# Entornos

Software es el conjunto de programas informáticos que actúan sobre el

Hardware para ejecutar lo que el usuario desee.

tres tipos de software:

-Sistemas: software base que sirve a que las aplicaciones puedan ejecutarse y funcionar. Ej.

sistemas operativos, controladores, drivers.

- Software de programación: conjunto de herramientas que nos permiten

desarrollar programas informáticos. Editores, compiladores o depuradores

etc. Sin embargo lo más habitual es usar un tipo de software que integre

todas estas herramientas→IDE ENTORNOS DE DESARROLLO INTEGRADOS.

- Aplicaciones informáticas: son un conjunto de programas que tienen una

finalidad más o menos concreta. Son ejemplos de aplicaciones: un procesador

de textos, una hoja de cálculo, el software para reproducir música, un

videojuego, etc.

**conceptos**

**fundamentales de software y las fases del llamado ciclo de vida de una**

**aplicación informática.**

Hardware: conjunto de dispositivos físicos que conforman un ordenador.

Hardware:

-CPU-(cerebro)

-MEMORIA PRINCIPAL O RAM

-UNIDAD DE ENTRADA/SALIDA

Esta relación software-hardware se puede ver desde dos puntos de vista:

● Desde el punto de vista del sistema operativo (SO)

Todas las aplicaciones necesitan recursos hardware durante su ejecución

Será siempre el sistema operativo el encargado de controlar todos estos

aspectos de manera "oculta" para las aplicaciones (y para el usuario).

● Desde el punto de vista de las aplicaciones

Aplicación: conjunto de programas, escritos en algún lenguaje de

Programación lenguajes de programación diferentes, todos ellos con

algo en común, estar escritos con sentencias de un idioma que el ser

humano puede aprender y usar fácilmente. el hardware de un ordenador sólo es capaz de interpretar señales eléctricas.

¿Cómo será capaz el ordenador de "entender" algo escrito en un lenguaje que no es el suyo?

un proceso de traducción de código

Desarrollo del software: Todo el proceso que ocurre desde que se concibe una

idea hasta que un programa está implementado en el ordenador y funcionando.

Ciclo de vida del software: Describe el desarrollo de software desde la fase

inicial hasta la fase final, proponiendo etapas que sirven como referencia para

realizar este proceso.

Modelo en cascada

pequeños desarrollos, ya que las etapas pasan de una a

otra sin retorno posible.

**Ciclos de vida del software**

Modelo en cascada con retroalimentación

Es uno de los modelos más utilizados. Proviene del modelo anterior,

pero se introduce una realimentación entre etapas, de forma que

podamos volver atrás en cualquier momento para corregir, modificar o

depurar algún aspecto. No obstante, si se prevén muchos cambios

durante el desarrollo no es el modelo más idóneo.

Es el modelo perfecto si el proyecto es rígido (pocos cambios, poco

evolutivo) y los requisitos están claros.

Ventajas: fácil de comprender, de planificar y de seguir, y existen

herramientas que lo soportan.

Inconvenientes: los cambios generan confusiones conforme el

proyecto avanza, es difícil para el cliente especificar todos los requisitos

al principio, cuanto más tarde se detecte un fallo más costoso será

hacer los cambios que implique (rehacer todas las fases). Cliente tendrá

que tener paciencia, no habrá una versión funcional hasta que esté muy

avanzado el proyecto. Si descubrimos un fallo al final= DESASTRE.

**3 Modelos de proceso incremental**

Para resolver el problema del tiempo del modelo en cascada, podemos

ir haciendo entregas con una funcionalidad parcial del software al

cliente y aumentarla en entregas posteriores. De esta forma se concibe

el desarrollo de software como un proceso en el que se van

produciendo diversos incrementos o entregables.

**Modelo iterativo incremental**

Es decir se realiza el modelo anterior (cascada) eliminando la

retroalimentación y haciendo los cambios necesarios en cada

incremento y entrega. Las fases se repiten y refinan, y van

propagando su mejora a las fases siguientes

4. Modelos evolutivos

Son más modernos que los anteriores. Tienen en cuenta la

naturaleza cambiante y evolutiva del software. (cambio en los

requerimientos).

Modelos iterativos, desarrollan versiones cada vez más

completas del software.

**4.1 Construcción de prototipos**

En la práctica, la construcción de prototipos ayuda al

ingeniero de sistemas y al cliente a entender de mejor

manera cuál será el resultado de la construcción cuando

los requisitos estén satisfechos.

Desventajas

El cliente considera la mayoría de las veces al prototipo

como el producto final.

La calidad del software o la factibilidad de

mantenimiento puede que no se tomen en cuenta.

Es conveniente seguir este modelo cuando el cliente es

capaz d definir un conjunto de objetivos generales para

el software pero no puede identificar los requerimientos

en detalle o cuando los desarrolladores tienen dudas

considerables.

**4.2 Modelo en espiral**

Es una combinación del modelo anterior con el modelo en cascada

añadiendo un nuevo elemento, el análisis de riesgo. En él, el software se

va construyendo repetidamente en forma de versiones que son cada

vez mejores, debido a que incrementan la funcionalidad en cada

versión. Es un modelo bastante complejo.

Cuando empieza este proceso evolutivo, el equipo de ingeniería del software gira

alrededor de la espiral en la dirección de las agujas del reloj, comenzando por el

centro. El primer circuito de la espiral puede producir el desarrollo de una

especificación de productos; los pasos siguientes en la espiral se podrían utilizar

para desarrollar un prototipo y progresivamente versiones más sofisticadas del

software. Cada paso por la región de planificación produce ajustes en el plan del

proyecto.

**4.- FASES EN EL DESARROLLO DEL SOFTWARE.**

Hemos visto en puntos anteriores que debemos elegir un modelo de ciclo de vida para

el desarrollo de nuestro software. Independientemente del modelo elegido, siempre

hay una serie de etapas que debemos seguir para construir software fiable y de

calidad.

Estas etapas son:

• Análisis de requisitos. Se especifican los requisitos funcionales y no funcionales del

sistema.

• Diseño. Se divide el sistema en partes y se determina la función de cada una.

• Codificación. Se elige un lenguaje de programación y se codifican los programas.

• Pruebas. Se prueban los programas para detectar errores y se depuran.

• Documentación. De todas las etapas, se documenta y guarda toda la información.

• Explotación. Instalamos, configuramos y probamos la aplicación en los equipos del

cliente.

• Mantenimiento. Se mantiene el contacto con el cliente para actualizar y modificar

la aplicación el futuro.

• Autoevaluación

**4.1. ANÁLISIS.**

Es la primera etapa del proyecto, la más complicada y la que más depende de la

capacidad del analista. En ella se especifican y analizan los requisitos funcionales y no

funcionales del sistema.

¿Cómo se hace?

Mediante:

• entrevistas

• dinámicas de grupos estructuradas (JAD, Joint Application Development)

• planificación conjunta de requisitos (dinámicas JAD dirigidas a la alta dirección)

• Brainstorming

• Prototipos (versiones iniciales del sistema)

• Casos de uso. Técnica UML (representación de escenarios que describan lo que hace

el sistema, no cómo lo hace)

Se especifican dos tipos de Requisitos:

Funcionales, que detallan:

• Qué funciones tendrá que realizar la aplicación.

• Qué respuesta dará la aplicación ante todas las entradas.

• Cómo se comportará la aplicación en situaciones inesperadas.

• ...

No funcionales, que tratan sobre las características del sistema:

• Tiempos de respuesta del programa,

• Legislación aplicable,

• Fiabilidad

• Mantenibilidad

• Sistema operativo

• Restricciones

**TÉCNICAS DE REPRESENTACIÓN DE LOS REQUISITOS**

• Diagramas de flujo de datos, DFD: representa el flujo de datos entre los

diferentes procesos, entidades externas y almacenes que forman el sistema.

• Diagramas de flujo de control, DFC: similar pero muestra el flujo de control en

lugar de los datos (es decir el orden en que se realizan los procesos)

• Diagramas de transición de estados, DTE: representa cómo se comporta el

sistema como consecuencia de sucesos externos.

• Diagrama Entidad / Relación, DER: representa los datos y la forma en que se

relacionan entre ellos (recordar lo visto en Bases de Datos)

• Diccionario de datos, DD: es una descripción detallada de los datos utilizados

por el sistema que gráficamente se encuentran representados en el DFD.

Todo lo recogido en esta fase queda

reflejado en el documento

Especificación de Requisitos del

Software (ERS)

**4.2. DISEÑO.**

Durante esta fase, donde ya sabemos lo que hay que hacer, el siguiente paso es

¿Cómo hacerlo?

Tenemos que traducir los requisitos en una representación de software.

Hay dos tipos de diseño:

• Estructurado: basado en el flujo de datos a través del sistema

• Orientado a objetos: el sistema se entiende como un conjunto de objetos que

tienen propiedades y comportamientos, existen eventos que activan

operaciones que modifican el estado de dichos objetos y los objetos

interactúan con otros.

-Diseño procedimental

-Diseño de interfaz

-Diseño arquitectónico

-Diseño de datos

• Diseño de datos: Está basado en los datos y las relaciones, definidos en el diagrama entidad relación y en

la descripción detallada de los datos.

• Diseño arquitectónico: Representa la estructura de los componentes del software, sus propiedades e

interacciones.

• Diseño de la interfaz: Describe como se comunica el software consigo mismo, con los sistemas que operan

con él y con las personas que lo utilizan. Resultado: creación de formatos de pantalla.

• Diseño a nivel de componentes o diseño procedimental:

El diseño a nivel de componentes se presenta a menudo después que se ha

terminado la primera iteración del diseño arquitectónico y el objetivo de esta

fase es traducir el diseño en software operacional.

El diseño a nivel de componentes define las estructuras de datos, los algoritmos,

las características de la interfaz y los mecanismos de comunicación asignados a

cada componente de software. esta fase permite revisar si los detalles de diseño

son correctos y consistentes con las representaciones iniciales de diseño

Construcciones lógicas para diseñar cualquier programa:

Secuencial: Implementa los pasos del proceso esenciales para la especificación

de cualquier algoritmo.

Condicional: Permite seleccionar un proceso u otro a partir de la evaluación de

una condición lógica. Si se cumple, realiza la tarea «Parte Si» y si no se cumple

realiza la tarea «Parte No»

Selección múltiple, es una extensión de la anterior. Decisiones sucesivas hasta que

ocurre una condición verdadera

Repetitiva proporciona los bucles

• «Repetir-hasta»: primero ejecuta una tarea del bucle y después comprueba la

condición, si no cumple se vuelve a realizar la tarea, si cumple, finaliza el bucle.

• «Hacer-mientras»: primero se comprueba la condición y después se realiza la tarea

del bucle repetidamente siempre y cuando la condición se cumpla; el bucle finaliza

cuando la condición no se cumple.

**4.2.1 NOTACIONES GRÁFICAS PARA EL DISEÑO**

Diagrama de flujo: es una herramienta muy usada para el diseño

procedimental. Se utilizan estos símbolos:

-Una caja indica un paso del proceso.

-Un rombo representa una condición lógica.

-Las flechas indican el flujo de control.

Diagrama de cajas

Tablas de decisión: permiten

representar en tablas una

combinación compleja de

condiciones y las acciones que

llevan a cabo combinando estas

condiciones:

• Hacer una lista de todas las acciones y condiciones

• Asociar conjuntos de condiciones con acciones específicas

• Determinar las reglas indicando las acciones que ocurren para un determinado

número de condiciones.

Pseudocódigo:

• Utiliza texto descriptivo para realizar el diseño de un algoritmo.

• A primera vista se parece a un lenguaje de programación.

• Depende mucho de quien lo escriba ya que no hay un estándar definido.

• No puede ser compilado.

4.- FASES EN EL DESARROLLO DEL SOFTWARE

4.2. DISEÑO.

4.2.1 NOTACIONES GRÁFICAS PARA EL DISEÑO

Ejemplo:

Inicio

Abrir fichero

Leer Registro del Fichero

Mientras no sea Fin de Fichero Hacer

Procesar Registro leído

Leer Registro del Fichero

Fin Mientras

Cerrar Fichero

Fin

**4.2.2 DISEÑO ORIENTADO A OBJETOS**

El diseño de software orientado a objetos es difícil. Hay que partir de un análisis

orientado a objetos en el que se definen todas las clases que son importantes para el

problema a resolver, las operaciones y los atributos asociados, las relaciones y

comportamientos y las comunicaciones entre clases.

Este diseño define 4 capas:

• Subsistema: se centra en el diseño de los subsistemas que implementan las

funciones principales del sistema

• Clases y objetos: Especifica la arquitectura de objetos global y jerarquía de clases.

• Mensajes: Indica cómo se realiza la colaboración entre los objetos

• Responsabilidades: Identifica las operaciones y atributos que caracterizan cada

clase.

Para este análisis y diseño se utiliza UML (Unified Modeling Language o Lenguaje de

Modelado Unificado)

Es un lenguaje de modelado basado en diagramas que sirve para expresar modelos. Es

el estándar utilizado por la mayor parte de las metodologías de desarrollo orientado a

objetos que existen hoy en día.

En los temas 5 y 6 veremos más sobre UML.

**4.3. CODIFICACIÓN.**

Una vez diseñado, se realiza el proceso de codificación. Esta tarea la realiza el programador y tiene que

cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.

El programador recibe las especificaciones de diseño y las transforma en un conjunto de instrucciones

escritas en un lenguaje de programación almacenadas dentro de un programa. A este conjunto de

instrucciones se le llama código fuente.

En el proyecto debe haber unas normas de codificación y estilo, claras y homogéneas. Estas normas facilitan

la corrección y mantenimiento de los programas, sobre todo cuando las realiza una persona que no los ha

desarrollado.

TIPOS DE CÓDIGO.

Las características deseables de todo código son:

• Modularidad: que esté dividido en trozos más pequeños.

• Corrección: que haga lo que se le pide realmente.

• Fácil de leer: para facilitar su desarrollo y mantenimiento futuro.

• Eficiencia: que haga un buen uso de los recursos.

• Portabilidad: que se pueda implementar en cualquier equipo.

Durante esta fase, el código pasa por diferentes estados:

Código Fuente:. Se escribe usando algún lenguaje de programación de alto nivel y

contiene el conjunto de instrucciones necesarias.

Código Objeto: es el código binario resultado de compilar el código fuente.

La compilación es la traducción de una sola vez del programa, y se realiza utilizando

un compilador. La interpretación es la traducción y ejecución simultánea del programa

línea a línea.

El código objeto no es directamente inteligible por el ser humano, pero tampoco por la

computadora. Es un código intermedio entre el código fuente y el ejecutable.

Código Ejecutable: Es el código binario resultante de enlazar los archivos de código

objeto con ciertas rutinas y bibliotecas necesarias. El sistema operativo será el

encargado de cargar el código ejecutable en memoria RAM y proceder a ejecutarlo.

También es conocido

**4.4.- PRUEBAS.**

En esta etapa ya tenemos el software y se trata de encontrar errores tanto de

codificación como de especificación o diseño.

Se realizan tareas de Verificación y Validación del software (V&V).

Verificación: son las actividades que tratan de comprobar si estamos diseñando

el producto correctamente, es decir si el software implementa correctamente

una función específica.

Validación: son las actividades que tratan de comprobar que el producto es

correcto es decir que se ajusta a los requisitos del cliente.

Objetivo de esta etapa: planificar y diseñar pruebas que detecten diferentes

clases de errores, con la menor cantidad de tiempo y esfuerzo posible.

Una prueba tiene éxito si descubre un error que no se había detectado

anteriormente.

Caso de prueba: documento que especifica los valores de entrada, salida esperada y las condiciones previas para la ejecución de la prueba.

Prueba de caja blanca, se centra en

validar la estructura interna del

programa (necesita conocer los detalles

procedimentales del código)

Pruebas de caja negra, se centran en

validar los requisitos funcionales sin

fijarse en el funcionamiento interno del

programa (necesitan saber la

funcionalidad que el código ha de

proporcionar)

No son excluyentes, pueden combinarse

ambas.

**4.5.- DOCUMENTACIÓN.**

Todas las etapas del desarrollo deben estar documentadas y es en esta etapa

donde tenemos que reunir todos los documentos generados, clasificarlos

según el nivel técnico de sus descripciones.

Los documentos:

• Actuaran como medio de comunicación entre los miembros del equipo

de desarrollo

• Serán un repositorio de información del sistema (mantenimiento)

• Proporcionarán información que ayude a planificar la gestión del

presupuesto y programar el proceso del desarrollo del software

• Algunos documentos deben indicar a los usuarios como utilizar y

administrar el sistema (manual de usuario)

Se divide en dos clases:

Documentación del proceso: registra el proceso de desarrollo y

mantenimiento. Los documentos indican planes, estimaciones y horarios que

predicen y controlan el proceso de software, informan sobre como usar los

recursos durante el proceso de desarrollo y las normas de cómo se ha de

implementar el proceso.

(se usa para gestionar todo el proceso de desarrollo del software)

Documentación del producto: describe el producto que está siendo

desarrollado. (se utiliza una vez que el sistema está en funcionamiento)

• Documentación del sistema punto de vista técnico (mantenimiento)

• Documentación del usuario, orientada a los futuros usuarios del

sistema.

**4.5.- DOCUMENTACIÓN. DOCUMENTACIÓN DEL USUARIO**

Hay diferentes tipos de usuarios:

Usuarios finales: utilizan el software para realizar alguna tarea, necesitan conocer

cómo el software les ayuda a realizarla no están interesados en los detalles

informáticos ni del programa.

Administradores del sistema: son responsables de la gestión de los programas

informáticos utilizados por los usuarios finales (gestores de red, técnicos, etc.)

Para atender a los distintos tipos de usuarios con diferentes niveles de experiencia

se definen una serie de documentos que deberán ser entregados con el sistema

de software.

Incluye todos los documentos que describen el sistema, desde la especificación de

requisitos hasta las pruebas de aceptación. Los documentos que describen el diseño,

la implementación y las pruebas del sistema son esenciales para entender y mantener

el software.

Debe incluir:

• Fundamentos del sistema (objetivos)

• Análisis y especificación de requisitos (ERS)

• Diseño (describe la arquitectura del sistema)

• Implementación (describe la forma en que se expresa el sistema en el lenguaje de

programación usado y las acciones en forma de comentarios dentro del programa)

• Plan de pruebas del sistema (describe cómo las ud. De programa son evaluadas

individualmente y todo el sistema se prueba tras la integración)

• Plan de pruebas de aceptación (describe las pruebas que ha de pasar el sistema

antes de que los usuarios las acepten)

• Diccionarios de datos (contiene la descripción de todos los términos que se

relacionan con el sistema de software en cuestión)

• La documentación ha de estar organizada en capítulos, secciones y

subsecciones.

• Tiene que ser legible y usable.

• Todos los documentos han de tener portada (nombre del proyecto, tipo de

documento, autor, fecha creación, versión, revisores, destinatarios y y clase de

confidencialidad del documento)

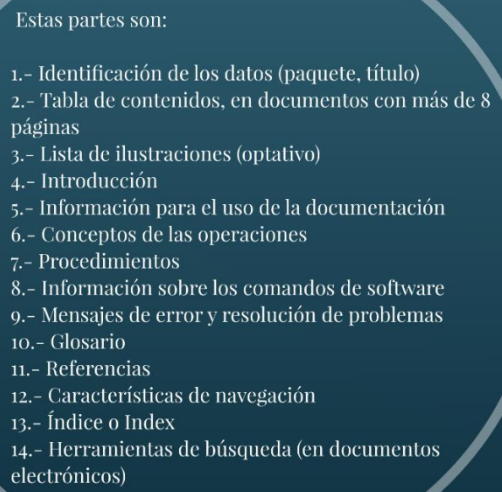
• Debe contener un índice. Y tiene que definirse un esquema de numeración

coherente (capítulo-página)

• Al final debe haber un glosario de términos ya que el documento irá dirigido a

un amplio espectro de lectores.

El Estandar (IEEE, 2001) Propone la siguiente estructura para los documentos.



**4.6.- EXPLOTACIÓN**

En esta etapa se lleva a cabo la instalación y puesta en marcha del producto

software en el entorno de trabajo del cliente.

Tareas:

• Definir la estrategia para la implementación del proceso. Normas y

procedimientos para tratar los problemas que surjan e ir probando el programa.

• Pruebas de operación para cada release (gestión de entregas) del producto

software se efectuarán pruebas de funcionamiento.

• Uso operacional del sistema. El sistema entrará en acción en el entorno previsto

de acuerdo con la documentación de usuario.

• Soporte al usuario. Se debe proporcionar asistencia y consultoría a los usuarios

cuando la soliciten. Registrada y supervisada.

**4.7.- MANTENIMIENTO**

El mantenimiento software consiste en la modificación de este producto de

software con objeto de corregir fallos mejorar el rendimiento u otros atributos o

adaptar el producto a un entorno modificado.

Mantenimiento adaptativo. Si se requiere cambiar el entorno de uso de la

aplicación (que incluye al sistema operativo, a la plataforma de hardware o, en el

caso de las aplicaciones web, al navegador), puede ser indispensable modificarla

para mantener su plena funcionalidad en estas nuevas condiciones. Es el más usual

debido a los rápidos cambios de la informática.

Mantenimiento correctivo. Corrige los defectos encontrados en el software, y que

originan un comportamiento distinto al deseado.

Mantenimiento perfectivo. Por distintas razones, el usuario puede solicitar

el agregado de nuevas funcionalidades o características no contempladas al

momento de la implementación del software. El mantenimiento perfectivo

adapta la aplicación a este requerimiento.

Mantenimiento preventivo: Modificación del producto de software sin

alterar las especificaciones del mismo con el fin de mejorar las tareas de

mantenimiento. Ej. restructuración de programas para mejorar su legibilidad

o añadir comentarios para mejorar la comprensión.



Tareas en el mantenimiento del software

• Implementación del proceso. Establecer planes y procedimientos, ejecutarlos,

registrarlo todo e informar a los usuarios sobre la situación

• Análisis de problemas y modificaciones. Realizar el análisis teniendo en cuenta

todo, el impacto en la organización, el sistema existente, los sistemas con los que

interacciona. Verificar el problema, desarrollar opciones para implementar la

modificación y documentarlo todo.

• Implementación de las modificaciones. Analizar y determinar que documentación

y unidades de software y versiones se deben modificar. Documentarlo todo

• Revisión/aceptación del mantenimiento. Revisiones con la organización que

autorice las modificaciones para determinar la integridad del sistema modificado.

Deberá obtener aprobación.

• Migración. Preparar, documentar y ejecutar un plan de migración. (Incluir a los

usuarios, notificar los palanes y las actividades)

• Retirada del software. Se retirará por petición del propietario. Crear un plan de

retirada, notificar a los usuarios. La operación de retirada e implantación del nuevo

software deberá hacerse en paralelo. Durante este periodo se deberá formar a los

usuarios.