Preparador Informática

www.preparadorinformatica.com

TEMA 48. INFORMÁTICA.

INGENIERÍA DEL SOFTWARE. CICLO DE DESARROLLO DEL SOFTWARE. TIPOS DE CICLOS DE DESARROLLO. METODOLOGÍAS DE DESARROLLO. CARACTERÍSTICAS DISTINTIVAS DE LAS PRINCIPALES METODOLOGÍAS DE DESARROLLO UTILIZADAS EN LA UNIÓN EUROPEA

TEMA 48 INF: INGENIERÍA DEL SOFTWARE. CICLO DE DESARROLLO DEL SOFTWARE. TIPOS DE CICLOS DE DESARROLLO. METODOLOGÍAS DE DESARROLLO. CARACTERÍSTICAS DISTINTIVAS DE LAS PRINCIPALES METODOLOGÍAS DE DESARROLLO UTILIZADAS EN LA UNIÓN EUROPEA.

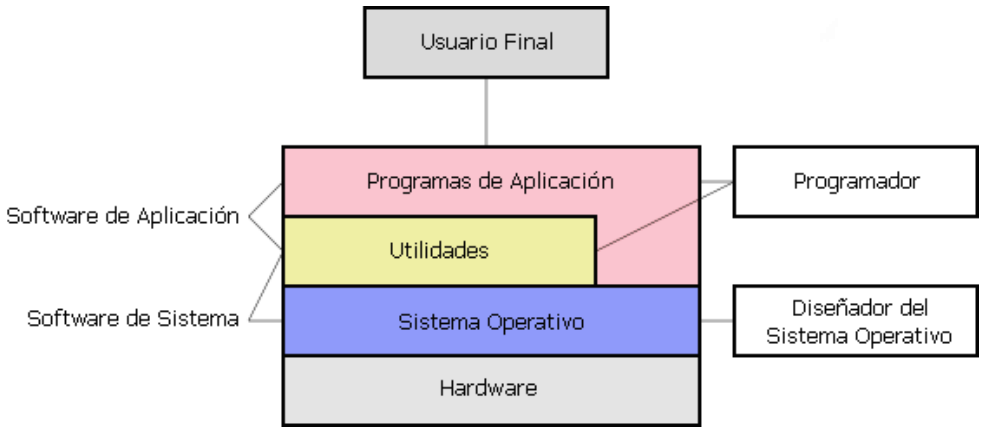
1. INTRODUCCIÓN

Un sistema operativo se trata de un programa que realiza procesos básicos en el sistema informático, en donde además permite que se puedan realizar el resto de las operaciones. Cuando se enciende el ordenador o el equipo, el hardware del mismo entrega el mando al sistema operativo quien comenzará un proceso de identificación de los dispositivos que tiene el sistema, ya sean de entrada, de salida o mixtos. Una vez verificada la existencia de los dispositivos y realizadas las correcciones que sean necesarias permitirá la apertura del mismo para que el usuario pueda comenzar a trabajar con las **aplicaciones informáticas.**

La importancia de este tema radica en el uso continuo, diario y constante que hacemos de distintas aplicaciones informáticas en nuestro día a día ya sea en nuestro ordenador, tablet, móvil, etc. En este tema nos centraremos en cómo se crean estas aplicaciones informáticas, es decir en la ingeniería del software y en el ciclo de desarrollo de software y sus metodologías de desarrollo.

2. CONCEPTO DE APLICACIÓN.

En informática, una **aplicación** es un programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de tareas. Las aplicaciones pertenecen al software de aplicación y están en contacto directo con el usuario final como podemos ver en el esquema siguiente:



Una solución informática se orienta a la automatización de ciertas tareas complicadas, como pueden ser la contabilidad, la redacción de documentos, o la gestión de almacenes. Ciertas aplicaciones desarrolladas a medida suelen ofrecer una gran potencia de uso y rapidez en la ejecución, ya que están

exclusivamente diseñadas para resolver un problema específico. Otros, llamados paquetes integrados de software, ofrecen menos potencia en cuanto a adaptabilidad al uso y requerimientos en cuanto al equipo utilizado, pero a cambio, incluyen un variado abanico de aplicaciones, como es el caso de los programas procesadores de textos, procesadores de hojas de cálculo, y manejadores de base de datos. Otros ejemplos de programas de aplicación pueden ser: presentaciones, diseño gráfico, cálculo, finanzas, compresión de archivos, presupuestos de obras, gestión de empresas, etc.

Gracias a los desarrolladores independientes y a las empresas que trabajan en el desarrollo de productos informáticos, la variedad que se puede encontrar en el mercado es muy amplia y variada.

3. INGENIERÍA DEL SOFTWARE

La **ingeniería de software** es una disciplina formada por un conjunto de métodos, herramientas y técnicas que se utilizan en el **desarrollo de las aplicaciones informáticas** (**software**).

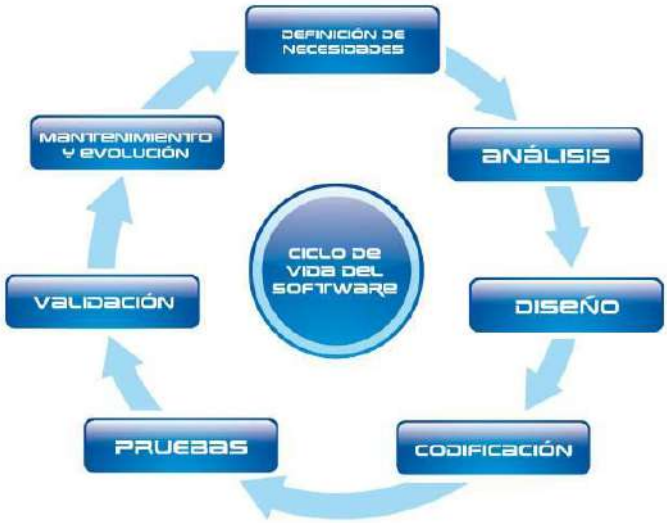
Esta disciplina trasciende la actividad de programación, que es el pilar fundamental a la hora de crear una aplicación. El ingeniero de software se encarga de toda la gestión del proyecto para que éste se pueda desarrollar en un plazo determinado y con el presupuesto previsto.

La ingeniería de software, por lo tanto, incluye el análisis previo de la situación, el diseño del proyecto, el desarrollo del software, las pruebas necesarias para confirmar su correcto funcionamiento y la implementación del sistema.

Cabe destacar que el proceso de desarrollo de software implica lo que se conoce como **ciclo de vida del software**.

4. CICLO DE VIDA DEL SOFTWARE

El ciclo de vida del desarrollo Software (SDLC en sus siglas inglesas), es una secuencia estructurada y bien definida de las etapas en Ingeniería de software para desarrollar el producto software deseado. El SDLC aporta una serie de pasos a seguir con la finalidad de diseñar y desarrollar un producto software de manera eficiente. Los pasos a seguir se pueden resumir en el siguiente gráfico:



En cada una de las etapas de un modelo de ciclo de vida, se pueden establecer una serie de objetivos, tareas y actividades que lo caracterizan. Veamos una pequeña descripción de cada una de las etapas del ciclo de vida del software:

- **Definición de necesidades y especificaciones del proyecto**: Esta etapa tiene como objetivo la creación de un documento en el cual se reflejan los requerimientos y funcionalidades que ofrecerá al usuario el sistema a implementar (qué, y no cómo, se va a implementar). El equipo debe tener las reuniones necesarias para intentar conseguir la máxima cantidad de información posible sobre lo que requieren. Los requisitos se contemplan y agrupan en requisitos del usuario, requisitos funcionales y requisitos del sistema. La recolección de todos los requisitos se lleva a cabo como se especifica a continuación:
    - Estudiando el software y el sistema actual u obsoleto.
    - Entrevistando a usuarios y a desarrolladores de Software.
    - Consultando la base de datos.
    - Recogiendo respuestas a través de cuestionarios.

  Después de la recolección de requisitos, el equipo idea un plan para procesar el software. En esta fase, el equipo analiza si el software puede hacerse para cubrir todos los requisitos del usuario y si hay alguna posibilidad de que el software ya no sea necesario. Se investiga si el proyecto es viable a nivel financiero, práctico, y a nivel tecnológico para que la organización acepte la

oferta. Hay varios algoritmos disponibles, los cuales ayudan a los desarrolladores a concluir si el proyecto software es factible o no.

- **Análisis**: Determinamos los elementos que intervienen en el sistema a desarrollar, su estructura, relaciones, evolución temporal, funcionalidades, tendremos una descripción clara de qué producto vamos a construir, qué funcionalidades aportará y qué comportamiento tendrá. En este paso los desarrolladores trazan su plan e intentan crear el mejor y más conveniente modelo de software para el proyecto.
- **Diseño**: El siguiente paso es diseñar el producto software con la ayuda de toda la información recogida sobre requisitos y análisis. Ya sabemos qué hacer, ahora tenemos que determinar cómo debemos hacerlo: cómo debe ser construido el sistema en cuestión, definimos en detalle entidades y relaciones de las bases de datos, seleccionamos el lenguaje que vamos a utilizar, el Sistema Gestor de Bases de Datos, etc. Para esta fase los ingenieros pueden crear diagramas lógicos, diagramas de flujo de datos, y en algunos casos pseudocódigos.
- **Codificación**: Esta fase también se puede denominar 'fase de programación'. Empezamos a codificar algoritmos y estructuras de datos, definidos en las etapas anteriores, en el correspondiente lenguaje de programación o para un determinado sistema gestor de bases de datos. En algunos proyectos (aunque es cada vez menos habitual) se pasa directamente a esta etapa; son proyectos muy arriesgados que adoptan un modelo de ciclo de vida de code & fix (codificar y corregir) donde se eliminan las etapas de especificaciones, análisis y diseño con la consiguiente pérdida de control sobre la gestión del proyecto.
- **Pruebas**: El objetivo de esta etapa es garantizar que nuestro programa no contiene errores de diseño o codificación. En esta etapa no deseamos saber si nuestro programa realiza lo que solicitó el usuario, en ésta deseamos encontrar la mayor cantidad de errores. Todos los programas contienen errores: encontrarlos es cuestión de tiempo. Lo ideal es encontrar la mayoría, si no todos, en esta etapa. Los errores pueden arruinar el software tanto a nivel crítico y hasta el punto de ser eliminado. Encontrar errores y su remedio a tiempo es la llave para conseguir un software fiable.

- **Validación**: Aquí se instala el software en máquinas de clientes. Esta etapa tiene como objetivo la verificación de que el sistema desarrollado cumple con los requerimientos expresados inicialmente por el cliente y que han dado lugar al presente proyecto. En muchos proyectos las etapas de validación y pruebas se realizan en paralelo por la estrecha relación que llevan. Sin embargo, tenemos que evitar la confusión: podemos realizarlos en paralelo, pero no como una única etapa.
- **Mantenimiento y Evolución**: En la mayoría de los proyectos a esta etapa se le asigna, no sólo el agregado de nuevas funcionalidades (evolución); sino la corrección de errores que surgen (mantenimiento). En la práctica esta denominación no es del todo errónea, ya que es posible que aun luego de una etapa de pruebas y validación exhaustiva, se filtren errores. Esta fase confirma el funcionamiento del software en términos de más eficiencia y menos errores. Si se requiere, los usuarios se forman, o se les presta documentación sobre como operar y como mantenerlo en funcionamiento. Esta fase puede que tenga que encarar retos originados por virus ocultos o problemas no identificados del mundo real.

5. TIPOS DE CICLOS DE DESARROLLO

Una vez conocidas las etapas, tendremos que analizar cómo abordarlas en su conjunto. Existen distintos tipos de ciclo de vida, y la elección de un modelo para un determinado tipo de proyecto es realmente importante. El Paradigma de desarrollo de Software ayuda al desarrollador a escoger una estrategia para desarrollar el software. El paradigma de desarrollo software tiene su propio set de herramientas, métodos y procedimientos, los cuales son expresados de forma clara, y define el ciclo de vida del desarrollo del software. Hay bastantes tipos de ciclos de desarrollo por lo que veremos sólo algunos de los más importantes a continuación:

5.1. MODELO LINEAL

Es el más sencillo de todos los modelos, consiste en descomponer la actividad global del proyecto en etapas separadas, que son realizadas de manera lineal, es decir, cada etapa se realiza una sola vez, a continuación de la etapa anterior

y antes de la etapa siguiente. Con un ciclo de vida lineal es muy fácil dividir las tareas, y prever los tiempos (sumando linealmente los de cada etapa).

Se destaca como ventaja la sencillez de su gestión y administración tanto económica como temporal, ya que se acomoda perfectamente a proyectos internos de una empresa para programas pequeños. Tiene como desventaja que no es apto para desarrollos que superen mínimamente requerimientos de retroalimentación entre etapas, es decir es muy costoso retomar una etapa anterior al detectar algún fallo.

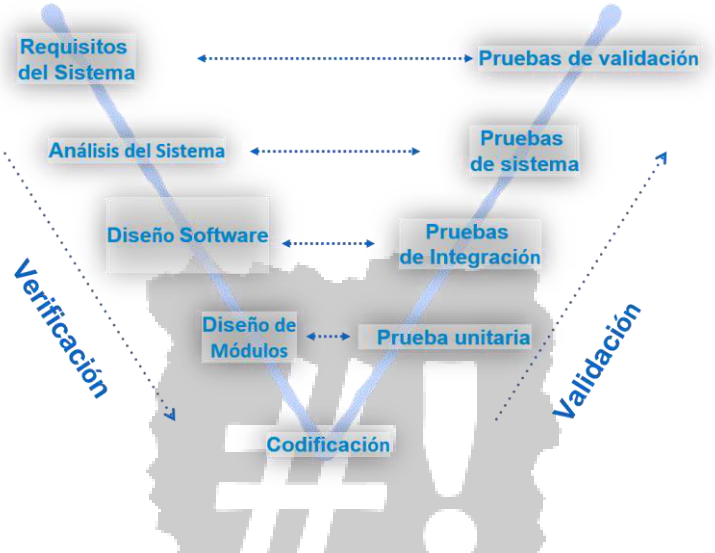
5.2. MODELO EN CASCADA

Es un ciclo de vida que admite iteraciones, contrariamente a la creencia de que es un ciclo de vida secuencial como el lineal. Después de cada etapa se realiza una o varias revisiones para comprobar si se puede pasar a la siguiente. Es un modelo rígido, poco flexible, y con muchas restricciones. Aunque fue uno de los primeros, y sirvió de base para el resto de los modelos de ciclo de vida.

Una de sus ventajas, además de su planificación sencilla, es la de proveer un producto con un elevado grado de calidad sin necesidad de un personal altamente cualificado. Se pueden considerar como inconvenientes: la necesidad de contar con todos los requerimientos (o la mayoría) al comienzo del proyecto, y, si se han cometido errores y no se detectan en la etapa inmediata siguiente, es costoso y difícil volver atrás para realizar la corrección posterior. Además, los resultados no los veremos hasta que no estemos en las etapas finales del ciclo, por lo que, cualquier error detectado nos trae retraso y aumenta el costo del desarrollo en función del tiempo que nos lleve la corrección de éstos.

5.3. MODELO EN V

Contiene las mismas etapas que el ciclo de vida en cascada. A diferencia de aquél, a éste se le agregaron dos subetapas de retroalimentación entre las etapas de análisis y mantenimiento, y entre las de diseño y pruebas.



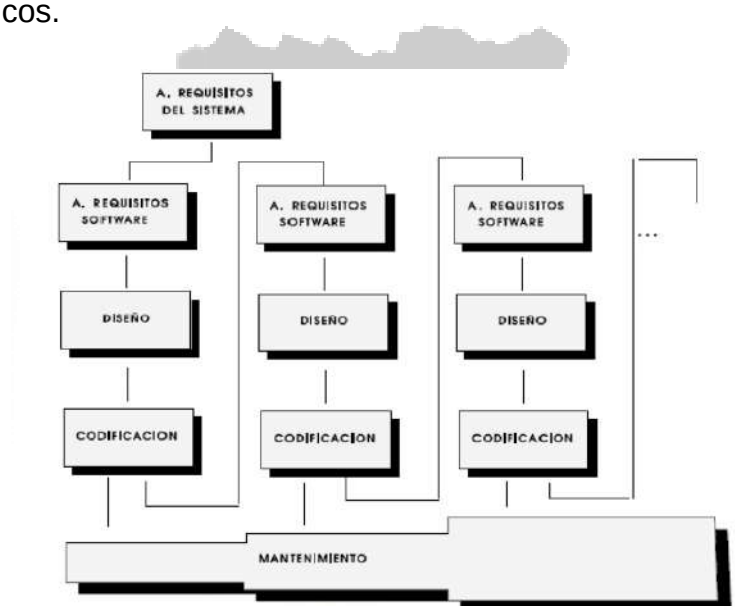
Las ventajas y desventajas de este modelo son las mismas del ciclo anterior, con el agregado de los controles cruzados entre etapas para lograr una mayor corrección. Podemos utilizar este modelo de ciclo de vida en aplicaciones, que si bien son simples (pequeñas transacciones sobre bases de datos, por ejemplo), necesitan una confiabilidad muy alta. Un ejemplo claro en el que no nos podemos permitir el lujo de cometer errores es una aplicación de facturación, en la que, si bien los procedimientos vistos individualmente son de codificación e interpretación sencilla, la aplicación en su conjunto puede tener matices complicados

5.4. MODELO INCREMENTAL

En una visión genérica, el proceso se divide en 4 partes: Análisis, Diseño, Código y Prueba. Sin embargo, para la producción del software, se usa el principio de trabajo en cadena o “Pipeline”, utilizado en muchas otras formas de programación. Con esto se mantiene al cliente en constante contacto con los resultados obtenidos en cada incremento.

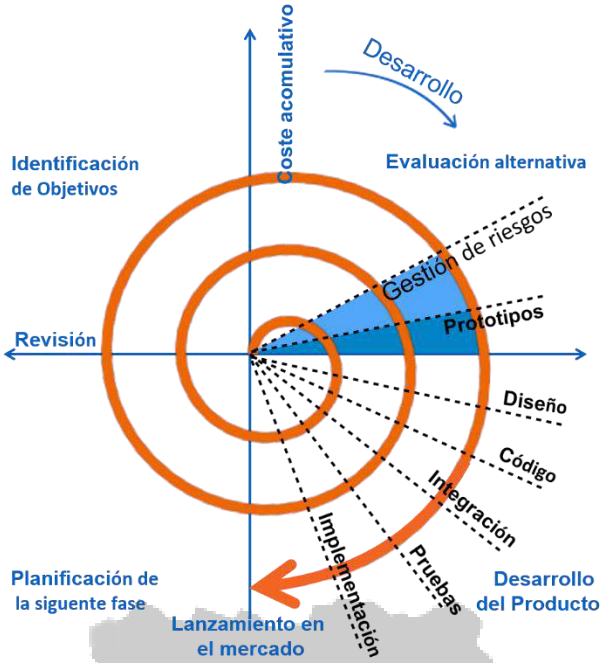
Es el mismo cliente el que incluye o desecha elementos al final de cada incremento a fin de que el software se adapte mejor a sus necesidades reales. El proceso se repite hasta que se elabore el producto completo. De esta forma el tiempo de entrega se reduce considerablemente.

El modelo incremental es de naturaleza interactiva, pero se diferencia de aquellos en que al final de cada incremento se entrega un producto completamente operacional. Es particularmente útil cuando no se cuenta con una dotación de personal suficiente. Los primeros pasos los pueden realizar un grupo reducido de personas y en cada incremento se puede añadir personal de ser necesario. Por otro lado, los incrementos se pueden planear para gestionar riesgos técnicos.



5.5. MODELO EN ESPIRAL

El modelo se basa en una serie de ciclos repetitivos para ir ganando madurez en el producto final. En este modelo se tiene más en cuenta el concepto de riesgo que aparece debido a las incertidumbres e ignorancias de los requerimientos proporcionados al principio del proyecto o que surgirán durante el desarrollo. A medida que el ciclo se cumple (el avance de la espiral), se van obteniendo prototipos sucesivos que van ganando la satisfacción del cliente o usuario. A menudo, la fuente de incertidumbres es el propio cliente o usuario, que en la mayoría de las oportunidades no sabe con perfección todas las funcionalidades que debe tener el producto.



La ventaja más notoria de este modelo de desarrollo de software es que puede comenzarse el proyecto con un alto grado de incertidumbre, se entiende también como ventaja el bajo riesgo de retraso en caso de detección de errores, ya que se pueden solucionar en la próxima rama de la espiral. Algunas de las desventajas son: el costo temporal que suma cada vuelta de la espiral, la dificultad para evaluar los riesgos y la necesidad de la presencia o la comunicación continua con el cliente o usuario. Se observa que es un modelo adecuado para grandes proyectos internos de una empresa, en donde no es posible contar con todos los requerimientos desde el comienzo y el usuario está en nuestro mismo ambiente laboral.

6. METODOLOGÍAS DE DESARROLLO. CARACTERÍSTICAS

Las metodologías para el desarrollo del software imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Una metodología de desarrollo de software tiene como principal objetivo aumentar la calidad del software que se produce en todas y cada una de sus fases de desarrollo. No existe una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigiéndose así que el proceso sea configurable. Las metodologías de desarrollo se pueden dividir en dos grupos de acuerdo con sus características y los objetivos que persiguen: ágiles y robustas.

6.1. METODOLOGÍAS ÁGILES.

Se caracterizan por hacer énfasis en la comunicación cara a cara, es decir, se basan en una fuerte y constante interacción, donde clientes desarrolladores y desarrolladores trabajan constantemente juntos, estableciéndose así una estrecha comunicación. Estas metodologías están orientadas al resultado del producto y no a la documentación; exige que el proceso sea adaptable, permitiendo realizar cambios de último momento. Se puede hacer mención dentro de las metodologías ágiles a:

Extreme Programming (XP)

Es una metodología ágil 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. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Kent Beck, el padre de XP, describe la filosofía de XP sin cubrir los detalles técnicos y de implantación de las prácticas. Posteriormente, otras publicaciones de experiencias se han encargado de dicha tarea.

Scrum

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto. Éstas son las verdaderas protagonistas, especialmente la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

RUP

La metodología RUP, abreviatura de *Rational Unified Process (o Proceso Unificado Racional*), es un proceso propietario de la ingeniería de software creado por Rational Software , adquirida por IBM, proporcionando técnicas que deben seguir los miembros del equipo de desarrollo de software con el fin de aumentar su productividad en el proceso de desarrollo.

La metodología RUP utiliza el enfoque de la orientación a objetos en su diseño y está diseñado y documentado el uso de la notación UML (Unified Modeling Language) para ilustrar los procesos en acción. Utiliza técnicas y prácticas probadas comercialmente.

Es un proceso considerado pesado y preferentemente aplicable a grandes equipos de desarrollo y grandes proyectos, pero el hecho de que es ampliamente personalizable que permite adaptarse a proyectos de cualquier escala.

Para la gestión del proyecto, la metodología RUP proporciona una solución disciplinada como las tareas y responsabilidades señaladas dentro de una organización de desarrollo de software.

6.2. METODOLOGÍAS ROBUSTAS O TRADICIONALES

Están guiadas por una fuerte planificación. Centran su atención en llevar una documentación exhaustiva de todo el proceso de desarrollo y en cumplir con un plan de proyecto, definido en la fase inicial del mismo. Entre las metodologías robustas se encuentran:

Métrica v3

MÉTRICA es una metodología de planificación, desarrollo y mantenimiento de sistemas de información. Es la metodología utilizada por la administración española. Ofrece a las Organizaciones un instrumento útil para la sistematización de las actividades que dan soporte al ciclo de vida del software dentro del marco que permite alcanzar los siguientes objetivos:

- Proporcionar o definir Sistemas de Información que ayuden a conseguir los fines de la Organización mediante la definición de un marco estratégico para el desarrollo de los mismos.
- Dotar a la Organización de productos software que satisfagan las necesidades de los usuarios dando una mayor importancia al análisis de requisitos.
- Mejorar la productividad de los departamentos de Sistemas y Tecnologías de la Información y las Comunicaciones, permitiendo una mayor capacidad de adaptación a los cambios y teniendo en cuenta la reutilización en la medida de lo posible.
- Facilitar la comunicación y entendimiento entre los distintos participantes en la producción de software a lo largo del ciclo de vida del proyecto, 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 obtenido.

Esta metodología tiene un enfoque orientado al proceso. Como punto de partida, MÉTRICA Versión 3 cubre el Proceso de Desarrollo y el Proceso de Mantenimiento de Sistemas de Información. MÉTRICA Versión 3 ha sido concebida para abarcar el desarrollo completo de Sistemas de Información 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.

SSADM (Método de análisis y diseño de sistemas estructurados)

Es un enfoque de sistemas para el análisis y diseño de sistemas de información. SSADM fue producida para la Agencia Central de Informática y Telecomunicaciones en el gobierno del Reino Unido, que es la oficina que se encarga del uso de la tecnología en el gobierno, a partir de 1980.

Se considera que SSADM representa el pináculo del enfoque riguroso en la documentación hacia el diseño del sistema.

Las tres técnicas más importantes que se utilizan en SSADM son los siguientes:

- Modelado de datos lógicos.
- Modelado de flujo de datos.
- Modelado Entidad Evento.

Merise

Merise es un método integrado de análisis, concepción y gestión de proyectos, desarrollado en Francia. El mismo provee un marco metodológico y un lenguaje común riguroso para los desarrollos informáticos. Involucra aspectos relacionados con: Capacitación del personal, análisis, diseño y validación de procesos, evaluación de equipos informáticos, gestión de tiempos, etc.

En cada etapa del ciclo de vida, se utilizan cada vez los formalismos de abstracción y se toman decisiones. Al principio de forma general, conforme se desarrolle el trabajo se irán detallando cada vez más las decisiones.

Su ciclo de vida contempla 3 grandes periodos: Concepción, Realización y Mantenimiento. Estos periodos, involucran un total de 6 etapas que conforman el ciclo de vida del desarrollo de un sistema de información.

Normalmente realiza el estudio de los tratamientos por un lado y el de los datos por otro. Utiliza el modelo Entidad/Relación para representar los datos y Diagramas de Encadenamiento de Procedimientos para representar los tratamientos.

7. CONCLUSIÓN

De todos es sabido la gran importancia que tienen para nuestro trabajo diario el uso de distintas aplicaciones. En todos los dispositivos que manejamos actualmente encontramos aplicaciones de todo tipo. También las empresas necesitan aplicaciones para la mayoría de sus gestiones diarias.

Es por ello que es necesario conocer cómo se crean estas aplicaciones, todas las fases necesarias para su creación basándonos en la ingeniería del software y en el ciclo de vida del software y, sobre todo, conocer qué tipos de ciclos de desarrollo y qué metodologías de desarrollo podemos usar en cada momento según la aplicación o software que queramos crear.

8. BIBLIOGRAFÍA

- Piattini, Mario G. y otros. ***Análisis y diseño de Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería del Software***. Editorial Ra-Ma.
- Prieto A., y otros. ***Introducción a la informática***. Editorial McGraw-Hill
- Montañez F. ***Aplicaciones informáticas de propósito general***. Ed. McGraw-Hill
- Rocandio F.J. Tecnologías de la información. Ed. McGraw-Hill
- http://tecnomundo.es/aplicaciones-informaticas/
- https://www.tutorialspoint.com/es/software_engineering/software_development_life_cycle.htm