicos o conceptuales.

  Ejemplo: Sistema sexagesimal, idioma español, lógica difusa.

SEGÚN SU ORIGEN

- ✓ **Naturales:** Sistemas generados por la naturaleza.

  Ejemplo: ríos, los bosques, las moléculas de agua.

- ✓ **Artificiales:** Sistemas que son productos de la actividad humana, son concebidos y construidos por el hombre.

  Ejemplo: tren, avión, idioma ingles.

SEGÚN SUS RELACIONES

- ✓ **Simples:** Sistemas con pocos elementos y relaciones.

  Ejemplo: los juegos de billar, péndulo, f(x)=x+2, palanca.

- ✓ **Complejos:** Sistemas con numerosos elementos y relaciones.

  Ejemplo: cerebro, universidad, cámara fotográfica.

SEGÚN SU CAMBIO EN EL TIEMPO

- ✓ **Estáticos:** Sistema que no cambia en el tiempo.

  Ejemplo: piedra, vaso de plástico, montañas.

- ✓ **Dinámicos:** Sistema que cambia en el tiempo.

  Ejemplo: Universo, átomo, la tierra, hongo.

Esta **clasificación es relativa** por que depende del periodo de tiempo definido para el análisis del Sistema.

SEGÚN EL TIPO DE VARIABLE QUE LO DEFINEN

- ✓ **Discretos:** Sistema definido por variables discretas.

  Ejemplo: lógica, alfabeto.

- ✓ **Continuos**: Sistema definido por variables continúas.

  Ejemplo: alternador, ríos.

OTRAS CLASIFICACIONES

- ✓ **Jerárquicos:** Sistemas cuyos elementos están relacionados mediante relaciones de dependencia o subordinación conformando una organización por niveles.

  Ejemplo: gobierno de una ciudad.

- ✓ **Sistema de control:** Sistema jerárquico en el cual unos elementos son controlados por otros.

  Ejemplo: lámparas.

- ✓ **Sistema de Control con retroalimentación:** Sistema de control en el cual elementos controlados envían información sobre su estado a los elementos controladores.

  Ejemplo: termostato.

También cabe plantear que los sistemas pueden clasificarse como:

- ✓ **Vivientes y no viviente:** Los sistemas vivientes están dotados de funciones biológicas.

  Ejemplo: como el nacimiento, la reproducción y la muerte.

- ✓ **Abstractos y concretos:** Un sistema abstracto. Es aquel en que todos sus elementos son conceptos.

  Un sistema concreto es aquel en el aquel por lo menos dos de sus elementos son objetivos o sujetos, o ambos.

1.2 Ciclo de vida de un proyecto de software.

El término **ciclo de vida** de un proyecto de software describe el desarrollo de software, desde la fase inicial hasta la fase final. El **propósito de este programa** es definir las distintas fases intermedias que se requieren para validar el desarrollo de la aplicación, es decir, para garantizar que el software

cumpla los requisitos para la aplicación y verificación de los procedimientos de desarrollo: se asegura de que los métodos utilizados sean apropiados.

Estos programas se originan en el hecho de que es muy costoso rectificar los errores que se detectan tarde dentro de la fase de implementación.

El ciclo de vida permite que los errores se detecten lo antes posible y por lo tanto, permite a los desarrolladores concentrarse en la calidad del software, en los plazos de implementación y en los costos asociados.

Consta de los siguientes procedimientos:

a.- Definición de objetivos: definir el resultado del proyecto y su papel en la estrategia global.

b.- Análisis de los requisitos y su viabilidad: recopilar, examinar y formular los requisitos del cliente y examinar cualquier restricción que se pueda aplicar.

c.- Diseño general: requisitos generales de la arquitectura de la aplicación.

d.- Diseño en detalle: definición precisa de cada subconjunto de la aplicación.

e.- Programación (programación e implementación): es la implementación de un lenguaje de programación para crear las funciones definidas durante la etapa de diseño.

f.- Prueba de unidad: prueba individual de cada subconjunto de la aplicación para garantizar que se implementaron de acuerdo con las especificaciones.

g.- Integración: para garantizar que los diferentes módulos se integren con la aplicación. Éste es el propósito de la prueba de integración que está cuidadosamente documentada.

h.- Prueba beta (o validación), para garantizar que el software cumple con las especificaciones originales.

i.- Documentación: sirve para documentar información necesaria para los usuarios del software y para desarrollos futuros.

j.- Mantenimiento: para todos los procedimientos correctivos (mantenimiento correctivo) y las actualizaciones secundarias del software (mantenimiento continuo).

El orden y la presencia de cada uno de estos procedimientos en el ciclo de vida de una aplicación dependen del tipo de modelo de ciclo de vida acordado entre el cliente y el equipo de desarrolladores.

Para facilitar una **metodología** común entre el cliente y la compañía de software, los modelos de ciclo de vida se han actualizado para reflejar las etapas de desarrollo involucradas y la documentación requerida, de manera que cada etapa se valide antes de continuar con la siguiente etapa. Al final de cada etapa se arreglan las revisiones de manera que no se encuentren errores.

1.2.1 Planificación y gestión del proyecto

El objetivo de la **planificación** del proyecto de software es proporcional a un marco de trabajo que permita al gestor hacer estimaciones razonables de recursos, costo y planificación temporal.

La **gestión del proyecto** comienza con un conjunto de actividades llamadas planificación del proyecto. Se debe realizar una estimación del trabajo a realizar, los recursos necesarios y el tiempo que transcurrirá. Se considera el tamaño del proyecto ya que afectaría la precisión y la eficiencia de las estimaciones.

La complejidad del proyecto y el grado de incertidumbre estructural afectan a la fiabilidad de la estimación.

1.2.2 Determinación de requerimientos

La **determinación de requerimientos** es el conjunto de actividades encaminadas a obtener las características necesarias que deberá poseer el nuevo desarrollo del sistema, es el estudio de un sistema, actividad o proceso,

para comprender cómo trabaja y dónde es necesario efectuar mejoras o cambios considerables.

Este es el primer paso en el análisis de sistemas y se puede decir que es el más importante.

1.2.3 Análisis y diseño

Análisis.

Es necesario determinar qué elementos intervienen en el sistemas a desarrollar, así como su estructura, relaciones, evolución en el tiempo, detalle de sus funcionalidades, que van a dar una descripción clara de que sistema vamos a construir, qué funcionalidades va a aportar y qué comportamiento va a tener, responde a la pregunta ¿que vamos hacer?

Diseño.

Tras la etapa anterior ya se tiene claro que debe hacer el sistema, ahora tenemos que determinar cómo va a hacerlo (¿cómo debe ser construido el sistema; aquí se definirán en detalle entidades y relaciones de las bases de datos, se pasará de casos de uso esenciales a su definición como casos expandidos reales, se seleccionara el lenguaje más adecuado, el Sistema Gestor de Base de Datos a utilizar en un caso, librerías, configuraciones hardware, redes, etc.).

La función del Análisis puede ser dar soporte a las actividades de un negocio, o desarrollar un producto que pueda venderse para generar beneficios.

El Software, que son Programas de computadora, con estructuras de datos y su documentación que hacen efectiva la logística, metodología o controles de requerimientos del Programa.

El Hardware, dispositivos electrónicos y electromecánicos, que proporcionan capacidad de cálculos y funciones rápidas, exactas y efectivas.

1.2.4 Programación

Nota: Puede instalar paquetes comprados a terceros o escribir programas diseñados a la medida del solicitante.

Lenguajes de Programación. Son utilizados para escribir programas de computadoras que puedan ser entendidos por ellas.

La elección depende del costo de cada alternativa, del tiempo disponible para escribir el software y de la disponibilidad de los programadores.

Los lenguajes de programación se clasifican en tres grandes categorías, maquinas, bajo nivel y alto nivel.

Lenguaje de maquina: El lenguaje de maquina es aquel cuyas instrucciones son directamente entendibles por la computadora y no necesitan traducción posterior para que la UCP pueda comprender y ejecutar el programa.

Las instrucciones en lenguaje maquina se expresan en términos de la unidad de memoria más pequeña **(bit) = digito binario 0 o 1,** en esencia una secuencia de bits que especifican la operación y las celdas de memoria implicadas en una operación.

Ejemplo. Instrucciones en lenguaje de maquina:

0010, 0000, 1001, 1001, 10001, 1110.

1.2.5 Pruebas e Implementación.

Pruebas: Antes de que pueda ser usado el sistema de información debe ser probado. Durante este proceso se debe poner en práctica todas las estrategias posibles para garantizar que el usuario inicial del sistema se encuentre libre de problemas.

Existen seis pruebas básicas:

a.- **Prueba de carga máxima:** Consiste en probar si el sistema puede manejar el volumen de actividades que ocurren cuando el sistema está en el punto más alto de su demanda de procesamiento.

b.- Prueba de almacenamiento: Determina si el sistema puede almacenar una alta cantidad proyectada de datos tanto en sus dispositivos de discos fijos y movibles.

c.- Prueba de tiempo de ejecución: Determina el tiempo de máquina que el sistema necesita para procesar los datos de una transición.

d.- Prueba de recuperación: Probar la capacidad del sistema para recuperar datos y restablecer después de una falla.

e.- Prueba de procedimientos: Evaluar la claridad, validez, seguridad así como su facilidad y sencillez de los manuales de procedimientos.

f.- Prueba de recursos humanos: Se determinan como utilizar los usuarios el sistema al procesar datos o procesar informes.

Implementación: Es la última fase del desarrollo de sistemas. Es el proceso de instalar equipos o software nuevo, como resultado de un análisis y diseño previo como resultado de la situación o mejoramiento de la forma de llevar a cabo un proceso automatizado.

Al implementar un sistema lo primero que debemos hacer es asegurarnos qué el sistema sea operacional o que funcione de acuerdo a los requerimientos del análisis y permitir que los usuarios puedan operarlos.

Existen varios enfoques de implementación:

a.- darle responsabilidad a los grupos

b.-Uso de diferentes estrategias para el enfrentamiento de usuarios.

c.-El analista necesita formular medidas de desempeño con los cuales evalúa a los usuarios.

Durante el proceso de implementación y prueba se deben poner en práctica todas las estrategias posibles para garantizar que el usuario inicial del sistema se encuentre libre de problemas lo cual se puede describir durante este proceso, para llevar a cabo las correcciones.

Actividad correspondiente a la unida 1

Contesta correctamente las siguientes preguntas

1. ¿Define que es un sistema?
2. ¿Define que es entropía?
3. ¿Define el ciclo de vida de un proyecto de software?
4. ¿Define que es programación de sistemas?
5. ¿Define que es el software?
6. ¿Define que es el hardware?
7. ¿Escribe las pruebas que se aplican a un sistema de información?
8. ¿Define que es implementación?

UNIDAD 2
INTRODUCCIÓN A LA INGENIERÍA DE SOFTWARE.

Objetivo: Comprenderá los elementos que integran la Ingeniería de Software y el aseguramiento de la calidad.

La **Ingeniería del Software** es una disciplina o área de la informática o ciencias de la computación, que ofrece métodos y técnicas para desarrollar y mantener software de calidad que resuelven problemas de todo tipo.

Hoy día es cada vez mas frecuente la consideración de la Ingeniería del Software como una nueva área de la ingeniería, y el Ingeniero del Software comienza a ser una profesión implantada en el mundo laboral internacional, con derechos, deberes y responsabilidades que cumplir, junto a una, y reconocida consideración social en el mundo empresarial y, por suerte, para esas personas con brillante futuro.

2.1 Definición de ingeniería de software.

Tipos de Definiciones de Ingeniería de Software:

a.-Es el estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas de software.

b.-Es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada requerida para desarrollar y operar (funcionar) y mantenerlos. Así como también desarrollo de software o producción de software.

c.-Es una disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iníciales de la especificación del sistema hasta el mantenimiento de este después que se utiliza.

Definición de Ingeniería: La ingeniería es el estudio y la aplicación de las distintas ramas de la tecnología. El profesional en este ámbito recibe el nombre de ingeniero.

La ingeniería también supone la aplicación de la inventiva y del ingenio para desarrollar una cierta actividad. Esto, por supuesto, no implica que no se utilice el método científico para llevar a cabo los planes.

Definición de Software: Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.

El **software** no son, solo programas, sino todos los documentos asociados y la configuración de datos que se necesitan para hacer que estos programas operen de manera correcta. Un sistema de software consiste en diversos programas independientes, archivos de configuración que se utilizan para ejecutar estos programas, un sistema de documentación que describe la estructura del sistema, la documentación para el usuario que explica como

utilizar el sistema y sitios web que permitan a los usuarios descargar la información de productos recientes.

El **software de computadora** es el producto que los ingenieros de software construyen y después mantienen en el largo plazo.

El software se forma con:

1.- Las instrucciones (programas de computadora) que al ejecutar se proporcionan las características, funciones y el grado de desempeño deseados.

2.- Las estructuras de datos que permiten que los programas manipulen información de manera adecuada.

3.- Los documentos que describen la operación y uso de los programas.

2.2 Historia de la ingeniería de software.

La Ingeniería del **Software**, término utilizado por primera vez por Fritz Bauer en la primera conferencia sobre desarrollo de software patrocinada por el Comité de Ciencia de la OTAN celebrada en Garmisch, Alemania, en octubre de 1968, puede definirse según Alan Davis como “la aplicación inteligente de principios probados, técnicas, lenguajes y herramientas para la creación y mantenimiento, dentro de un coste razonable, de software que satisfaga las necesidades de los usuarios.

El término **ingeniería del software** empezó a usarse a finales de la década de los sesenta, para expresar el área de conocimiento que se estaba desarrollando en torno a las problemáticas que ofrecía el software en ese momento.

En esa época, el crecimiento espectacular de la demanda de sistemas de computación cada vez más y más complejos, asociado a la inmadurez del propio sector informático (totalmente ligado al electrónico) y a la falta de

métodos y recursos, provocó lo que se llamó la crisis del software (en palabras de Edsger Dijkstra) entre los años 1965 y 1985.

Durante esa época muchos proyectos importantes superaban con creces los presupuestos y fechas estimados, algunos de ellos eran tan críticos (sistemas de control de aeropuertos, equipos para medicina, entre otros) que sus implicaciones iban más allá de las pérdidas millonarias que causaban.

Así pues, desde 1985 hasta el presente, han ido apareciendo herramientas, metodologías y tecnologías que se presentaban como la solución definitiva al problema de la planificación, previsión de costes y aseguramiento de la calidad en el desarrollo de software.

Entre las que se encuentran la **programación estructurada**, la programación **orientada a objetos**, a los aspectos, **las herramientas CASE**, el lenguaje de **programación ADA**, la documentación, los estándares, **CORBA**, los servicios **Web y el lenguaje UML (entre otros)** fueron todos anunciados en su momento como la solución a los problemas de la ingeniería del software, la llamada “bala de plata” (por silver bullet). Y lo que es más, cada año surgen nuevas ideas e iniciativas encaminadas a ello.

2.3 Características del software.

El software es un elemento del sistema que es lógico. Por tanto, el software tiene características considerablemente **distintas al hardware:**

a.-El software se desarrolla, no se fabrica en un sentido clásico.

b.-El software no se estropea.

c.-La mayoría de software se construye a medida, en lugar de ensamblar componentes existentes.

a.- El software se desarrolla, no se fabrica en un sentido clásico.

A pesar de que existen similitudes entre el **desarrollo del software y la manufactura del hardware,** las dos actividades serian diferentes en lo fundamental. En ambas la alta calidad se alcanza por medio del buen diseño, la fase de manufactura del hardware puede incluir problemas de calidad existentes en el software.

Figura 3. Curva del software.

El software es inmune a los males ambientales que desgasten el hardware. Por lo tanto la curva de tasas de fallas para el software debería tener la forma de la “curva idealizada”.

Los defectos sin descubrir causan tasas de fallas altas en las primeras etapas de vida de un programa. Sin embargo, los errores se corrigen y la curva se aplana: el software no se desgasta, pero si se deteriora.

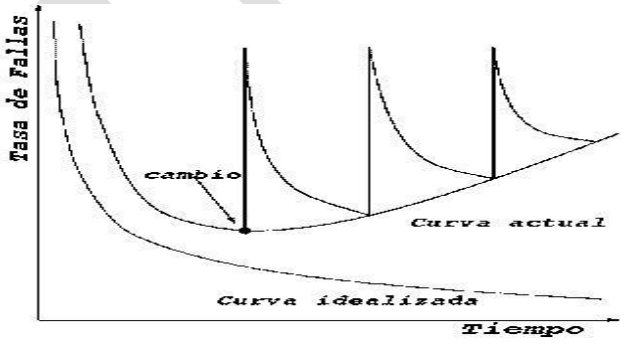


Figura 4. Curva real de fallos del software.

A pesar de que la industria tiene una tendencia hacia la construcción por componentes, la mayoría del software aun se construye a la medida.

La mayoría del software se construye a medida, en lugar de ensamblar componentes existentes.

Un componente de software se debe diseñar e implementar de forma que puede utilizarse en muchos programas diferentes.

Los componentes reutilizables modernos encapsulan tanto los datos como el proceso se aplican a estos, lo que permite al ingeniero de software crear nuevas aplicaciones nuevas a partir de partes reutilizables.

2.4 Mitos del software

Son creencias acerca del software y de los procesos empleados para construirlo, se pueden rastrear hasta los primeros días de la computación.Los mitos tienen ciertos atributos.

Los gestores con responsabilidad sobre el software, como los gestores en la mayoría de las disciplinas, están normalmente bajo la presión de cumplir las propuestas, hacer que no se retrase el proyecto y mejorar la calidad. Un gestor de software se agarra frecuentemente a un mito del software.

- **Mito:** Si fallamos en la planificación podemos añadir más programadores y recuperar el tiempo perdido.
- **Realidad:** Ley de Brooks: "Agregar gente a un proyecto atrasado, lo atrasa aún mas".
- **Razón:** Crear software no es una tarea particionable, como dice el Principio de Brooks: "Gestar a un bebé tarda 9 meses, no importa cuántas mujeres sean asignadas a la tarea."
- **Mito:** Una declaración general de los objetivos es suficiente para comenzar a escribir los programas; podemos dar los detalles más adelante.
- **Realidad:** Una mala definición inicial es la principal causa del trabajo en vano. Es esencial una descripción formal y detallada del ámbito de la información, funciones, rendimiento, interfaces y criterios de validación.

Esto solo puede determinarse después de una exhaustiva comunicación entre el cliente y el analista.

- **Mito:** Los requisitos del proyecto cambian continuamente pero los cambios pueden acomodarse fácilmente.
- **Realidad:** El impacto del cambio varía según el momento en el que se introduzca:

2.5 Capas de la ingeniería de software.

El fundamento de la **ingeniería de software** es la capa del proceso. El proceso de la ingeniería de software es la unión que mantiene juntas las capas de tecnología y que permiten un desarrollo racional y oportuno de la ingeniería de software.

Los métodos de la ingeniería de software indican cómo construir técnicamente el software. Los métodos abarcan una gran gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento.

Dado lo anterior, el objetivo de la ingeniería de software es lograr productos de software de calidad (tanto en su forma final como durante su elaboración), mediante un proceso apoyado por métodos y herramientas.

Las Cuatro Capas de La ingeniería de software son:

1.- Un enfoque de Calidad**, 2.-** Procesos**, 3.-** Métodos**, 4.-** Herramientas

Figura 5. Capas de La ingeniería de software.

Dichas capas se describen a continuación:

1) **Un enfoque de Calidad:** Es la base o cimientos de la ingeniería de software.
2) **Procesos:** Es el fundamento de la ingeniería de software.
3) **Métodos:** Indican cómo construir técnicamente el software.
4) **Herramientas:** Proporcionan un soporte automático o semi automático a los procesos y a los métodos.

2.6 El proceso del software

Un proceso de software es el **conjunto de actividades**, métodos, prácticas y tecnologías aplicables a todos los proyectos de software. Un proceso básico (también conocido como ciclo de vida básico) está conformado por el análisis, diseño, codificación, pruebas y mantenimiento.

Actividades requeridas para desarrollar un sistema de software.

1.- Especificación**, 2.-** Diseño**, 3.-** Validación**, 4.-** Evolución.

Las actividades varían dependiendo de la organización y del tipo de sistema a desarrollarse.

Características del proceso de software:

- **Entendible:** Se encuentra el proceso bien definido y es entendible.
- **Visible:** El proceso es visible al exterior.
- **Soportable:** Puede el proceso ser soportado por herramientas CASE.
- **Aceptable:** El proceso es aceptado por aquellos involucrados en el.
- **Confiable:** Los errores del proceso son descubiertos antes de que se conviertan en errores del producto.
- **Robusto:** Puede continuar el proceso a pesar de problemas inesperados.
- **Mantenible:** Puede el proceso evolucionar para cumplir con los objetivos organizacionales.

- **Rapidez:** Que tan rápido puede producirse el sistema.
- **Problemas**: Normalmente, las especificaciones son incompletas o anómalas.

No existe una distinción precisa entre la especificación, el diseño y la manufactura,

Solo hasta que el sistema se ha producido se puede probar. (El software no se puede remplazar siempre durante el mantenimiento).

2.7 Software de alta calidad.

Las inspecciones de software surgen a partir de la necesidad de producir software de alta calidad. Algunos grupos de **desarrollo** creen que la calidad del software es algo en lo que deben preocuparse una vez que se ha generado el código.

La garantía de la calidad del software es una actividad de protección que se aplica a lo largo de todo **el proceso** de ingeniería de software. La SQA (Software **Quality Assurance**) engloba:

* Un enfoque de gestión de calidad

* Tecnología de Ingeniería de Software efectiva (métodos y herramientas)

* Revisiones técnicas formales que se aplican durante el proceso del software

* Una estrategia de prueba multi escalada

* Un control de la documentación del software y de los cambios realizados

* Un **procedimiento** que asegure un ajuste a los estándares de desarrollo de

* software

* Mecanismos de **medición** y de generación de informes.

El control de la calidad es una serie de revisiones, y pruebas utilizados a lo largo del ciclo del desarrollo para asegurar que cada producto cumple con los requisitos que le han sido asignados.

La **principal** meta de un equipo desarrollador de software debería ser siempre producir software catalogado como de alta calidad. Pero para ello se deben tener en cuenta algunas ideas previas.

Productos software son realizados por personas para personas. Así, las personas desarrolladoras deben tener en cuenta claramente que son otras personas las que utilizarán sus productos, los que pueden estar sujetos a fallos constantes. Aún a pesar de los avances actuales en **Inteligencia Artificial**, los asistentes software para el desarrollo de software no son demasiado confiables como para que la mano humana no intervenga en este **proceso.**

El instituto de la ingeniería del software **(CEI)** ha desarrollado un modelo completo de un amplio proceso basado en un conjunto de capacidades de software y de sistemas que deben de estar presentes conforme las organizaciones alcanzan diferentes grados de capacidad y madurez.

MODELO DE CAPACIDAD DE MADUREZ (IMCM)

Nivel 0: Incompleto. El área del proceso (por ejemplo, la gestión de requisitos) aún no se realiza o todavía no alcanza todas las metas y objetivos definidos para el nivel 1 de capacidad.

Nivel 1: Realizado. Todas las metas específicas de área del proceso (como las definió la IMCM) han sido satisfechas. Las tareas de trabajo requeridas para producir el producto específico han sido realizadas.

Nivel 2: Administrado. Todos los criterios del nivel 1 han sido satisfechos. Además, todo el trabajo asociado con el área de proceso se ajusta a una política organizacional definida; toda la gente que ejecuta el trabajo tiene acceso a recurso adecuados para realizar su labor; los clientes están implicados de manera activa en el área de proceso, cuando esto se requiere;

todas las tareas de trabajo y productos están “monitoreadas, controlados y revisados; y son evaluados en apego a la descripción del proceso”

Nivel 3: Definido. Todos loa criterios del nivel 2 se han cumplido. Además, el proceso esta “adaptado al conjunto de procesos de estándar de la organización, de acuerdo con las políticas de adaptación de esta misma, y contribuye a la información de los productos del trabajo, mediciones y otras mejorías del proceso para los activos del proceso organizacional”.

Nivel 4: Administrativo en forma cuantitativa. Todos los criterios del nivel 3 han sido cumplidos. Además, el área del proceso se controla y mejora mediante mediciones y evaluación cuantitativa. “Los objetivos cuantitativos para la calidad y el desempeño del proceso están establecidos y se utilizan como un criterio para administrar el proceso”.

Nivel 5: Mejorado. Todos loa criterios del nivel 4 han sido satisfechos. Además, el área del proceso “se adapta y mejora mediante el uso de medios cuantitativos (estadísticos) para conocer las necesidades cambiantes del cliente y mejorar de manera continua la eficacia del área del proceso que se está considerando”.

2.8 Factores de calidad y productividad.

La calidad del software desarrollado, así como la **productividad** del programador son factores de difícil pero no imposible medida. Existen una serie de factores que influyen en la calidad y productividad, que son los siguientes y que ayudan a realizar dicha medida:

La capacidad individual.

En este factor interviene la competencia del individuo y su familiaridad con el área de la aplicación. La comunicación entre los miembros del equipo.

La complejidad del producto.

Este factor depende del tipo de aplicación a desarrollar y es de difícil estimación, ya que muchas veces hasta la fase de desarrollo no es posible comprender en toda su perspectiva las complicaciones que conlleva su realización.

Utilización de una notación adecuada.

Este factor es de gran importancia para facilitar la comunicación entre las partes involucradas (incluido el usuario).

Como en el resto de las actividades industriales, en el desarrollo de software, también es importante realizar una buena planificación del trabajo y una buena asignación de recursos a los distintos miembros del equipo.

Una mala planificación termina con una mala aplicación o una aplicación terminada a destiempo (disgusto del peticionario), lo cual supone un fracaso.

Empleo de métodos sistemáticos.

Es importante que se empleen técnicas que sean de amplio consenso y bien conocidas por los integrantes del equipo de desarrollo de la aplicación. También es fundamental que estas técnicas se empleen de manera sistemática sobre todas las aplicaciones de características semejantes con objeto de facilitar el análisis de coste y tiempo, y también para poder observar la trayectoria profesional de los miembros del equipo.

Conocer el tiempo disponible.

Este factor esta vinculado a otros anteriores, ya que es básico conocer el tiempo que puede aportar cada miembro del equipo y en que plazos, sobre todo en función de las tareas a realizar y de la mejor o peor productividad de determinados miembros en cada una de ellas.

Existencia de facilidades y recursos externos.- Este factor, es determinante en la medida en que se conozcan productos o herramientas (automáticas o no)

que faciliten las labores de desarrollo e integración de la aplicación. En mayor medida cuando se conocen aplicaciones parecidas de fácil tras portabilidad y modificación que puedan servir de base a la que hay que realizar.

Como en el resto de las actividades industriales, en el desarrollo de software, también es importante realizar una buena planificación del trabajo y una buena asignación de recursos a los distintos miembros del equipo. Una mala planificación termina con una mala aplicación o una aplicación terminada a destiempo (disgusto del peticionario), lo cual supone un fracaso. Varios fracasos consecutivos de este mismo estilo suponen la ruina para la mayor parte de las empresas del sector, debido a la competencia existente

Actividad correspondiente a la unidad 2

Relaciona la columna enumerada al paréntesis
Correcto

1.-Es una disciplina o área de la informática Que ofrece métodos y técnicas para desarrollar y mantener software de calidad	( ) Software
2.-Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.	( ) Capacidad individual
3.-Es una serie de revisiones, y pruebas utilizadas a lo largo del ciclo del desarrollo para asegurar que cada producto cumple con los requisitos que le han sido asignados.	( ) Ingeniería de software
4.-En este factor interviene la competencia del individuo y su familiaridad con el área de la aplicación, la comunicación entre los miembros del equipo.	( ) Control de calidad

UNIDAD 3
PARADIGMAS DE LA INGENIERÍA DE SOFTWARE.

Objetivo: Comprenderá la diferencia de aplicar un enfoque estructurado vs. Orientado a objetos en el desarrollo de un proyecto de software.

Paradigma: Es un conjunto de tres elementos (Métodos, Herramientas, Procedimientos), que facilitan el control sobre el proceso de desarrollo de software y suministran las bases para construir software de calidad de una forma productiva.

Métodos: Que indican cómo construir el software técnicamente e incluyen un amplio espectro de métodos para la planificación, la estimación, el análisis, el diseño, codificación, prueba y mantenimiento.

Herramientas: Automáticas y semiautomáticas que apoyan la aplicación de los métodos. Cuando se integran las herramientas de forma que la información creada por una herramienta puede ser usada por otra, se establece un sistema para el soporte del desarrollo de software, llamado Ingeniería de Software Asistida por Computadora (CASE).

Procedimientos: Que definen la secuencia en la que se aplican los métodos, las entregas, los controles de calidad y guías para evaluación del progreso.

Paradigmas de programación:

a) Los que soportan técnicas de programación de bajo nivel (copia de ficheros frente a estructuras de datos compartidos).

b) Los que soportan métodos de diseño de algoritmos (programación dinámica).

c) Los que soportan soluciones de programación de alto nivel.

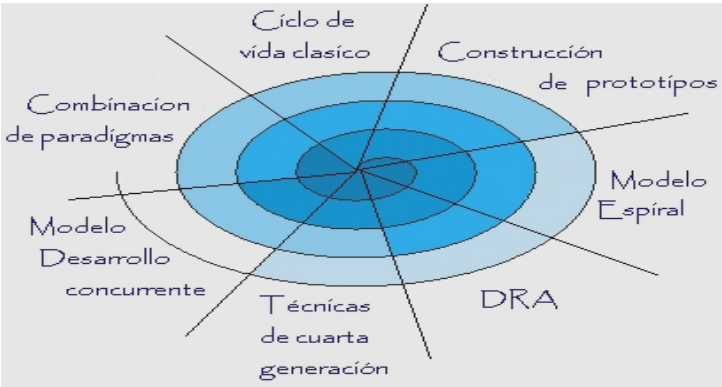


Figura 6. Gama de paradigmas de la ingeniería de software.

3.1 El enfoque estructurado.

En el Enfoque Estructurado se usan los DFD (Diagramas de Flujos de Datos) como principal herramienta para entender al sistema antes de plasmarlo a código fuente. DFD es un diagrama en el que participan procesos (métodos), flujo de datos (argumentos) y archivos (base de datos). Hay de diferentes niveles dependiendo la complejidad del sistema que analiza.

Muestran en forma visual sólo el flujo de datos entre los distintos procesos, entidades externas y almacenes que conforman un sistema.

Cuando los analistas de sistemas indagan sobre los requerimientos de información de los usuarios, deben ser capaces de concebir la manera en que los datos fluyen a través del sistema u organización, los procesos que sufren estos datos y sus tipos de salida.

Figura 7. Diagrama de un Enfoque Estructurado.

Hablando de lenguajes Tiene muchas diferencias con la OO (orientado a objetos). Un mínimo cambio en el código puede llegar alterar al resto del programa, cosa que en un OO bien encarado eso no sucede lo cual es una ventaja porque así no se pierde tiempo en arreglar cosas ya hechas.

3.1.1 Diagramas de flujos de datos.

Un diagrama de flujo de datos (DFD por sus siglas en español e inglés) es una descripción grafica de un procedimiento para la resolución de un problema. Son frecuentemente usados para descubrir algoritmos y programas de computador.

Los diagramas de flujos están compuestos por figuras conectadas con flechas. Para ejecutar un proceso comienza por el Inicio y se siguen las acciones indicadas por cada figura: El tipo de figura indica el tipo de paso que representa el Software.

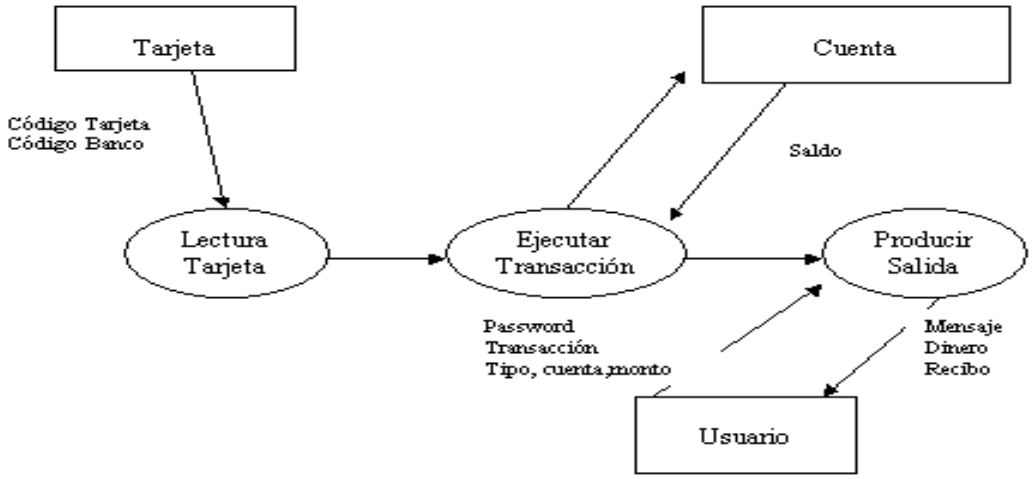


Figura 8. Diagrama de flujo para el caso de un Cajero Automático.

DFD es un software diseñado para contribuir y analizar algoritmos se puede crear diagramas de flujos de datos para la representación de algoritmos de programación estructurada a partir de las herramientas de edición que para este propósito suministra el programa .

La interfaz grafica de DFD facilita en gran medida el trabajo con diagramas ya que simula la representación estándar de diagramas de flujo en hojas de papel.

Los componentes de un Diagrama de Flujo son:

*Proceso *Flujo *Almacén *Terminador

Proceso

El primer componente de diagrama de flujo de datos se conoce como Proceso. Cuando un flujo de datos entra en un proceso sufre una transformación, y muestra una parte del sistema que transforman en Entradas y Salidas.

Figura 9. Proceso.

Un proceso no es origen ni final de los datos, sólo lugar de transformación de ellos. Un proceso puede trasformar un dato en varios.

Es necesario un proceso entre una Entidad Externa y un Almacén de datos. Un proceso puede representarse señalando una localización. La localización expresa la unidad o área dentro de la organización donde se realiza el proceso.

Flujo de Datos

Un flujo se representa gráficamente por medio de una flecha que entra y sale de proceso; el flujo se usa para describir el movimiento, de bloques o paquetes de información de una parte del sistema a otra.

Figura 10. Flujo de Datos.

Se representan los Datos, es decir, Bits, caracteres, mensajes, números, de puntos, flotante y los diversos tipos de información con los que las computadoras pueden tratar.

Almacén de Datos

Se utiliza para modelar una colección de paquetes de datos en reposo. Se denota por dos líneas paralelas, de modo característico el nombre que se

utiliza para identificar los paquetes que entran y salen del almacén por medios de flujo.

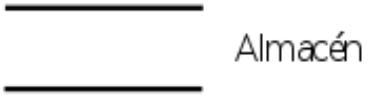


Figura 11. Almacén de Datos.

No puede crear, destruir ni transformar datos, no puede estar comunicado directamente con otro almacén o Entidad externa. El flujo de datos (Entrada y Salida) no lleva nombre cuando incide sobre su contenido completo. Los almacenes se conectan por flujos a los procesos. Así el contexto en el que se muestra en un DFD (Diagrama de Flujo de Datos) es uno de los siguientes:

A) Un flujo desde un almacén. B) Un flujo hacia un almacén.

Entidad Externa

Representa personas, organizaciones, o sistemas que no pertenecen al sistema. En el caso de que las entidades externas se comunicasen entre sí, esto no se contemplaría en el diagrama, por estar fuera del ámbito de nuestro sistema.

Figura 12. Entidad externa

Puede aparecer en los distintos niveles de DFD para mejorar su comprensión, aunque normalmente sólo aparecerá en el diagrama de contexto. Pueden aparecer varias veces en un mismo diagrama, para evitar entrecruzamientos de líneas. Suministra información acerca de la conexión del sistema con el mundo exterior. **EJEMPLO:**

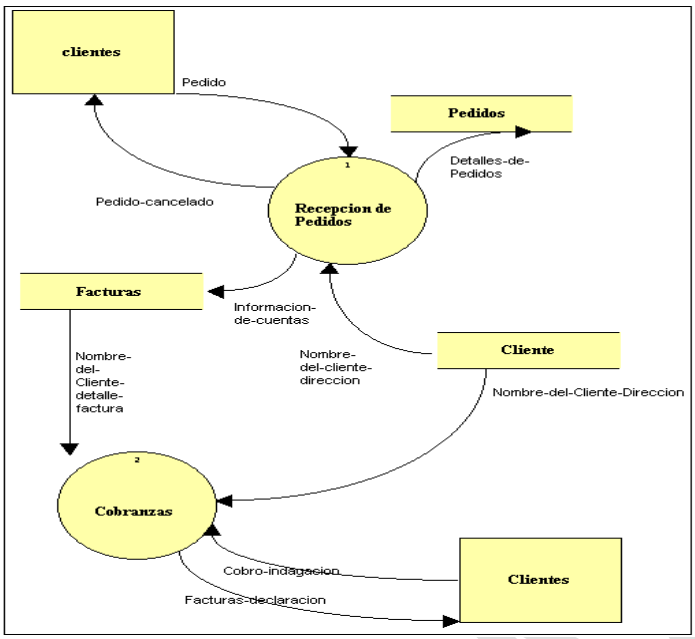


Figura 13: FDF

3.1.2 Diccionarios de datos.

Un diccionario de datos: es un conjunto de metadatos (En general, un grupo de metadatos se refiere a un grupo de datos). Que contiene las características lógicas y puntuales de los datos que se van a utilizar en el sistema que se programa, incluyendo nombre, descripción, alias, contenido y organización.

Identifica los procesos donde se emplean los datos y los sitios donde se necesita el acceso inmediato a la información, se desarrolla durante el análisis de flujo de datos y auxilia a los analistas que participan en la determinación de los requerimientos del sistema, su contenido también se emplea durante el diseño.

Los diccionarios de datos son buenos complementos a los diagramas de flujo de datos, los diagramas de entidad-relación, etc.

Existen muchos esquemas de anotación usados por los analistas de sistemas el que sigue es uno de los más usados.

CONTENIDO DEL DICCIONARIO DE DATOS

El Diccionario de datos debe contener la siguiente información:

Nombre: el nombre principal del elemento; del flujo de datos, del repositorio de datos o de una entidad externa.

Alias: otros nombres usados para la entrada, dado que un mismo elemento puede ser conocido por diferentes nombres.

Definición: Exposición clara y precisa de las características genéricas y diferenciales del objeto.

Descripción: Explicar las diversas partes o circunstancias, que componen la definición, de los objetos.

Dónde se usa/cómo se usa: Un listado de los procesos que usan un elemento de datos, o del control de cómo lo usan.

Descripción del contenido: El contenido es representado mediante una anotación que se describe en la siguiente tabla.

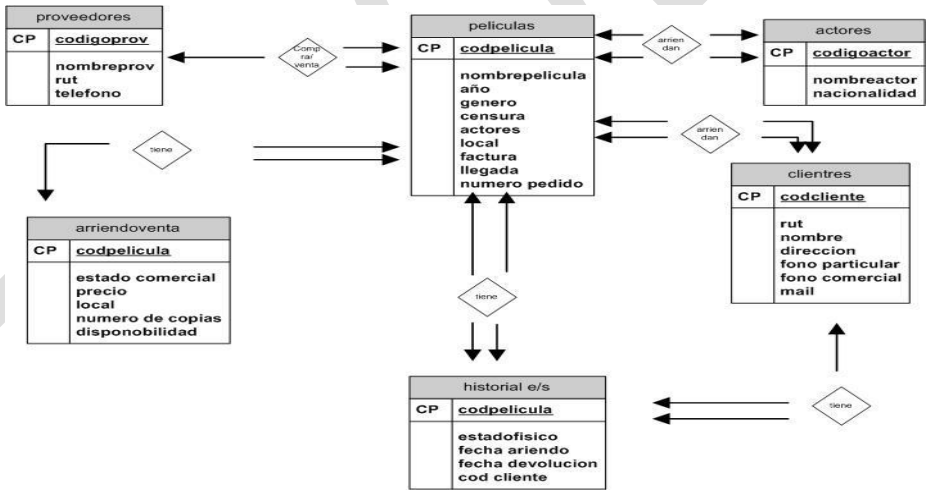


Figura 14: Diccionario de datos

3.1.3 Diseño de módulos.

La definición de Módulo es una dimensión que convencionalmente se toma como unidad de medida, Unidad de diseño que presenta una división de Software clara y manejable con sus interfaces definidas, Puede representar un programa, subprograma o rutina.

En programación, un módulo es un software que agrupa un conjunto de subprogramas y estructuras de datos. Los módulos son unidades que pueden ser compiladas por separado y los hace reusables y permite que múltiples programadores trabajen en diferentes módulos en forma simultánea, produciendo Ahorro en los tiempos de desarrollo.

Criterios del Diseño Modular.

El objetivo principal del diseño estructurado es desarrollar una estructura de programa en la que queden bien definidas las divisiones entre elementos homogéneos y la comunicación entre ellos. Dichos elementos son llamados módulos y su identificación hará mucho más sencillo un Software, así como su control y mantenimiento.

Conceptos de División

La subdivisión de un sistema en subsistemas y de éstos en módulos se hace de acuerdo a una serie de conceptos que se anuncian a continuación

- ✓ **Abstracción**: El nivel de abstracción disminuye a medida nos adentramos en terrenos más técnicos.

  Encontramos tres tipos de abstracciones:

  - **1.- De procesos 2.-De datos 3.-De control.**
- ✓ **Refinamiento:** Mientras la abstracción nos permite conceptualizar con independencia los detalles de bajo nivel, el refinamiento nos permite avanzar hacia estos.
- ✓ **Modularidad:** División del Software en elementos con función propia distinguibles de otros que se comunican e intercambian información.
- ✓ **Diseño estructurado**: Nos da una guía para modularizar un problema.

Definir los módulos del sistema de información, y la manera en que van a interactuar unos con otros, intentando que cada módulo trate total o parcialmente un proceso específico y tenga una interfaz sencilla.

Consistencia en el DFD.

- ❖ Cada proceso en un diagrama “padre” es una consolidación del DFD “hijo”.
- ❖ Balanceo de DFD’s: Las E/S de un proceso “padre” deben correspondenderse con las E/S del DFD “hijo” que lo explica.

Jerarquía de DFD's:

- ✓ En un DFD completo cada proceso tiene un número único que lo identifica en función de su situación en la jerarquía.
- ✓ Cada DFD tiene también un número único que coincide con el proceso que describe.
- ✓ Las hojas o nodos terminales corresponden a “procesos primitivos” indescomponibles.
- ✓ Para cada proceso primitivo existirá una mini especificación.

Un modelo de datos es un lenguaje orientado a describir una Base de Datos. Típicamente un Modelo de Datos permite describir:

- ✓ Las estructuras de datos de la base: El tipo de los datos que hay en la base y la forma en que se relacionan.
- ✓ Las restricciones de integridad: Un conjunto de condiciones que deben cumplir los datos para reflejar correctamente la realidad deseada.
- ✓ Operaciones de manipulación de los datos: típicamente, operaciones de agregado, borrado, modificación y recuperación de los datos de la base.

3.1.4 Descomposición en procesos.

En cualquier momento nos puede aparecer un proceso que no necesite descomposición y es lo que denominaremos Proceso Primitivo (PP). En ellos, se detallará la entrada y salida que tenga, además de la descripción asociada que explique lo que realiza.

DFD: Construcción.

- ✓ Representar el diagrama de contexto.
- ✓ Representar el DFD de primer nivel, indicando los distintos subsistemas funcionales en que se descompone nuestro sistema.
- ✓ Descomponer cada uno de los procesos que aparecen en el DFD de primer nivel, hasta llegar a un nivel suficiente de detalle.

Se recomienda el utilizar cuatro niveles de descomposición de diagramas.

Nivel 0: Diagrama de contexto

Nivel 1: Subsistemas

Nivel 2: Funciones de cada subsistema

Nivel 3: Subfunciones asociadas

Nivel 4: Procesos necesarios para el tratamiento de cada subfunción

Entiende todo proceso como un **: "**CONJUNTO DE TAREAS LOGICAMENTE RELACIONADAS QUE EXISTEN PARA OBTENER UN RESULTADO BIEN DEFINIDO DENTRO DE UN NEGOCIO".

Descomposición de procesos.

Los procesos de descomponen en "subprocesos", hasta llegar a los procesos primitivos.

Descomposición funcional.

- ✓ Cada proceso se puede explotar, refinar o descomponer en un DFD más detallado.
- ✓ El DFD de un sistema es realmente un conjunto de DFD's dispuestos jerárquicamente.
- ✓ Los niveles de la jerarquía están determinados por la descomposición funcional de los procesos.
- ✓ La raíz de la jerarquía es el "diagrama de contexto", que es el más general de todos.

3.2 El Enfoque Orientado a Objetos

La orientación a objetos puede describirse como el conjunto de disciplinas que desarrollan y modernizan software que facilitan la construcción de sistemas complejos a partir de componentes.

El atractivo intuitivo de la orientación a objetos es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible. Estos conceptos y herramientas orientados a objetos son tecnologías que permiten que los problemas del mundo real sean expresados de modo fácil y natural.

Las técnicas orientadas a objetos proporcionan mejoras y metodologías para construir sistemas de software complejos a partir de unidades de software modula rizado y reutilizable. Se necesita un nuevo enfoque para construir software en la actualidad.

Este nuevo enfoque debe ser capaz de manipular tanto sistemas grandes como pequeños y debe crear sistemas fiables que sean flexibles, mantenible y capaces de evolucionar para cumplir las necesidades del cambio.

La orientación a objetos trata de cubrir las necesidades de los usuarios finales, así como las propias de los desarrolladores de productos software.

Estas tareas se realizan mediante la modelización del mundo real. El soporte fundamental es el modelo objeto.

Un objeto es la instancia de una clase. Una clase es la representación abstracta de un concepto en el mundo real, y proporciona la base a partir de la cual creamos instancias de objetos específicos. Como ejemplo, puede crear una clase que defina a un cliente. Después puede crear una nueva instancia de la clase cliente para tener un objeto utilizable de Cliente. Para poder crear un objeto de la clase cliente, debe crear una nueva instancia basada en esa clase.

Todos los objetos están compuestos de tres cosas:

Interfaz: La Interfaz es el conjunto de métodos, propiedades, eventos y atributos que se declaran como públicos en su alcance y que pueden invocar los programas escritos para usar nuestro objeto.

Implementación: Al código dentro de los métodos se le llaman Implementación. Algunas veces también se le llama comportamiento**,** ya que este código es el que efectivamente logra que el objeto haga un trabajo útil. Es importante entender que las aplicaciones del cliente pueden utilizar nuestro objeto aunque cambiemos la implementación, siempre que no cambiemos la interfaz. Siempre que se mantengan sin cambio nuestro nombre de método, su lista de parámetro y el tipo de datos de devolución, podremos cambiar la implementación totalmente.

Estado: El estado o los datos de un objeto es lo que lo hace diferente de otros objetos de la misma clase. El estado se describe a través de las variables del Miembro o de la Instancia. Las variables del miembro son aquellas declaradas, de tal manera que están disponibles para todo el código dentro de la clase. Por lo general, las variables del miembro son Privadas en su alcance. Algunas veces, se les conoce como variables de la instancia o como atributos. Observe

que las propiedades no son variables del Miembro, ya que son un tipo de método que funciona para recuperar y establecer valores.

3.2.1 Análisis

El modelo de análisis se extiende luego para describir la manera en que interactúan los actores y el sistema para manipular el modelo del dominio de aplicación. Los desarrolladores usan el modelo de análisis, junto con los requerimientos no funcionales, para preparar la arquitectura del sistema que se desarrolla durante el diseño de alto nivel.

Las actividades del análisis: la identificación de objetos, su comportamiento, sus relaciones, su clasificación y su organización.

El **modelo de análisis** está compuesto por tres modelos individuales: el modelo funcional, representado por casos de uso y escenarios, el modelo de objetos de análisis, representado por diagramas de clase y objeto, y el modelo dinámico, representado por gráficas de estado y diagramas de secuencia.

Conceptos de análisis

Particularmente describimos: El análisis para el enfoque orientado a objetos; En lugar de considerar el **SW** desde una perspectiva básica de **entrada-proceso-salida** como los métodos estructurados se basa en modelar el sistema mediante los objetos que forman parte de el y las relación estáticas (herencia y composición)o dinámicas (uso entre otros objetos).El análisis identifica las clases y objetos relativamente en el dominio del problema, el diseño proporciona detalles sobre la arquitectura, las interfaces y los componentes la implementación transforma el diseño en código y la pruebas checan tanto la arquitectura como la interfaz y los componentes. El enfoque para el análisis orientado a objetos esta basado en cinco capas. Esas cinco

capas consisten de capa **clase/objeto**, capa de estructura, capa de atributos, capa de servicios, y capa de tema.

Estas capas dan mayor poder a la representación de la complejidad del análisis y el diseño en sistemas flexibles.

1.- Capa Clase/Objeto: Esta capa del análisis y diseño indica las clases y objetos.

2.- Capa de Estructura: Esta capa captura diversas estructuras de clases y objetos, como las relaciones uno a muchos.

3.- Capa de Atributos: Esta capa detalla los atributos de las clases.

4.- Capa de Servicios: Esta capa indica los mensajes y comportamientos de los objetos.

5.- Capa de Tema: Esta capa divide el diseño en unidades de implementación o asignaciones de equipos.

Análisis de Clases y Objetos

Objeto: Es una **abstracción** de algo en un dominio de un problema que refleja las capacidades de un sistema para llevar información acerca de ello, interactuar con ello o ambas cosas. Una encapsulación de valores de atributos y sus servicios exclusivos.

Clase: Una descripción de uno o más objetos con un conjunto de atributos y servicios uniformes, incluyendo una descripción de cómo crear nuevos objetos en la clase.

En el análisis de clases y Objetos se identifican los objetos y las clases como candidatos para la aplicación a desarrollar. Las clases y objetos se pueden obtener haciendo un análisis gramatical de la descripción del problema, subrayando términos que podrían ser representados con objetos dentro del sistema.

Para representar las clases, los objetos y las clases objetos, se utiliza la siguiente notación

Los objetos que tienen ocurrencia de una clase son representados por un cuadro sombreado rodeado por la clase. Debido a que los objetos tienen ocurrencias de una clase, no es posible que objetos existan independientemente de su clase.

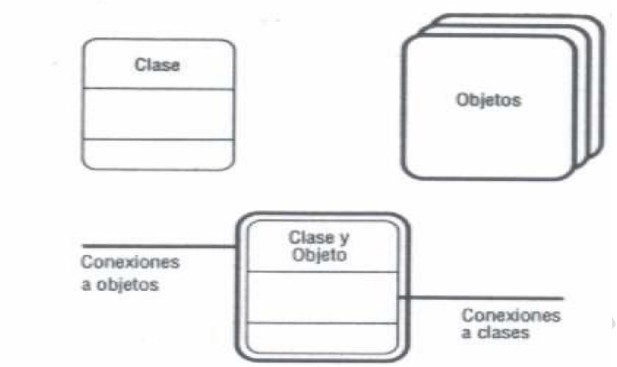


Figura 15. Clasificación de las clases, los objetos y las clases objetos.

3.2.2 Diseño.

Durante el diseño de objetos cerramos el hueco entre los objetos de aplicación y los componentes hechos, identificando objetos de solución adicionales y refinando los objetos existentes.

El diseño de objetos incluye:

- ✓ **Especificación de servicios:** La cual describimos con precisión cada interfaz de clase.
- ✓ **Selección de componentes:** La cual identificamos componentes hechos y objetos de solución adicionales.
- ✓ **Reestructuración del modelo de objetos:** El cual transformamos el modelo de diseño de objetos para mejorar su comprensibilidad y extensibilidad.
- ✓ **Optimización del modelo de objetos**: La cual transformamos el modelo de diseño de objetos para tratar criterios de desempeño, como el tiempo de respuesta o la utilización de la memoria.

El diseño de objetos, al igual que el diseño del sistema, no es algorítmico, a veces es difícil distinguir claramente del análisis y el diseño Orientado a Objetos (OO).

Diseño Orientado a Objetos

El **enfoque** plantea que el **análisis** es razonablemente independiente de la tecnología, en cambio el **diseño** viene a ser entonces cada vez más orientado hacia un lenguaje OO particular y a un ambiente de desarrollo.

Las actividades de diseño OO están agrupadas en los cuatro componentes principales del sistema final: **El componente del problema, el componente de interfaz humana, el componente de manejo de datos y el componente de manejo de tareas**, todos ellos están expandido a lo largo de las cinco capas con que cuenta el diseño OO, mismas que se presentaron anteriormente en el análisis OO.

Actividad correspondiente a la unidad 3

Encierra la respuesta correcta

1.-¿Es un conjunto de tres elementos que facilitan el control sobre el proceso de desarrollo de software?

a) Mitos b) Herramientas c) Paradigma

2.-¿Están compuestos por figuras conectadas con flechas. Para ejecutar un proceso comienza por el Inicio y se siguen las acciones indicadas por cada figura?

a) Enfoque estructurado b) Diagramas de flujos c) Proceso

3. ¿Recibe el nombre del primer componente de diagrama de flujo de datos?

a) Enfoque estructurado b) Paradigma c) Proceso

4. Se utiliza para modelar una colección de paquetes de datos en reposo. Se denota por dos líneas paralelas.

a) Almacén de datos b) Entidad externa c) diccionario de datos

5. Esta capa captura diversas estructuras de clases y objetos, como las relaciones uno a muchos.

a) Capa de Atributos b) Capa de Servicios c) Capa de estructura

UNIDAD 4
MODELOS DE PROCESOS DEL SOFTWARE

Objetivo: Identificará los diferentes modelos de proceso que se aplican en el desarrollo de software.

Cada modelo es una descripción de un proceso software que se presenta desde una perspectiva particular. Alternativamente, a veces se usan los términos ciclo de vida y Modelo de ciclo de vida.

Cada modelo describe una sucesión de fases y un encadenamiento entre ellas. Según las fases y el modo en que se produzca este encadenamiento, tenemos diferentes modelos de proceso.

Un modelo es más adecuado que otro para desarrollar un proyecto dependiendo de un conjunto de características de éste. Existe una gran variedad de modelos diferentes entre los que tenemos los que se describen a continuación.

4.1 Modelo de cascada

El modelo lineal (o modelo en cascada).

Es el más antiguo de todos los modelos de Ingeniería del Software. El modelo lineal presenta una estructura secuencial (de ahí el nombre de Modelo en cascada) formada por seis fases o etapas:

- **Análisis del Sistema.**
- **Análisis de Requisitos de Software.**
- **Diseño.**
- **Codificación.**
- **Prueba.**
- **Mantenimiento.**

Las fases incluyen dentro de sí determinadas tareas que clasifican de una forma clara el trabajo a realizar. El desarrollo de las fases, como he mencionado antes, se produce de manera secuencial. Una vez se produce el análisis tanto del Sistema como de los requisitos del software demandado por el cliente, (fases en las que la intervención del cliente es absolutamente necesaria), se procede a la fase de diseño de la arquitectura global del software.

Un diseño elaborado de forma cuidadosa llevará a una rápida codificación. Tras haber traducido el programa a un lenguaje comprensible para el ordenador, se comprueban los elementos de forma individual y más tarde de manera homogénea (todos los sistemas a la vez). Una vez entregado el software al cliente, la fase de Mantenimiento comprenderá las actualizaciones y las correcciones de errores que sean necesarias en el programa.

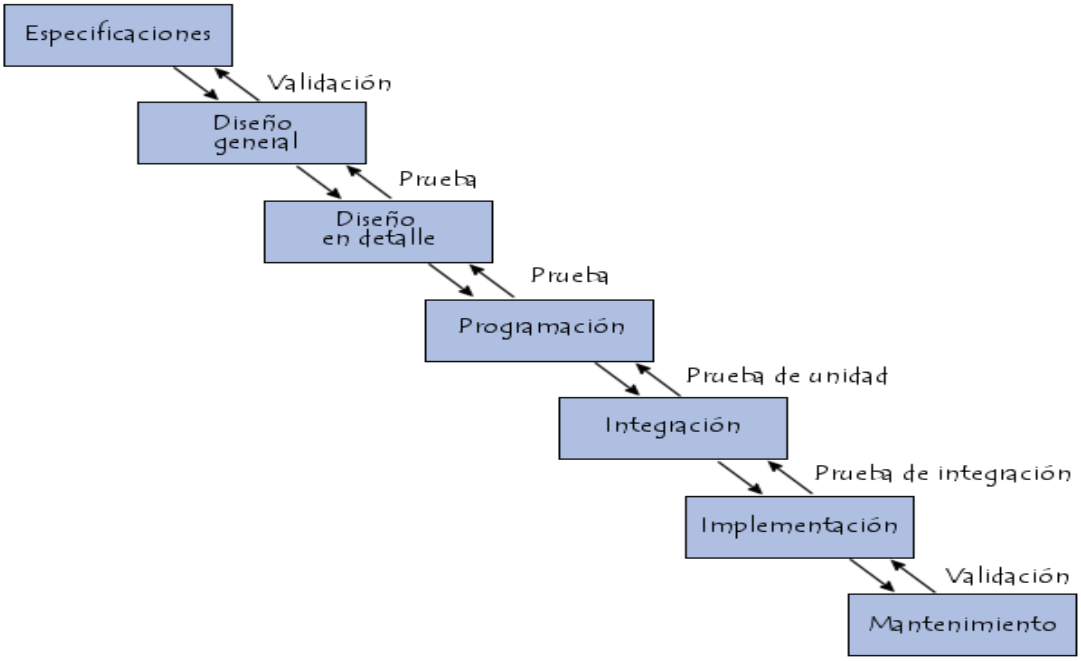


Figura 16: Modelo cascada

4.2 El modelo en espiral.

El modelo espiral para la ingeniería de software ha sido desarrollado para cubrir las mejores características tanto del ciclo de vida clásico, como de la creación de prototipos, añadiendo al mismo tiempo un nuevo elemento: el análisis de riesgo. El modelo representado mediante la espiral de la **figura 16**, define cuatro actividades principales:

- **Planificación:** Determinación de objetivos, alternativas y restricciones.
- **Análisis de riesgo:** Análisis de alternativas e identificación/resolución de riesgos.
- **Ingeniería:** Desarrollo del producto del "siguiente nivel",
- **Evaluación del cliente:** Valorización de los resultados de la ingeniería.

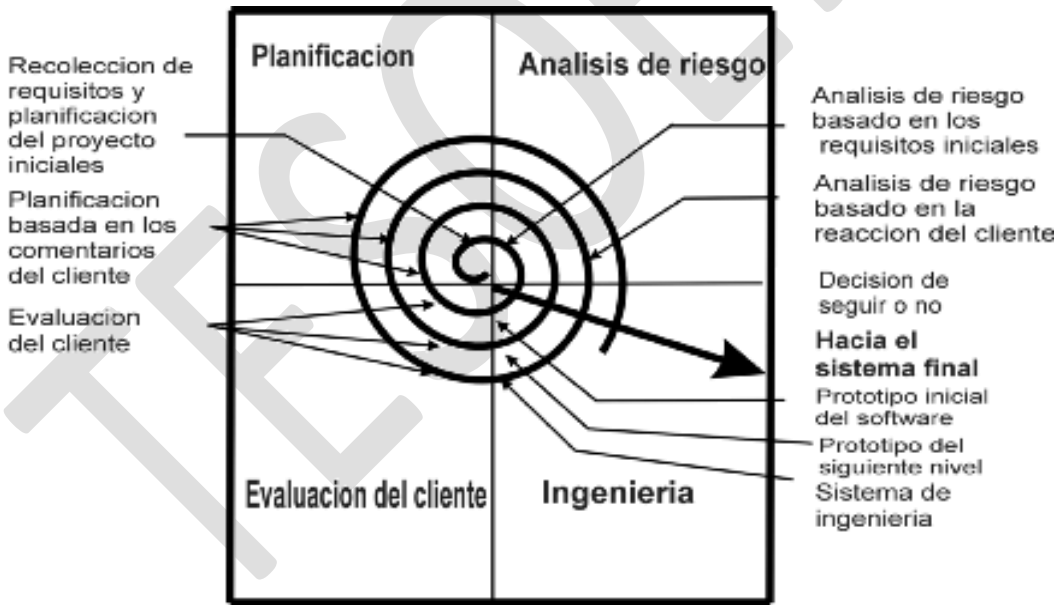


Figura 17: Modelo Espiral

Durante la primera vuelta alrededor de la espiral se definen los objetivos, las alternativas y las restricciones, y se analizan e identifican los riesgos. Si el análisis de riesgo indica que hay una incertidumbre en los requisitos, se puede usar la creación de prototipos en el cuadrante de ingeniería para dar asistencia tanto al encargado de desarrollo como al cliente.

El cliente evalúa el trabajo de ingeniería (cuadrante de evaluación de cliente) y sugiere modificaciones. Sobre la base de los comentarios del cliente se produce la siguiente fase de planificación y de análisis de riesgo. En cada bucle alrededor de la espiral, la culminación del análisis de riesgo resulta en una decisión de "seguir o no seguir".

Con cada iteración alrededor de la espiral (comenzando en el centro y siguiendo hacia el exterior), se construyen sucesivas versiones del software, cada vez más completa y, al final, al propio sistema operacional.

El paradigma del modelo en espiral para la ingeniería de software es actualmente el enfoque más realista para el desarrollo de software y de sistemas a gran escala.

4.3 Modelo incremental.

Descripcion

- Combina elementos del modelo lineal con la filosofía de creación de prototipos.
- El primer incremento a menudo es un producto esencial (núcleo).
- A partir de la evaluación se planea el siguiente incremento y así sucesivamente.
- Es interactivo por naturaleza.
- Es útil cuando el personal no es suficiente para la implementación completa.

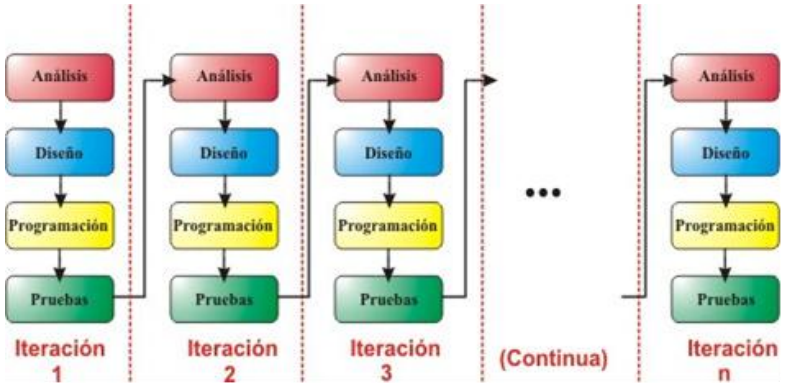


Figura 18: Modelo Incremental

Ventajas.

- Se puede financiar el proyecto por partes.
- Apropiado para proyectos grandes de larga duración.
- No se necesita tanto personal al principio como para una implementación completa.

4.4 Proceso de desarrollo unificado.

Durante varios años se ha utilizado el modelo tradicional en cascada, demostrando en la práctica que no refleja en la realidad la complejidad inherente al proceso de desarrollo de software.

Como una alternativa de solución a este problema, se definieron posteriormente los modelos iterativos e incrementales que trabajan adecuadamente con niveles altos de riesgo, y permiten entregar liberaciones de software en etapas tempranas; tal es el caso del Proceso Unificado propuesto por IBM, que incluye prácticas claves y aspectos relacionados a la planeación estratégica y administración de riesgos.

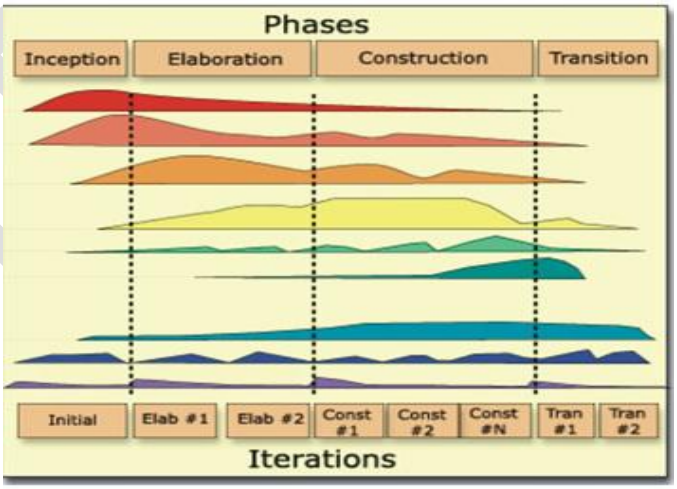


Figura 19: Proceso de desarrollo unificado

El proceso **unificado conocido como RUP**, es un modelo de software que permite el desarrollo de software a gran escala, mediante un proceso continuo de pruebas y retroalimentación, garantizando el cumplimiento de ciertos estándares de calidad.

Fase de concepción.

Esta fase tiene como propósito definir y acordar el alcance del proyecto con los patrocinadores, identificar los riesgos potenciales asociados al proyecto, proponer una visión muy general de la arquitectura de software y producir el plan de las fases y el de iteraciones.

Fase de elaboración.

Se seleccionan los casos de uso que permiten definir la arquitectura base del sistema y se desarrollaran en esta fase, se realiza la especificación de los casos de uso seleccionados y el primer análisis del dominio del problema, se diseña la solución preliminar.

Fase de construcción.

El propósito de esta fase es completar la funcionalidad del sistema, para ello se deben clarificar los requerimientos pendientes, administrar los cambios de acuerdo a las evaluaciones realizados por los usuarios y se realizan las mejoras para el proyecto.

Fase de transición.

El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados en las pruebas de aceptación, capacitar a los usuarios y proveer el soporte técnico necesario.

4.5 Proceso Software Personal.

PROCESO DEL SOFTWARE PERSONAL (PSP)

Cada desarrollador usa distintos procesos para construir un software, estos pueden ser no eficientes o exitosos o también pueden cambiar a diario, pero existe un proceso.

WATTS HUMPHREY dice que para cambiar un proceso inefectivo se tiene que pasar por cuatro fases y estas requieren capacitación e instrumentación. PSP

resalto la medida personal al profesional de la planeación, también hace responsables al profesional de la planeación del proyecto y la calidad de todos los productos.

Existen 5 actividades de marco de trabajo que son:

1. **Planeación:** Aquí se selecciona los requisitos y se desarrolla el tamaño y la estimación de los recursos. Estas mediciones se anotan en las plantillas y al final se identifican las tareas de desarrollo y se crea un programa del proyecto.
2. **Diseño de alto nivel:** Se analizan los factores externos y se construyen prototipos cuando hay incertidumbre.
3. **Revisión del diseño de alto nivel:** Se aplican los métodos de verificación a los errores que se descubrieran en el diseño.
4. **Desarrollo:** Se refina y revisa el diseño y se verifica el código y se compila, además todas las mediciones se guardan para los resultados de trabajo.
5. **Análisis de resultados:** Aquí se determina la efectividad del proceso, analizando todos los datos que se tienen.

El **PSP** destaca que cada ingeniero tiene la necesidad de identificar los errores y de entender la importancia y los tipos de errores que suelen cometerse.

Descripción.

- En el año de 1995 el PSP fue propuesto por Watts Humphrey, este inicialmente estaba dirigido para estudiantes.
- Para 1997 con el lanzamiento del libro "An Introduction to the Personal Software Process" el PSP ya estaba destinado a los ingenieros.
- El PSP se caracteriza porque es de uso personal y se aplica a programas pequeños de menos de 10.000 líneas de código.

- El PSP sirve para producir software de calidad, donde cada ingeniero debe trabajar en la necesidad de realizar trabajo de calidad.
- El PSP busca proporcionar un marco de trabajo para el personal involucrado en el proceso de desarrollo de software.

Objetivos de PSP.

- Lograr una disciplina de mejora continua en el proceso de desarrollo.
- Mejorar la calidad del proceso de desarrollo.
- En general, PSP provee calidad y productividad.

Desventajas de aplicar PSP.

- El tiempo requerido para conocerlo.
- El costo emocional por mantener una disciplina.

Ventajas de aplicar PSP.

- La idea de que ganamos en talento y habilidad.
- La estimulación por nuevas ideas.
- Tomar control del propio trabajo.
- La convicción de que es lo mejor que se puede hacer.

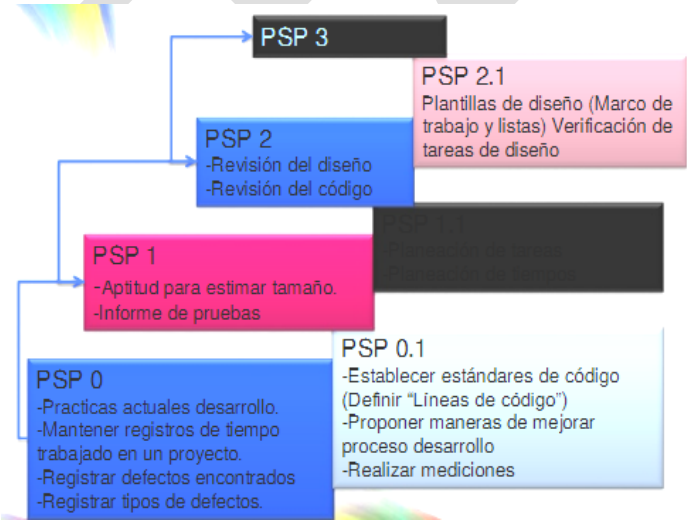


Figura 20: Proceso de Software Personal.

ACTIVIDAD CORRESPONDIENTE A LA UNIDAD 4

Realiza un cuadro sinóptico general de los modelos de proceso de software.

UNIDAD 5

TECNICAS, HERRAMIENTAS Y ESTUDIOS PREVIOS

Objetivo: Diseñara esquemas relacionales de base de datos.

5.1 Técnicas de recopilación de información.

Los analistas utilizan una variable de métodos a fin de recopilar los datos sobre una situación existente, como entrevistas, cuestionario, inspección de registros y observación. Cada uno tiene ventajas y desventajas. Generalmente, se utilizan dos o tres técnicas para complementar el trabajo de cada una y ayudar a asegurar una investigación completa. A continuación se verán cada una de ellas.

5.1.1 E n t r e v i s t a

Las entrevistas se utilizan para recabar información de forma verbal, a través de preguntas que propone el analista. Quienes responden pueden ser gerentes o empleados, los cuales son usuarios actuales del sistema, existen usuarios potenciales del sistema propuesto o aquellos que proporcionaran datos o serán afectadas por la aplicación propuesta.

El analista puede entrevistar al personal de forma individual o en grupos. Para poder recabar los datos mediante la entrevista.

La entrevista es una forma de conversación. Al analizar las características de los sistemas con personal seleccionado cuidadosamente, por sus conocimientos sobre ese sistema los analistas pueden conocer los datos que no están disponibles en ninguna otra forma.

En las investigaciones de sistemas, existen dos formas de información la forma cualitativa y cuantitativa:

La información cualitativa: está relacionada con opiniones, políticas y algunas otras descripciones.

La información cuantitativa: trata con números, frecuencia o cantidades. Las entrevistas dan la mejor fuente de información cualitativa; los otros métodos tienden a ser más útiles en la recopilación de datos cuantitativos.

Determinación del tipo de entrevista.

La estructura de las entrevistas varía. Si el objetivo de la entrevista radica en adquirir información general, es conveniente elaborar una serie de preguntas sin estructura, con una sección de preguntas y respuestas libres. Sin embargo, cuando los analistas necesitan adquirir datos más específicos sobre la aplicación o desean asegurar una alta confiabilidad en las respuestas a las preguntas que han propuesto a sus entrevistados, las entrevistas estructuradas son mejores

Las entrevistas estructuradas utilizan preguntas estandarizadas. El formato de respuestas para las preguntas puede ser abiertas o cerradas; para las respuesta abierta permiten a los entrevistados dar cualquier respuesta que parezca apropiada. Con las respuestas cerradas se proporciona al usuario un conjunto de respuestas que se pueda seleccionar.

Por ejemplo:

Si dijera, "hola, fui enviado para encontrar una forma de mojar el rendimiento y de reducir los errores que presentan aquí. O la introducción: "Estamos aquí para resolver su problema", es igualmente mala.

A través de la entrevista, los analistas deben preguntarse a sí mismos las siguientes interrogantes:

¿Qué es lo que me está diciendo la persona?

¿Por qué me lo está diciendo a mí?

¿Qué se está olvidando?

¿Qué espera esta persona que haga yo?

Si se considera cada elemento de la información contra estas preguntas, los analistas tendrán más conocimientos no solamente de la información adquirida sino también de su importancia.

5.1.2 C u e s t i o n a r i o.

Los cuestionarios proporcionan una alternativa muy útil para las entrevistas; sin embargo, existen ciertas características que pueden ser apropiadas en algunas situaciones e inapropiadas en otras.

La recopilación de datos mediante cuestionarios. Para los analistas los cuestionarios puede ser la única forma posible de relacionarse con un gran número de personas para conocer varios aspectos del sistema.

Cuando se llevan a cabo largos estudios en varios departamentos, se puede distribuir los cuestionarios a todas las personas apropiadas para recabar hechos con relación al sistema. Por supuesto, no es posible observar las expresiones o relaciones de quienes responden a los cuestionarios.

Las preguntas estandarizadas pueden proporcionar datos más confiables. Por otra parte, las características anteriores también son desventajas de los cuestionarios. Aunque su aplicación puede realizarse con un mayor número de individuos, es muy rara una respuesta total. Puede necesitarse algún seguimiento de los cuestionarios para motivar al personal que responda; todas las respuestas se encontraran en una proporción entre el 25 o 35%, que es lo más común.

Existen dos formas de cuestionarios para recabar datos; cuestionario abierto y cerrado, se aplican dependiendo de los analistas que conocen de antemano todas las posibles respuestas de las preguntas y pueden incluirlos. Con frecuencia se utilizan ambas formas en los estudios de sistemas.

Cuestionarios abiertos.

Los cuestionarios pueden ser abiertos y se aplica cuando se quiere conocer los sentimientos, opiniones y experiencias generales; también son útiles al explorar el problema básico.

Por ejemplo:

Un analista que utiliza cuestionarios para estudiar los métodos de verificación de un crédito, en un medio ambiente de ventas al menudeo, podría recabar más información provechosa de una pregunta abierta de este tipo: ¿Cómo podría simplificarse y mejorarse el proceso de verificación de crédito para los clientes?

Cuestionarios cerrados.

El cuestionario cerrado limita las respuestas posibles del interrogado. Por medio de un cuidadoso estilo en la pregunta, el analista puede controlar el marco de referencia. Este formato es el mejor método para obtener información sobre los hechos. También forzar a los individuos para que tomen una posición y forma de opinión sobre los aspectos importantes.

Selección de quienes recibirán el cuestionario.

Aquellas personas que reciban el cuestionario deben seleccionarse de a cuerdo con la información que puedan proporcionar. Escribir o imprimir un cuestionario no significa que se pueda distribuir ampliamente sin un análisis previo. Lo pueden contestar personas no calificadas y si el cuestionario no es anónimo, y no será posible retirar sus respuestas de la muestra. La realización de esto también es desgastante y cara.

5.1.3 Recopilación y análisis de documentos.

Recopilación de datos: Deberá dirigirse al registro de aquellos hechos que permitan conocer y analizar lo que realmente sucede en la unidad o tema que

se investiga. Esto consiste en la recolección, síntesis, organización y comprensión de los datos que se requieren.

Se conocen dos tipos de fuentes:

1. Primarias: que contienen información original no abreviada ni traducida.

2. Secundarias: obras de referencia que auxilian al proceso de investigación.

Se conoce otra división que se conforma por las siguientes fuentes:

-Documentales

-De campo.

Las fuentes de recolección de datos son todos los registros de aquellos hechos que permitan conocer y analizar lo que realmente sucede en el tema que se investiga.

Concluida la parte preparatoria de la investigación se inicia la fase de recopilación de datos.

Para recabar la información existente sobre el tema, el investigador se auxilia de instrumentos como las fichas de trabajo; hay diversos tipos de fichas de trabajo como:

Fichas de trabajo para fuentes documentales, fichas de trabajo de una revista, fichas de trabajo de un periódico, para investigación de campo, para la observación, fichas bibliográficas y hemerográficas.

Análisis e interpretación de información.

La interpretación de los resultados de la indagación lleva inmediatamente a la solución.

El análisis del instrumento de recolección de información de campo (encuesta), fue utilizando el análisis individual de preguntas que se realiza con base en los porcentajes que alcanzan las distintas respuestas de cada pregunta.

Para llevar a cabo este tipo de análisis se diseño una forma donde se tabulan las respuestas en base a la cantidad de personas que contestaron cada respuesta y el porcentaje que representa del total de la muestra.

Redacción y presentación del informe.

El objetivo del informe es presentar a los lectores el proceso que se realizó para presentar una solución al problema planteado, para lo cual es necesario hacer la presentación del problema, los métodos empleados para su estudio, los resultados obtenidos, las conclusiones a las que se llegaron y las recomendaciones en base a estas.

Con respecto a la estructura del informe, ésta es sencilla y sigue fielmente los pasos fundamentales del diseño de la investigación, ya que el informe debe ser la respuesta a lo planteado por el diseño de investigación.

5.1.4 Observación y técnica “STROBE”.

Confirma o niega la narración organizacional de entrevistas o cuestionarios.

La observación es sistemática:

1. Sigue una metodología estándar y una clasificación estándar para el análisis de los elementos organizacionales que influencian la toma de decisiones.

2. Permite que otros analistas apliquen el mismo marco de trabajo analítico a la misma organización.

3. Limita el análisis a la organización como existe durante la etapa de su ciclo actual de vida.

Elemento STROBE son 7:

1. Ubicación de la oficina. La ubicación de la oficia de un tomador de decisiones particular con respecto a las demás oficinas.

2. Ubicación del escritorio del tomador de decisiones. Proporciona pistas sobre el ejercicio de poder por el tomador de decisiones.

3. Equipo de oficina fijo. Se conforma de archiveros, libreros, etc.

4. Propiedades. Todo el equipo pequeño que se usa para procesar información (calculadoras, pantallas de video, lápices, etc.).

5. Revistas y periódicos del negocio. Estas revelan si el tomador de decisiones busca información externa o se apoya más en información interna.

6. Iluminación y color de la oficina. Nos indica la manera en que el tomador de decisiones recopila información.

7. Vestimenta usada por los tomadores de decisiones.

El analista de sistemas puede obtener una comprensión de la credibilidad exhibida por los gerentes de la organización observando la vestimenta que usan en el trabajo.

Mediante el uso de STROBE el analista de sistemas puede obtener una mejor comprensión sobre la manera en que los gerentes recopilan, procesan, almacenan y usan información.

5.2 Herramientas CASE.

Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, calculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Objetivos

1. Mejorar la productividad en el desarrollo y mantenimiento del software.

2. Aumentar la calidad del software.
3. Mejorar el tiempo y coste de desarrollo y mantenimiento de los sistemas informáticos.
4. Mejorar la planificación de un proyecto
5. Aumentar la biblioteca de conocimiento informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
6. Automatizar, desarrollo del software, documentación, generación de código, pruebas de errores y gestión del proyecto.
7. Ayuda a la reutilización del software, portabilidad y estandarización de la documentación
8. Gestión global en todas las fases de desarrollo de software con una misma herramienta.
9. Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

Clasificación de las H.CASE

Aunque no es fácil y no existe una forma única de clasificarlas, las herramientas CASE se pueden clasificar teniendo en cuenta los siguientes parámetros:

1. Las plataformas que soportan.
2. Las fases del ciclo de vida del desarrollo de sistemas que cubren.
3. La arquitectura de las aplicaciones que producen.
4. Su funcionalidad.

La siguiente clasificación es la más habitual basada en las fases del ciclo de desarrollo que cubren, **Upper CASE (U-CASE),** herramientas que ayudan en las fases de planificación, análisis de requisitos y estrategia del desarrollo, usando, entre otros diagramas UML.

- **Middle CASE (M-CASE),** herramientas para automatizar tareas en el análisis y diseño de la aplicación.
- **Lower CASE (L-CASE),** herramientas que semiautomática la generación de código, crean programas de detección de errores, soportan la depuración de programas y pruebas. Además automatizan la documentación completa de la aplicación. Aquí pueden incluirse las herramientas de Desarrollo rápido de aplicaciones.

Existen otros nombres que se le dan a este tipo de herramientas, y que no es una clasificación excluyente entre si, ni con la anterior

- **Integrated CASE (I-CASE),** herramientas que engloban todo el proceso de desarrollo software, desde análisis hasta implementación.
- **Meta CASE,** herramientas que permiten la definición de nuestra propia técnica de modelado, los elementos permitidos del metamodelo generado se guardan en un repositorio y pueden ser usados por otros analistas, es decir, es como si definiéramos nuestro propio UML, con nuestros elementos, restricciones y relaciones posibles.
- **CAST (Computer-Aided Software Testing),** herramientas de soporte a la prueba de software.
- **IPSE (Integrated Programming Support Environment),** herramientas que soportan todo el ciclo de vida, incluyen componentes para la gestión de proyectos y gestión de la configuración.

Por funcionalidad podríamos diferenciar algunas como:

- Herramientas de generación semiautomática de código.
- Editores UML.
- Herramientas de Refactorización de código.

- Herramientas de mantenimiento como los sistemas de control de versiones

5.2.1 Estructuradas.

Aparecieron a fines de los 60's con la Programación Estructurada, posteriormente a mediados de los 70's extendidas con el Diseño Estructurado y a fines de los 70's con él. Análisis Estructurado. Versiones más recientes incorporan Diagramas Entidad-Relación y Diagramas de Transición de Estados.

Ejemplos de metodologías estructuradas impulsadas por organismos gubernamentales lo constituyen: MERISE (Francia), METRICA (España), SSADM (Reino Unido). ejemplo

Otras metodologías estructuradas en el ámbito académico y comercial son: Gane & Sarson, Ward & Mellor, Yourdon & DeMarco y Information Engineering. Esta última propuesta por James Martin pone un énfasis adicional en el modelado de datos y la incorporación de los desarrollos informáticos dentro del contexto organizacional (planificación, objetivos, etc.).

5.2.2 Orientadas a Objetos.

Su historia va unida a la evolución de los lenguajes de programación orientada a objeto, los más representativos: a fines de los 60's SIMULA, a fines de los 70's Smalltalk-80, la primera versión de C++ por Bjarne Stroustrup en 1981 y actualmente Java. Sólo a fines de los 80's comenzaron a consolidarse algunas metodologías Orientadas a Objetos.

Algunas de las más representativas en el ámbito comercial son: OMT de Rumbaugh, OOAD de Grady Booch, OOSE de I.Jacobson, la propuesta por Peter Coad & Edward Yourdon, la propuesta por S. Shaler & S.J. Mellor y la propuesta por J.Martin y J.J.Odell. En los últimos años se ha realizado un

esfuerzo de estandarización notacional materializado en UML (Unified Modeling Language), el cual aglutina característica de algunas de las propuestas OO más conocidas. En 1997 UML fue aprobado como notación estándar por el OMG.

5.3 Desarrollo de prototipos

¿Qué es un Prototipo?

Es un modelo a escala o facsímil de lo real, pero no tan funcional para que equivalga a un producto final, ya que no lleva a cabo la totalidad de las funciones necesarias del sistema final. Proporcionando una retroalimentación temprana por parte de los usuarios acerca del Sistema.

Importancia de Definir su Objetivo.

Siempre se debe establecer cuál es su objetivo, ya que un prototipo puede ser útil en diferentes fases del proyecto, por ello su objetivo debe ser claro. Durante la fase de análisis se usa para obtener los requerimientos del usuario. En la fase de diseño se usa para ayudar a evaluar muchos aspectos de la implementación seleccionada.

Propósitos del Prototipo.

En la fase de Análisis de un proyecto, su principal propósito es obtener y validar los requerimientos esenciales, manteniendo abiertas, las opciones de implementación. Esto implica que se debe tomar los comentarios de los usuarios, pero debemos regresar a sus objetivos para no perder la atención. En la fase de Diseño, su propósito, basándose en los requerimientos previamente obtenidos, es mostrar las ventanas, su navegación, interacción, controles y botones al usuario y obtener una retroalimentación que nos permite mejorar el Diseño de Interfaz.

Características de los Prototipos.

El proceso de desarrollo y empleo de prototipos tiene las siguientes características:

Los prototipos se crean con rapidez.

Los prototipos evolucionan a través de un proceso iterativo.

Los prototipos tienen un costo bajo de desarrollo.

El profesional de Sistema por medio de la observación, evaluación y la retroalimentación, obtendrá cómo reaccionan los usuarios al trabajar con el prototipo, y que tan conveniente es el acoplamiento entre las necesidades y las características modeladas en el sistema. A través de la recopilación de tales reacciones, el profesional, irá descubriendo nuevas perspectivas del prototipo, incluso si los usuarios se encuentran satisfechos con él, o si habrá dificultades para vender o implantar el sistema.

Desarrollo de Prototipo.

Para decidir si el prototipo debe incluirse o no Ciclo de Desarrollo de Sistema de Información, el profesional considera los siguientes factores:

Problemas no estructurado, novedosos y complejos, de información personalizada del usuario, ya que sus salidas no son predecibles y definidas Problemas de ambiente Inestable, el profesional también debe evaluar el contexto del sistema.

Etapas del Prototipo

El desarrollo de un prototipo se lleva a cabo en forma ordenada a través de las siguientes etapas.

ACTIVIDAD CORRESPONDIENTE A LA UNIDAD 5

Realiza un mapa mental de las técnicas de recopilación de información.

UNIDAD 6

DISEÑO Y ARQUITECTURA DE PRODUCTOS DE SOFTWARE

Objetivo: Comprenderá las arquitecturas en el diseño del software dependiendo del tipo de dominio de la aplicación.

Diseño y arquitectura de productos de software.

Consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información.

6.1 Descomposición modular.

El diseño modular es una metodología de desarrollo de programas complejos, que utiliza la filosofía TOP-DOWN, conocida también como diseño descendente o refinamiento por pasos sucesivos; o comúnmente conocido por los programadores como “Divide y Vencerás”; puesto que enfrenta un problema desde lo abstracto (TOP) hacia lo particular (DOWN).

Esta técnica consiste en dividir el problema en un conjunto de sub problemas, y estos a su vez en otros de mayor facilidad de trabajo; generando los Módulos Funcionales.

Son deseables para la programación estructurada; que permite resolver cada módulo de forma independiente y, si no se sabe resolver alguno de ellos, puede asignársele un nombre y pasar al desarrollo de otro módulo, más adelante retomar el módulo incompleto para darle solución al obtener mayor conocimiento sobre dicho proceso, es decir, la solución del problema no se detiene por falta de conocimiento de algún proceso en particular.

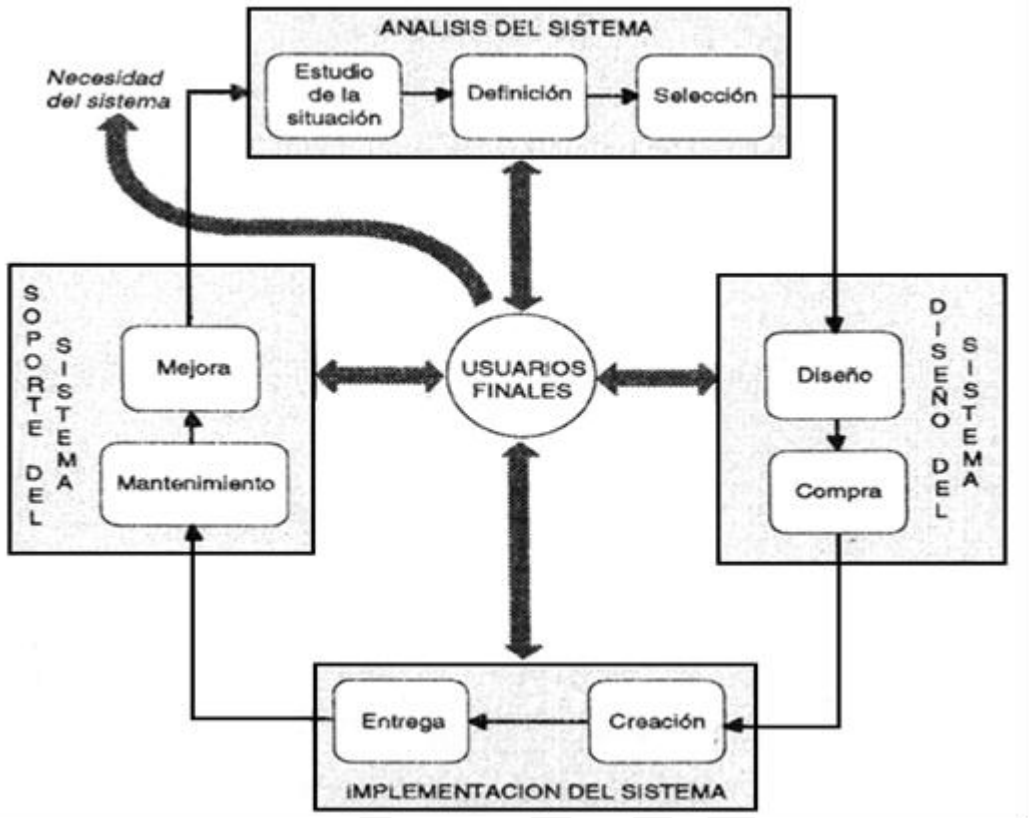


Figura 21. Descomposición modular

6.2 Arquitecturas de dominio específico.

El reto para el diseño es diseñar el software y hardware para proporcionar características deseables a los sistemas distribuidos y, al mismo tiempo, minimizar los problemas propios a estos sistemas. Es necesario comprender las ventajas y desventajas de las diferentes arquitecturas de sistemas distribuidos. Aquí se tratan dos tipos genéricos de arquitecturas de sistemas distribuidos**: Arquitectura cliente-servidor.**

En este caso el sistema puede ser visto como un conjunto de servicios que se proporcionan a los clientes que hacen uso de dichos servicios. Los servidores y los clientes se tratan de forma diferente en estos sistemas.

Arquitecturas de objetos distribuidos. Para esta arquitectura no hay distinción entre servidores y clientes, y el sistema puede ser visto como un

conjunto de objetos que interaccionan cuya localización es irrelevante. No hay distinción entre un proveedor de servicios y el usuario de estos servicios.

Ambas arquitecturas se usan ampliamente en la industria, pero la distribución de las aplicaciones generalmente tiene lugar dentro de una única organización. La distribución soportada es, por lo tanto, intraorganizacional. También se pueden tomar dos tipos más de arquitecturas distribuidas que son más adecuadas para la distribución interorganizacional: arquitectura de sistemas peer-to-peer (p2p) y arquitecturas orientadas a servicios.

Los sistemas **peer-to-peer** han sido usados principalmente para sistemas personales, pero están comenzando a usarse para aplicaciones de empresa.

Los componentes en un sistema distribuido pueden implementarse en diferentes lenguajes de programación y pueden ejecutarse en tipos de procesadores completamente diferentes.

Los modelos de datos, la representación de la información y los protocolos de comunicación pueden ser todos diferentes.

Un sistema distribuido, por lo tanto, requiere software que pueda gestionar estas partes distintas, y asegurar que dichas partes se puedan comunicar e intercambiar datos. El término middleware se usa para hacer referencia a ese software; se ubica en medio de los diferentes componentes distribuidos del sistema.

Los sistemas distribuidos se desarrollan normalmente utilizando una aproximación orientada a objetos. Estos sistemas están formados por partes independientes pobremente integradas, cada una de las cuales puede interaccionar directamente con los usuarios o con otras partes del sistema.

Algunas partes del sistema pueden tener que responder a eventos independientes.

Existen cuatro modelos de dominio específico:

1. **Modelos genéricos que son abstracciones de varios sistemas reales.**
2. **Modelos de referencia que son modelos abstractos y describen a una clase mayor de sistemas.**
3. **Modelo genérico: flujo de datos de un compilador.**
4. **Modelo de Referencia: La arquitectura OSI.**

6.2.1 Diseño de Software Arquitectura Multiprocesador.

Un sistema multiproceso o multitarea es aquel que permite ejecutar varios procesos de forma concurrente, la razón es porque actualmente la mayoría de las CPU's sólo pueden ejecutar un proceso cada vez.

La única forma de que se ejecuten de forma simultánea varios procesos es tener varias CPU's (ya sea en una máquina o en varias, en un sistema distribuido.

La ventaja de un sistema multiproceso reside en la operación llamada cambio de contexto. Esta operación consiste en quitar a un proceso de la CPU, ejecutar otro proceso y volver a colocar el primero sin que se entere de nada.

Es necesario conocer ampliamente como están interconectados dichos procesadores, y la forma en que el código que se ejecuta en los mismos ha sido escrito para escribir aplicaciones y software que aproveche al máximo sus prestaciones.

Para lograrlo, es necesario modificar varias facetas del sistema operativo, la organización del código de las propias aplicaciones, así como los lenguajes de programación.

Se configuran dos computadoras de gran capacidad interconectados electrónicamente entre si. Esta configuración recibe el nombre de multiproceso y se caracteriza porque permite proceso de datos continuo aún en el caso de que surjan problemas de funcionamiento en alguno de las computadoras.

Los sistemas de software compuestos de procesos múltiples no necesariamente son sistemas distribuidos. Si más de un procesador está disponible, entonces se puede implementar la distribución, pero los diseñadores del sistema no siempre consideran lo puntos de distribución durante el proceso de diseño. El enfoque de diseño para este tipo de sistemas es esencialmente el mismo que para los de tiempo real.

Ventajas

• Es económica.

• El uso de componentes comúnmente disponibles, en grandes cantidades, permite ofrecer mayor rendimiento, a un precio menor que el de máquinas con procesadores especialmente diseñados (como por ejemplo las máquinas de procesadores vectoriales y de propósito específico).

• Adicionalmente, las computadoras paralelas son inherentemente escalables, permitiendo actualizarlas para adecuarlas a una necesidad creciente.

• Las arquitecturas “tradicionales” se actualizan haciendo los procesadores existentes obsoletos por la introducción de nueva tecnología a un costo posiblemente elevado. Por otro lado, una arquitectura paralela se puede actualizar en términos de rendimiento simplemente agregando más procesadores.

Desventajas

• En ocasiones se menciona también la limitante física; existen factores que limitan la velocidad máxima de un procesador, independientemente del factor económico.

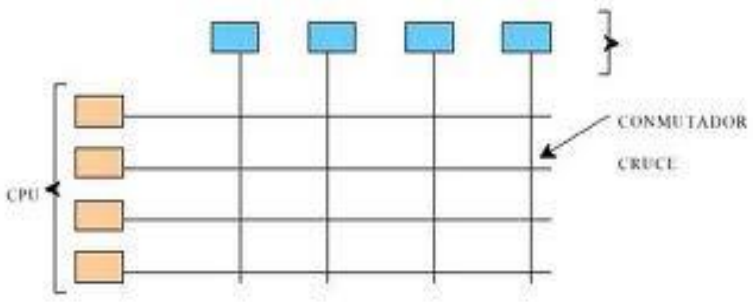


Figura 22. Diseño de software Arquitectura Multiprocesador

Hablando de las arquitecturas hardware como software que se han desarrollado para conseguir hacer trabajo paralelo o distribuido. Estas arquitecturas fueron inventadas para superar los problemas de rendimiento y responden a distintos enfoques para conseguir sacar más rendimiento gracias al paralelismo. Algunas arquitecturas pretenden tener una mejor relación velocidad/precio y otras ser las máquinas más rápidas del mercado.

Las soluciones hardware fueron las primeras en aparecer**, las soluciones software** tuvieron que esperar hasta que el hardware diese la potencia necesaria y en el caso de los sistemas distribuidos se tuvo que esperar a que en los años 70 se desarrollaran las redes de área local. En el capítulo Sistemas operativos se explicará con detalle el software distribuido a nivel de sistema operativo.

Soluciones hardware. Las soluciones hardware han estado en el mercado de la computación desde sus inicios. Para muchas empresas la única manera de crear mejores máquinas era crear arquitecturas paralelas a partir de las que ya poseían.

Como el número de estos tipos de máquinas es elevado, existen numerosas y diversas soluciones. Quizás la división más conocida y básica sea

la taxonomía de Flint. Flint dividió las soluciones hardware en cuatro categorías:

SISD: un flujo de instrucciones único trabaja sobre un flujo de datos único, a esta categoría pertenecen las CPUs simples y las superescalares.

SIMD: un flujo de instrucciones único trabaja sobre un flujo de datos múltiple, en esta categoría tenemos los computadores matriciales.

MISD: un flujo de instrucciones múltiple trabaja sobre un flujo de datos único, resultado teórico de la clasificación, el modelo que más se acerca son los computadores sistólicos.

MIMD: un flujo de instrucciones múltiple trabaja sobre un flujo de datos múltiple, estos son los multiprocesadores y multicomputadores.

Se irá viendo cómo se explota el paralelismo en el hardware a varios niveles: desde lo más pequeño (procesadores) hasta lo más grande (multicomputadores).

6.2.2 Diseño de software de arquitectura Cliente/Servidor.

Esta arquitectura es un modelo para el desarrollo de sistemas de información en el que las transacciones se dividen en procesos independientes que cooperan entre sí para intercambiar información, servicios o recursos.

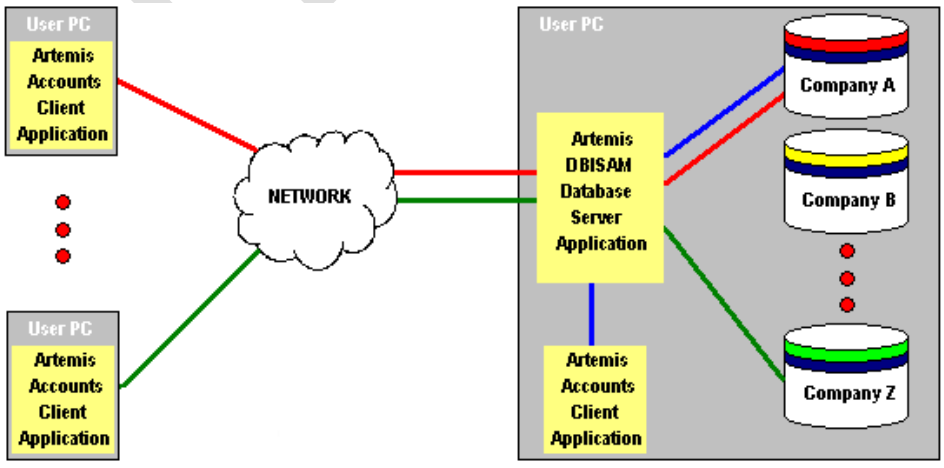


Figura 23: Diseño de software de arquitectura Cliente/Servidor.

Se denomina cliente al proceso que inicia el diálogo o solicita los recursos y servidor al proceso que responde a las solicitudes.

Las aplicaciones se dividen de forma que el servidor contiene la parte que debe ser compartida por varios usuarios, y en el cliente permanece sólo lo particular de cada usuario.

Los clientes realizan generalmente funciones como:

• Manejo de la interfaz de usuario.

• Captura y validación de los datos de entrada.

• Generación de consultas e informes sobre las bases de datos.

Por su parte los servidores realizan, entre otras, las siguientes funciones:

• Gestión de periféricos compartidos.

• Control de accesos concurrentes a bases de datos compartidas

• Enlaces de comunicaciones con otras redes de área local o extensa.

Entre las principales características de la arquitectura cliente/servidor se pueden destacar las siguientes:

• El servidor presenta a todos sus clientes una interfaz única y bien definida.

• El cliente no necesita conocer la lógica del servidor, sólo su interfaz externa.

• El cliente no depende de la ubicación física del servidor, ni del tipo de equipo físico en el que se encuentra, ni de su sistema operativo.

• Los cambios en el servidor implican pocos o ningún cambio en el cliente.

6.2.3 Diseño de software distribuido.

Los Sistemas cuyos componentes hardware y software, que están en ordenadores conectados en red, se comunican y coordinan sus acciones mediante el paso de mensajes, para el logro de un objetivo. Se establece la comunicación mediante un protocolo prefijado por un esquema cliente-servidor.

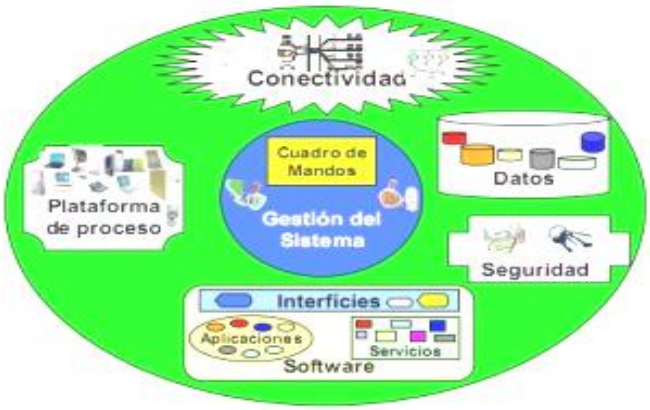


Figura 24: Diseño de software distribuido

Características:

• **Concurrencia.-** Esta característica de los sistemas distribuidos permite que los recursos disponibles en la red puedan ser utilizados simultáneamente por los usuarios y/o agentes que interactúan en la red.

• **Carencia de reloj global.-** Las coordinaciones para la transferencia de mensajes entre los diferentes componentes para la realización de una tarea, no tienen una temporización general, está más bien distribuida a los componentes.

• **Fallos independientes de los componentes.-** Cada componente del sistema puede fallar independientemente, con lo cual los demás pueden continuar ejecutando sus acciones. Esto permite el logro de las tareas con mayor efectividad, pues el sistema en su conjunto continua trabajando.

Procesamiento central (Host).- Uno de los primeros modelos de ordenadores interconectados, llamados centralizados, donde todo el procesamiento de la organización se llevaba a cabo en una sola computadora, normalmente un Mainframe, y los usuarios empleaban sencillos ordenadores personales.

Los problemas de este modelo son

• Cuando la carga de procesamiento aumentaba se tenía que cambiar el hardware del Mainframe, lo cual es más costoso que añadir más computadores personales clientes o servidores que aumenten las capacidades.

• El otro problema que surgió son las modernas interfaces gráficas de usuario, las cuales podían conllevar a un gran aumento de tráfico en los medios de comunicación y por consiguiente podían colapsar.

6.2.4 Diseño Software Tiempo Real.

El software de tiempo real está muy acoplado con el mundo externo, esto es, el software de tiempo real debe responder al ámbito del problema en un tiempo dictado por el ámbito del problema. Debido a que el software de tiempo real debe operar bajo restricciones de rendimiento muy rigurosas, el diseño del software esta conducido frecuentemente, tanto por la arquitectura del hardware como por la del software.

Las características del sistema operativo, por los requisitos de la aplicación y tanto por los extras del lenguaje de programación como prospectos de diseño.

La computadora digital se ha convertido en una maquina omnipresente en la vida diaria de todos nosotros. Las computadoras nos permiten ver juegos, así como contar el tiempo, optimizar el gasto de gasolina de nuestras últimas generaciones de coches y programar a nuestros aparatos.

Todas estas interacciones con las computadoras sean útiles o intrusivas son ejemplos de computación **de tiempo real.** La computadora está controlando algo que interactúa con la realidad sobre una base de tiempo de hecho, el tiempo es la esencia de la interacción.

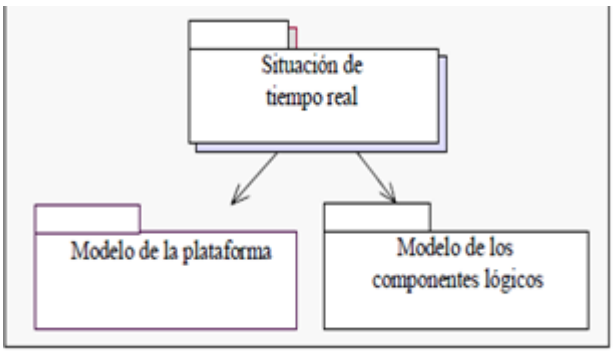


Figura 25: Diseño de software T-R

Diseño del sistema

Parte del proceso de diseño implica decidir qué capacidades del sistema tienen que implementarse en software y cuáles en hardware.

Los sistemas de tiempo real incluyen:

1. Un reloj de tiempo real
2. Un manejador de interrupciones
3. Un planificador
4. Un gestor de recursos
5. Un despachador

El orden de estas actividades en el proceso depende del tipo de sistema que se esté desarrollando y de los requerimientos de su proceso y plataforma.

Un caso real El diseño de un teléfono inalámbrico

El Eole 400 es un teléfono inalámbrico de la clase CT0 de Alcatel, que admite múltiples auriculares, con un contestador automático de estado sólido integrado en la base. Está basado en otros dos productos anteriores, el France Telecom X1 (para el auricular y el circuito de radio de la base) y el Eole 300 (para el resto de la base).

Modelo de la Plataforma: Modela los recursos hardware/software que constituyen la plataforma en que ejecuta la aplicación y que es independiente de ella.

Modelo de los componentes lógicos: Modela el comportamiento de tiempo real de los componentes software con los que se construye la aplicación.

ACTIVIDAD CORRESPONDIENTE A LA UNIDAD 6

Realiza un cuadro sinóptico de diseño y arquitectura de productos de software.

SOLUCION DE LA ACTIVIDAD CORRESPONDIENTE A LA UNIDAD 1

Contesta correctamente las siguientes preguntas

1. ¿Define que es un sistema?

Es un conjunto de unidades recíprocamente relacionadas, de un todo organizado y complejo, entre el sistema y su ambiente admiten cierta arbitrariedad.

2. ¿Define que es entropía?

Es la tendencia de los sistemas a desgastarse, a desintegrarse, para el relajamiento de los estándares y un aumento de la aleatoriedad, esta aumenta con el tiempo ya que si aumenta la información, disminuye la entropía

3. ¿Define el ciclo de vida de un proyecto de software?

Describe el desarrollo de software, desde la fase inicial hasta la fase final. El propósito de este programa es definir las distintas fases intermedias que se requieren para validar el desarrollo de la aplicación del software

4. ¿Define que es programación de sistemas?

Es el encargado de desarrollar software puede instalar paquetes comprados a terceros o escribir programas diseñados a la medida del solicitante.

5¿Define que es el software?

que son Programas de computadora, con estructuras de datos y su documentación que hacen efectiva la logística, metodología o controles de requerimientos del Programa.

6¿Define que es el hardware?

Es un dispositivo electrónico y electromecánico, que proporcionan capacidad de cálculos y funciones rápidas, exactas y efectivas.

SOLUCION DE LA ACTIVIDAD CORRESPONDIENTE A LA UNIDAD 2

Relaciona las columnas correctamente

1.-Es una disciplina o área de la informática que ofrece métodos y técnicas para desarrollar y mantener software de calidad. (2) Software

2.-Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un Sistema de computación. (4) Capacidad individual.

3.-Es una serie de revisiones, y pruebas utilizadas a lo largo del ciclo del desarrollo para asegurar que cada producto cumple con los requisitos que le han sido asignados. (1) Ingeniería de software

4.-En este factor interviene la competencia del individuo y su familiaridad con el área de la aplicación. La comunicación entre los miembros del equipo. (3) Control de la calidad

SOLUCION DE LA ACTIVIDAD CORRESPONDIENTE A LA UNIDAD 3

Encierra la respuesta correcta

1.-Es un conjunto de tres elementos que facilitan el control sobre el proceso de desarrollo de software.

a) Mitos b) Herramientas c) Paradigma

2.-Están compuestos por figuras conectadas con flechas. Para ejecutar un proceso comienza por el Inicio y se siguen las acciones indicadas por cada figura

a) Enfoque estructurado b) Diagramas de flujos c) Proceso

3. Recibe el nombre del primer componente de diagrama de flujo de datos se

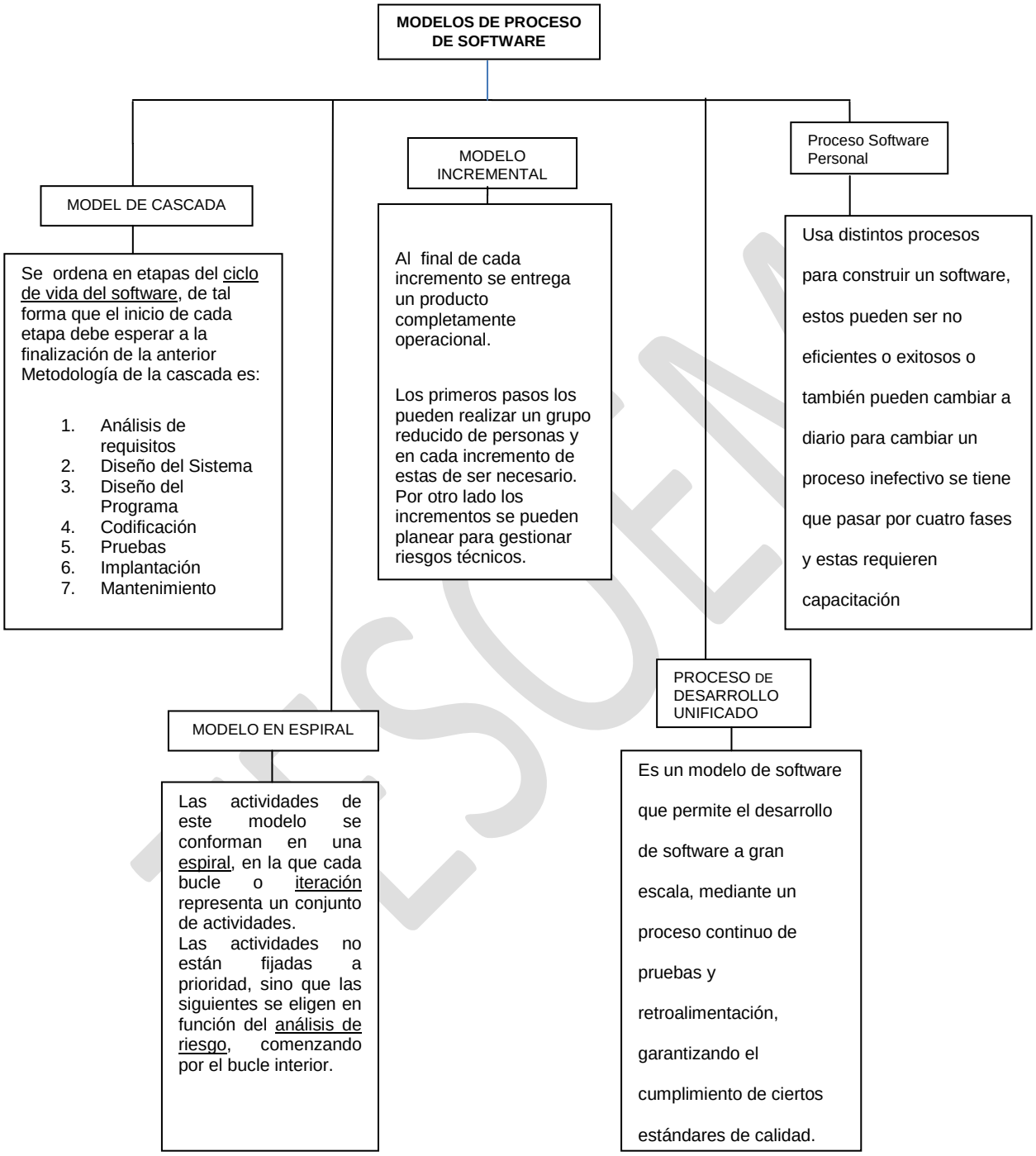
a) Enfoque estructurado b) Paradigma c) Proceso

4. Se utiliza para modelar una colección de paquetes de datos en reposo. Se denota por dos líneas paralelas

a) Almacén de datos b) Entidad externa c) diccionario de datos

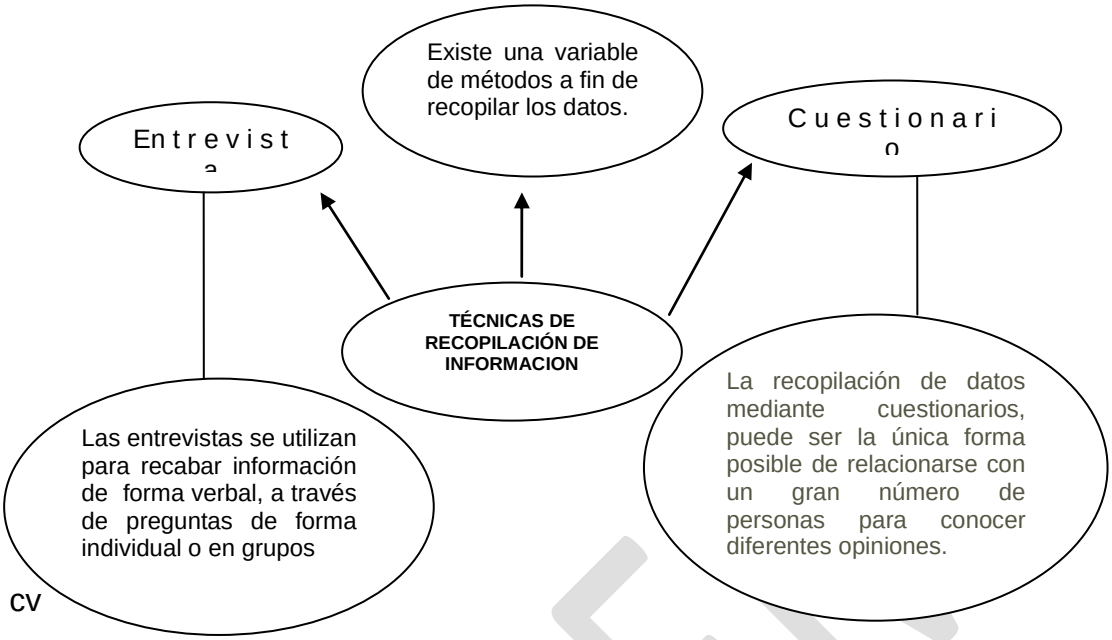
SOLUCION DE LA ACTIVIDAD CORRESPONDIENTE A LA UNIDAD 4

Realiza un cuadro sinóptico general de los modelos de proceso de software



SOLUCION DE LA ACTIVIDAD CORRESPONDIENTE A LA UNIDAD 5

Realiza un mapa mental de las técnicas de recopilación de información



HERRAMIENTAS CASE

Son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

Estas herramientas nos pueden ayudar en el ciclo de vida de desarrollo del software como en el proceso de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, recopilación automática, documentación o detección de errores.

OBJETIVOS

1. Mejorar la productividad en el desarrollo y mantenimiento del software.
2. Aumentar la calidad del software.
3. Automatizar, desarrollo del software, documentación, generación de código, pruebas de errores y gestión del proyecto.
4. Ayuda a la reutilización del software, portabilidad y estandarización de la documentación
8. Gestión global en todas las fases de desarrollo de software con una misma herramienta.
9. Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

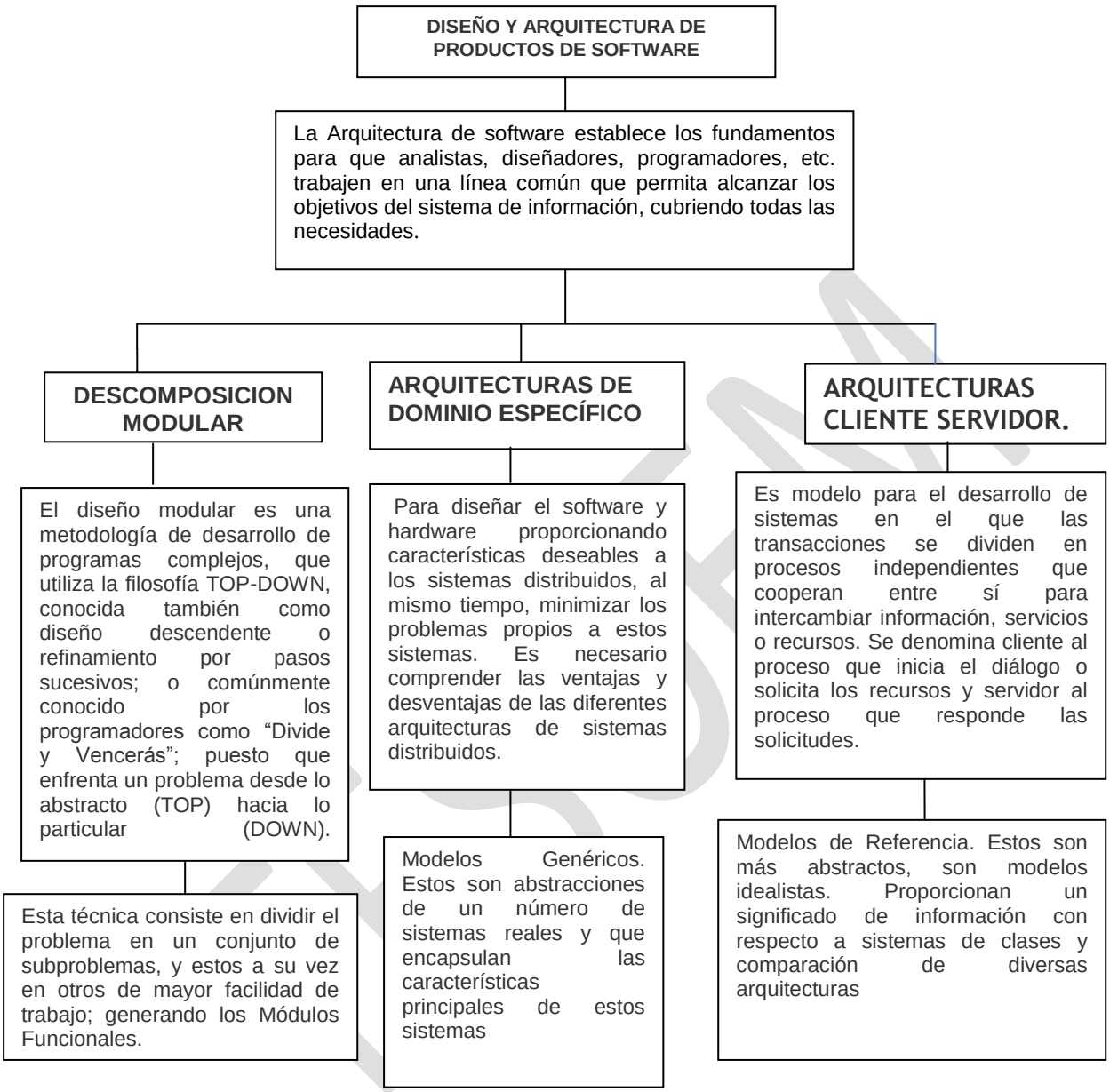
CLASIFICACIÓN

Aunque no es fácil y no existe una forma única de clasificarlas, las herramientas CASE se pueden clasificar teniendo en cuenta los siguientes parámetros

1. Las plataformas que lo soportan.
2. Las fases del ciclo de vida del desarrollo de sistemas que lo cubren.
3. La arquitectura de las aplicaciones que producen.
4. Su funcionalidad.

SOLUCION DE LA ACTIVIDAD CORRESPONDIENTE A LA UNIDAD 6

Realiza un cuadro sinóptico de diseño y arquitectura de productos de software



BIBLIOGRAFIA

1. Kendall, Kenneth E. Análisis y Diseño de Sistemas. Prentice-Hall. 2001
2. Laudon & Laudon 8/E. Management Information Systems. Prentice-Hall. 2003.
3. Pressman Roger S. Ingeniería del software. McGraw-Hill. 2001.
4. Sommerville, Ian. Ingeniería de software.Prentice-Hall. 2001.
5. Yourdan, Edward. Análisis Estructurado Moderno. Prentice-Hall. 1999.
6. Jacobson,Ivar. El Proceso unificado de desarrollo de software. Addison Wesley. 2000.
7. Fowler, Martin. UML Gota a Gota. Addison Wesley.