**TEMA 1**

**1.1 Conceptos de los programas informáticos.**

* **Programa informático:** conjunto de **instrucciones** escritos en un lenguaje de programación que se ejecutan de manera **secuencial**.
* Las **instrucciones** se dividen en **microinstrucciones**.

**1.2 Conceptos de los lenguajes de programación.**

* Un sistema informático sólo es capaz de entender código escrito en código máquina **(1s y 0s**).
* Un lenguaje de programación permite interactuar el código con el procesador al escribir programas utilizando un mayor nivel de abstracción en el código.

**1.2.1 Definición de lenguaje de programación.**

* Conjunto de **instrucciones** + **operadores** + **reglas de sintaxis** y **semánticas** que se ponen a disposición de los programadores para que se comuniquen con el hardware y el software.
* Hay diferentes niveles de lenguajes de programación **(bajo, medio, alto)**, pero se suelen usar más los de nivel alto como **C++**, ya que son más entendibles y fáciles.
* Nadie programa en nivel bajo, pero si se puede dar el caso de programar en nivel medio con **"Assembler"**, por ejemplo, para desarrollar drivers de periféricos, ya que ocupan poco y son óptimos.

**1.2.2 Características de los lenguajes de programación más usados.**

**Tipos de lenguajes:**

**Lenguajes de primera generación: Lenguaje máquina.**

* Es a nivel de procesador (bajo), las instrucciones son en binario (0s y 1s), sólo hay uno y es el código máquina.

**Lenguajes de segunda generación: Lenguaje ensamblador.**

* Escriben programas muy optimizados que aprovechan al máximo el hardware.
* Permiten al programador escribir código legible (medio) y dependen directamente del hardware.

**Lenguajes de tercera generación: Lenguajes de alto nivel.**

* Son independientes de la máquina en la que se van a ejecutar.
* El código es sencillo y comprensible, lo que permite ejecutarlo en diferentes máquinas según el compilador usado, pero al ser programas escritos en nivel alto se ejecutan más lento que los de nivel bajo.

**Lenguajes de cuarta generación: De propósito específico.**

* Permiten desarrollar aplicaciones sofisticadas en poco tiempo, por ejemplo, para realizar consultas a una base de datos con una sola instrucción.
* Están orientados básicamente al manejo de base de datos.

**Lenguajes de quinta generación.**

* Son lenguajes específicos para inteligencia artificial, como Prolog o Lisp.

**1.2.3 El nivel de abstracción en los lenguajes de programación.**

* El nivel de abstracción de un lenguaje implica lo alejado que está del código máquina.
* Cuanto más parecido sea al lenguaje humano más nivel de abstracción.

**Tipos de niveles:**

* **Bajo:** Binario.
* **Medio:** Assembler.
* **Alto:** Java o cualquier otro de alto nivel.

**1.2.4 Los lenguajes de programación según la forma de ejecución.**

* **Compilados (C++):** Son los lenguajes que deben ser compilados antes de poder ejecutarse.
* **Interpretados (Python):** Estos lenguajes se ejecutan línea a línea y no hace falta compilar todo el programa.
* Además, no generan un código objeto, en vez de ejecutar el código el so lo hace un intérprete que hay dentro del mismo y lo traduce en tiempo real.
* **Virtuales (Java/JVM):** Son iguales a los compilados, solo que en vez de ejecutarse en la propia máquina lo hace en una virtual.

**1.2.5 Los lenguajes de programación según el paradigma de programación.**

El paradigma de programación de un lenguaje de programación se basa en:

* El **método** para llevar a cabo los cálculos en el proceso.
* La **forma** en la que deben estructurarse las tareas que debe realizar el programa.

**Tipos de paradigmas:**

* **Imperativos o estructurados (C):** Se basan en sentencias imperativas, básicamente realizan una determinada operación una tras otra.
* **Orientado a objetos (Java):** Intentan abstraer conceptos de la vida real y representarlos con objetos.
* **Funcional (Haskell):** Basados en modelos matemáticos.
* **Lógico (Prolog):** Basados en modelos matemáticos y resolver preguntas planteadas, se suelen usar para investigación.

**TEMA 2**

**2.1 Elementos que intervienen en el desarrollo de aplicaciones.**

**2.1.1 Los elementos más importantes.**

* **Código fuente:** Son las instrucciones que codifican los programadores.
* **Código objeto:** Es el resultado de compilar/traducir el código fuente y es un código intermedio.
* **Código ejecutable:** El que se puede ejecutar, como un .exe.

**2.1.2 Herramientas implicadas para la obtención de código ejecutable.**

* **Compilador:** Traduce el código fuente escrito con lenguaje de nivel alto a lenguaje máquina. Además, detecta los errores y da el ejecutable depurado.
* **Máquina virtual:** Hace exactamente lo mismo que el compilador, pero interpretando directamente el **Bytecode** (lenguaje de bajo nivel) en una máquina virtual sin importar el hardware de la máquina física.

**2.1.3 Etapas del proceso de obtención de código ejecutable.**

La compilación es la traducción de un programa escrito (código fuente) y estas son las etapas:

* **Código fuente:** líneas de texto con los pasos que se deben seguir para ejecutar el programa.
* **Análisis lexicográfico:** a partir del código fuente genera una salida compuesta de tokens qué son los componentes léxicos.
* **Análisis sintáctico-semántico:** se comprueba el texto de entrada en base a una gramática dada, la del lenguaje de programación.
* **Generador de código intermedio:** es el que transforma el código lenguaje más próximo a la plataforma de ejecución.
* **Optimizador de código:** realiza una serie de transformaciones de mejora del código, se obtiene un código optimizado.
* **Generador de código:** el compilador convierte el programa sintácticamente correcto en una serie de instrucciones que deben ser interpretadas por una máquina. A partir del generador de código se obtiene el código objeto.
* **Enlazador:** programa que a partir del código generado y los recursos necesarios (bibliotecas) quita los recursos que no necesita y enlaza los que necesite al código objeto finalmente genera un código ejecutable.

**2.1.4 Fases de desarrollo de una aplicación**

* **Análisis:** Son los requisitos que necesita el programa **(lenguaje natural)**.
* **Diseño:** En esta fase se realizarán los diagramas de clases o de comportamiento.
* **Codificación:** Se pasa del diseño al código.
* **Pruebas:** Probar el código, durante y después de la codificación.
* **Documentación:** Documentar el código.
* **Mantenimiento:** Corregir errores o amplificaciones o modificaciones de software.
* **Explotación:** Se prepara el software para su distribución.

**TEMA 3**

**3.1 ¿Qué es un IDE?**

**3.1.1 Definición de Entorno de Desarrollo Integrado (IDE).**

* Un IDE es un programa que nos ofrece todas las herramientas que necesitamos para programar en un único programa. Ejemplos: Eclipse y NetBeans.

**3.1.5 Las diferentes herramientas que podemos encontrar en un IDE.**

* Los componentes básicos comunes que se suelen encontrar en un IDE son el editor de texto, el compilador, el intérprete, el depurador y el cliente (para el control de versiones).

**3.1.5.1 Compilador.**

* El compilador del IDE es el que realiza las diferentes fases del análisis del código y traduce a lenguaje máquina.

**3.1.5.2 Ejecutar de forma virtual.**

* Permite ejecutar el programa de forma virtual sin tener un ejecutable final.

**3.1.5.3 Depurador.**

* Es la herramienta que nos permite probar y depurar el código fuente del programa. También detecta errores.

**3.1.5.4 Control de versiones.**

* Hace posible tener un registro histórico de las tareas hechas o versiones del código fuente.

**3.1.5.5 Refactorización.**

* Permite reestructurar el código fuente, mejorándolo sin alterar la funcionalidad.

**3.1.5.6 Documentación.**

* Ayudan a documentar el código mientras se va construyendo, por ejemplo: JavaDoc.

**3.1.5.7 El gestor de proyectos.**

* Generaran de forma automática todas las dependencias del código fuente y ayuda a tenerlas localizadas.

**3.1.5.8 El editor de texto.**

* Es un editor de texto plano donde se escribe el código fuente, dispone de autocompletado de código, coloreado de sintaxis e inspecciones de clases y objetos.

**3.1.5.9 Vistas.**

* Ventanas auxiliares de diferentes tipos de contenidos, como el valor de las variables, el debug, etc.

**3.1.5.10 Añadir y modificar la barra de herramientas.**

- Todas las barras de herramientas que aparecen en los IDEs, se pueden personalizar.

**3.1.5.11 Configurar diferentes interfaces.**

- Los IDEs permiten configurar diferentes interfaces con distintas herramientas.

**3.1.5.12 Comandos personalizados y atajos de teclado.**

- Combinaciones de teclas para ejecutar tareas de forma más rápida y eficiente.

**3.1.6 El uso básico de un IDE.**

- Un IDE sirve para codificar programas en un determinado lenguaje de programación con tal de obtener un ejecutable que haga una determinada función, facilitando la tarea del programador.

**TEMA 4**

**4.1.1.2 Herramientas para hacer las pruebas y detectar errores.**

* **El depurador:** El depurador es una herramienta que permite ver el proceso del programa paso a paso.

**¿Qué herramientas podemos usar para la depuración?**

**1. Los puntos de ruptura o breakpoint:**

* Cuando se alcanza un punto de ruptura (breakpoint) el programa se detiene, los "breakpoints" se establecen en líneas concretas del código.
* Aquí también se puede hacer "step in" que te lleva al interior de las funciones o métodos, o "step over" que ejecuta la línea actual y pasa al siguiente nivel sin entrar en funciones.

**2. Puntos de seguimiento:**

* Al contrario que los "breakpoints" no detienen el programa y se sitúan en cualquier línea del código.

**El analizador de código:** ayuda a identificar los errores en tiempo real a medida que se programa.

**Los errores:** Suelen marcarse en el margen de color rojo, si no se solucionan no se puede compilar.

**Los Warnings:** Son avisos de que algo podría ir mal, pero no son errores fatales que nos impidan compilar.

**4.2.2 ¿Qué son los casos de pruebas?**

* Son las condiciones que se establecen con el objetivo de determinar si la aplicación funciona correctamente.
* Son pruebas a nivel atómico de cada función o bucle, cuantas más líneas de código más pruebas habrá que hacer.

**4.2.3 Tipos de pruebas (Menor a mayor nivel).**

* **Pruebas unitarias:** para verificar el correcto funcionamiento de una unidad de código.

**1. Pruebas de caja blanca:**

1. El método del camino básico.
2. Complejidad ciclomática.
3. Caminos independientes.

**2. Pruebas de caja negra:**

1. Particiones equivalentes.
2. Valores límites.

* **Pruebas funcionales:** para detectar errores del código teniendo en cuenta los requerimientos del cliente.
* **Pruebas de integración:** pruebas sobre las relaciones entre las diferentes partes del software.
* **Pruebas de sistema:** para detectar errores relacionados con los requisitos del programa.
  1. Pruebas de carga.
  2. Pruebas de estrés.
  3. Pruebas de seguridad.
* **Pruebas de aceptación:** las pruebas que realizan los usuarios finales.
  1. Pruebas alfa
  2. Pruebas beta

**TEMA 5**

**5.1.1.1 Qué es el refactoring.**

* El refactoring son cambios para que el código sea más visible, flexible y modificable, debe aplicarse en un código que ya funcione y verificar que siga funcionando después de la refactorización.

**¿Para qué refactorizar si ya funciona el código?**

* Para facilitar la tarea de trabajar con el código y disminuir la probabilidad de generar errores.

**Ventajas de refactoring:**

* Evitar problemas derivados de los cambios de los mantenimientos posteriores.
* Simplificar el diseño.
* Ayuda a entenderlo mejor.
* Será más fácil detectar errores.
* Agiliza la programación.

**Inconvenientes de refactoring:**

* No es fácil de aplicar si se tiene poca experiencia.
* Si se llega a un exceso de querer optimizar el código de una forma obsesiva.
* Dedicar mucho tiempo a la refactorización.
* Posible impacto de la refactorización al resto del software.

**¿Cuándo se realiza el refactoring?**

* Es recomendable hacerlo durante la fase de mantenimiento.

**5.1.1.3 Los malos olores (bad smells).**

* Es el código que está mal hecho y el cuál se puede corregir, mejorar o refactorizar.

**5.2.2 Los comentarios.**

* Son frases cortas que pretenden aportar información sobre una parte del código y se consideran parte de este.
* Se añaden normalmente con los caracteres // o /\*\*/ y se pueden escribir en bloque o en línea.

**5.2.3 Herramientas de documentación.**

* Son herramientas que ayudan a documentar proyectos proporcionando unos comentarios en un determinado formato.

**5.3 Control de versiones.**

**¿Para qué sirve un programa de control de versiones?**

* Compartir archivos de diferentes programadores.
* Bloquear archivos que se están editando.
* Fusionar archivos con diferentes cambios.

**¿Qué funcionalidades tiene un programa de control de versiones?**

* Comparar cambios en el código fuente.
* Coordinar las tareas entre diferentes programadores.
* Guardar versiones anteriores del código fuente.
* Seguimiento de los cambios realizados en el código: con un historial de cambios realizados en el código fuente pudiendo conocer el momento del cambio y el autor.
* Restaurar a una versión de código anterior.
* Control de los usuarios.
* Crear ramas (forks) del proyecto que permiten desarrollar varias versiones de un mismo programa a la vez.

**5.3.1 Partes de un sistema de control de versiones.**

* **Repositorio:** es donde se almacenan los datos actualizados e históricos de cambios. Normalmente es un servidor.
* **Módulo:** conjunto de directorios y/o archivos dentro del repositorio que pertenecen a un proyecto común.
* **Revisión:** es una versión determinada de la información que se almacena.
* **Etiqueta (Tag):** darle a cada uno de los ficheros del módulo en desarrollo en un momento preciso un nombre común para asegurarse de reencontrar ese estado de desarrollo posteriormente bajo ese nombre.
* **Rama (branch):** cuando se genera un duplicado del código fuente sobre el que se va a trabajar dentro de los directorios y/o archivos dentro del repositorio.
* **Trunk:** rama principal de código fuente.
* **Merge**: operación de fusión de diferentes ramas.
* **Commit:** operación de confirmación de cambios en el sistema de control de versiones.
* **Changeset:** conjunto de cambios que hace un usuario y sobre los que se realiza una operación “Commit” de manera simultánea y que son identificados mediante un número único en el sistema de control de versiones.

**5.3.2 Tipos de sistemas de control de versiones.**

* **Locales:** cuando se copian los archivos a otro directorio en el mismo equipo.
* **Centralizados:** tienen un único servidor que contiene todos los archivos versionados y varios clientes que descargan los archivos desde ese lugar central.
* **Distribuidos:** cada usuario tiene su propio repositorio en local.

**5.3.3 Operaciones que se pueden hacer en un sistema de versiones.**

* **Commit (subir):** sube una copia de los cambios hechos en local que se integra sobre el repositorio.
* **CheckOut (bajar o desplegar):** cuando crea una copia de trabajo local desde el repositorio.
* **Update (actualización):** cuando se integran los cambios que se han hecho en el repositor en la copia de trabajo local.