Unidad 1: Elementos del desarrollo de software

1. Programa y aplicación informática.

Poner un programa en marcha es sinónimo de ejecutarlo. Cuando ejecutamos un programa sólo vemos los resultados, pero no el guion seguido por el ordenador para conseguir esos resultados. Este guion es realmente el programa.

Los pasos a seguir en la realización de un programa, una vez que ya tenemos claro qué queremos hacer (o hemos solucionado el problema que queremos solventar con un programa), son los siguientes:

* Editar el programa. (Escribir el código).
* Compilarlo.
* Ejecutarlo.
* Depurarlo

Por lo general, un programa no es capaz de ejecutarse por sí mismo, sino que necesita de otros programas externos que lo complementen. Este complemento se llama librería,

Para todo ello, usaremos herramientas que facilitan el trabajo llamadas Entornos de Desarrollo Integrado (IDE).

Un programa informático es un conjunto de instrucciones diseñadas y creadas a través del razonamiento lógico y que son almacenadas en ficheros de texto respetando la sintaxis de un determinado lenguaje de programación.

Una aplicación informática es el conjunto de uno o más programas enlazados o relacionados entre sí, junto con la documentación generada durante el proceso de desarrollo de dicha aplicación.

Un sistema es el conjunto de elementos relacionados entre sí para la consecución de un determinado fin.

1. Lenguajes de programación
   1. **Definición de lenguaje**

Un lenguaje de programación, está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones, permitiendo especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias.

Al proceso por el cual se escribe, se prueba, se depura, se compila y se mantiene el código fuente de un programa informático se le llama programación.

* 1. **Elementos de un lenguaje de programación.**

La sintaxis de un lenguaje de programación describe las combinaciones posibles de los símbolos que forman un programa sintácticamente correcto. El significado que se le da a una combinación de símbolos es manejado por su semántica.

* 1. **Tipos de lenguajes de programación**
* Lenguajes interpretados: Un programa escrito en un lenguaje interpretado requiere de un programa auxiliar (el intérprete), que traduce los comandos de los programas según sea necesario.
* Lenguajes compilados: Un programa escrito en un lenguaje “compilado” se traduce a través de un programa anexo llamado compilador que, a su vez, crea un nuevo archivo independiente que no necesita ningún otro programa para ejecutarse a sí mismo. Este archivo se llama ejecutable.

Un programa escrito en un lenguaje compilado posee la ventaja de no necesitar un programa anexo para ser ejecutado una vez que ha sido compilado. Además, como sólo es necesaria una traducción, la ejecución se vuelve más rápida.

Sin embargo, no es tan flexible como un programa escrito en lenguaje interpretado, ya que cada modificación del archivo fuente (el archivo comprensible para los seres humanos: el archivo a compilar) requiere de la compilación del programa para aplicar los cambios.

Por otra parte, un programa compilado tiene la ventaja de garantizar la seguridad del código fuente. En efecto, el lenguaje interpretado, al ser directamente un lenguaje legible, hace que cualquier persona pueda conocer los secretos de fabricación de un programa y, de ese modo, copiar su código o incluso modificarlo.

Por lo tanto, existe el riesgo de que los derechos de autor no sean respetados. Por otro lado, ciertas aplicaciones aseguradas necesitan confidencialidad de código para evitar las copias ilegales (transacciones<bancarias, pagos en línea, comunicaciones seguras...).

Por último, algunos lenguajes pertenecen a ambas categorías dado que el programa escrito en estos lenguajes puede, en ciertos casos, sufrir una fase de compilación intermediaria, en un archivo escrito en un lenguaje ininteligible (por lo tanto, diferente al archivo fuente) y no ejecutable (requeriría un intérprete).

Como veremos más adelante, éste es el caso de Java, en el que se realiza un doble proceso de compilación e interpretación.

* 1. **Tipos de código generados por un lenguaje de programación**

Código fuente es el código escrito por un programador en un determinado lenguaje de programación, definiendo el conjunto de instrucciones que debe realizar la computadora.

Este código fuente, comprensible por el ser humano ya que ha sido escrito siguiendo unas reglas definidas por él mismo, debe ser traducido al lenguaje máquina del procesador. Este proceso de traducción se llama compilación y el resultado de dicho proceso es el código objeto.

Código objeto es el código máquina generado tras compilar un determinado código fuente.

Un archivo ejecutable es un programa completo, que puede ser ejecutado por el ordenador, obtenido después de enlazar varios códigos objetos y librerías.

* 1. **Paradigmas de programación.**

Ejemplos de paradigmas de programación son los siguientes:

* Imperativo
* Declarativo
* Estructurado
* Orientado a objetos
* Funcional
* Lógico
  1. **La programación estructurada.**

La programación estructurada tiene un teorema fundamental: Cualquier programa puede ser elaborado utilizando únicamente las tres estructuras básicas (secuencia, selección, iteración).

Entre las ventajas de la programación estructurada, se encuentran las siguientes:

* Programas más fáciles de entender.
* Reducción del esfuerzo de pruebas.
* Reducción de los costos de mantenimiento.
* Programas más sencillos y más rápidos.
* Aumento de la productividad del programador.
* Los programas quedan mejor documentados internamente.

Por el contrario, la programación estructurada acarrea algunos inconvenientes. El principal es que se obtiene un único bloque de programa, que cuando se hace demasiado grande puede resultar problemático en cuanto a su manejo, esto se resuelve empleando la programación modular.

* 1. **Programación orientada a objetos.**

La programación orientada a objetos se caracteriza por la idea central, el objeto. Podemos decir que cualquier cosa, en serio, cualquier cosa, que nos rodea es un objeto.

Las características fundamentales de este enfoque son las siguientes:

* Existe la posibilidad de representar directamente las entidades del mundo real en los escenarios informáticos, sin necesidad de deformarlas.
* Se facilita enormemente la reutilización y modificación del software.
* Se puede trabajar en equipo, desarrollando en paralelo módulos para una aplicación con un esfuerzo mínimo en coordinación.

Existen una serie de pasos que sirven de guía a la hora de modelar un sistema empleando orientación a objetos:

1. Identificar los objetos que intervienen en él.
2. Agrupar en clases a todos aquellos objetos que tengan características y comportamiento comunes.
3. Identificar los datos y operaciones\* de cada una de las clases.
4. Identificar las relaciones que puedan existir entre las clases.

La programación orientada a objetos presenta ciertas características:

* Protección de la información.
* Rápido desarrollo.
* Fácil mantenimiento.
* Polimorfismo.
* Herencia.

Un modelo de objetos es más cercano a la realidad que un modelo funcional. Además, el modelo orientado a objetos facilita la integridad de módulos que han sido realizados por separado para no correr riesgos en el manejo de los datos.

1. El lenguaje de programación Java.
   1. **Historia del lenguaje Java.**

Java fue lanzado por Sun en el otoño de 1995. Tres de las principales razones que llevaron a crear Java son:

* Creciente necesidad de interfaces mucho más cómodas e intuitivas que los sistemas de ventanas que proliferaban hasta el momento.
* Fiabilidad del código y facilidad de desarrollo. Gosling observó que muchas de las características que ofrecían C o C++ aumentaban de forma alarmante el gran coste de pruebas y depuración. Por ello en los sus ratos libres creó un lenguaje de programación donde intentaba solucionar los fallos que encontraba en C++.
* Enorme diversidad de controladores electrónicos. Los dispositivos electrónicos se controlan mediante la utilización de microprocesadores de bajo precio y reducidas prestaciones, que varían cada poco tiempo y que utilizan diversos conjuntos de instrucciones. Java permite escribir un código común para todos los dispositivos.
  1. **Características del lenguaje Java**
* Lenguaje simple.
* Orientado a objetos.
* Distribuido.
* Compilado e interpretado.
* Robusto.
* Seguro.
* Independiente a la arquitectura.
* Portable.
* Multihilo.
* Dinámico.

1. Fases en el desarrollo de una aplicación.
   1. **Metodologías para el desarrollo de software.**

En la literatura sobre este tema existen muchas definiciones sobre lo que es una metodología. Más o menos todas ellas coinciden en que debería tener al menos las siguientes características:

* Define como se divide un proyecto en fases y las tareas a realizar en cada una.
* Para cada una de las fases está especificado cuales son las entradas que reciben y las salidas que producen.
* Tienen alguna forma de gestionar el proyecto.

Teniendo esto en cuenta establecemos la siguiente definición:

Una metodología es un modo sistemático de producir software.

* 1. **Ciclo de vida del software.**

Usualmente se consideran las etapas: especificación y análisis de requisitos, diseño del sistema, implementación del software, aplicación y pruebas, entrega y mantenimiento.

Cada fase estará definida por un conjunto de tareas a realizar en ella y por el producto generado antes de pasar a la siguiente etapa. Este producto podrá ser considerado como un entregable o no, en función de si es para entregarlo al cliente como parte del proyecto o forma parte de la documentación interna.

Un aspecto esencial dentro de las tareas del desarrollo del software es la documentación de todos los elementos y especificaciones en cada fase. Cada etapa tiene como entrada uno o varios documentos procedentes de las etapas anteriores y produce otros documentos de salida.

Etapas:

* Análisis.
* Diseño.
* Codificación.
* Pruebas.
* Explotación.
* Mantenimiento.
  1. **Análisis.**

Durante esta fase, se estudia a fondo el problema que pretende resolverse, tratando de comprender necesidad del cliente y de definir las especificaciones del producto que pretende desarrollarse.

Tras esa primera descripción, el producto a desarrollar se define a través de requisitos que se van obteniendo a partir de las sucesivas reuniones con el cliente.

Un requisito puede definirse como la capacidad que el sistema debe tener porque el cliente lo ha pedido explícita o implícitamente.

La obtención de requisitos es difícil por varias razones:

* La naturaleza de los requisitos es cambiante.
* Surgen nuevos requisitos en cualquier momento.
* El cliente puede no tenerlos claros.
* Pueden existir malos entendidos debidos a:
* Falta de conocimientos por parte del equipo desarrollador sobre el problema.
* Falta de conocimientos técnicos (informáticos) por parte del cliente para expresarse con claridad.

Los requisitos se dividen en dos tipos:

* Requisitos funcionales: dicen qué debe hacer el sistema, en el sentido de servicios proporcionados al usuario.
* Requisitos no funcionales: hablan de características del sistema, como pueden ser la fiabilidad, la mantenibilidad, el sistema operativo, la plataforma hardware, el aspecto gráfico, etc.

Otros elementos importantes en la fase de análisis son los casos de uso, que constituyen otra forma de especificar los requisitos de un sistema. Un caso de uso:

* Describe la interacción entre un actor externo al sistema y el sistema con texto en lenguaje natural.
* Representa los requisitos funcionales desde el punto de vista del usuario y por lo tanto produce un resultado observable por él.
* Es iniciado por un único actor.
* Realiza una funcionalidad concreta.

Un proceso clave en la fase de análisis y toma de requisitos son las entrevistas con el cliente, a través de las cuales se van comprendiendo sus necesidades que, posteriormente, serán traducidas a requisitos y casos de uso. El proceso habitual suele ser:

En una reunión final, se dará por cerrado el análisis, aprobando el Documento de Análisis que contiene todos los requisitos y casos de uso, así como las especificaciones del sistema. Este documento se entiende como un contrato por el que el equipo de desarrollo se compromete a desarrollar una aplicación que realice exactamente las tareas que se recogen en dicho documento.

* 1. **Diseño.**

Para dejar constancia de los diseños se deben utilizar lenguajes lo más formales posible, como tablas, diagramas y pseudocódigo.

UML son las siglas de lenguaje unificado de modelado (unified modeling language). Se trata de un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema (en nuestro caso, un sistema software).

Por lo general, el producto de esta fase es el Documento de Diseño que, entre otros elementos, suele contener:

* Una descripción y un boceto del interfaz de la aplicación. En ocasiones se construyen maquetas sin funcionalidad.
* Diagrama UML de Componentes, indicando la arquitectura del sistema y las relaciones entre los distintos subsistemas (base de datos, interfaz, etc.)
* Diagrama UML de Clases, mostrando las distintas entidades que aparecen en el modelo de datos. Si la aplicación se va a realizar siguiendo la POO, se indican las clases, con sus atributos, métodos y relaciones entre ellas.
* Diagrama UML de Secuencias, mostrando el flujo de la aplicación en los distintos casos de uso.
* Descripción de cada una de las clases que conformarán el modelo de datos.
  1. **Implementación o codificación.**

Para realizar el trabajo, este debe ser dividido en pequeños módulos que luego se reparten entre individuos o pequeños grupos. Este reparto se realiza atendiendo a un gráfico de dependencias entre tareas, siempre con la idea en mente de paralelizar tanto trabajo como sea posible.

El producto de la fase de codificación es el programa propiamente dicho, listo para ser testeado en la fase de pruebas. Otros productos de esta fase son:

* Los manuales de configuración e instalación
* El manual de usuario
* La documentación del código, donde se recoge la funcionalidad de cada clase.
  1. **Pruebas.**

Las pruebas consisten en la comprobación de que la salida obtenida en cada fase corresponde a las especificaciones de entrada correspondientes.

Las pruebas que se van haciendo durante el ciclo de vida son:

* Pruebas unitarias
* Pruebas de integración
* Pruebas de sistema
* Pruebas de tensión
* Pruebas de desempeño
  1. **Documentación.**

Es muy importante distinguir entre:

* La documentación orientada a futuros desarrollos o modificaciones de mantenimiento (documentación de diseño, implementación y pruebas, manual técnico, manual de referencia),
* La documentación de uso y aplicación (introducción de uso rápido, manual de configuración, manual de usuario, manual de interfaz).

En un proyecto no es necesario que estén todos, sobre todo en proyectos pequeños, pero los documentos habituales son:

* Documentos previos de Ingeniería:
* Estudio de viabilidad
* Análisis de riesgo
* Análisis costo-beneficio
* Informe de decisión de sistemas
* Plan de proyecto
* Documentos de Desarrollo del Proyecto:
* Requisitos funcionales
* Requisitos de datos
* Especificaciones de sistema/subsistema
* Especificaciones del programa
* Especificaciones de la base de datos
* Plan de Pruebas
* Documentos de usuarios y de explotación:
* Manual de usuario
* Manual de operación
* Manual de mantenimiento
* Plan de instalación
  1. **Explotación y mantenimiento.**

Dentro del mantenimiento se deben incluir no solamente las correcciones de errores detectados posteriormente por el cliente, sino también las modificaciones necesarias para actualización, e incluso las peticiones de cambios por parte del cliente.

1. Metodologías ágiles.

El desarrollo ágil de software refiere a métodos basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan mediante la colaboración de grupos auto organizados y multidisciplinarios.

La mayoría desarrolla el software en lapsos cortos, lo que minimiza los riesgos.

Cada iteración del ciclo de vida incluye todas las fases tradicionales vistas anteriormente, pero realizadas de manera simultánea y, evidentemente, más ligera, ya que sólo debe realizarse una pequeña parte del

proyecto.

Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, sino que la meta es tener una «demo» (sin errores) al final de cada iteración.

Los métodos ágiles enfatizan las comunicaciones cara a cara en vez de la documentación.