Adrián Tirado Ramos

1º De DAW

Entorno Desarrollo

RESUMEN UD1

ÍNDICE

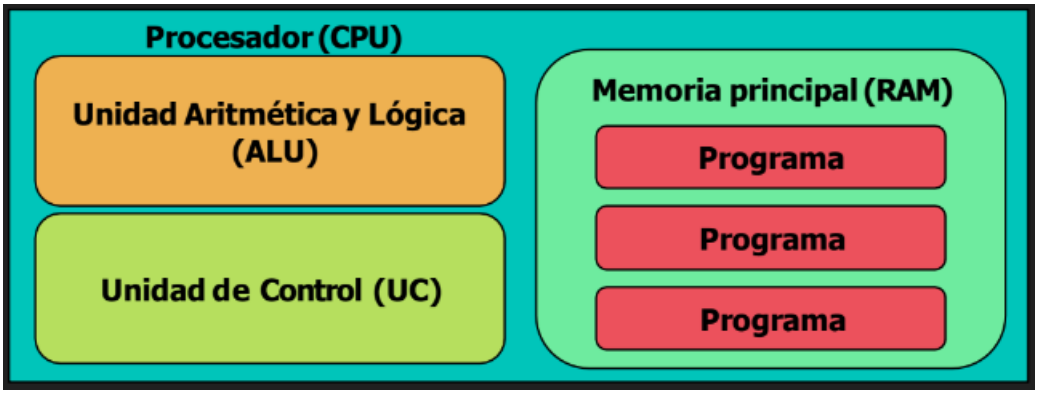
Resumen

Qué es un ordenador

Un ordenador es un dispositivo que se compone de elementos físicos (hardware) y elementos lógicos (software) que acepta datos de entrada (teclado, ratón), los procesa y muestra una salida(pantalla). Estas órdenes que introduce el usuario deben de ser traducidas para que las entienda el ordenador.

Los primeros ordenadores en aparecer se originaron en la década de los 40's

Diagrama de un ordenador



Qué es el software

Es la parte intangible (lógica) de un Sistema Informático que se desarrolla para llevar a cabo una tarea específica.

Este se comunica con el hardware para comunicarle que es lo que debe de realizar.

Se encarga de traducir las instrucciones de los usuarios.

El término fue usado por Charles Babbage en el siglo XIX.

Alan Turing coge el relevo para descifrar la máquina Enigma.

Conceptos clave:

- El software se desarrolla, no se fabrica
- El software es intangible, por lo que no es un elemento físico
- El software no se daña (genéricamente hablando), y una copia produce un duplicado del software original
- El software puede personalizarse (aplicación de propósito específico)
- El software permite el uso del ordenador
- El software se crea usando un lenguaje de programación y una metodología de programación

Qué es una aplicación informática

Una aplicación se compone de un conjunto de instrucciones y se desarrolla usando un lenguaje de programación

Las instrucciones le indican al ordenador el programa o pasos que debe ejecutar

Si el ordenador no entiende alguna instrucción, lo notificará mediante un mensaje

En este ciclo pasaremos de ser usuarios a programadores de aplicaciones

Algunos ejemplos:

- Sistemas operativos (Windows, Linux, MacOS, RedHat, Solaris)
- Aplicaciones de contabilidad y ofimática (Word, Excel, Paint, …)
- Aplicaciones de gestión de bases de datos (Access, MySQL, …)
- Aplicaciones de diseño gráfico (Adobe Photoshop)
- Aplicaciones de correo electrónico (Gmail, Outlook, Hotmail, …)
- Videojuegos

Ejemplo de aplicación informática (en este caso es un script de Linux)

```
# Function to initialise a template SSH config file
init() {
  filePath="$1"

  if [[ -z "$filePath" ]]; then
    printf "%s\n" "${red}Please provide a file path after --init option! ${end}"
    exit 1
  elif [ -f "$filePath" ]; then
    read -r -p "${red}$filePath already exists. Do you want to overwrite it? [y/n] ${end}" ans
    ans="$(tr '[:upper:]' '[:lower:]' <<< "$ans")"

    if [[ $ans == "n" || $ans == "no" ]]; then
      exit 0
    elif [[ $ans == "y" || $ans == "yes" ]]; then
      create_template "$filePath"
    else
      printf "%s\n" "${red}Invalid option! ${end}"
      exit 1
    fi
  else
    create_template "$filePath"
  fi
}
```

Conceptos clave:

- **Programa**: conjunto de pasos o instrucciones que indican al ordenador cómo realizar una tarea
- **Ejecutar**: iniciar un programa y ponerlo en ejecución
- **Librería**: conjunto de programas y funciones que realizan tareas concretas (BBDD, informes...)
- **Entorno de desarrollo (IDE)**: herramienta que facilita y posibilita el desarrollo de software

Software a medida vs. estándar

Software a medida

Se desarrolla según los requerimientos de una empresa u organismo

Se adecuan a la actividad de dicha empresa u organismo

Características:

- Necesita un tiempo de desarrollo
- Se adapta a las necesidades específicas de dicha empresa (no trasladable a otras)
- Puede contener errores y necesitar de una etapa de adaptación y mantenimiento
- Más costoso que el software estándar

Software estándar:

Software genérico (válido para cualquier cliente)

Resuelve múltiples necesidades, y suele incluir herramientas de configuración

Características:

- Se compra ya desarrollado (sólo se puede configurar o actualizar)
- Suele tener menos errores que el software a medida (más testeado generalmente)
- Suele ser más barato que el software a medida
- Suele incluir funciones que nunca serán usadas y/o puede carecer de opciones que sí son importantes para la empresa

Qué es un lenguaje de programación

Es un lenguaje artificial (creado) por programadores que permiten traducir ciertas instrucciones para que las entienda el ordenador, son muchísimo más complejos que el lenguaje máquina (binario).

Se componen al igual que los lenguajes artificiales de los humanos con cierta gramática y semántica

Conceptos clave:

- Lenguaje máquina: lenguaje sólo comprensible por el ordenador (binario)
- Programa: conjunto de instrucciones escritas en un lenguaje de programación
- Instrucción: cada sentencia u órdenes que forman parte de un programa
- Palabra reservada: cada símbolo o conjunto de caracteres que componen la sintaxis del lenguaje
- Semántica: reglas que definen la combinación de los símbolos de un lenguaje de programación

Tipos de lenguajes de programación

Existen muchos tipos de lenguajes diferentes que evolucionan a lo largo de los años, ya que un lenguaje de programación recientemente creado es más amigable con el usuario que otros más anteriores para facilitarle el proceso de programación al programador

Hay veces que se necesita hacer un programa pesado (lento y de gran tamaño) comparado a otros programas, por lo que facilitarle el proceso de programación al programador hace que ese proceso sea más liviano

Hay muchos lenguajes de programación como C, C++, PowerShell, JavaScript, …

Lenguaje máquina

Operaciones complejas e ilegibles (01010)

Necesita ser traducido al ordenador

Primer lenguaje de programación (muy dependiente de hardware)

Al ser muy dependiente del hardware, al cambiar un componente, puede ser que cambie más o menos las órdenes dependiendo del componente (los más importantes CPU y RAM).

Ahora, solo se suele usar para programas como los drivers, módulos de Sistemas Operativos

Lenguaje de medio nivel (ensamblador)

Facilita la programación

Se centra en el hardware, pero usa términos legibles para el programador

Hay que compilarlo (traducirlo) para que lo entienda el ordenador

Trabaja con registros del procesador y direcciones de memoria

A pesar de ser prácticamente una mejora del lenguaje máquina, sigue siendo difícil de comprender y usar.

Lenguaje de alto nivel

Programación sencilla e intuitiva

Son cercanos a los lenguajes humanos (IF, WHILE, DO, …)

Incorporan librerías y funciones predeterminadas que ayudan al programador en la operación de programación

Suelen ofrecer frameworks, que incorporan funciones y componentes extra

La mayoría de los lenguajes de programación actuales se engloban en esta categoría

Por ejemplo:

C/C++:

Uno de los más potentes y utilizados, ligado a Unix y a Linux

Estructurados y ligados a funciones

Incluyen el concepto de puntero (necesario para gestionar la memoria, pero un concepto complejo)

Combinan comandos de alto nivel

Programas eficientes, rápidos y pequeños

Necesidad de compilar los programas

Java

Simplificación de C++(no admite sobrecarga de operadores ni herencia múltiple)

Manejo más eficiente de cadenas de caracteres (String)

Lenguaje 100%Orientado a objetos

Pensado para trabajar con redes y protocolos de red (TCP/IP,HTTP,HTTPS...)

Portable, seguro y permite la programación concurrente

Es Un lenguaje virtual interpretado

Python

Lenguaje de alto nivel muy portable

Lenguaje interpretado

Orientado a objetos

Permite incrustarse en otros lenguajes como C/C++

Uno de los lenguajes más simples y sencillos de aprender

Lenguaje JavaScript

Se utiliza mucho en programación web (Angular, React...)

Se parece mucho a Java, C y C++

Es Un lenguaje de scripting, seguro y fiable

Se ejecuta en el lado del cliente (Frontend)

Su código es visible(hay que tener en cuenta aspectos de seguridad)

Lenguaje PHP

Lenguaje web de propósito general usado en el servidor (Backend)

Se emplea en aplicaciones como Prestashop, WordPress...

Es Un lenguaje multiplataforma

Orientado al desarrollo de webs dinámicas

Buena integración con MySQL

Orientado a objetos

Lenguaje interpretado

Paradigma de programación

Programación estructurada

Consiste en una secuencia ordenada y organizada de instrucciones

Es fácil de entender

Programas sencillos y rápidos

Posee tres estructuras básicas: secuencia, selección e iteración

Inconveniente: un único bloque de programa (es inmanejable si crece)

Programación funcional

Consiste en descomponer el programa en funciones o módulos

Programa modular y estructurado

Inconveniente: el programa puede crecer en gran medida y ser complejo

Programación orientada a objetos

Consiste en representar entidades del mundo real

Permite reutilizar código

Principio de separación de responsabilidades

Patrones de diseño y paradigmas avanzados (MVC...)

Polimorfismo, herencia y encapsulación

Programación lógica

Consiste en representar predicados y relaciones

Uso de lógica de predicados de primer orden

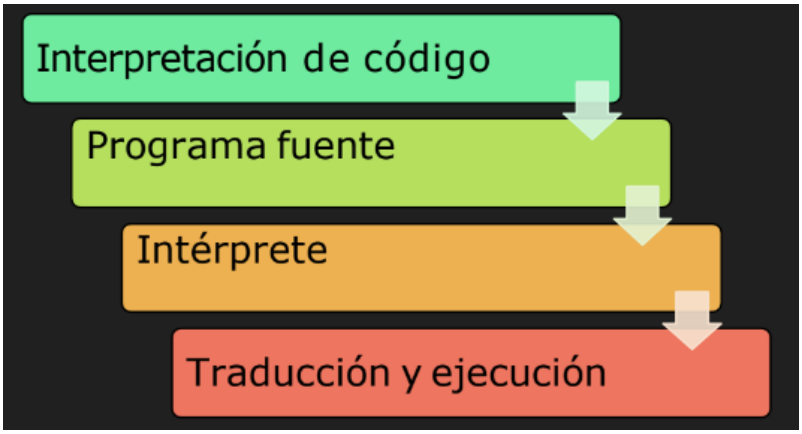
Muy utilizado en aplicaciones de inteligencia artificial

El proceso de traducción/compilación

4.1 Traductores de código

4.2 Interpretación de código

4.3 Compilación de código



Traductores de código

Un traductor traduce un lenguaje de alto nivel a ensamblador o lenguaje máquina

Existen dos tipos de traductores: intérpretes y compiladores, en función de la forma de ejecutar un lenguaje existen los siguientes tipos:

- Lenguajes compilados
- Lenguajes interpretados
- Lenguajes virtuales

Lenguajes compilados

Necesitan un compilador para traducir el código fuente a código máquina

Se ejecutan más rápida que los programas interpretados o virtuales

Precisan de un programa enlazador que permite unir el código objeto con el código objeto de librerías

Aunque el código es más seguro, no son tan flexibles para modificarlos como los lenguajes interpretados

Ejemplo: C/C++

Lenguajes interpretados

No generan código objeto

El intérpretees un programa que tiene que estar cargado en memoria

El intérprete “interpreta” cada sentencia del programa y lo ejecuta

Las instrucciones se traducen “on the fly” (al vuelo), a medida que van ejecutándose

Ejemplo: PHP

Lenguajes virtuales

Son más portables que los lenguajes compilados

El código se genera tras la compilación en un código intermedio o bytecode

El código puede ser interpretado por una máquina virtual instalada en cualquier equipo

Son más lentos, pero más versátiles

Ejemplos: Java

Algunas definiciones importantes

- Código fuente: código de un programa, escrito por un programador en un lenguaje de programación
- Compilar: proceso que traduce el código fuente en código objeto (comprensible por un ordenador)
- Código objeto: código máquina generado tras la compilación del código fuente
- Librería: código externo que se incluye al programa para proporcionar funcionalidad adicional
- Archivo ejecutable: programa ejecutable que puede ser ejecutado en el ordenador
- Incluye librerías (tras ser enlazadas), complementos, módulos...

4.2 Interpretación de código

Un intérprete traduce el código fuente línea a línea

El intérprete tiene que estar en memoria ejecutándose para ejecutar el programa

El código fuente del programa también ser carga en memoria para poder ser interpretado

Compilación de código

Un compilador traduce código fuente a código máquina

El compilador se instala en cada máquina en la que queramos compilar el programa

El compilador depende de la arquitectura hardware de la máquina

El proceso de compilación supone las siguientes fases:

Preprocesado: se traducen y se ejecutan los comandos de preprocesamiento

Generación de código intermedio: generación de código máquina

Enlazado: se enlazan el código objeto con librerías externas

Fases del desarrollo de una aplicación

El software se desarrolla siguiendo una metodología de desarrollo cada metodología o paradigma suele tener varias fases en común

Las fases más comunes:

1. Fase inicial

Planificación del proyecto

Estimación de costes (para saber si el proyecto va a ser viable)

La fase más compleja

Se requiere de expertos en planificación de proyectos

Se desarrollan documentos importantes para el proyecto

2. Fase de análisis

Analizar el problema

Recopilar, examinar y formular los requisitos del cliente (especificación requisitos)

Análisis de restricciones, entrevistas con el cliente y los usuarios finales

Se genera un documento vinculante (contrato) entre el cliente y el desarrollador

3. Fase de diseño

Determinar los requisitos generales de la arquitectura de la aplicación

Se define cada subconjunto de la aplicación

Los documentos más técnicos

En esta fase se involucra a los jefes de proyecto, arquitectos de software y analistas

4. Fase de implementación

Implementar el software usando lenguajes de programación (librerías, frameworks...)

Crea documentación muy detallada en la que se incluye y documenta el código fuente

Parte del código suele comentarse sobre el propio código fuente generado

Se detallan las entradas, salidas, parámetros... de cada uno de los módulos del programa.

El detalle es máximo, pensando en el mantenimiento y soporte futuro que tendrá el programa

5. Pruebas

Realizan pruebas para garantizar que la aplicación se ha programado según las especificaciones

2 categorías de pruebas:

Pruebas funcionales

Prueba que la aplicación hace lo que tiene que hacer (en el equipo cliente)

Pruebas estructurales

Se realizan pruebas técnicas sobre el sistema (pruebas de estrés, carga, …)

6. Explotación

Instala el software en el entorno real de uso

Se trabaja con el software de forma cotidiana

Se recogen los errores y las correcciones en un nuevo documento de mantenimiento

Los programadores y analistas revisan esos fallos para mejorar el software y aprender de los errores

7. Mantenimiento

Se realizan procedimientos correctivos sobre el programa

Siempre hay que tener delante la documentación técnica de la aplicación

Las operaciones de mantenimiento se deben documentar para dejar constancia de los cambios

8. Retirada

El software llega al final de su vida útil

No resulta rentable seguir ampliándolo o manteniéndolo

Llegados a este punto, se reinicia este ciclo:

- Comprando un nuevo software
- Desarrollando un nuevo software (a medida)

5.2 Documentación

La documentación es vital

En cada fase se generan 1 o más documentos

Es vital para saber cómo usar la aplicación final

Como mínimo, cada aplicación debe tener estos documentos:

- Manual de usuario: explica cómo usará el usuario como usar la aplicación
- Manual técnico: documentación dirigida a técnicos
- Manual de instalación: detalla el proceso de instalación de la aplicación

5.3 Roles del desarrollo de software

Detallamos los roles del proceso de desarrollo de software:

Arquitecto de software:

Decide cómo realiza el proyecto y cómo va a estructurarse

Amplio conocimiento de las tecnologías, los frameworks y las librerías

Decide y forma los recursos del desarrollo de un proyecto

Jefe de proyecto:

Dirige el curso del proyecto

Puede ser un analista con experiencia, un arquitecto o una persona dedicada a ese puesto en exclusividad

Debe saber gestionar un equipo y lidiar con los tiempos (de cada tarea/apartado)

Trata de manera continua fluida con el cliente

Analista de sistemas:

Realiza un estudio exhaustivo del problema que va a llevarse a cabo

Efectúa el análisis el diseño de todo el sistema

Requiere mucha experiencia, y también suele involucrarse en reuniones con el cliente

Analista programador:

Conoce en profundidad el lenguaje de programación

Se encarga de codificar las tareas encomendadas por el analista o el analista programador

Sumisiones la de codificar y probar los diferentes módulos de la aplicación

5.4 Paradigmas de desarrollo clásicos

Los paradigmas de desarrollo clásicos son inflexibles

Cada etapa debe finalizar para pasar a la siguiente

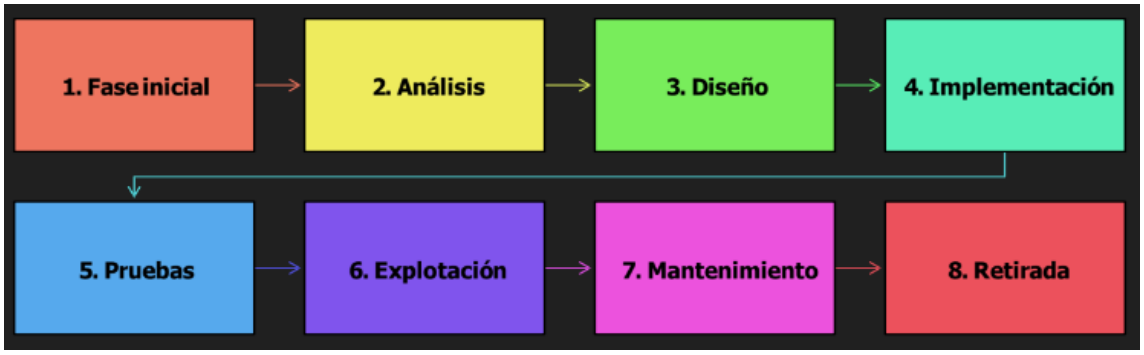
¿Qué pasa si detectamos un error grave de diseño durante la implementación?

El paradigma más clásico es el modelo en cascada

En esta asignatura trabajaremos con metodologías ágiles (SCRUM)

El modelo en cascada es muy inflexible

Cualquier error detectado en una fase muy tardía implica sobrecoste y desperdicios



5.5 Metodologías ágiles

En esta asignatura usaremos metodologías ágiles

Estas metodologías responden mejor ante un cambio de especificaciones o un error

Las metodologías ágiles se basan en el manifiesto ágil, que valora:

A los individuos y su interacción, por encima de los procesos y las herramientas Al software que funciona, por encima de la documentación exhaustiva

A la colaboración con el cliente, por encima de la negociación contractual

A la respuesta al cambio, por encima del seguimiento de un plan