Tema 1: Desarrollo de software

Entornos de desarrollo
1º Desarrollo de Aplicaciones Multiplataforma

1 Software y programa. Tipos de software

Dos partes del ordenador: **hardware** y **software**

Software

- conjunto de programas informáticos
- actúan sobre el hardware
- ejecutar lo que el usuario desee
- tres tipos de software
  - sistema operativo
  - software de programación
  - aplicaciones

1 Software y programa. Tipos de software

Sistema operativo

- software base
- ha de estar instalado y configurado en el ordenador
- para que las aplicaciones puedan ejecutarse y funcionar
- Windows, Linux, Mac OS X …

1 Software y programa. Tipos de software

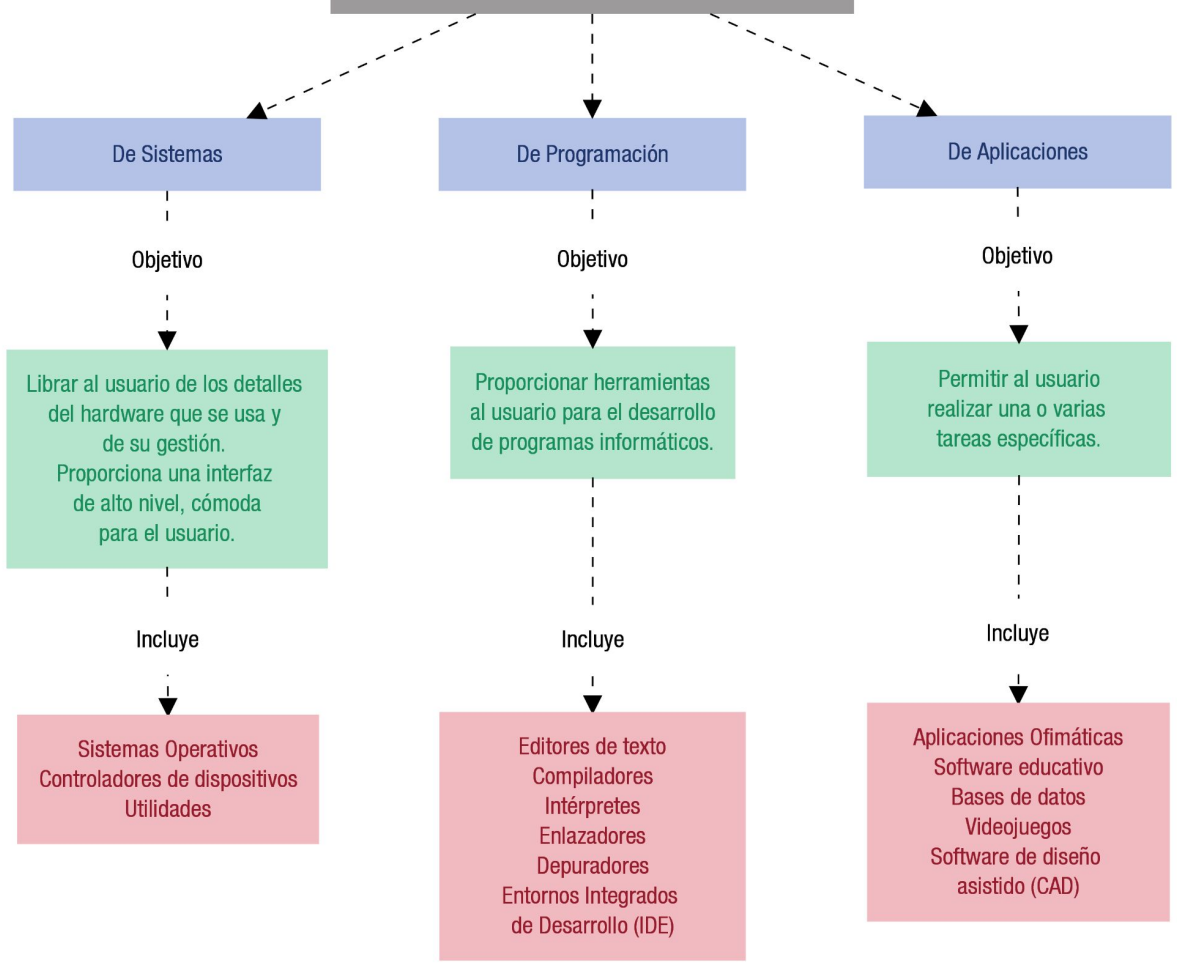
Software de programación

- conjunto de herramientas que nos permiten desarrollar programas informáticos

Aplicaciones informáticas

- conjunto de programas que tienen una finalidad más o menos concreta.
- un procesador de textos, una hoja de cálculo, el software para reproducir música, un videojuego, etc.

TIPOS DE SOFTWARE



2 Relación hardware-software

Relación indisoluble entre hardware y software, necesitan estar instalados y configurados correctamente para que el equipo funcione

El software se ejecutará sobre los dispositivos físicos

Dos puntos de vista

- Sistema operativo
- Aplicaciones

2 Relación hardware-software

Desde el punto de vista del sistema operativo

- encargado de coordinar al hardware durante el funcionamiento del ordenador
- intermediario con las aplicaciones
- recursos hardware durante su ejecución
    - tiempo de CPU, espacio en memoria RAM, gestión de los dispositivos de Entrada/Salida, etc.

2 Relación hardware-software

Desde el punto de vista de las aplicaciones

- escritos en algún lenguaje que el hardware del equipo debe interpretar y ejecutar
- multitud de lenguajes de programación diferentes escritos con sentencias de un idioma que el ser humano puede aprender y usar
- hardware de un ordenador sólo es capaz de interpretar señales eléctricas 0 y 1 (código binario)
- proceso para esta traducción

2 Relación hardware-software

Para fabricar un programa informático que se ejecuta en una computadora:

A. Hay que escribir las instrucciones en código binario para que las entienda el hardware .
B. Sólo es necesario escribir el programa en algún lenguaje de programación y se ejecuta directamente.
C. Hay que escribir el programa en algún Lenguaje de Programación y contar con herramientas software que lo traduzcan a código binario.
D. Los programas informáticos no se pueden escribir: forman parte de los sistemas operativos.

2 Relación hardware-software

Para fabricar un programa informático que se ejecuta en una computadora:

A. Hay que escribir las instrucciones en código binario para que las entienda el hardware .
B. Sólo es necesario escribir el programa en algún lenguaje de programación y se ejecuta directamente.
C. Hay que escribir el programa en algún Lenguaje de Programación y contar con herramientas software que lo traduzcan a código binario.
D. Los programas informáticos no se pueden escribir: forman parte de los sistemas operativos.

3 Desarrollo de software

Proceso que ocurre desde que se concibe una idea hasta que un programa está implementado en el ordenador y funcionando

Serie de pasos de obligado cumplimientopara garantizar la eficiencia, fiabilidad, seguridad y que respondan a las necesidades de los usuarios finales

Etapas

3 Desarrollo de software

Según estimaciones, el 26% de los grandes proyectos de software fracasan, el 48% deben modificarse drásticamente y sólo el 26% tienen rotundo éxito.

La principal causa del fracaso de un proyecto es la falta de una buena planificación de las etapas y mala gestión de los pasos a seguir.

¿Por qué el porcentaje de fracaso es tan grande?

¿Por qué piensas que estas causas son tan determinantes?

3 Desarrollo de software

Porque los errores en la planificación inicial se propagarán en cascada al resto de etapas del desarrollo.

Según estimaciones, el 26% de los grandes proyectos de software fracasan, el 48% deben modificarse drásticamente y sólo el 26% tienen rotundo éxito.

La principal causa del fracaso de un proyecto es la falta de una buena planificación de las etapas y mala gestión de los pasos a seguir.

¿Por qué el porcentaje de fracaso es tan grande?

¿Por qué piensas que estas causas son tan determinantes?

3.1 Ciclos de vida del software

Pasos a seguir para desarrollar un programa

3.1 Ciclos de vida del software

Modelo en Cascada

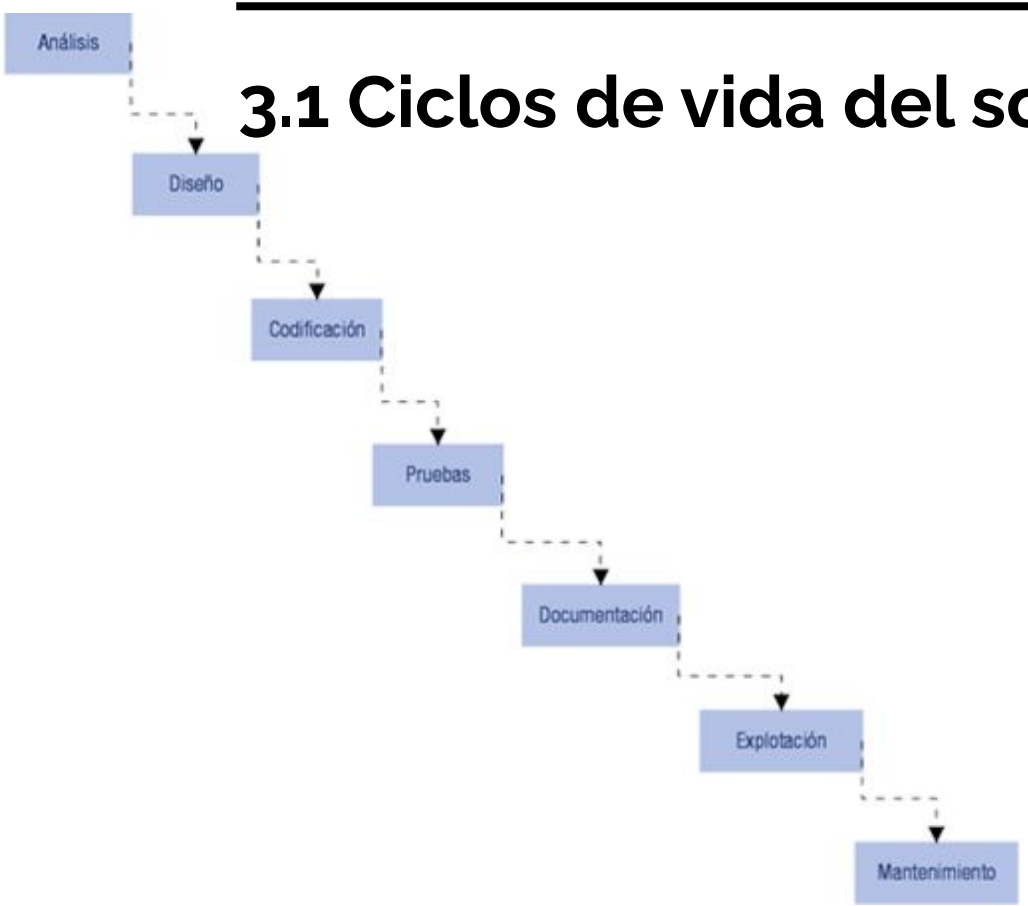
Modelo de vida clásico del software

Se requiere conocer de antemano todos los requisitos del sistema

Aplicable a pequeños desarrollos, ya que las etapas pasan de una a otra sin retorno posible

Se presupone que no habrá errores ni variaciones del software

3.1 Ciclos de vida del software



3.1 Ciclos de vida del software

Modelo en Cascada con Realimentación

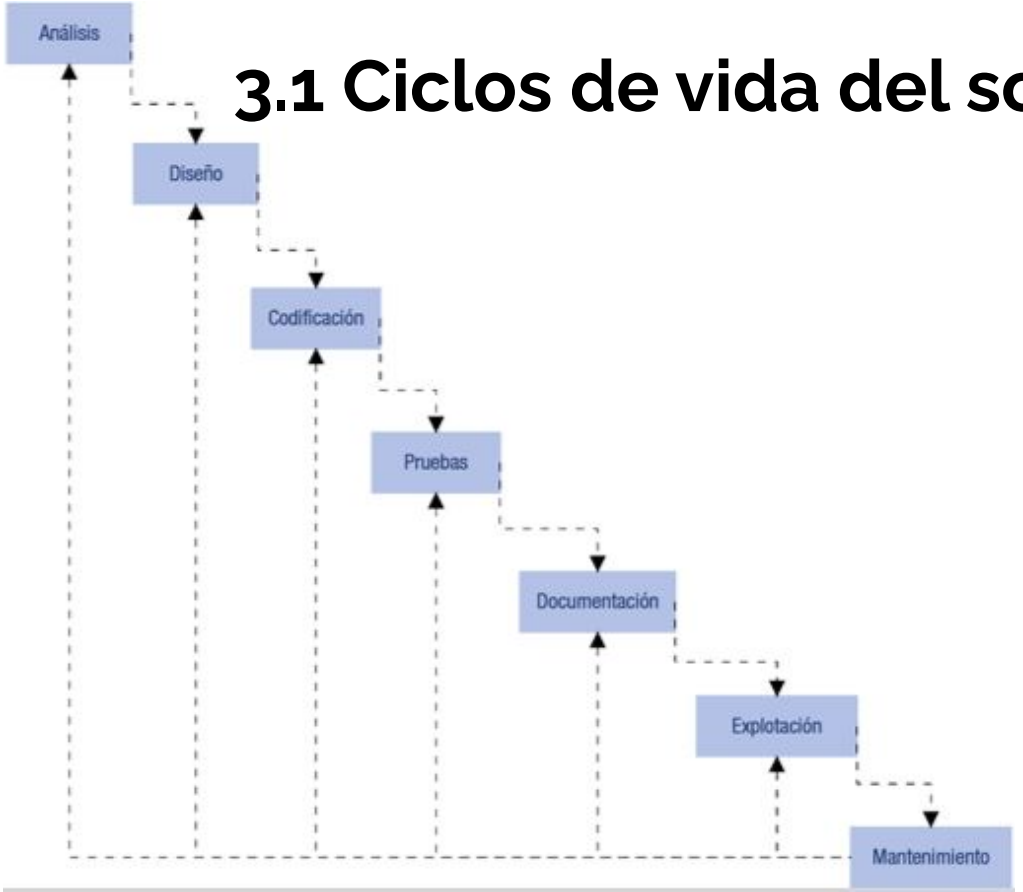
Realimentación entre etapas, podemos volver atrás

Corregir, modificar o depurar aspectos

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

3.1 Ciclos de vida del software



3.1 Ciclos de vida del software

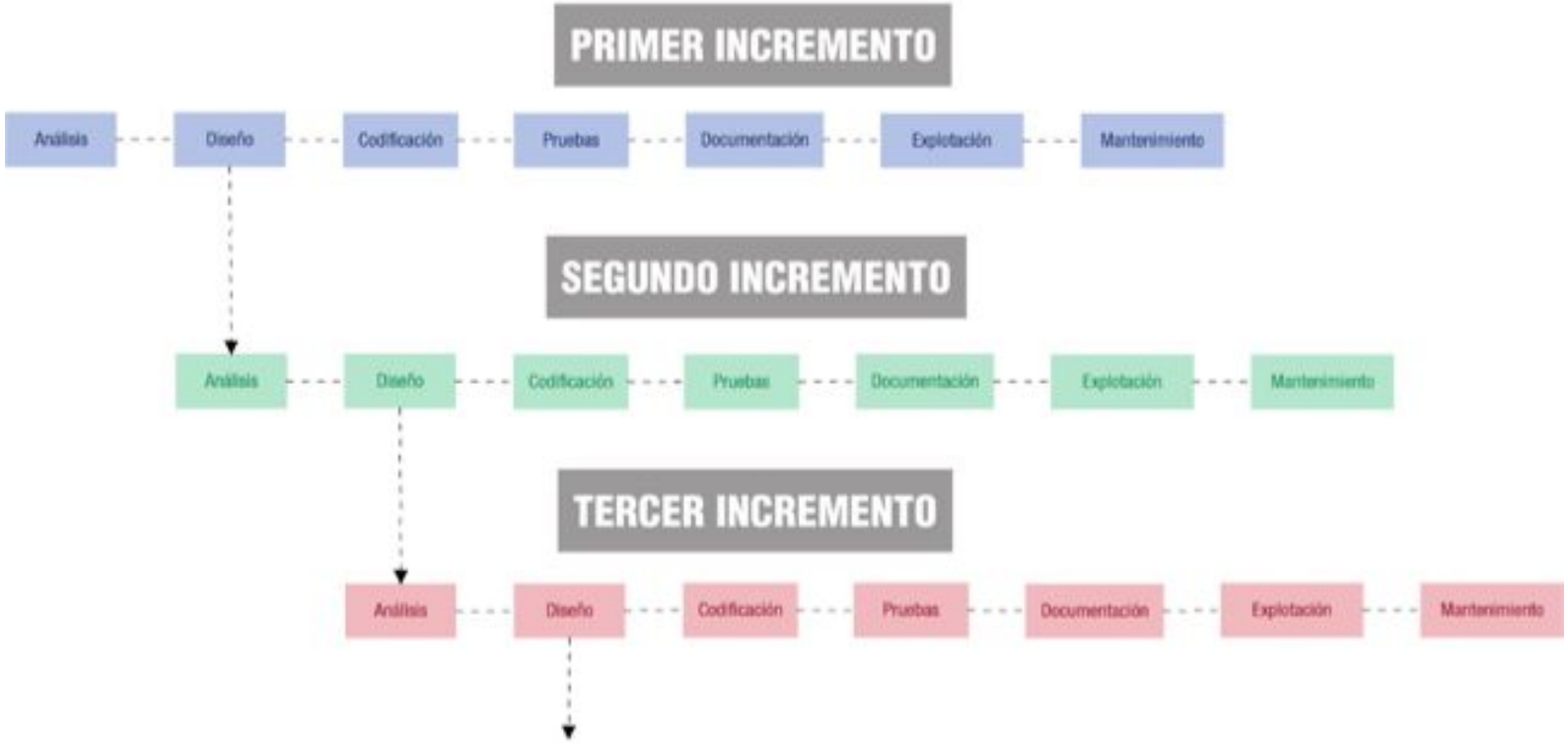
Modelo Evolutivo: Iterativo Incremental

Tienen en cuenta la naturaleza cambiante y evolutiva del software

Versiones cada vez más completas

Fases se repiten y refinan, y van propagando su mejora a las fases siguientes

3.1 Ciclos de vida del software



3.1 Ciclos de vida del software

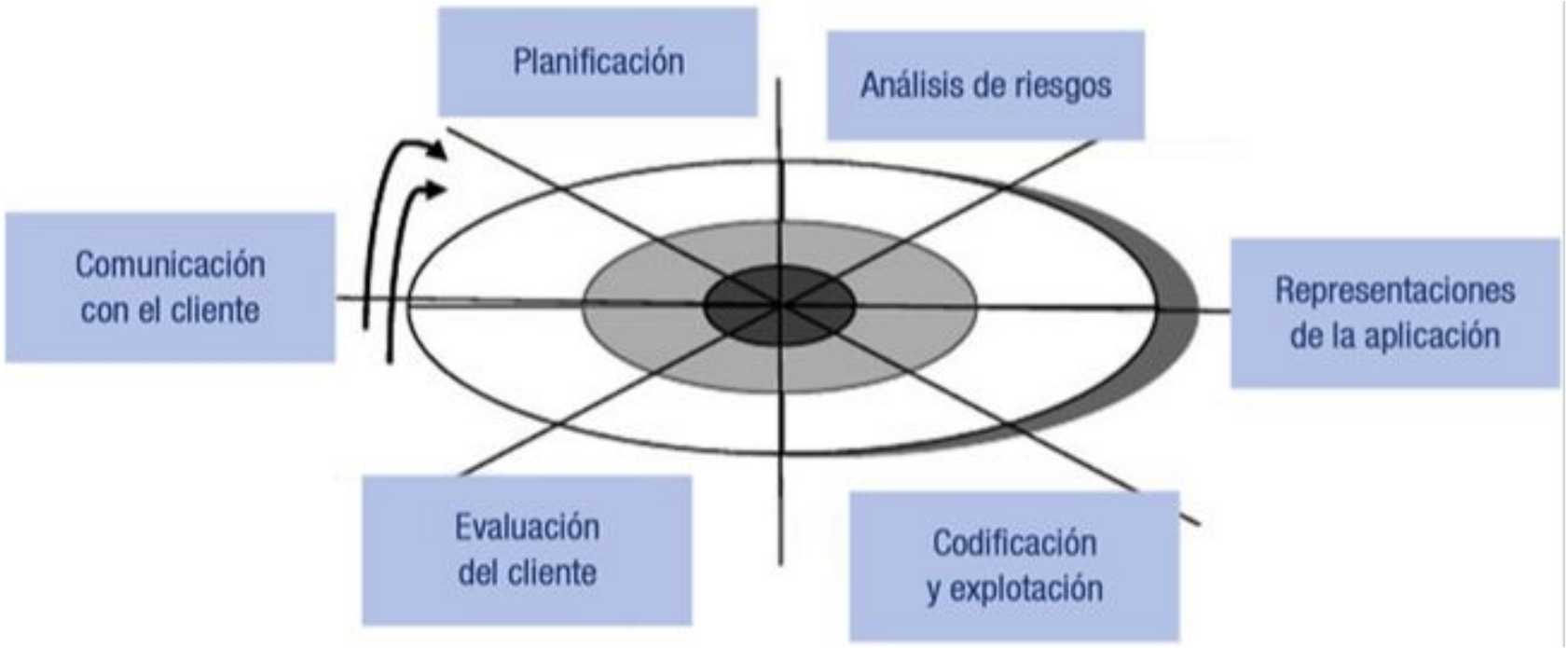
Modelo Evolutivo: en Espiral

6 regiones de tareas: Comunicación con el cliente, Planificación, Análisis de riesgos, Representación de la aplicación, Codificación y explotación y Evaluación del cliente

Se adapta completamente a la naturaleza evolutiva del software

Reduce los riesgos antes de que sean problemáticos

3.1 Ciclos de vida del software



3.1 Ciclos de vida del software

Si queremos construir una aplicación pequeña, y se prevé que no sufrirá grandes cambios durante su vida, ¿sería el modelo de ciclo de vida en espiral el más recomendable?

A. Sí
B. No

3.1 Ciclos de vida del software

Si queremos construir una aplicación pequeña, y se prevé que no sufrirá grandes cambios durante su vida, ¿sería el modelo de ciclo de vida en espiral el más recomendable?

A. Sí
B. No

La aplicación no sufrirá grandes cambios

3.2 Herramientas de apoyo al desarrollo del software

Para llevar a cabo varias de las etapas vistas en el punto anterior contamos con **herramientas informáticas**

Finalidad principal es **automatizar las tareas** y ganar **fiabilidad** y **tiempo**

Esto nos va a permitir centrarnos en los **requerimientos del sistema** y el **análisis** del mismo, que son las causas principales de los **fallos del software**

3.2 Herramientas de apoyo al desarrollo del software

Herramientas CASE (Computer Aided Software Engineering)

Conjunto de aplicaciones que se utilizan en el desarrollo de software

Objetivo de reducir costes y tiempo del proceso

Mejorando la productividad del proceso

3.2 Herramientas de apoyo al desarrollo del software

Herramientas CASE (Computer Aided Software Engineering)

¿En qué fases del proceso nos pueden ayudar?

En el diseño del proyecto, en la codificación de nuestro diseño a partir de su apariencia visual, detección de errores…

3.2 Herramientas de apoyo al desarrollo del software

Herramientas CASE (Computer Aided Software Engineering)

Permiten:

- Mejorar la planificación del proyecto
- Darle agilidad al proceso
- Poder reutilizar partes del software en proyectos futuros
- Hacer que las aplicaciones respondan a estándares
- Mejorar la tarea del mantenimiento de los programas
- Mejorar el proceso de desarrollo, al permitir visualizar las fases de forma gráfica

3.2 Herramientas de apoyo al desarrollo del software

Herramientas CASE (Computer Aided Software Engineering)

Clasificación (en función de las fases del ciclo de vida del software en la que ofrecen ayuda):

- U-CASE: fases de planificación y análisis de requisitos
- M-CASE: análisis y diseño.
- L-CASE: programación del software, detección de errores del código, depuración de programas y pruebas y en la generación de la documentación

3.2 Herramientas de apoyo al desarrollo del software

Herramientas CASE (Computer Aided Software Engineering)

Ejercicio de investigación

- Historia
- Otras clasificaciones
- Diferencias entre unas herramientas CASE y otras
- Ejemplos

4 Lenguajes de programación

Programas informáticos: escritos usando algún lenguaje de programación

Lenguaje de Programación

- **idioma** creado de forma artificial
- formado por un conjunto de **símbolos** y **normas** que se aplican sobre un **alfabeto** para obtener un **código**
- el **hardware** de la computadora pueda **entender** y **ejecutar**

4 Lenguajes de programación

Los lenguajes de programación son los que nos permiten comunicarnos con el hardware del ordenador

4 Lenguajes de programación

Tener clara la **función** de los lenguajes de programación

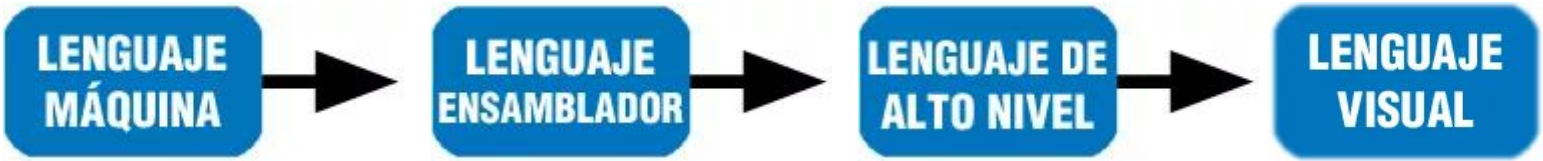
Multitud de lenguajes de programación, cada uno con unos símbolos y unas estructuras diferentes

Cada lenguaje está **enfocado** a la programación de tareas o áreas determinadas

Elección del lenguaje a utilizar en un proyecto

4 Lenguajes de programación

Evolución



4 Lenguajes de programación

Lenguaje máquina

- Sus instrucciones son combinaciones de **unos y ceros**
- Es el único lenguaje que **entiende** directamente el **ordenador** (no necesita traducción)
- Fue el **primer lenguaje** utilizado
- Es **único para cada procesador** (no es portable de un equipo a otro)
- Hoy día nadie programa en este lenguaje

4 Lenguajes de programación

Lenguaje ensamblador (bajo nivel)

- Sustituyó al lenguaje máquina para **facilitar** la labor de programación
- En lugar de unos y ceros se programa usando **mnemotécnicos** (instrucciones complejas)
- Necesita **traducción** al lenguaje máquina para poder ejecutarse
- Sus instrucciones son sentencias que hacen referencia a la ubicación física de los archivos en el equipo
- Difícil de utilizar

4 Lenguajes de programación

Lenguaje de alto nivel

- Sustituyeron al lenguaje ensamblador para **facilitar** más la labor de programación
- En lugar de mnemotécnicos, se utilizan s**entencias y órdenes** derivadas del idioma **inglés** (necesita traducción al lenguaje máquina)
- Son más cercanos al **razonamiento humano**
- Son utilizados hoy día

4 Lenguajes de programación

Lenguajes visuales:

- En lugar de sentencias escritas, se programa **gráficamente** usando el ratón y diseñando directamente la apariencia del software
- Su correspondiente código se **genera automáticamente**
- Necesitan traducción al lenguaje máquina
- Son completamente portables de un equipo a otro

4 Lenguajes de programación

Lenguaje máquina

- Secuencia de números (1 y 0)
- Lenguaje propio de cada computadora

Lenguaje ensamblador (bajo nivel)

- Códigos parecidos al inglés
- Varía con el tipo de procesador
- Ejemplo: LOAD x, ADD y

C, C++

Lenguaje de alto nivel

- Instrucciones en lenguaje familiar
- Notaciones matemáticas conocidas: X + Y
- Independiente de la máquina

Java, C#, Python, JavaScript

4.1 Concepto y características

Concepto

Conjunto de:

- **Alfabeto**: símbolos permitidos
- **Sintaxis**: normas de construcción de los símbolos del lenguaje
- **Semántica**: significado de las construcciones para hacer acciones válidas

4.1 Concepto y características

Características

Según lo cerca que esté del lenguaje humano

- **Lenguajes de Programación De alto nivel**: están más próximos al razonamiento humano
- **Lenguajes de Programación De bajo nivel**: están más próximos al funcionamiento interno de la computadora
  - Lenguaje Ensamblador
  - Lenguaje Máquina

4.1 Concepto y características

Características

Según la técnica de programación utilizada:

- **Lenguajes de Programación Estructurados**
  - Ejemplos: Pascal, C, etc.
- **Lenguajes de Programación Orientados a Objetos**
  - Ejemplos: C++, Java, Ada, Delphi, etc.
- **Lenguajes de Programación Visuales**: técnicas anteriores permitiendo programar gráficamente, código correspondiente generado de forma automática
  - Ejemplos: Visual Basic.Net, Borland Delphi, etc.

4.2 Lenguajes de programación estructurados

Escribir lenguajes de programación que permite sólo el uso de tres tipos de sentencias o estructuras de control

- **Sentencias secuenciales**: se da de forma natural en el lenguaje, sentencias se ejecutan en el orden en el que aparecen en el programa
- **Sentencias selectivas** (condicionales): variables booleanas (sí/no)
  - if, switch
- **Sentencias repetitivas** (iteraciones o bucles): se ejecuta mientras se de una condición
  - for y while

4.2 Lenguajes de programación estructurados

4.2 Lenguajes de programación estructurados

Ventajas

- Programas son **fáciles** de leer, sencillos y rápidos
- El **mantenimiento** de los programas es sencillo
- La **estructura** del programa es sencilla y clara

4.3 Lenguajes de programación orientados a objetos

Tratan a los programas no como un conjunto ordenado de instrucciones sino como un **conjunto de objetos** que colaboran entre ellos para realizar acciones

Los objetos son **reutilizables** para proyectos futuros

4.3 Lenguajes de programación orientados a objetos

Desventaja: no es una programación tan intuitiva

Alrededor del 55% del software que producen las empresas se hace usando esta técnica

- El código es **reutilizable**
- Si hay algún **error**, es más **fácil de localizar** y **depurar** en un objeto que en un programa entero

4.3 Lenguajes de programación orientados a objetos

Características:

- Los objetos tendrán una serie de **atributos** que los diferencian unos de otros
- Se define **clase** como una colección de objetos con características similares
- Mediante los llamados **métodos**, los objetos se comunican con otros produciéndose un cambio de estado de los mismos
- Los **objetos** son como unidades individuales e indivisibles que forman la base de este tipo de programación

Ejemplos: C++, JavaScript, Python, Java, etc.

5 Fases en el desarrollo y ejecución del software

Independientemente del modelo elegido, siempre hay una serie de etapas que debemos seguir para construir software fiable y de calidad

5 Fases en el desarrollo y ejecución del software

¿Crees que debemos esperar a tener completamente cerrada una etapa para pasar a la siguiente?

Sí.

No.

5 Fases en el desarrollo y ejecución del software

¿Crees que debemos esperar a tener completamente cerrada una etapa para pasar a la siguiente?

Sí

No

Hay que dejar siempre una "puerta abierta" para volver atrás e introducir modificaciones

5.1 Análisis

Primera etapa del proyecto, la más **complicada** y la que más depende de la **capacidad del analista**

Se **especifican y analizan los requisitos funcionales y no funcionales** del sistema

comunicación entre el analista y el cliente

5.1 Análisis

Requisitos:

- **Funcionales**
  - funciones tendrá que realizar la aplicación
  - respuesta que dará la aplicación ante todas las entradas
  - cómo se comportará la aplicación en situaciones inesperadas
- **No funcionales**
  - Tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.

5.1 Análisis

Documento ERS (**Especificación de Requisitos Software**)

- La **planificación** de las reuniones que van a tener lugar
- Relación de los **objetivos** del usuario cliente y del sistema
- Relación de los **requisitos** funcionales y no funcionales del sistema
- Relación de **objetivos** prioritarios y temporización
- Reconocimiento de **requisitos** mal planteados o que conllevan contradicciones

5.1 Análisis

Ejemplos requisitos funcionales

- Si desean que la lectura de los productos se realice mediante códigos de barras
- Si van a detallar las facturas de compra y de qué manera la desean
- Si los trabajadores de las tiendas trabajan a comisión, tener información de las ventas de cada uno
- Si van a operar con tarjetas de crédito
- Si desean un control del stock en almacén

5.1 Análisis

Ejemplos requisitos no funcionales

- El procedimiento de desarrollo de software a usar debe estar definido explícitamente (en manuales de procedimientos) y debe cumplir con los estándares ISO 9000
- El sistema debe ser desarrollado utilizando las herramientas CASE XYZ.
- Debe especificarse un plan de recuperación ante desastres para el sistema a ser desarrollado.

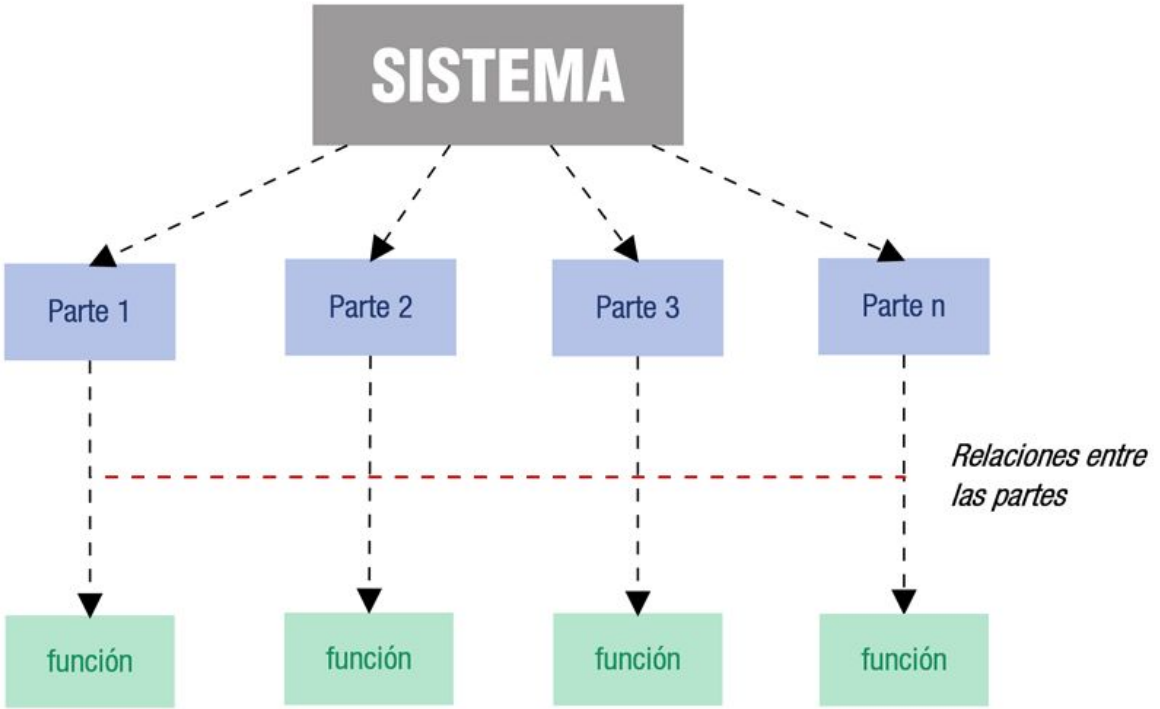
5.2 Diseño

Dividir el sistema en partes y establecer qué relaciones habrá entre ellas

Decidir **qué hará** exactamente cada parte

Crear un **modelo funcional-estructural** de los requerimientos del sistema global, para poder dividirlo y afrontar las partes por separado

5.2 Diseño


SISTEMA
Parte 1
Parte 2
Parte 3
Parte n
Relaciones entre las partes
función
función
función
función

5.2 Diseño

Decisiones, ejemplos:

- Entidades y relaciones de las bases de datos
- Selección del lenguaje de programación que se va a utilizar
- Selección del Sistema Gestor de Base de Datos

5.3 Codificación. Tipos de código

Proceso de programación

Elegir un lenguaje de programación, codificar toda la información anterior y llevarlo a código fuente

Esta tarea la realiza el programador y tiene que cumplir con todos los datos impuestos en el análisis y en el diseño de la aplicación

5.3 Codificación. Tipos de código

Las características del código

- **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

5.3 Codificación. Tipos de código

Código pasa por diferentes estados:

- **Código Fuente**
  - escrito por los programadores en algún editor de texto
  - se escribe usando algún lenguaje de programación de alto nivel
  - contiene el conjunto de instrucciones necesarias

5.3 Codificación. Tipos de código

Código pasa por diferentes estados:

- **Código Objeto**
  - código binario resultado de compilar el código fuente
  - no es directamente inteligible por el ser humano, pero tampoco por la computadora
  - es un código intermedio entre el código fuente y el ejecutable
  - sólo existe si el programa se compila, ya que si se interpreta se traduce y se ejecuta en un solo paso
    - **Compilación**: traducción de una sola vez del programa, se realiza utilizando un compilador
    - **Interpretación**: traducción y ejecución simultánea del programa línea a línea

5.3 Codificación. Tipos de código

Código pasa por diferentes estados:

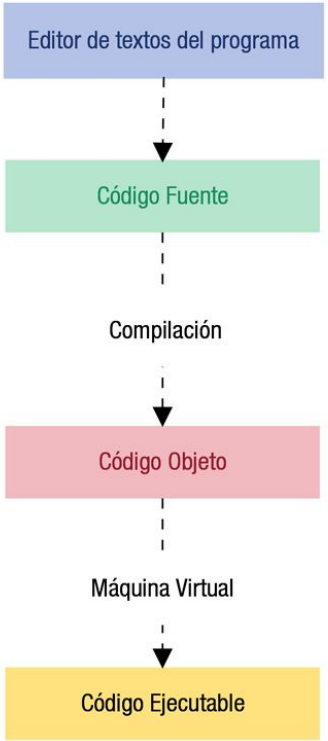
- **Código Ejecutable**
  - código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias
  - es directamente inteligible por la computadora
  - el sistema operativo será el encargado de cargar el código ejecutable
  - conocido como código máquina

5.3 Codificación. Tipos de código

Los **programas interpretados** no producen código objeto

- El paso de fuente a ejecutable es directo
  - Java
  - JavaScript
  - C#
  - Python
  - PHP

5.4 Fases en la obtención de código



5.4.1 Código fuente

Conjunto de instrucciones que la computadora deberá realizar, escritas por los programadores en algún **lenguaje de alto nivel**

No es directamente ejecutable por la máquina

Deberá ser **traducido** al lenguaje máquina, que la computadora será capaz de entender y ejecutar

5.4.1 Código fuente

Elaboración previa de un **algoritmo:** conjunto de **pasos a seguir** para obtener la solución del problema

Lo diseñamos en pseudocódigo

La codificación posterior a algún Lenguaje de Programación determinado será más rápida y directa

5.4.1 Código fuente

1. Se debe partir de las **etapas anteriores** de análisis y diseño
2. Se diseñará un **algoritmo** que simbolice los pasos a seguir para la resolución del problema
3. Se elegirá un **Lenguajes de Programación de alto nivel** apropiado para las características del software que se quiere codificar
4. Se procederá a la **codificación** del algoritmo antes diseñado

5.4.1 Código fuente

Obtenemos un **documento** con la codificación de todos los módulos, funciones, bibliotecas y procedimientos necesarios

Habrá que **traducirlo** a un código equivalente en código binario: **código objeto**

Dos tipos según su **licencia**:

- **Código fuente abierto**: está disponible para que cualquier usuario pueda estudiarlo, modificarlo o reutilizarlo
- **Código fuente cerrado**: no tenemos permiso para editarlo

5.4.1 Código fuente

Para obtener código fuente a partir de toda la información necesaria del problema:

A. Se elige el Lenguaje de Programación más adecuado y se codifica directamente.
B. Se codifica y después se elige el Lenguaje de Programación más adecuado.
C. Se elige el Lenguaje de Programación más adecuado, se diseña un algoritmo y se codifica.

5.4.1 Código fuente

Para obtener código fuente a partir de toda la información necesaria del problema:

A. Se elige el Lenguaje de Programación más adecuado y se codifica directamente.
B. Se codifica y después se elige el Lenguaje de Programación más adecuado.
C. Se elige el Lenguaje de Programación más adecuado, se diseña un algoritmo y se codifica.

5.4.2 Código objeto

Código intermedio

Resultado de traducir código fuente a un código equivalente formado por código binario que aún **no puede ser ejecutado directamente por la computadora**

Código resultante de la **compilación** del código fuente

Sólo se genera código objeto una vez que el código fuente está **libre de errores** sintácticos y semánticos

5.4.2 Código objeto

Dos formas de proceso de traducción:

- **Compilación**
  - se realiza sobre todo el código fuente en un solo paso
  - el software responsable se llama compilador
- **Interpretación**:
  - se realiza línea a línea y se ejecuta simultáneamente
  - el software responsable se llama intérprete
  - detección de errores es más detallada
  - normalmente no guardan el resultado de dicha traducción

5.4.3 Código ejecutable

Resultado de **enlazar los archivos de código objeto**, consta de un **único archivo** que puede ser directamente **ejecutado por la computadora**

No necesita **ninguna aplicación externa**

Este archivo es ejecutado y controlado por el **sistema operativo**

5.4.3 Código ejecutable

Enlazar todos los archivos de código objeto, a través de un software llamado **linker** (**enlazador**)

- Toma los **objetos generados** en la compilación, la información de todos los recursos necesarios (**biblioteca**) y enlaza el código objeto con su(s) biblioteca(s)
- Finalmente produce un **fichero ejecutable** o una biblioteca

5.4.3 Código ejecutable

Tipo de código	Relación	Características
Código fuente		1. Escrito en Lenguaje Máquina pero no ejecutable
Código objeto		2. Escrito en algún Lenguaje de Programación de alto nivel, pero no ejecutable
Código ejecutable		3. Escrito en Lenguaje Máquina y directamente ejecutable

5.4.3 Código ejecutable

Tipo de código	Relación	Características
Código fuente	**2**	1. Escrito en Lenguaje Máquina pero no ejecutable
Código objeto	**1**	2. Escrito en algún Lenguaje de Programación de alto nivel, pero no ejecutable
Código ejecutable	**3**	3. Escrito en Lenguaje Máquina y directamente ejecutable

5.5 Máquinas virtuales

Tipo especial de software cuya misión es **separar el funcionamiento del ordenador de los componentes hardware instalados**

Podremos **desarrollar y ejecutar una aplicación sobre cualquier equipo**, independientemente de las características concretas de los componentes físicos instalados

Garantiza la **portabilidad** de las aplicaciones

5.5 Máquinas virtuales

Funciones principales:

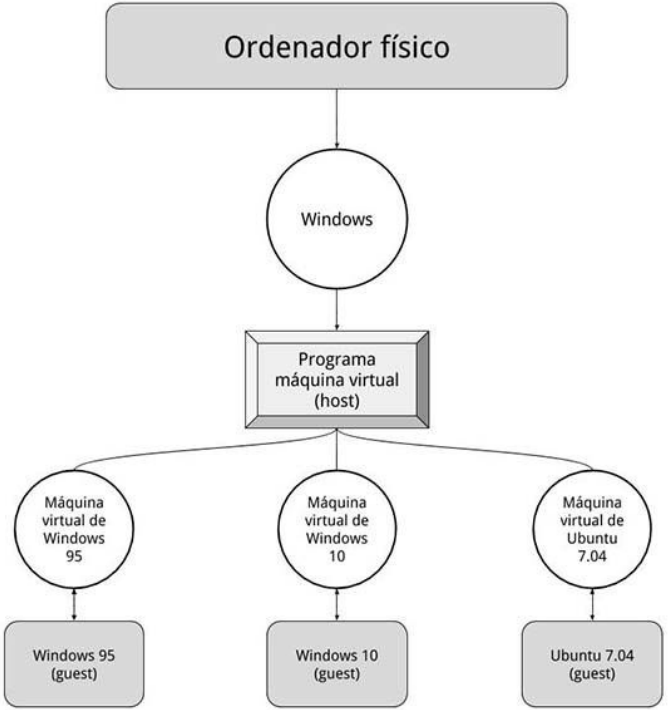
- Conseguir que las aplicaciones sean **portables**
- Reservar **memoria** para los objetos que se crean
- **Comunicarse con el sistema** donde se instala la aplicación (huésped), para el control de los dispositivos hardware implicados en los procesos
- Cumplimiento de las **normas de seguridad** de las aplicaciones

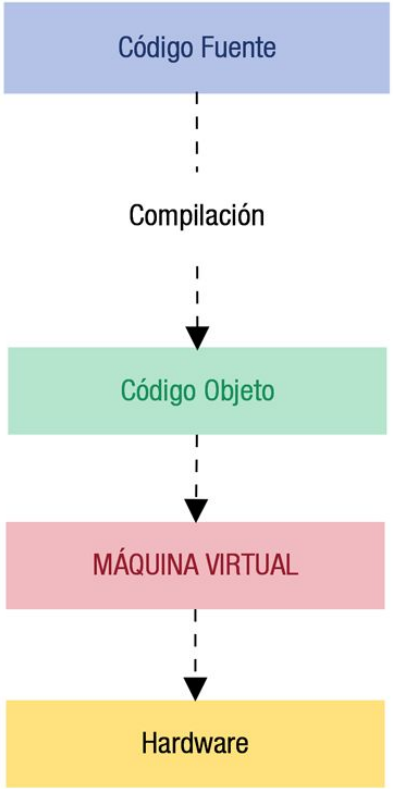
5.5 Máquinas virtuales

Características:

- Cuando el código fuente se compila se obtiene **código objeto** (bytecode, código intermedio)
- Funciona como una capa de software de bajo nivel y actúa como **puente entre** el bytecode de la **aplicación** y l**os dispositivos físicos del sistema**
- La Máquina Virtual **verifica todo el bytecode** antes de ejecutarlo.

5.5 Máquinas virtuales





5.5 Máquinas virtuales

Usos:

- Probar otros sistemas operativos
- Ejecutar programas antiguos
- Usar aplicaciones disponibles para otros sistemas
- Probar una aplicación en distintos sistemas
- Seguridad adicional

Pega principal: el rendimiento

5.6 Pruebas

Una vez obtenido el software, la siguiente fase del ciclo de vida es la r**ealización de pruebas**

Se realizan sobre un **conjunto de datos de prueba**, que consisten en un conjunto seleccionado y predefinido de datos límite a los que la aplicación es sometida

Es imprescindible para asegurar la **validación y verificación del software** construido

5.6 Pruebas

Pruebas unitarias

Probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente)

5.6 Pruebas

Pruebas de integración

Una vez que se han realizado con éxito las pruebas unitarias

Comprobar el **funcionamiento del sistema completo**: con todas sus partes interrelacionadas

Beta Test: se realiza sobre el entorno de producción donde el software va a ser utilizado por el cliente (a ser posible, en los equipos del cliente y bajo un funcionamiento normal de su empresa)

El período de prueba será normalmente el **pactado con el cliente**

5.7 Documentación

Todas las etapas en el desarrollo de software deben quedar perfectamente documentadas

Para dar toda la **información a los usuarios de nuestro software** y poder acometer **futuras revisiones** del proyecto

En **todas las fases** del mismo, para pasar de una a otra de forma clara y definida

Una correcta documentación permitirá la **reutilización de parte de los programas en otras aplicaciones**

5.7 Documentación

Guía técnica

Refleja:

- El diseño de la aplicación
- La codificación de los programas
- Las pruebas realizadas

5.7 Documentación

Guía técnica

¿A quién va dirigido?

- Al personal técnico en informática (analistas y programadores)

5.7 Documentación

Guía técnica

¿Cuál es su objetivo?

- Facilitar un correcto desarrollo, realizar correcciones en los programas y permitir un mantenimiento futuro

5.7 Documentación

Guía de uso

Quedan reflejados:

- Descripción de la funcionalidad de la aplicación
- Forma de comenzar a ejecutar la aplicación
- Ejemplos de uso del programa
- Requerimientos software de la aplicación
- Solución de los posibles problemas que se pueden presentar

5.7 Documentación

Guía de uso

¿A quién va dirigido?

- A los usuarios que van a usar la aplicación (clientes)

5.7 Documentación

Guía de uso

¿Cuál es su objetivo?

- Dar a los usuarios finales toda la información necesaria para utilizar la aplicación

5.7 Documentación

Guía de instalación

¿A quién va dirigido?

- Al personal informático responsable de la instalación, en colaboración con los usuarios que van a usar la aplicación (clientes)

5.7 Documentación

Guía de instalación

¿Cuál es su objetivo?

- Dar toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa

5.8 Explotación

Ya hemos hecho

- las pruebas que nos demuestran que el software es fiable y carece de errores
- la documentado todas las fases

Siguiente paso es la explotación

- Parecido al mantenimiento
- Diferencia en base al momento en que se realiza

5.8 Explotación

Instalación, **puesta a punto** y **funcionamiento** de la aplicación en el **equipo final del cliente**

Instalación

- los programas son transferidos al computador del usuario cliente
- posteriormente configurados y verificados
- Beta Test: últimas pruebas que se realizan en los propios equipos del cliente y bajo cargas normales de trabajo

5.8 Explotación

Configuración

- Asignamos los **parámetros de funcionamiento normal** del cliente y probamos que la aplicación es **operativa**
- Se puede hacer de forma **automática** si la aplicación es sencilla

Producción normal

- La aplicación pasa a manos de los usuarios finales
- Explotación del software

5.9 Mantenimiento

¿Con la entrega de nuestra aplicación **hemos terminado nuestro trabajo**?

En otro sector laboral puede ser así, pero el caso de la **construcción de software** es diferente

Necesitamos un **mantenimiento**, la etapa más **larga** de todo el ciclo de vida del software

5.9 Mantenimiento

El software es **cambiante** y deberá **actualizarse y evolucionar con el tiempo**

Deberá **adaptarse** de forma paralela a las mejoras del hardware en el mercado

Afrontar **situaciones nuevas** que no existían cuando el software se construyó

Surgen errores que habrá que ir corrigiendo

Nuevas versiones del producto

5.9 Mantenimiento

Se pacta con el cliente un **servicio de mantenimiento** de la aplicación

Tendrá un **coste** temporal y económico

Mantenimiento se define como el proceso de **control**, **mejora** y **optimización** del software

5.9 Mantenimiento

Tipos de cambios:

- **Perfectivos**: mejorar la funcionalidad del software
- **Evolutivos**: nuevas necesidades del cliente. Modificaciones, expansiones o eliminaciones de código
- **Adaptativos**: modificaciones, actualizaciones... para adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, etc.
- **Correctivos**: La aplicación tendrá errores en el futuro