C.F.G.S. DESARROLLO DE APLICACIONES INFORMÁTICAS

MÓDULO: Análisis y Diseño de Aplicaciones Informáticas de Gestión

Unidad 4

Metodologías de Desarrollo del Software

ÍNDICE DE CONTENIDOS

OBJETIVOS

Una vez introducido el concepto de ciclo de vida del software, qué indicaba qué pasos deben realizarse en cualquier proyecto relacionado con el desarrollo de aplicaciones, ahora debemos introducir un nuevo concepto que es el de **metodología de desarrollo del software**", que nos indicará cómo deben realizarse esos pasos del ciclo de vida.

El ciclo define el ***qué hacer***, y la metodología el ***cómo hacerlo***. Para ello es preciso tener una visión general de las metodologías, así como de sus fundamentos dentro de la ingeniería del software para posteriormente ir detallando algunas de las metodologías más potentes y de uso más extendido en las próximas unidades.

En esta unidad es preciso que el alumno siga profundizando en su conocimiento sobre la ingeniería del software, comprendiendo la necesidad de planificar el trabajo, y de trabajar en equipo. Por ello va a ser tratada esta unidad como introductoria para el desarrollo de software de calidad.

1.- Introducción a las metodologías de desarrollo.

En la unidad anterior vimos el **ciclo de vida del software** y sus diferentes modelos de desarrollo de software asociados. El **ciclo de vida** del software indica qué es lo que hay que obtener a lo largo del proceso de desarrollo del proyecto pero no muestra cómo hacerlo. Por tanto, surge la pregunta, **¿cómo se obtienen los distintos productos parciales o finales del ciclo de vida del software?**

Para responder a esta pregunta, la **ingeniería del software** ha realizado distintas aproximaciones a lo largo del tiempo, definiendo lo que se ha denominado como **metodologías de desarrollo del software**.

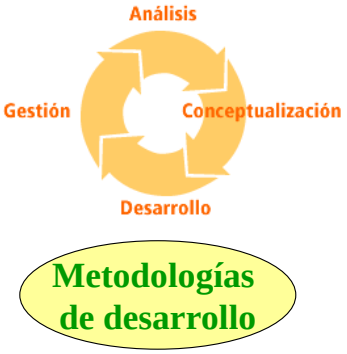
Aunque no hay una definición única para la metodología de desarrollo, podemos consensuar que **las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas, herramientas y soporte documental para el desarrollo de productos software**.

Para entender qué significa esto, podemos ver a la metodología como un **libro de recetas de cocina**, en el que se van indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado. Además, el *recetario* nos permite ir indicando qué personas deben participar en el desarrollo de las actividades y qué papel deben de tener, y detallar la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla.

El **análisis de los riesgos** constituye una pieza fundamental en el diseño y desarrollo de sistemas de software seguros. Los riesgos que afectan a un sistema de información son de distinta índole:

- **naturales** (inundaciones, incendios, etc.) o
- **lógicos** (fallos propios, ataques externos, virus, etc.).

Por tanto, el riesgo y la **incertidumbre** son inherentes al desarrollo del software, y en este contexto, **la metodología es imprescindible para solucionar los problemas de desarrollo del software ya que permite considerar y controlar, en la medida de lo posible, el riesgo y la incertidumbre.**



Como **arquitectos de software**, debemos tener un plano en el que apoyarnos, y la metodología nos va a permitir disponer de ese plano para satisfacer las necesidades de los clientes.

Normalmente, la metodología consistirá en un **conjunto de fases** descompuestas en subfases (módulos, etapas, pasos, etc.).

- Esta descomposición del proceso de desarrollo, **guía** a los desarrolladores de software con una serie de técnicas que han de elegir para cada situación del proyecto.
- La descomposición también facilita la **planificación**, gestión, control y evaluación de los proyectos.
- Las **técnicas,** indican cómo debe ser realizada cada actividad del proceso. Estas técnicas combinan el empleo de unos modelos o representaciones gráficas junto con el empleo de unos procedimientos detallados.
- Una técnica puede ser utilizada en una o más actividades de la metodología de desarrollo de software. Para garantizar la calidad del proceso es necesario que el cliente colabore en la construcción de las aplicaciones, y esto se consigue a través de las técnicas y estrategias que proporciona la metodología. Una metodología, por tanto, representa el camino para **desarrollar software de una manera sistemática**.

PARA SABER MÁS

Éste es un buen momento para que repases el concepto de ingeniería del software y el uso de las metodologías de desarrollo a través de la visita a esta web:

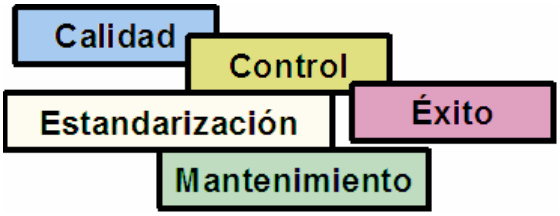
Wikipedia: ¿Qué es la Ingeniería del Software?
http://es.wikipedia.org/wiki/Ingeniería_de_software

1.1.- Fundamentos.

Vimos la necesidad de aplicar técnicas de la **ingeniería** a los problemas informáticos asociados a la programación. Por tanto, en un principio los métodos aplicados se han basado en el **método científico**, si bien adaptado a las necesidades y particularidades del desarrollo del software. Posteriormente, y a lo largo del tiempo, estos principios se han ido ajustando por la experiencia adquirida en su uso. ¿Y qué pretendemos aplicando estas metodologías? Varias cosas como vemos a continuación:

- Conseguir aplicaciones informáticas de **calidad** que den respuesta a las necesidades de los clientes y estén libres de **errores**. El aplicar de una metodología experimentada debe ayudarnos a alcanzar software de calidad con **éxito**.
- Un buen **control** de los proyectos para evitar retrasos, desarrollar más rápido y ajustarse al presupuesto. Sin una Metodología es fácil que en un proyecto se nos dispare el presupuesto y no se cumplan los plazos. Por **ejemplo**, si no hay una planificación no sabremos en qué fases debemos detenernos más o menos.

- Un método de trabajo **estándar** para todos los proyectos, lo cual permite satisfacer a los clientes, desarrolladores, directivos o auditores. Por **ejemplo**, podremos intercambiar a las personas que desarrollan el proyecto, y cualquier experto puede conocer el proyecto sin necesidad de haber trabajado en él.
- Construir una aplicación que esté bien documentada y sea fácil de **mantener**. Así, cuando en el futuro se presente la necesidad de realizar cambios o ajustes en el sistema, será más fácil hacerlos.



Sabemos que las **metodologías** descomponen el proceso. Pero, ¿en qué elementos lo descomponemos?

- El proceso queda descompuesto en **tareas**, que son actividades elementales en que se dividen los procesos.
- Para cada tarea definimos un **procedimiento**, que es la forma en que ejecutaremos la tarea.
- Con el procedimiento se establece también un **medio de comunicación** entre el cliente y los desarrolladores.
- Como resultado de llevar a cabo el procedimiento vamos a obtener un **producto**.
- Este producto será intermedio en cada una de las fases y servirá de base para la fase siguiente.
- En la última fase el producto será final y será el que se entregue al cliente.

Ahora bien, ¿cómo son esos procedimientos?

Los **procedimientos** pretenden ayudarnos a realizar la tarea. Por tanto, deben ofrecernos unas **técnicas** que nos permitan llevar a cabo cada procedimiento. En realidad, no hay una técnica exclusiva para cada procedimiento, sino que podremos utilizar la misma técnica en procedimientos distintos.

¿Y en qué consistirán las **técnicas**?

Las **técnicas** utilizan representaciones gráficas y textos formales para realizar los procedimientos. Además, la metodología también nos proporciona **herramientas** de ayuda a la realización de las técnicas. Por **ejemplo**, una herramienta puede ser un programa informático de dibujo que permita realizar los gráficos, o un programa que permita comprobar si el dibujo tiene errores. Con estas técnicas desarrollaremos cada uno de los **productos**.

Una metodología puede seguir uno o varios modelos de ciclo de vida, es decir, el ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto y la metodología indica cómo hay que obtener los distintos productos parciales y finales

Ejemplo

Veamos un ejemplo sobre algo tan cotidiano como es el hacer una tortilla de patata:

Metodología → Receta.

Tareas → Pelar y picar patatas, batir huevos, freír patatas, mezclar y hacer.

Para la tarea **batir huevos**

Procedimiento → Abrir los huevos, echarlos en un recipiente, quitar restos de cáscara, batir hasta bien mezclados.

Técnicas → Ilustraciones del libro de recetas.

Herramientas → Tenedor, batidora.

AUTOEVALUACIÓN:

1.- Si disponemos de un programa informático para realizar gráficos de organigramas, y lo utilizamos en una metodología de desarrollo. ¿A qué se está refiriendo?

a) A un procedimiento.

b) A una tarea.

c) A una herramienta.

d) A una técnica.

1.2.- De la tradición al método estructurado.

El desarrollo del software ha evolucionado desde la programación **artesanal** hasta la **ingeniería del software**. En paralelo han ido evolucionando las **metodologías de desarrollo**. Esta evolución podríamos concretarla en una serie de grandes revoluciones:

- **Estado inicial.**

En esta etapa inicial de desarrollo del software los desarrolladores estaban más centrados en la **codificación del código** que en comprender las **necesidades de los usuarios**. Esto provocaba que los sistemas desarrollados fuesen **difíciles de mantener**, y en muchas ocasiones los **clientes no quedasen satisfechos** con el resultado porque no se cumplía con sus necesidades. Como se carecía de una planificación y de un método de desarrollo se producían distintos problemas:

- Los resultados son **imprevisibles**, no se conocen los plazos de realización y además el resultado depende de las personas concretas que desarrollen el sistema ya que no hay unas pautas comunes de desarrollo
- Es **difícil controlar y revisar** el desarrollo del producto, ya que no existen mecanismos de control ni fases establecidas. Por tanto, es difícil detectar errores.
- Los **cambios** en el equipo de desarrollo influyen en todo el proceso y en el mantenimiento de los productos terminados. Si cambian las personas a mitad de proceso de desarrollo al no haber una documentación estandarizada es difícil continuar el trabajo. Y cuando el producto está finalizado y hay que actualizarlo, es muy difícil por el mismo motivo, la falta de documentación completa y estándar.

■ **Diseño estructurado.**

Para resolver los problemas del desarrollo tradicional que hemos indicado, surgieron distintas propuestas de desarrollo de software que vamos a englobar en lo que se denomina **desarrollo estructurado.** Nació a finales de los sesenta y se utilizó en las empresas de desarrollo de software a partir de la mitad de los **setenta**. El desarrollo estructurado incluye una serie de técnicas y conceptos que siguen métodos de ingeniería. Veámoslos:

- **Programación estructurada** que permite hacer programas más entendibles y fáciles de modificar.
- **Diseño estructurado** que posibilita la división de los programas en **módulos** y se introduce el concepto de **abstracción.**
- **Análisis estructurado o descendente,** que se centra en el estudio de las **especificaciones** y requisitos que debe cumplir el programa para satisfacer las demandas del cliente. La **programación estructurada** se centraba en cómo debía ser el programa informático, el **diseño estructurado** en cómo debíamos diseñar el programa para que la programación fuese más fácil de hacer, sencillo de mantener y de mayor calidad, pero ¿y si no sabemos exactamente qué debe hacer el programa?, ¿y si no hemos entendido lo que quiere el cliente? Para centrarnos en estos aspectos surge el análisis estructurado. Antes del análisis estructurado, las especificaciones del programa se hacían en un documento no estándar y sin estructura alguna, por lo que era difícil de entender. El análisis estructurado intenta solucionar esto proporcionando una serie de técnicas estándar que permiten detallar claramente y sin redundancia los requisitos del programa.
- **Técnicas estructuradas** de desarrollo de software**.** Estas técnicas son gráficas (diagramas) y textuales (documentos), modulares para poder hacer unas partes independientes de otras, y poco redundantes para evitar que los cambios obliguen a modificaciones grandes del análisis.

El **diseño estructurado** ha ido evolucionando a lo largo del tiempo para adaptarse a las nuevas necesidades como la programación de **sistemas en tiempo real**.

En la siguiente tabla podemos ver cómo han surgido metodologías de desarrollo del software que siguen los principios del desarrollo estructurado:

AÑOS	EVOLUCIÓN
1968	Aparece la **programación estructurada** de DIJKSTRA
1974	Surgen las **técnicas de programación estructurada** de WARNIER y JACKSON
1975	Aparece el **diseño estructurado** (MYERS y YOURDON)
1977	Aparece el **análisis estructurado** (GANE y SARSON)
1978	Nace la metodología de desarrollo de software francesa **MERISE**
1981	Nace la metodología de desarrollo de software británica **SSADM**
1985	Análisis y Diseño estructurado para **sistemas de tiempo real** (WARD y MELLOR)
1989	Nace la metodología de desarrollo de software española **METRICA**
2001	Nueva versión de la metodología **METRICA Versión 3**

PARA SABER MÁS

Para profundizar en los principios del desarrollo estructurado puedes consultar esta web donde encontrarás una información muy completa:

Wikipedia: Programación estructurada
http://es.wikipedia.org/wiki/Programación_estructurada

Edsger Wybe Dijkstra (1930-2002) fue un importante informático de origen holandés que desarrolló los principios de la programación estructurada. Para conocerlo un poco podemos leer sus escritos. Aquí tienes una interesante página web donde encontrar toda la información que te interese:

Escritos de Edsger Wybe Dijkstra
http://www.smaldone.com.ar/documentos/ewd.shtml

1.3.- Del desarrollo estructurado a los objetos.

En el apartado anterior hemos visto una importante **evolución** desde el estado inicial que podríamos describir como **artesanal** hasta una metodología estructurada que sigue unos principios científicos. Pero, ¿cuáles son las **tendencias actuales** en las metodologías de desarrollo de la ingeniería del software?

- **Desarrollo orientado al objeto.**

Durante los años **ochenta** surgieron los lenguajes de **programación orientada a objetos**, que a diferencia de la programación estructurada, no separan datos y procesos sino que los tratan de forma conjunta en los denominados objetos que representan conceptos. La esencia del desarrollo orientado a objetos es la **identificación y organización de conceptos en el diseño de la aplicación,** y no tanto de su representación final en un lenguaje de programación. Una de las ventajas de utilizar objetos es que se crean **librerías** (bibliotecas de clases de objetos) que permiten reutilizar los elementos según se necesiten, de esta forma no es necesario partir siempre desde cero. Otra ventaja es que al ser muy flexibles en la concepción de los conceptos representados por objetos, permite tratar mejor la incertidumbre y adaptarse a las necesidades cambiantes de sistemas complejos.

Ha habido **múltiples metodologías** orientadas a objetos a lo largo del tiempo. Actualmente, la que tiene más relevancia es la metodología del **proceso unificado** que trata de integrar los diferentes métodos existentes. Esta metodología utiliza como método el **Lenguaje de Modelado Unificado (UML)**. La empresa Rational Software Corporation (perteneciente a IBM) distribuyó esta metodología como **RUP (Rational Unified Process)** en el año 2000.

- **Desarrollo ágil.**

A veces hay situaciones en las que no es necesario un desarrollo completo de una metodología. Algunos desarrolladores piensan que en estas situaciones deben utilizarse metodologías más **ligeras** que permitan realizar desarrollos rápidos. En ese sentido en los últimos años está teniendo bastante éxito las metodologías denominadas **ágiles** que intentan buscar un equilibrio entre el uso de las metodologías y la programación directa de aplicaciones.

Un **ejemplo** de este tipo de metodologías es la **Programación Extrema (XP)** desarrollada en 1996, aunque no es la única. La metodología RUP puede adaptarse al desarrollo ágil y seguir sus principios y propuestas. Más adelante volveremos a hablar de estas metodologías.

PARA SABER MÁS

En este enlace encontrareis información de la programación extrema con un poco de humor.

Programación Extrema (XP)

http://www.marquetti-asociados.com.ar/paradigma.php

AUTOEVALUACIÓN:

2.- Indica el orden correcto en el que han aparecido las siguientes metodologías de desarrollo del software:

a) Métrica, RUP, XP, SSADM

b) Métrica, Merise, metodología orientada a objetos, metodología estructurada

c) Métrica, metodologías orientadas a objetos, metodologías ágiles

d) Desarrollo tradicional, Métrica, XP, metodologías orientadas a objetos

1.4.- ¿Para qué una metodología de desarrollo?

Hemos explicado los esfuerzos que se han seguido para desarrollar **metodologías de desarrollo** a lo largo del tiempo, pero:

- ¿De verdad era necesario todo este esfuerzo?
- ¿Tan **importante** es realizar un análisis y un diseño de las aplicaciones antes de comenzar a programar?
- ¿No sería más fácil y rápido empezar directamente a introducir código en el ordenador con un buen lenguaje de programación?

A veces ocurre que realizamos el diseño de nuestro software únicamente con los **requisitos** que el cliente nos indicó al principio, y cuando estamos en la etapa final del desarrollo el cliente nos solicita un **cambio**. Este cambio puede ser difícil de realizar, pues si lo hacemos, altera muchas cosas que no habíamos previsto, y esto puede ocasionar un **retraso** en el proyecto con el consiguiente malestar del cliente. Esto podría haberse resuelto si se hubiese aplicado una **estrategia** correcta, con una metodología de desarrollo que implique más al cliente en el desarrollo y le vaya mostrando los sucesivos avances, por ejemplo, mediante el empleo de **prototipos**.

Dos conocidos ejemplos ilustran la importancia de utilizar una buena metodología de desarrollo que asegure la **calidad del software**.

- En 1995, un profesor de matemáticas de la universidad de Lynchburg descubrió un error en los **microprocesadores Pentium** de la compañía Intel. Aunque Intel corrigió el error rápidamente, ya se habían distribuido casi dos millones de procesadores defectuosos. Intel sustituyó los microprocesadores pero ya se había producido la mala imagen de la empresa y tuvo que soportar importantes pérdidas. El **fallo** ocurrió por un error de diseño en el algoritmo de división de punto flotante (operaciones matemáticas con decimales).

- Apenas un año después, la **nave espacial** europea Ariane 5 explotó cuarenta segundos después de su lanzamiento perdiéndose casi 500 millones de euros. La comisión de expertos que investigó el desastre descubrió, que el **fallo** se produjo en un pequeño programa reutilizado del sistema del cohete Ariane 4. Lo irónico del caso, es que este código no tenía ninguna utilidad en el nuevo sistema.

Si se hubiesen establecido los mecanismos adecuados de **supervisión** que proponen las metodologías de desarrollo, probablemente se habrían evitado estos errores.

PARA SABER MÁS

Hemos hablado de la conveniencia de proponer prototipos al cliente. Si quieres profundizar más en este tema puedes ver los siguientes enlaces:

http://www.sidar.org/recur/desdi/traduc/es/visitable/tecnicas/Prototyping.htm

Podemos saber más sobre los prototipos realizados en papel en este enlace:

Universidad Politécnica de Madrid: Prototipado en papel
http://is.ls.fi.upm.es/xavier/usabilityframework/actividades/ed_prototipado.php

1.5.- Hacia el ideal.

Una vez que nos hemos convencido de la importancia de utilizar una **metodología de desarrollo**, nos surge la siguiente pregunta, ¿qué metodología elegimos?

Como veremos en lo que queda de unidad la respuesta a esta pregunta no es única, en realidad deberemos determinar el **alcance** de nuestra aplicación y luego ver cuál es la **metodología** que mejor responde a nuestras necesidades. Además, la metodología influye en el entorno de desarrollo e incluso en la **estructura** de la propia empresa informática de desarrollo de software. Así, cada empresa debe ver qué metodología se ajusta mejor a su propia estructura.

También cabe preguntarse, ¿cómo debe ser una metodología de desarrollo **ideal**?, ¿qué características debe reunir?

Las vamos a enumerar a continuación:

- Descripción **clara** de las fases, tareas, productos, técnicas y herramientas.
- Debe cubrir **todo el ciclo** de vida del software, desde el inicio hasta el mantenimiento.
- Evaluaciones y **controles** periódicos de los productos obtenidos.
- **Planificación** del proyecto para poder realizar su control y facilitar la labor a los directivos.
- Mecanismos de **comunicación** fáciles entre los desarrolladores y el cliente para satisfacer sus necesidades.
- **Flexibilidad** para adaptarse a cualquier proyecto de manera que una empresa de desarrollo de software pueda utilizar siempre la misma metodología sin necesidad de cambiar.
- **Fácil** de aprender a manejar por los desarrolladores.
- La metodología debe estar soportada por **herramientas CASE** que automaticen y faciliten los procesos.
- Mecanismo de control y mejora de la **calidad** del desarrollo.
- Soporte al **mantenimiento** de las aplicaciones, no sólo hay que desarrollar programas sino mantenerlos después.
- Posibilidad de **reutilización** de los componentes software desarrollados en otros proyectos para que no haya que partir siempre de cero.

Estas son las características **ideales**, pero aunque hay miles de metodologías es difícil encontrar una metodología que las cumpla todas, por lo que debemos conocer la posibilidad y oportunidad de distintas metodologías.

AUTOEVALUACIÓN:

3.- A la hora de elegir una metodología de desarrollo debo:

a) Utilizar en cada proyecto una distinta según las necesidades.

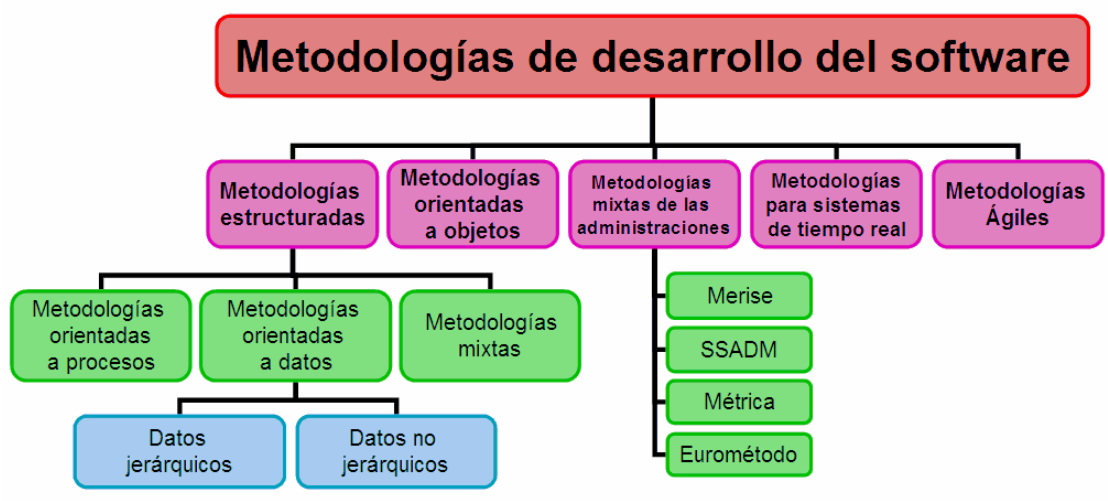
b) Seleccionar una y aplicarla a todos los proyectos siempre.

c) Escoger la metodología ideal según se ha descrito antes.

d) Estudiar la estructura y experiencia de mi empresa, las necesidades del proyecto y escoger la que más se ajuste a mis necesidades.

2.- Clasificación de las metodologías.

Cuando estudiamos nuevos conceptos o ideas, siempre tendemos a establecer **clasificaciones** para entender mejor su naturaleza. Clasificamos grupos de música, modelos de coches, tipos de ordenadores, etc. En esta línea nosotros vamos a intentar clasificar las **metodologías** de desarrollo según el enfoque que utilizan y su uso. Así, como podemos ver en la imagen siguiente tenemos metodologías:



2.1.- Introducción a las metodologías estructuradas.

En definitiva la metodología de desarrollo lo que pretende es **resolver un problema** o necesidad, y para ello parte de la petición del cliente y con sucesivas fases obtiene una solución informática.

Ya vimos que en las metodologías **estructuradas** se realiza una aproximación a la resolución del problema **descendente**. Es decir, se pasa de una visión más general del problema con un nivel de **abstracción** alto (cercano a las personas), a un nivel de abstracción más bajo (cercano a la máquina). Para ello, estas metodologías proponen la creación de **modelos** que representen los procesos o acciones a realizar, los flujos de información y las estructuras de datos necesarias para almacenar la información.

El modelo general que representa a un sistema informático consta de **Entrada-Proceso-Salida**. Los datos se introducen en el sistema, el cual los procesa para obtener unos resultados a la salida. Las metodologías estructuradas utilizan este esquema para realizar su enfoque del desarrollo del software.

Según se enfoque el desarrollo desde un punto de vista u otro tenemos:

- Las metodologías **orientadas a procesos**.
- Las **orientadas a datos**.
- También existen metodologías **mixtas** que toman en consideración tanto los procesos como los datos, añadiendo el factor tiempo según un modelo de eventos.

Aunque fundamentalmente las metodologías **estructuradas** respondieron inicialmente a un ciclo de vida de **cascada**, algunas metodologías estructuradas han evolucionado con el tiempo hacia otros modelos de ciclos de vida incluyendo la posibilidad de realizar prototipos.

AUTOEVALUACIÓN:

4.- Queremos desarrollar un nuevo sistema software que responda a las necesidades de una tienda de alquiler de películas de vídeo y juegos de ordenador. Al aplicar una metodología analizo primero la información acerca de los clientes, los tipos de películas en DVD, los tipos de juegos según plataforma, etc. Estoy utilizando una metodología estructurada:

a) Orientada a procesos.
b) Orientada a datos.
c) Mixta.
d) Ninguna respuesta es correcta.

2.2.- Metodologías estructuradas orientadas a procesos.

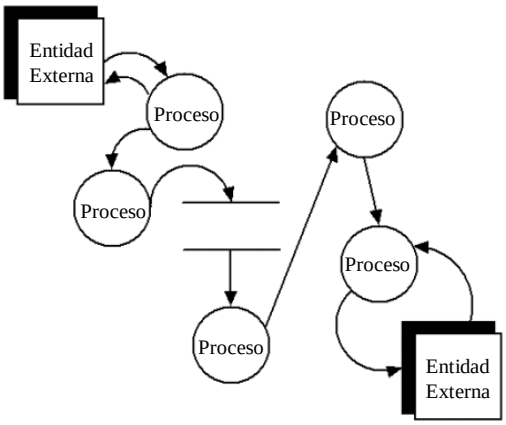
Estas metodologías orientadas a **procesos** estudian cómo son transformados los **flujos de datos** por los procesos**,** desde la entrada hasta la salida. Por tanto, hacen más hincapié en los procesos que en los datos.

¿Cómo vamos a analizar estos flujos de información?

Siempre recurrimos a **representaciones gráficas** porque son las más útiles y fáciles de interpretar. Estas representaciones gráficas son acompañadas por documentos de texto desarrollando lo que se denomina **especificación estructurada**.

La especificación estructurada incluye estas **técnicas**:

- **Diagramas de Flujo de Datos (DFD)**: son una representación gráfica que representa los procesos que debe llevar a cabo el sistema informático y los datos que hay a la entrada y salida de cada proceso. Utilizando el concepto de **abstracción**, los procesos se van describiendo al principio como procesos amplios y complejos, para ir posteriormente descomponiéndolos en procesos más **simples**. Este proceso se irá repitiendo para realizar un refinamiento progresivo del sistema. En la imagen podemos ver una herramienta informática de ayuda para desarrollar DFD. En concreto se trata de la modelización de un almacén, en la que aparece un proceso de gestión de compras y unas entidades de proveedores y almacén.



- **Diccionario de Datos (DD)**: son las descripciones de todos los datos del sistema y de los elementos que aparecen en el DFD.
- **Especificaciones de procesos**: es la descripción detallada de los procesos, es decir, explican cómo se obtienen las salidas a partir de las entradas.

Para la realización de estas técnicas dispondremos de una serie de **herramientas** informáticas de ayuda que permitan facilitar la realización de los DFD y automatizar parte del proceso. En la imagen anterior tenemos un DFD con procesos, entidades y un almacén de datos.

■ Fases de la especificación estructurada.

Pero, ¿cuáles son las **fases** de esta metodología?, ¿qué **tareas y procedimientos** asociados deben realizarse? Vamos a verlo a continuación:

1. **Fase de planificación inicial.**
2. **Fase de análisis:**
   - **Descripción del modelo físico actual**: se trata de describir el sistema actual del cliente que podrá estar informatizado o no. Para ello, se realizan una serie de DFD que describen los sistemas actuales.
   - **Elaboración del modelo lógico actual**: a partir del modelo anterior, se realiza un nuevo modelo que no dependa de elementos físicos, por ejemplo la organización actual de la empresa o las ubicaciones en una u otra planta del edificio. Para ello, se realizan también DFD.

3. **Fase de diseño**:
   - **Desarrollo de un nuevo modelo lógico**: una vez estudiado el sistema actual, hay que introducir las nuevas necesidades del cliente y elaborar un modelo lógico alternativo que resuelva los problemas del cliente y satisfaga sus requisitos. Para ello se realiza una especificación estructurada completa, es decir, se incluyen DFD, DD y especificación de procesos.
   - **Elaboración de un conjunto de modelos físicos**: a partir del modelo lógico se incluyen los detalles físicos de la empresa, como por ejemplo la distribución del edificio de la empresa, y se realizan distintas propuestas alternativas.
   - **Estimación de los costes y tiempos** de cada opción del modelo físico.
   - **Selección de un modelo** en función de los intereses del cliente.
   - Recopilación del modelo elegido en un **documento de especificación** que permita su posterior implementación en la empresa.
4. **Fase de codificación**: mediante la programación estructurada.
5. **Resto de fases**: incluye las fases de prueba, control y mantenimiento.

Hay diferentes metodologías que siguen este modelo aunque con pequeños cambios. Entre otras, las más importantes son las desarrolladas por DeMarco (1979), Gane y Sarson (1979) y Yourdon/Constantine (1989).

PARA SABER MÁS

Puedes saber más sobre los DFD si visitas esta página:

Monografías: Manual sobre los DFD

http://www.monografias.com/trabajos12/diflu/diflu.shtml

2.3.- Metodologías estructuradas orientadas a datos.

Dentro del modelo básico **Entrada-Proceso-Salida**, estas metodologías se centran en el estudio de los datos a la **entrada y de los resultados** a la salida. Para ello definen cómo son esos datos, los agrupan en estructuras de datos, y posteriormente describen los procesos.

La razón para hacer esto está en la creencia de que **los datos son más estables** que los procesos en las empresas y sus sistemas, por lo que si identifico con éxito la estructura y naturaleza de estos datos podré desarrollar posteriormente, a partir de esa estructura de datos, un modelo de procesos más estable que si lo hago al contrario. En realidad, esto no siempre es cierto, sino que depende de cada **organización** o empresa.

¿Y todas las **estructuras** de datos son iguales?

No exactamente, por eso, en función de cómo sean esas estructuras de datos, podemos clasificar a estas metodologías en orientadas a datos jerárquicos y orientadas a datos no jerárquicos.

- **Metodologías estructuradas orientadas a datos jerárquicos.**

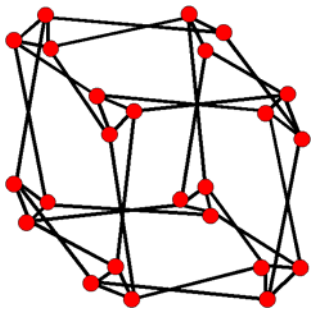
En este caso la estructura de los datos es **jerárquica** y la estructura de control del programa que se deriva del estudio de los datos también es jerárquica. Esto significa que hay una **relación de jerarquía entre los datos** y pueden ser representados mediante árboles. Por ejemplo, como puede verse en la figura, una estructura de datos jerárquica sería la representación del árbol genealógico de una familia.

En esta metodología, el procedimiento a seguir consiste en **definir** primero las estructuras de los datos de entrada y salida (análisis de los datos), **mezclarlas** todas en una estructura jerárquica de programa y después **ordenar** detalladamente la **lógica procedimental** del programa (análisis de los procesos) para que se ajuste a esta estructura jerárquica. Por tanto, en esta metodología se crea en primer lugar el **diccionario** de datos y luego se estudian las **actualizaciones** y modificaciones que deben realizarse a esos datos para obtener las salidas definiendo de esta manera los procesos.

Como ejemplos de este tipo de metodologías tenemos las propuestas por Jackson (1975), Cameron (1989), Warnier (1974) y Warnier/Orr (1981).

- **Metodologías estructuradas orientadas a datos no jerárquicos.**

En esta metodología se analizan los datos para crear un modelo que integre las entidades y las relaciones entre ellas. Estas entidades representan los elementos de la organización, por ejemplo una entidad podría ser un proveedor de la empresa. En este caso, los datos no tienen por qué responder a una estructura jerárquica sino que pueden relacionarse de cualquier otra forma, como por ejemplo en la figura, donde no hay una estructura jerárquica.



Un ejemplo de este tipo de metodología es "***Information Engineering***" (IE) desarrollada por Martin y Finkelstein en 1981.

AUTOEVALUACIÓN:

5.- La metodología de Jackson es una metodología:

a) Orientada a procesos.

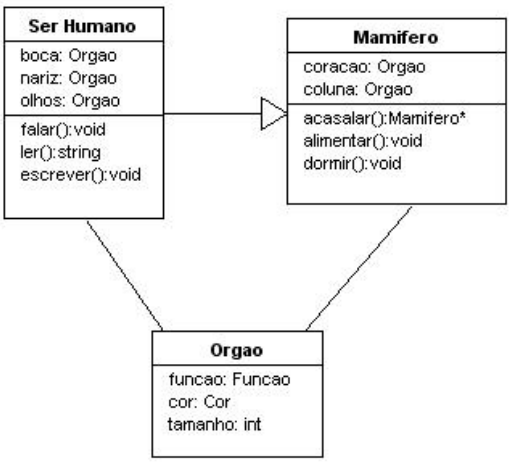
b) Orientada a datos jerárquicos.

c) Orientada a datos no jerárquicos.

d) Orientada a objetos.

2.4.- Metodologías orientadas a objetos: RUP

Vimos cómo **evolucionaron** las metodologías para responder al paradigma de la programación orientada a objetos. Ha habido varias metodologías, pero la que se ha consolidado actualmente y tiene más presencia en la ingeniería del software es el **Proceso Unificado (RUP)** que utiliza las técnicas proporcionadas por el **Lenguaje de Modelado Unificado** (UML). RUP ha unificado distintas metodologías y técnicas en una sola metodología. Así, RUP constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.



Sus principales **características** son:

- Forma **disciplinada** de asignar y organizar tareas y responsabilidades (quién, cómo, qué, cuándo).
- Desarrollo **iterativo e incremental**.
- Proporciona mecanismos de **gestión** del proyecto administrando horarios y recursos.
- Facilita la **gestión de requisitos** a través de un proceso completo para su recogida y documentación guiado por **casos de uso**.
- Centrada en la arquitectura para buscar su **robustez** con un producto que tenga el comportamiento deseado.
- Basada en **componentes reutilizables**.
- Modelado visual del software utilizando el estándar **UML** (asegura el uso de herramientas **CASE**).
- Resulta fácil dividir el sistema en varios **subsistemas** independientes.

- Facilita el **control de cambios** a lo largo de todo el proceso, guardando todas las versiones del proyecto.
- Permite la verificación de la **calidad** del software a través de diferentes mecanismos de control en los que participa todo el equipo de desarrollo.
- **Adaptable** a cualquier tipo de proyecto y organización independientemente del tamaño o complejidad.

Hemos indicado que RUP asigna y organiza disciplinadamente las tareas, para ello define estos elementos:

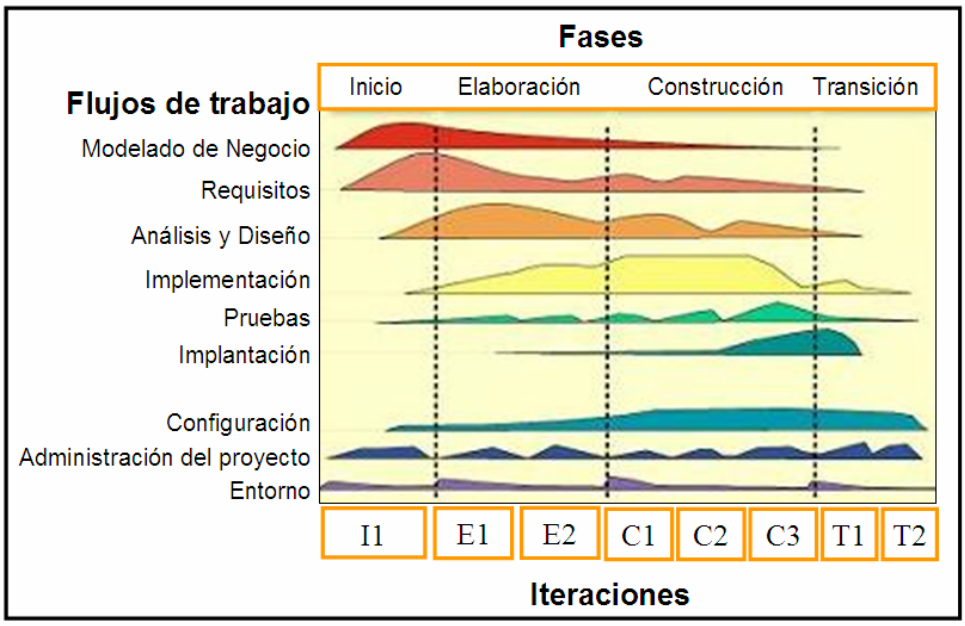
- **Perfiles o roles** de las personas y entidades implicadas (quién).
- **Actividades** que guían el proceso (cómo).
- **Artefactos o productos intermedios** a obtener (qué).
- **Flujos de trabajo:** indican la secuencia de actividades y los procedimientos a seguir (cuándo).
  - De **desarrollo** e ingeniería, incluyen:
    - **Modelado de Negocio**: describe la estructura y dinámica del negocio.
    - **Requisitos**: descripción de las necesidades del negocio mediante casos de uso.
    - **Análisis y Diseño**: describe la arquitectura de software mediante distintos modelos.
    - **Implementación**: desarrollo del software según el diseño y cumpliendo los requisitos.
    - **Pruebas**: para asegurar que el comportamiento es correcto y satisface las necesidades.
    - **Implantación**: puesta en marcha y configuración del sistema.
  - De ayuda y **apoyo**, incluyen:
    - **Configuración** y gestión de cambios: controla los productos intermedios.
    - **Administración** del proyecto: establece estrategias de trabajo, horarios y recursos.
    - **Entorno**: para controlar la infraestructura ligada al desarrollo del proyecto.

En las características, hemos dicho que la metodología RUP sigue un ciclo de vida **iterativo** e **incremental**. Para ello **RUP** divide el proceso de desarrollo en **ciclos**, teniendo un producto al final de cada ciclo. Las **fases** o etapas de desarrollo son:

- **Fase de Inicio**: El objetivo es estudiar la viabilidad del proyecto, para ello se hace un **plan temporal**, se identifican los riesgos, se detectan los casos de uso, se hace una estimación de costos, etc.
- **Fase de Elaboración**: El objetivo es determinar la arquitectura óptima, para lo que se hace un **plan de proyecto**, se completan los casos de uso y se eliminan los riesgos.

- **Fase de Construcción**: El objetivo es la elaboración de un producto totalmente operativo y eficiente y el manual de usuario**.**
- **Fase de Transición**: El objetivo es implantar el producto y ponerlo a disposición de los usuarios. Como consecuencia de esto suelen surgir nuevos requisitos a ser analizados, lo que presentamos en esta fase es una versión inicial o beta del sistema.

Cada una de estas etapas es desarrollada mediante un **ciclo de iteraciones**, que reproducen el ciclo de vida a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes. Cada iteración obtiene un producto intermedio operativo revisable por el cliente y que será un punto de partida para la siguiente. Con esto se consigue una **retroalimentación** con el cliente en cada iteración. Hay un alto grado de iteración y solapamiento, lo que lleva a una forma de trabajo muy dinámica donde es posible realizar distintas etapas al mismo tiempo. Se eliminan fronteras entre fases debido a la naturaleza iterativa del desarrollo orientado al objeto.



En la imagen podemos ver una descripción de las fases, las iteraciones (el sistema es flexible por lo que podría haber más o menos de las que aparecen, una para la fase de inicio I1, dos para la fase de elaboración E1 E2…), y los flujos de trabajo (las figuras de colores indican cuando se realizan los flujos de trabajo y con qué volumen de trabajo).

Una **característica** importante de una metodología orientada a objetos como RUP es que establecen mecanismos para tratar el **riesgo y la incertidumbre**, por lo que son necesarios cuando nos enfrentamos a desarrollos de software donde la incertidumbre es una componente significativa.

PARA SABER MÁS

Puedes profundizar en el estudio de la programación orientada a objetos haciendo una visita a esta página web de la enciclopedia libre wikipedia:

Wikipedia: artículo sobre Programación orientada a objetos
http://es.wikipedia.org/wiki/Programación_orientada_a_objetos

AUTOEVALUACIÓN:

6.- La metodología RUP se corresponde con el modelo de ciclo de vida de:

a) Cascada.
b) No corresponde a ningún modelo concreto sino que tiene características de varios modelos como iterativo, incremental y con reutilización.
c) Prototipado.
d) Espiral.

2.5.- Modelado de objetos con UML.

Hemos dicho que RUP utiliza UML como técnica de **modelado** de los objetos, pero ¿en qué consiste UML?

El **Lenguaje Unificado de Modelado** (UML) consiste en un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la **semántica** esencial de estos diagramas y los **símbolos** en ellos utilizados. UML es de libre uso y se trata de una estandarización o consolidación de muchas notaciones y modelos usados anteriormente. Por tanto, UML es un **lenguaje gráfico** para visualizar, especificar, construir y documentar un sistema de software. Como técnica de trabajo de la metodología, UML proporciona varios tipos de **diagramas** que muestran diferentes aspectos de las entidades o elementos representados.

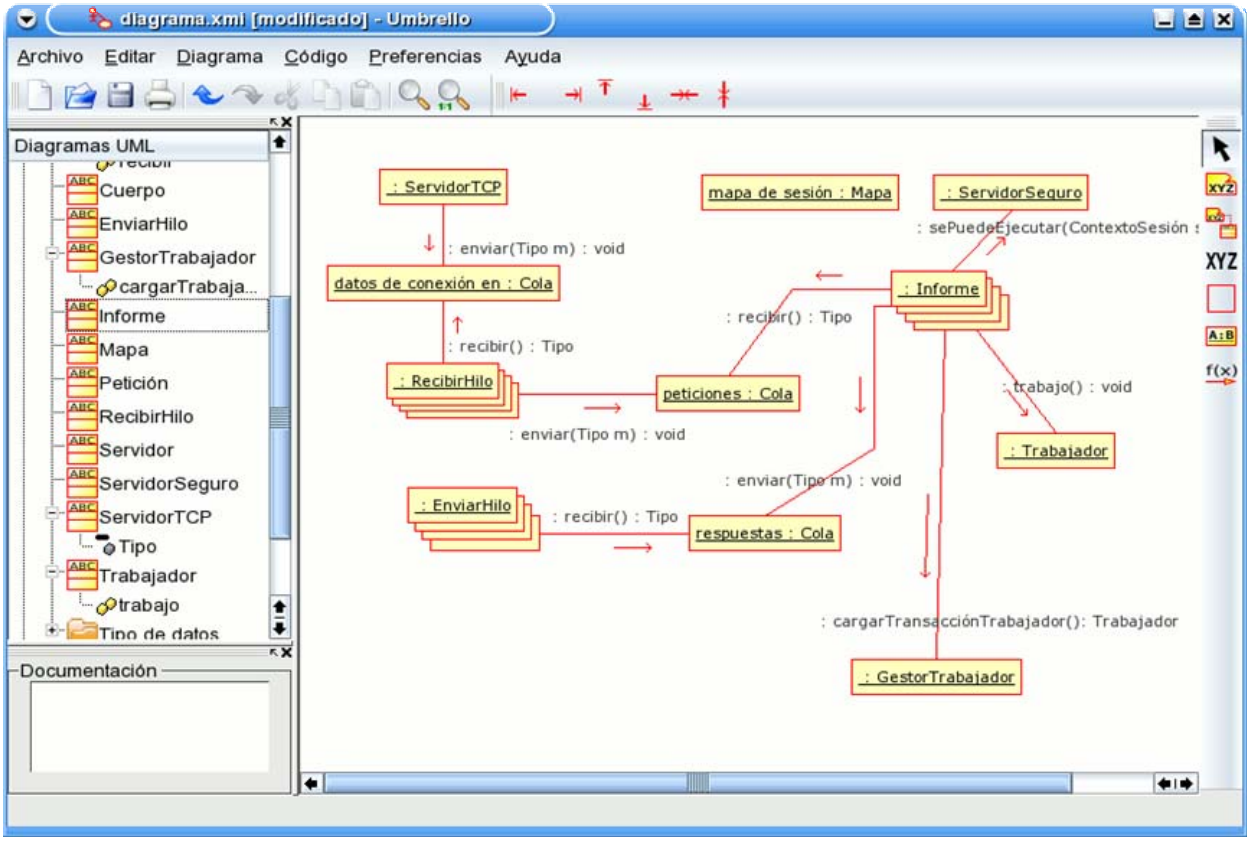
Podemos clasificar estos **diagramas** en:

- **Diagramas de estructura**: describen los elementos del sistema. Incluye:
  - Diagrama de clases.
  - Diagrama de componentes.
  - Diagrama de objetos.

- **Diagramas de comportamiento**: indican lo que deben hacer los elementos del sistema, para lo que incluye:
  - Diagrama de actividades.
  - Diagrama de casos de uso.
  - Diagrama de estados.

- **Diagramas de Interacción**: indican el flujo de control y de datos entre los elementos del sistema, para lo que utiliza:
  - Diagrama de secuencia,
  - Diagrama de comunicaciones,
  - Diagrama de tiempos, etc.

Una de las grandes ventajas de UML es que existen múltiples herramientas CASE que lo soportan, lo cual facilita y automatiza el trabajo. En la imagen podemos ver un diagrama de comunicaciones que representa la gestión de una cola de trabajos realizado con una herramienta CASE.



AUTOEVALUACIÓN:

7.- Podemos decir que UML:

a) Es un lenguaje gráfico que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos.

b) Pertenece a una empresa y hay que pagar por su uso.

c) Es una técnica desarrollada por la metodología RUP.

d) Todas las respuestas son correctas.

PARA SABER MÁS

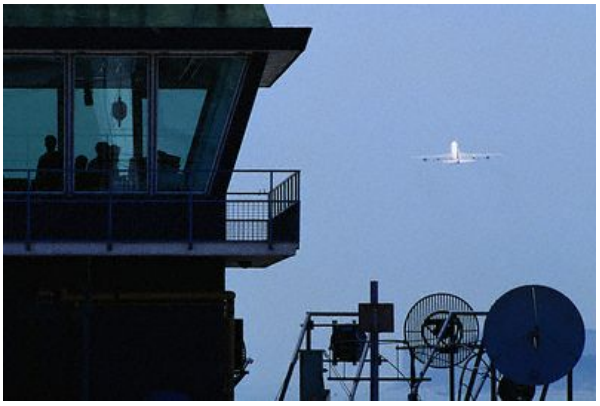
Puedes profundizar en UML visitando esta web que incluye un completo tutorial para principiantes:

Manuales clikear: UML

http://www.clikear.com/manuales/uml/

2.6.- Metodologías para sistemas de tiempo real.

Cuando viajamos en un **avión** sabemos que el piloto debe solicitar **permiso** para despegar o para conocer la ruta que debe seguir, o la altura de vuelo. Todo esto se lo indica un controlador de vuelo que se ayuda de un sistema informático. ¿Qué pasaría si hay muchos aviones en vuelo que saturan el sistema informático y el controlador aéreo tiene que esperar para dar una orden a dos aviones que intentan aterrizar al mismo tiempo en el mismo aeropuerto? Desde luego lo mejor es no estar en esos aviones.



Un **sistema informático** que debe captar señales (en este caso del radar y de los sistemas de comunicación de los aviones) sin perder ninguna y que debe contestar a las mismas antes de un determinado momento, es **un sistema de tiempo real**. En estos sistemas la **velocidad de respuesta** es fundamental porque el usuario, en este caso el controlador aéreo y el piloto, no pueden esperar.

No debemos confundir los **sistemas de tiempo real** con los **sistemas interactivos**, en los que hay que responder lo más rápido posible al usuario pero no hay riesgo de perder señales o datos a la entrada, ni un límite de tiempo en la respuesta.

- Por **ejemplo**, un sistema interactivo que controla las operaciones de venta en un supermercado interesa que responda rápido, pero si tarda, el único problema es que hay que esperar. No perderemos datos porque no habrá otra venta hasta que termine la anterior, y no hay un límite de tiempo porque el cliente puede esperar. Esto es un inconveniente pero no un problema crítico como un accidente aéreo.
- Otro **ejemplo** sería un sistema que controle las máquinas de una fábrica donde las tareas deben sincronizarse perfectamente.

Para desarrollar estos **sistemas** será necesario disponer de metodologías de desarrollo del software que permitan especificar este tipo de situaciones de tiempo real y las posibles soluciones. ¿Qué incluirán estas metodologías? Deberán establecer mecanismos para modelar sistemas que:

- Controlen la comunicación y **sincronización** entre tareas.
- Gestionen los **procesos concurrentes** que se ejecutan en paralelo.
- Respondan ante **eventos externos** como puede ser una nueva señal.
- Reciban datos y **señales continuas** que no pueden perderse.
- **Interrumpan** los procesos para pasar a otra acción: por ejemplo al producirse una señal de alarma.

Para enfrentarse a estos sistemas las metodologías de desarrollo tradicionales evolucionaron para adaptarse. Así por ejemplo, tenemos la metodología de Hatley y Pirbhay (1987) que es una ampliación de la metodología estructurada de Yourdon/Constantine que ya vimos. También hay una evolución de las metodologías orientadas a objetos para desarrollos de sistemas de tiempo real, como la extensión de UML realizada por Douglass (1998), o la metodología RUP que contempla el desarrollo de software para sistemas de tiempo real.

AUTOEVALUACIÓN:

8.- El sistema informático que controla los procesos de una central nuclear:

a) Es un sistema de tiempo real.
b) Es un sistema interactivo.
c) Debe desarrollarse con una metodología orientada a objetos.
d) Debe desarrollarse con la metodología RUP.

2.7.- Metodologías de desarrollo ágil.

Hemos estado describiendo las metodologías de desarrollo de software que han ido evolucionando a lo largo del tiempo. Cabría preguntarse:

- ¿Qué **nuevas metodologías** se están desarrollando?,
- ¿Siguen todas las metodologías los modelos tradicionales o están apareciendo otras posibilidades?

Algunos desarrolladores creen que las metodologías **tradicionales** generan demasiada burocracia y exigen demasiado esfuerzo, sobre todo para empresas de desarrollo pequeñas y en desarrollos de proyectos pequeños. Por otro lado, el mercado competitivo actual de los productos tecnológicos, no sólo exige calidad, coste e innovación, sino también rapidez y flexibilidad. En este contexto, **el mercado necesita ciclos de desarrollo más cortos.**

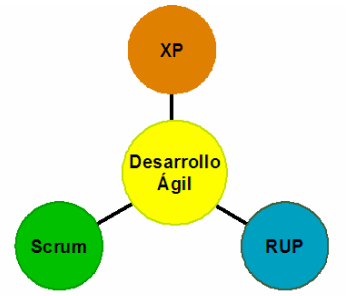
Para solucionar estos problemas se han propuesto una serie de **principios** y valores que aligeren la carga de las metodologías tradicionales. Con el desarrollo de estas ideas y conceptos han aparecido un nuevo tipo de metodologías que se denominan de **desarrollo ágil**. Sin embargo, **algunos expertos consideran que el desarrollo ágil no constituye realmente una nueva metodología**, sino un conjunto de recomendaciones y principios aplicables a las metodologías tradicionales para hacerlas más flexibles, rápidas y adaptables.

Este enfoque está mostrando su **efectividad** en proyectos pequeños, con requisitos inestables o cambiantes (incertidumbre) y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad del producto final.

Las metodologías ágiles se basan en el trabajo en equipo y pretenden:

- **Centrarse en el desarrollo** y en **satisfacer al cliente**, es decir, producir un sistema con las funcionalidades correctas. Esto significa que el sistema final tiene que incluir sólo el mínimo número de características necesarias para satisfacer por completo al cliente real.
- Mejorar las predicciones y previsiones para **cumplir plazos** y ajustarse a los recursos.
- Eliminar riesgos tomando en consideración la **incertidumbre**.
- **Disminuir costes,** por ejemplo, deben eliminarse actividades relacionadas con algunos productos intermedios, como documentos formales de especificaciones que no tienen una relación directa con el resultado final del producto.

Las **metodologías ágiles** están basadas fundamentalmente en metodologías **orientadas a objetos**, algunas de las más utilizadas son: Programación Extrema (XP), Scrum (Schwaber y Beedle 2001), o Rational Unified Process (RUP) que por su flexibilidad puede seguir los principios de la metodología ágil. De hecho vemos que RUP se adapta a cualquier necesidad (sistemas tradicionales, sistemas con gran incertidumbre, sistemas de tiempo real, desarrollo ágil…).



AUTOEVALUACIÓN:

9.- Indica para qué tipo de proyecto sería adecuada una metodología ágil.

a) Un sistema de control aéreo en tiempo real de un aeropuerto.

b) La informatización de un banco nacional.

c) El desarrollo de una aplicación que consiste en la gestión y mantenimiento de una gran base de datos.

d) El desarrollo de una aplicación web para compartir por Internet archivos y mensajes de los empleados de una empresa.

2.7.1.- Desarrollo ágil: programación extrema.

La **Programación extrema (eXtreme Programming XP)** se trata de un proceso ágil de desarrollo de software formulado por Kent Beck (1999). Es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizada en proyectos de corto plazo, con equipo pequeño y que requieren flexibilidad. La metodología **consiste** en:

- Un desarrollo **incremental** con iteraciones cortas y programación rápida.
- Cuya particularidad es dar mayor valor al individuo, a la colaboración con el cliente (que forma parte del equipo), pues son los requisitos para llegar al éxito del proyecto.
- Los principios y prácticas que propone son de sentido común pero llevadas al extremo, de ahí proviene su nombre.

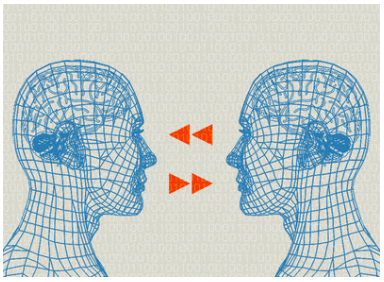
La **programación extrema** fue creada pensando en las siguientes circunstancias:

- Proyectos en contextos de **incertidumbre**, donde los requisitos tienen altas probabilidades de **cambiar** con el tiempo o son imprecisos, por ejemplo, porque el cliente no tiene claro lo que quiere, o porque el cambio de requisitos está ligado al propio problema a resolver.

- Proyectos con **alto riesgo,** por ejemplo, proyectos con una fecha de entrega que es indispensable cumplir, o proyectos totalmente novedosos para la industria con un alto riesgo técnico.
- Proyectos con un **grupo pequeño** de desarrolladores (máximo de 12), aunque el equipo completo sea bastante más extenso (incluye a jefes de equipo o representantes de clientes).

Las **características** fundamentales del método de programación extrema son:

- **Desarrollo iterativo e incremental,** realizando mejoras sucesivas.
- **Pruebas continúas,** se realizan automatizadas e incluso se escribe el código de la prueba antes de la codificación.
- **Programación por parejas,** se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto para buscar una mayor calidad del código, ya que el código es revisado y discutido mientras se escribe. Cada **miembro** lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el piloto y el copiloto: mientras uno conduce, el otro consulta el mapa.



- **Buen ambiente de trabajo,** centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo.
- **Frecuente comunicación** entre los usuarios y los desarrolladores. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- **Corrección de todos los errores** antes de añadir nueva funcionalidad, se hacen entregas frecuentes al cliente.
- **Refactorización del código**, es decir, **reescribir** ciertas partes del código para aumentar su legibilidad y su fácil mantenimiento pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo, ni pérdida de funcionalidad.
- **Propiedad del código compartida,** en lugar de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve que todo el personal pueda corregir y mejorar cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
- **Simplicidad** al desarrollar y codificar los módulos del sistema. Es la mejor manera de que las cosas funcionen, cuando todo funcione se podrá añadir funcionalidad si es necesario.
- **Reutilización del código** a partir de la creación de patrones o modelos estándar.

El **ciclo de vida ideal de XP** consiste de seis fases:

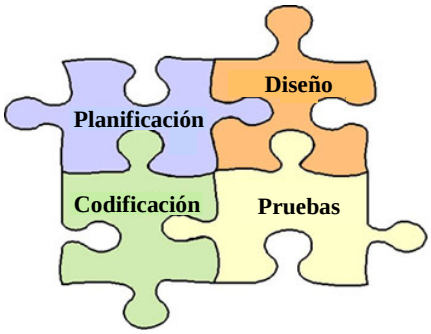
- Exploración de las necesidades.
- Planificación de la entrega con estimaciones.
- Iteraciones de desarrollo.
- Implantación del producto.
- Mantenimiento del producto implantado.
- Muerte o abandono del proyecto.

En las **iteraciones** se siguen estos pasos:

- El **cliente** define sus necesidades.
- El programador estima el **esfuerzo** necesario para su implementación.
- El cliente **selecciona** qué desarrollar de acuerdo con sus prioridades y las restricciones de tiempo.
- El desarrollador realiza lo **solicitado**.

En todas las iteraciones de este ciclo, tanto el cliente como el programador aprenden.

Para llevar a cabo lo solicitado por el cliente, esta metodología incluye una serie de prácticas que se pueden agrupar en cuatro grandes bloques:



- **planificación** (planning).
- **diseño** (designing).
- **codificación** (coding).
- **pruebas** (testing).

Sin embargo, estos bloques no deben realizarse en orden, sino que cada uno consta de una serie de actividades, y todas ellas se irán realizando de manera evolutiva.

AUTOEVALUACIÓN:

10.- La refactorización consiste en:

a) Desarrollar el software por parejas intercambiándose el código.

b) Reutilizar el código a partir de patrones refactorizados.

c) Reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

d) Programar con un código muy simple y fácil de mantener.

PARA SABER MÁS

Como hemos visto hay otras metodologías ágiles como por ejemplo Scrum. Si quieres leer más acerca de esta importante metodología visita la web que te proponemos.

Wikipedia: metodología Scrum
http://es.wikipedia.org/wiki/Scrum

2.8.- Desarrollo de software libre.

En los últimos años ha surgido con gran fuerza y éxito el fenómeno del **software libre** que proporciona el **código fuente de las aplicaciones** (código abierto). ¿Qué metodología emplean sus desarrolladores? No es sencillo responder a esta pregunta porque en el mundo del software libre hay entornos de trabajo muy **distintos** entre sí. Vamos a comenzar explicando qué es el software libre, después veremos quién lo desarrolla y terminaremos indicando qué metodologías se usan dependiendo de quién lo desarrolle.

Podemos definir el software libre como el software distribuido con su código fuente de manera que el usuario lo pueda utilizar para analizarlo, modificarlo, redistribuirlo y ejecutarlo en cuantos ordenadores desee.

Por tanto, no debe confundirse este concepto con el de **software gratuito**, ya que lo importante del software libre es la disposición del código abierto. El **software libre** es distribuido según una serie de licencias que indican los derechos que tiene el usuario y los que se reserva el cliente, una de estas **licencias** es la **GPL** (GNU General Public License, Licencia Pública General GNU) que da máximas libertades al usuario para su utilización, pero que protege al creador del código para que nadie pueda apropiarse del código y convertirlo en software propietario limitando sus derechos originales. Este concepto de software libre surge en contraposición al de **software propietario**, en el cual se limitan los derechos de manera que no se libera el código fuente, se prohíbe la copia, la distribución, la modificación, etc.

Según la definición anterior cualquiera podría desarrollar software libre. De hecho la procedencia del software libre es variada. El modelo de desarrollo tradicional de software libre y de código fuente abierto, se ha venido caracterizando por la participación de una red de programadores voluntarios que contribuyen a un proyecto software concreto. Esto ha cristalizado en una **Comunidad de Software Libre,** que se basa en la articulación de metodologías de trabajo que potencian la participación cooperativa de programadores dispersos por todo el planeta, para la elaboración colectiva de programas de software libre.

Dos elementos han hecho posible este modelo: la red mundial Internet y el afán de miles de desarrolladores de software por compartir sus conocimientos y experiencias. Dentro de este software libre debemos citar el sistema operativo **Linux** y sus aplicaciones asociadas, desarrolladas bajo licencia GPL.

En este modelo, la **integración** se configura como proceso habitual de mejora de un programa reutilizando y mejorando otros trabajos de programadores.

La ventaja de este modelo cooperativo es disponer de gran cantidad de desarrolladores que apliquen sus conocimientos y esfuerzos para la resolución de los problemas que plantea el diseño, la depuración y la corrección del código fuente, lo que permite alcanzar los niveles de calidad y estabilidad que hoy tienen.

En estos proyectos hay un **grupo** que lidera el desarrollo y que acepta o no las modificaciones realizadas por los desarrolladores y las incorpora a la versión principal. Este modelo asegura la **evolución continua** del software con un factor de crecimiento enorme, pues toda la comunidad participa del desarrollo.

En este entorno de trabajo cooperativo se han establecido una serie de **principios** y consejos para el desarrollo de software libre de código abierto, existiendo múltiples **coincidencias con los principios del desarrollo ágil**, como son:

- la comunicación.
- la retroalimentación.
- el enfoque centrado en el código y las personas.
- la organización menos formal y jerárquica.
- la flexibilidad ante el cambio.
- la reutilización o la sencillez del código.

Por ello, en el desarrollo del software libre triunfan las metodologías ágiles.

Sin embargo, actualmente el **desarrollo de software libre** no está limitado a estas comunidades de desarrollo, sino que cualquier entidad puede desarrollar programas o aplicaciones y distribuirlos con una licencia de software libre y de fuentes abiertas. Han aparecido **Organizaciones para la promoción del software libre** y de fuentes abiertas, también todo tipo de empresas, grandes y pequeñas, están participando en su desarrollo. De esta manera han surgido escenarios mixtos de software libre de código abierto con software propietario, y nuevas licencias que dan unos derechos pero limitan otros.

En este contexto, la **industria del software** (incluyendo grandes, medianas y pequeñas empresas) está adaptando su oferta de productos y servicios teniendo en cuenta la presencia y dimensiones del software libre, así como su demanda creciente.

- Hay **empresas** que están desarrollando software libre, centrando su negocio en los servicios asociados al mismo y no tanto en el propio desarrollo o distribución del producto.
- También las **administraciones públicas** han apostado por el software libre contribuyendo a su desarrollo, adquiriendo y desarrollando aplicaciones de código abierto. Así por ejemplo, la Junta de Andalucía ha desarrollado **Guadalinex**, una nueva distribución de Linux adaptada a las necesidades de la propia administración, de los centros escolares y de cualquier ciudadano con sus distintas posibilidades de configuración.

En este mundo de empresas, organizaciones y entidades trabajando con software libre, las metodologías de desarrollo de software utilizadas **varían** mucho y dependen del proyecto concreto a desarrollar y de la propia institución, por lo que las metodologías utilizadas siguen el mismo esquema de mercado que en el desarrollo de software propietario.

AUTOEVALUACIÓN:

11.- El sistema operativo Windows es software libre porque:

a) Puedo instalarlo libremente en cualquier ordenador.

b) Puedo copiarlo libremente.

c) Puedo modificarlo, mejorarlo y adaptarlo a mis necesidades porque tengo acceso al código fuente.

d) Ninguna respuesta es correcta porque Windows es software propietario de la empresa Microsoft.

PARA SABER MÁS

Si quieres saber más sobre la licencia de software libre GPL, puedes visitar esta web donde encontrarás tanto la versión original de la licencia en inglés como su traducción a español.

http://es.tldp.org/htmls/gpl.html

http://es.tldp.org/htmls/gpl.html

3.- Metodologías de las administraciones europeas.

Las **administraciones públicas**, que también desarrollan aplicaciones software, ¿qué metodología utilizarán? ¿Sería conveniente disponer de una metodología estándar para toda la administración pública?

Con la intención de **estandarizar** los diferentes proyectos informáticos que utilizaban y desarrollaban diferentes administraciones del mismo país, los organismos públicos comenzaron a desarrollar a finales de los años 70 y principios de los 80, metodologías de desarrollo software basadas en el **método estructurado**. Sin embargo, como las distintas administraciones necesitaban dar respuesta a proyectos de desarrollo muy diferentes, modificaron y ampliaron las metodologías existentes para adaptarlas a sus necesidades.

De esta manera, surgen una serie de metodologías en **diferentes países** europeos que tienen enfoques mixtos. Por **ejemplo**, a partir del desarrollo estructurado, realizan una metodología con diferentes enfoques para centrarse tanto en los procesos como en las estructuras de datos, los eventos, etc. Estas **metodologías** han ido evolucionando para adaptarse a nuevas necesidades y paradigmas incluyendo la **modelización de sistemas de tiempo real** o la **programación orientada a objetos**, por lo que su principal característica es que son metodologías mixtas.

Vamos a ver varios **ejemplos** de este tipo de metodologías, la metodología francesa Merise, la británica SSADM, la española Métrica, y el Eurométodo propuesto por la Unión Europea para dar homogeneidad a los proyectos ligados a su organización.

3.1.- Merise.

El proyecto Merise nace en 1977 dentro del centro CTI (Centre Technique d'Information) perteneciente al **Ministerio de Industria Francés**. Su objetivo era desarrollar una metodología de desarrollo del software para cubrir las necesidades tanto de la empresa pública como de las empresas en general. En 1978 se presentó su primera versión para el sector público, y fue en **1982** cuando el modelo se revisó y se extendió a las empresas del sector privado. Merise puede ser utilizado para el desarrollo de **todo tipo de sistemas de información**, desde aquellos que utilizan bases de datos hasta los que procesan eventos en tiempo real, y pretende que los proyectos se completen con éxito dentro del costo y tiempos planeados.

La **metodología Merise** aportó un **ciclo de vida más largo** que los existentes anteriormente con un conjunto definido de **etapas**. Abarca los aspectos relacionados con:

- La recopilación y validación de la información.
- Capacitación de personal.
- Evaluación de equipos informáticos.
- Análisis, diseño y validación de los procesos.
- Implementación, gestión de costos y tiempos y el desarrollo del código.

Además del ciclo de vida mas largo, esta metodología **introduce dos ciclos complementarios**:

- El **ciclo de abstracción,** se basa en la percepción de tres niveles de abstracción (de lo más abstracto a los más concreto): **conceptual** (finalidades generales de la empresa), **organizativo** (descripción de la organización necesaria para alcanzar los objetivos) y **físico** (concreción de los medios software y hardware necesarios). Los tres niveles se desarrollan simultáneamente.

- El **ciclo de decisión**. En cada etapa del ciclo de vida, cada vez se va bajando más en el nivel de abstracción (según el ciclo de abstracción), y se toman **decisiones**, al principio de forma global, y después de forma más detallada, conforme se va progresando en el trabajo.

Un aspecto muy importante de Merise es que se ocupa al mismo tiempo del estudio de los datos y de los procesos. Para llevar a cabo las diferentes actividades Merise utiliza distintas técnicas.

AUTOEVALUACIÓN:

12.- Merise es una metodología de desarrollo de software…

a) Estructurada orientada a datos jerárquicos.
b) Estructurada orientada a datos no jerárquicos.
c) Estructurada orientada a procesos.
d) Estructurada con un enfoque mixto.

3.2.- SSADM.

En 1980 el **gobierno británico** plantea la necesidad de crear una metodología para unificar y **estandarizar** los proyectos de software de las distintas administraciones. Así se desarrolló, entre el Central Computing and Telecommunications Agency (CCTA) y Learmonth and Burchett Management Systems (LBMS), la metodología SSADM (Structures Systems Analysis and Design Method).

Veamos algunas de las **características** fundamentales de SSADM que nos permitan acercarnos a esta metodología:

- Centrada en los **usuarios**, atendiendo a sus requisitos y su participación.
- Define de forma clara el proceso de **producción**, dando especificaciones acerca de qué hacer, cuándo y cómo.
- Utiliza **tres puntos de vista,** para orientarse a datos, eventos y procesos.
- **Flexibilidad** en herramientas y técnicas de implementación, proporcionando un conjunto de procedimientos para llevar a cabo las distintas tareas del análisis y diseño.

AUTOEVALUACIÓN:

13.- SSADM es una metodología de desarrollo de software:

a) Estructurada orientada a datos jerárquicos.
b) Estructurada orientada a datos no jerárquicos.
c) Estructurada orientada a procesos.
d) Estructurada con un enfoque mixto.

3.3.- Descripción general de métrica.

Al igual que ocurrió en otras administraciones públicas europeas, en **España** surgió la necesidad de crear una metodología de desarrollo de software común para todas las administraciones y organismos públicos.

La propuesta del **Ministerio de Administraciones Públicas** fue el desarrollo de la metodología **Métrica** para que todas las organizaciones siguiesen el mismo modelo y unificasen los criterios para aportar homogeneidad y eficiencia a las aplicaciones informáticas. El ámbito original de aplicación fue la administración general del estado español, pero puede ser utilizada por otras administraciones o empresas privadas. La primera versión es de **1989**, la segunda versión apareció en 1993 y en el año 2001 apareció la tercera versión (Métrica V3).

¿Qué objetivos persigue la última versión de **Métrica**? Podemos indicar que los **objetivos** generales de Métrica son:

- Satisfacer las **necesidades** de los usuarios dando una mayor importancia al análisis de requisitos.
- Mejorar la **productividad** permitiendo una mayor capacidad de adaptación a los cambios y teniendo en cuenta la reutilización en la medida de lo posible.
- Facilitar la **comunicación** entre todos los participantes en el desarrollo teniendo en cuenta su papel y responsabilidad, así como las necesidades de todos y cada uno de ellos.
- Facilitar la operación, **mantenimiento** y uso de los productos software obtenidos.
- Utilizar las distintas tecnologías que actualmente están conviviendo y los aspectos de gestión que aseguran que un proyecto cumple sus objetivos en términos de **calidad**, coste y plazos.

Hemos visto los objetivos que persigue, ¿cómo intenta alcanzar esos objetivos? Veamos las **características** de Métrica con las que cumple con los objetivos marcados:

- Tiene en cuenta los **métodos de desarrollo** más extendidos, así como los últimos **estándares** de ingeniería del software y calidad, además de referencias específicas en cuanto a seguridad y gestión de proyectos. Cumple con el estándar propuesto por **Eurométodo**.
- En una única estructura, Métrica cubre el desarrollo **estructurado** (con un enfoque mixto orientado a procesos, datos y eventos) y el desarrollo **orientado a objetos** (utiliza UML).
- La automatización de las actividades propuestas en la estructura de Métrica es posible ya que sus técnicas están soportadas por una amplia variedad de herramientas **CASE**.
- Ha sido concebida para abarcar el desarrollo completo de sistemas sea cual sea su complejidad y magnitud, por lo cual su estructura responde a desarrollos **máximos** y deberá adaptarse y dimensionarse en cada momento de acuerdo a las características particulares de cada proyecto.
- Es una **guía formal** que descompone cada uno de los procesos en actividades, y éstas a su vez en tareas. Para cada tarea se describe su contenido haciendo referencia a sus principales acciones, productos, técnicas, prácticas y participantes.
- Aunque propone una secuencia de pasos, que podría identificarse con una estructura de cascada, hace una propuesta **flexible** en la que deja libertad para escoger el orden de realización de las actividades, pudiéndose realizar en paralelo.
- Puede ser utilizada libremente con la única restricción de citar la fuente de su propiedad intelectual, es decir, el Ministerio de Administraciones Públicas de España.

AUTOEVALUACIÓN:

14.- Métrica es una metodología de desarrollo de software:

a) Estructurada orientada a datos.

b) Estructurada y orientada a objetos.

c) Estructurada orientada a procesos.

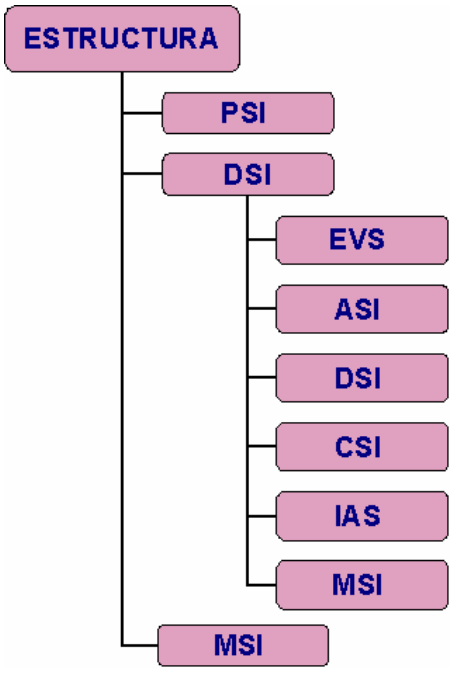
d) Orientada a objetos.

3.3.1.- Métrica: estructura principal.

Hemos visto los **fundamentos** de Métrica, pero nos falta ver cuál es la estructura real que propone, y cuáles son las etapas, tareas, actividades, productos...

Para ello debemos ver el **ciclo de vida** de Métrica que propone una serie de procesos que se descomponen en actividades, y además, se incluyen los procesos de interface para dar soporte a los aspectos organizativos del proyecto. Vamos a describir cada uno de estos procesos:

En este apartado vamos a tratar de explicarte la **estructura** de la metodología MÉTRICA. ¿Qué procesos forman parte de esta metodología?



- **Planificación de Sistemas de Información (Proceso PSI):** Su finalidad es dar soporte a la planificación e implantación de sistemas de información facilitando una visión general, necesaria para posibilitar su integración y un modelo de información global de la organización. De esta manera, podremos instalar por ejemplo, sistemas para la toma de decisiones como los que vimos en la unidad 2.

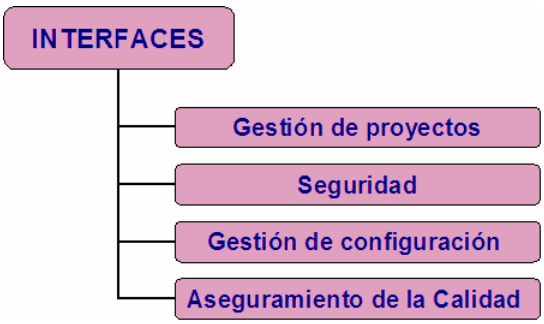
- **Desarrollo de sistemas de información (Proceso DSI)**: Contiene todas las actividades y tareas que se deben llevar a cabo para desarrollar un sistema, cubriendo desde el análisis de requisitos hasta la instalación del software. Es el **proceso** más importante de los identificados en el **ciclo de vida** de un sistema y se relaciona con todos los demás. Está dividido en distintos subprocesos:

- ✓ **Estudio de Viabilidad del Sistema (Proceso EVS)**: El propósito de este proceso es analizar un conjunto concreto de necesidades con la idea de proponer una solución a corto plazo. Los criterios con los que se hace esta propuesta serán aspectos económicos, técnicos, legales y operativos. Los **resultados** del **Estudio de Viabilidad del Sistema** constituirán la base para tomar la decisión de seguir adelante o abandonar.
- ✓ **Análisis del Sistema de Información (Proceso ASI)**: Su propósito es conseguir la especificación detallada del sistema a través de un catálogo de requisitos y una serie de modelos que cubran las necesidades de los clientes, y que serán la entrada para el proceso de Diseño del Sistema de Información.
- ✓ **Diseño del Sistema de Información (Proceso DSI)**: Su propósito es obtener la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del sistema de información, el plan de pruebas, etc.
- ✓ **Construcción del Sistema de Información (Proceso CSI)**: Tiene como objetivo final la construcción y prueba de los distintos componentes del sistema de información a partir del conjunto de especificaciones lógicas y físicas del mismo, obtenido en el proceso DSI. Se desarrollan los procedimientos de operación y seguridad y se elaboran los manuales de usuario final y de explotación.
- ✓ **Implantación y Aceptación del Sistema (Proceso IAS)**: Su objetivo principal es la entrega y aceptación del sistema en su totalidad, que puede comprender varios sistemas de información desarrollados de manera independiente según se haya establecido en el proceso de Estudio de Viabilidad del Sistema, y se prepara la implantación y puesta en marcha del sistema.

- **Mantenimiento del Sistema de Información (Proceso MSI):** comprende actividades y tareas de modificación, mejora y actualización de todos los componentes del sistema según las nuevas necesidades del cliente.

3.3.2.- Métrica: interfaces.

La estructura de Métrica incluye también un conjunto de **interfaces** que definen una serie de actividades de tipo organizativo o de soporte al proceso de desarrollo y a los productos, para complementar y garantizar el éxito del proyecto desarrollado.



Incluye estos procesos:

- **Gestión de Proyectos**: Tiene como finalidad principal la planificación, el seguimiento y control de las actividades y de los recursos humanos y materiales que intervienen en el desarrollo de un sistema de información. Como consecuencia de este control es posible conocer en todo momento qué **problemas** se producen y resolverlos o paliarlos lo más pronto posible, lo cual evitará desviaciones temporales y económicas. Contempla proyectos de desarrollo nuevos, proyectos de ampliación y mejora de los ya existentes.

- **Seguridad:** Incorpora mecanismos de seguridad adicionales a los que se proponen en la propia metodología para reducir los riesgos lógicos, asegurando el desarrollo de cualquier tipo de sistema con integridad y consistencia.

- **Gestión de Configuración**: Su finalidad es identificar, definir, proporcionar información y controlar los cambios en la configuración del sistema, así como las modificaciones y versiones de los mismos. Este proceso permitirá conocer el **estado** de cada uno de los productos, garantizando que no se realizan cambios incontrolados y que todos los participantes en el desarrollo disponen de la versión adecuada de los productos que manejan.

- **Aseguramiento de la Calidad**: están orientadas a verificar la calidad de los productos. Son actividades que evalúan la calidad y que son realizadas por un grupo de asesoramiento de la calidad independiente de los responsables de la obtención de los productos.

Métrica también incluye la descripción de las técnicas que se utilizan en los procesos y los perfiles de las personas que participan en el desarrollo del software.

PARA SABER MÁS

Podemos visitar la página web oficial del Ministerio de Administraciones Públicas donde están todos los documentos de la metodología para descargar y usar libremente:

Ministerio de Administraciones Públicas: Métrica V3

http://www.csi.map.es/csi/metrica3/

3.4.- Eurométodo.

A lo largo de esta unidad hemos visto diferentes **metodologías** de desarrollo de software, que las empresas y administraciones públicas pueden utilizar en función de sus intereses y necesidades. Cada **método** aporta su estructura con unas particularidades determinadas. Cuando una empresa privada o un organismo público pretenden adquirir un **sistema de información** informático debe establecerse una buena colaboración y relación de entendimiento entre el proveedor y el cliente.

Uno de los principales **obstáculos** en la consecución de este mutuo entendimiento es la **variedad de metodologías** de desarrollo existentes, cada una de ellas con sus conceptos y terminología propios y en general, con un **vocabulario** procedente de la ingeniería del software, que no siempre es fácilmente comprendido por usuarios, compradores, responsables de la contratación, etc. Ante una determinada oferta de contratación pueden acudir distintos proveedores, y cada uno de ellos, puede ofrecer un análisis realizado con una metodología distinta, lo cual dificulta la toma de decisión sobre la opción más adecuada. ¿Habría alguna solución?

Ésa fue la pregunta que se hizo la **Comisión Europea**, es decir, ¿es posible establecer un marco común para toda la **Unión Europea** de manera que clientes y desarrolladores puedan utilizar especificaciones comunes?

Con esta idea se pensó en la elaboración de un **Eurométodo** que en realidad no es una metodología de desarrollo de software, sino un **marco** con el que pueden guiarse desarrolladores y clientes tanto de empresas privadas como administraciones públicas, para adquirir sistemas de información software con garantías.

Podemos definir Eurométodo como una metodología para la adquisición de sistemas de información y servicios relacionados, desarrollada en el marco de un proyecto de instituciones europeas bajo supervisión de la Comisión Europea. Se trata de una metodología de carácter público que puede utilizarse libremente. Sus características de diseño permiten su utilización tanto en el ámbito público como en el privado.

Con **Eurométodo**, (actualmente promocionado por distintos motivos por la Comisión Europea como ISPL) la Comisión Europea pretende fomentar la **apertura** del mercado de sistemas de información, mejorar la movilidad de las personas entre los distintos países, facilitar la organización de proyectos internacionales y ayudar a los clientes a valorar las ofertas de los proveedores.

Para alcanzar estas pretensiones, Eurométodo se ha desarrollado con dos **orientaciones** diferentes:

- **Como marco metodológico** proporciona un conjunto de conceptos y una terminología, que aporta las siguientes ventajas:

    - ✓ **Mejorar la relación cliente-proveedor**: Los conceptos y la terminología de Eurométodo se han diseñado para facilitar el entendimiento mutuo entre el cliente, el proveedor, y los desarrolladores. También permite entender mejor las propuestas de los proveedores y crear una situación de cooperación eficaz entre clientes y proveedores, que favorezca al máximo que los proyectos se rematen felizmente para ambas partes.
    - ✓ **Armonizar los métodos**: para lo que existen **diccionarios puente** entre los distintos métodos y Eurométodo. También, evita quedar ligado a un determinado proveedor o a un determinado método de desarrollo.

- **Como método** sirve para definir, planificar y ejecutar la adquisición de un sistema de información y los servicios relacionados. Esto aporta las siguientes ventajas:

    - ✓ Permite gestionar y valorar la situación del problema y los **riesgos** asociados: Anima a los clientes y proveedores a controlar los costes y los plazos previstos, gestionar los riesgos y mejorar el entendimiento mutuo.
    - ✓ Poder expresar los **requisitos** del sistema con mayor claridad.
    - ✓ Valorar mejor el **objetivo** de la adquisición.
    - ✓ Determinar mejor la **estrategia de adquisición**, en función de la situación específica del problema a resolver.
    - ✓ Mejorar el "**plan de entregas**" del sistema software, el aspecto más relevante de las relaciones cliente-proveedor a nivel contractual.

Eurométodo es **compatible** con las técnicas estructuradas, los modelos orientados a objetos y está abierto a cualquier evolución que estos enfoques u otros puedan experimentar en el futuro. Así por ejemplo, se han establecido las conexiones conceptuales y metodológicas oportunas entre Métrica y Eurométodo.

PARA SABER MÁS

En la página web del Ministerio de Administraciones Públicas dispones de todos los documento del Eurométodo.

Ministerio de Administraciones Públicas: Eurométodo

http://www.csi.map.es/csi/pg5e40.htm

AUTOEVALUACIÓN:

15.- Eurométodo es:

a) Una metodología de desarrollo de software estructurada mixta.

b) Una metodología de desarrollo de software orientada a objetos.

c) La armonización entre SSADM, Métrica y Merise.

d) Todas las respuestas anteriores son correctas.

SOLUCIONES AUTOEVALUACIÓN

1.- c
2.- c
3.- d
4.- b
5.- b
6.- b
7.- a
8.- a
9.- d
10.- c
11.- d
12.- d
13.- d
14.- b
15.- d

GLOSARIO.

Abstracción.

La abstracción consiste en un mecanismo para reducir la información secundaria y poder centrarse en la importante. Así, en un programa, la abstracción de datos proporciona la definición de un dato o concepto y las operaciones aplicables al concepto, pero sin especificar la implementación concreta del concepto; y la abstracción de programas permite hacer subprogramas que podemos usar sin saber exactamente cómo están realizados internamente.

Casos de uso.

Es una técnica para la captura de requisitos de un nuevo sistema o una actualización software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico. Explican por tanto, qué debe realizar el sistema. Normalmente, en los casos de usos se evita el empleo de lenguaje técnico, prefiriendo en su lugar un lenguaje más cercano al usuario final.

GNU.

Es un proyecto de desarrollo de software libre basado en el sistema operativo Unix impulsado por el programador americano Richard Stallman en 1983 con la licencia pública general GNU (GPL). Su nombre es un acrónimo recursivo en inglés que significa “GNU no es Unix” y se lee ñu, como el animal de su logotipo.

Herramientas CASE.

Las Herramientas CASE (Computer Aided Software Engineering, o 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, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Incertidumbre.

Hablamos de incertidumbre cuando no tenemos un conocimiento seguro y claro de algo, por ejemplo, cuando no podemos determinar claramente las tareas en las que se divide un proceso. En presencia de incertidumbre la percepción del mundo es mucho más compleja. La incertidumbre oscurece las relaciones causa y efecto, dificultando la comprensión de los fenómenos. Si hay incertidumbre no todo es cognoscible, ni predecible, por mucho esfuerzo que se haga. La predicción metereológica es un ejemplo de tarea con presencia de incertidumbre.

Lógica Procedimental.

La lógica procedimental define el algoritmo (secuencia ordenada de instrucciones a seguir para alcanzar la solución) de un programa o de un proceso.

Método científico.

El método científico se entiende como el estudio sistemático, controlado, empírico y crítico de las hipótesis acerca de la explicación de distintos fenómenos. Descartes hizo una importante aportación al método científico mediante su “Discurso del Método”.

Microprocesador.

Es un conjunto de circuitos electrónicos integrados que realizan las operaciones de cálculo y control de un ordenador. Es el “cerebro” de la máquina.

Módulo.

Un módulo puede ser definido de varias formas, pero por lo general debe ser un componente de un sistema o programa informático mayor, y operar en ese sistema independientemente de las operaciones de otros componentes.

Objeto.

Un objeto es una representación detallada, concreta y particular de "algo". Tal representación determina su identidad, su estado y su comportamiento particular en un momento dado. Por ejemplo, podríamos definir un objeto llamado “cuenta de banco” cuyo estado está marcado por su “saldo”, y su comportamiento por las operaciones “ingresar” y “extraer dinero” de la “cuenta”.

Programación Estructurada.

A finales de los años sesenta surgió una nueva forma de programar que no solamente daba lugar a programas fiables y eficientes, sino que además estaban escritos de manera que facilitaba su comprensión posterior. Se demuestra mediante teoremas que todo programa puede escribirse utilizando únicamente las tres instrucciones de control siguientes: Secuencia de instrucciones, Instrucción condicional, y la Iteración o bucle de instrucciones.

Riesgo.

Hay riesgo cuando existe incertidumbre sobre la ocurrencia de un suceso cuya consecuencia es desfavorable. Por ejemplo, si no hacemos una buena predicción de los plazos de entrega de un programa tenemos el riesgo de incumplir un contrato.

Sistemas en tiempo real.

La informática o programación en tiempo real está relacionada con los sistemas hardware y software que se ven limitados por problemas de tiempo. Por ejemplo, el sistema informático que utilizan los controladores aéreos debe seguir el movimiento de los aviones y responder al ritmo en que realizan las operaciones de navegación, despegue y aterrizaje.