TEMA 1

1. Introducción

El ordenador se compone de 2 partes bien diferenciadas: hardware (parte física) y software (parte lógica).

2. Software. Tipos de Software

Software: Conjunto de programas y datos que coexisten en la computadora.

Programas: Listado de instrucciones que indican al ordenador qué es lo que tiene que hacer.

Datos: Información con la que trabajan las instrucciones de los programas. Es un conjunto de información que interpretada por un programa adecuado representa números, texto, imágenes, sonido, vídeo, etc..

2.1. Tipo de software según su función.

Software de sistema: Es el que trabaja directamente con el hardware haciéndolo funcionar, y poniéndolo al servicio del resto del software y por tanto del usuario. Fundamentalmente son los sistemas operativos y los drivers.

Software de aplicación: Aquel que el usuario utiliza para realizar su trabajo (ofimática, contabilidad, etc)

Software de programación: Aquel utilizado para el desarrollo de programas.

Software de utilidades: Aquel utilizado para analizar, configurar, optimizar y mantener un equipo, incluyendo el hardware, sistema operativo, software de aplicaciones y almacenamiento de datos.

2.2. Tipo de software según licencia

• **Software Libre:** Esto implica que sigue las siguientes 4 libertades:

• Ejecutar el programa, para cualquier propósito

• Estudiar el funcionamiento del programa, y adaptarlo a sus necesidades.

• Redistribuir copias.

• Mejorar el programa, y poner sus mejoras a disposición del público, para beneficio de toda la comunidad.

• **Software de fuente abierta:** Cualquier programa cuyo código fuente se pone a disposición para su uso o modificación, conforme los usuarios u otros desarrolladores lo consideren conveniente:

• El software que se distribuye debe ser redistribuido a cualquier persona sin ningún tipo de restricción.

• El código fuente debe estar disponible (de modo que la parte receptora sea capaz de mejorarlo o modificarlo).

• La licencia puede requerir que versiones mejoradas del software lleven un nombre o una versión diferente del software original.

• **Software con copyleft:** No permiten a los redistribuidores agregar ninguna restricción adicional cuando lo redistribuyen o modifican. Es decir, la versión modificada debe tener las mismas restricciones que la original.

• **Freeware:** Se usa comúnmente para programas que permiten la redistribución pero no la modificación (y su código fuente no está disponible en ningún caso).

• **Shareware:** Software con autorización de redistribuir copias, pero debe pagarse cargo por licencia de uso continuado.

• **Software privativo:** Aquél cuyo uso, redistribución o modificación están prohibidos o necesitan una autorización.

• **Software comercial:** El desarrollado por una empresa que pretende ganar dinero por su uso.

3. Relación entre Hardware y Software.

Existe una relación entre el hardware y el software, ya que los dispositivos hardware necesitan estar instalados y configurados con el software correspondiente para que funcionen correctamente.

La relación software-hardware la podemos ver desde dos puntos de vista:

• **Desde el punto de vista del sistema operativo**: El sistema operativo es el intermediario entre el usuario y el hardware. Todas las aplicaciones necesitan recursos hardware durante su ejecución (tiempo de CPU, espacio en memoria RAM, tratamiento de interrupciones, gestión de los dispositivos de Entrada/Salida, etc.). Será siempre el sistema operativo el encargado de controlar todos estos aspectos de manera “transparente" para las aplicaciones (y para el usuario).

• **Desde el punto de vista de las aplicaciones:** Una aplicación no es otra cosa que un conjunto de programas escritos en algún lenguaje de programación (generalmente de alto nivel) que el hardware del equipo debe interpretar y ejecutar. Esto nos hace plantearnos una cuestión: ¿Cómo será capaz el ordenador de "entender" algo escrito en un lenguaje que no es el suyo?. Tiene que haber un proceso de traducción de código para que el ordenador ejecute las instrucciones del programa.

4. Desarrollo de Software

Proceso (ciclo de vida) que ocurre desde que se concibe una idea de programación hasta que un programa está implementado en el ordenador y funcionando. Este proceso consta de una serie de pasos de obligado cumplimiento, pues sólo así podremos garantizar que los programas creados sean eficientes, fiables, seguros y respondan a las necesidades de los usuarios finales.

4.1. Modelos de ciclo de vida del Software

Los modelos de ciclo de vida más utilizados son: Modelos en cascada y Modelos evolutivos.

4.1.1. Modelo en cascada

• **Modelo clásico en Cascada:** En este modelo para empezar una etapa es necesario finalizar la etapa anterior sin posibilidad de vuelta atrás (imposible usar en casi ningún proyecto).

• **Modelo en Cascada con Realimentación:** Extensión del modelo anterior, introduciendo una realimentación entre etapas, de forma que podamos volver atrás en cualquier momento para corregir, modificar o depurar algún aspecto. (modelo para proyectos rígidos).

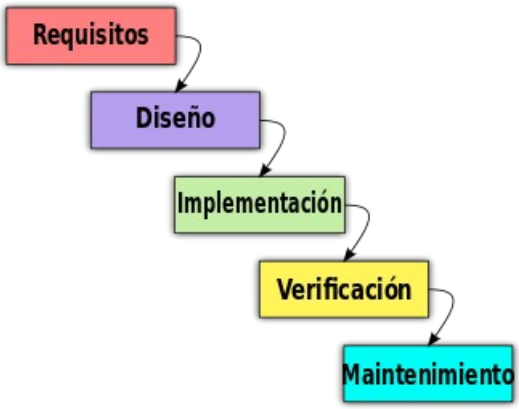
• **Ventajas modelos en cascada**:

Es fácil de comprender, planificar y seguir, la calidad del producto resultante es alta, permite personal poco cualificado.

• **Inconvenientes modelos en cascada**: Tenemos la necesidad de tener todos los requisitos definidos desde el principio, es difícil volver atrás si se comenten errores, y el producto no disponible hasta que está completamente terminado.

• **Recomendado en:** Proyectos similares a ya realizados con éxito anteriormente, con requisitos estables y bien comprendidos. Y además, cuando los clientes no necesitan versiones intermedias, por tanto, se entrega el producto final (sin versiones posteriores).

Un ejemplo visual sería:



4.1.2. Modelos evolutivos (Permiten desarrollar versiones sucesivas).

- **Modelo iterativo incremental:** Basado en el modelo en cascada con realimentación, donde las fases se repiten y refinan, y van propagando su mejora a las fases siguientes.

- **Modelo en Espiral:** Es una combinación del modelo iterativo con el modelo en cascada. 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.

4.1.2.1. Modelo iterativo incremental

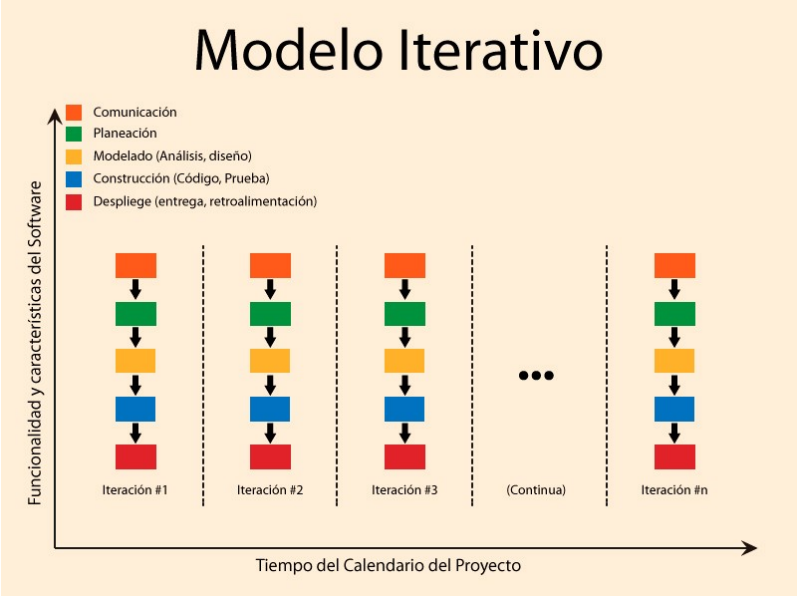
Se entrega el proyecto en partes pequeñas pero utilizables, llamadas “incrementos”, construido sobre el proyecto entregado anteriormente.

Ventajas: No se necesitan conocer todos los requisitos al comienzo y además permite entregar rápidamente partes operativas del producto final.

Inconvenientes: No podemos estimar el coste y esfuerzo final total, existe el riesgo de no acabar nunca y no es recomendable para sistemas en tiempo real y de alto nivel de seguridad

Recomendado cuando: Los requisitos no están definidos totalmente y es posible que haya grandes cambios. Y cuando se prueban o introducen nueva tecnologías.

Ejemplo visual:



4.1.2.2. Modelo espiral

Se parece al modelo Iterativo incremental teniendo en cuenta en cada ciclo el análisis de riesgos.

• **Determinación de objetivos**: Cada ciclo comienza identificando los objetivos y las restricciones

• **Análisis de riesgo**: Se identifican los riesgos (mal diseño, error de implementación, ...) y la manera de resolverlos.

• **Desarrollar y probar**: Verificar que la solución es aceptable.

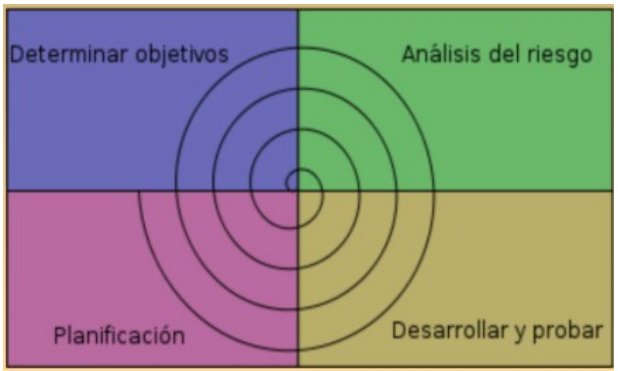
• **Planificacion**: Revisar y evaluar todo lo hecho y decidir si se continúa; en caso afirmativo, planificar fases del ciclo siguiente.

• **Ventajas**: No requiere definición completa de requisitos para empezar a funcionar. Se hace un análisis de riesgo en todas las etapas, por tanto, reduce riesgos del proyecto.

• **Inconvenientes**: El costo del proyecto aumenta a medida que la espiral crece, y es difícil evaluar riesgos, por tanto, el éxito del proyecto depende de la fase de análisis de riesgos.

• **Recomendado en** proyectos de gran tamaño que necesitan constantes cambios.

Ejemplo visual:



4.2. Herramientas de ayuda al desarrollo.

Las herramientas CASE son un conjunto de aplicaciones que se utilizan en el desarrollo de software con el objetivo de reducir costes y tiempo del proceso, mejorando por tanto la productividad del proceso. La tecnología CASE trata de automatizar las fases del desarrollo de software para que mejore la calidad del proceso y del resultado final. En concreto, estas herramientas 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, permitiendo visualizar las fases de forma gráfica

¿En qué fases del proceso nos pueden ayudar? En todas las etapas.

4.2.1. CLASIFICACIÓN de las herramientas CASE

Normalmente, las herramientas CASE se clasifican en función de las fases del ciclo de vida del software en la que ofrecen ayuda:

• **U-CASE:** ofrece ayuda en las fases de planificación y análisis de requisitos.

• **M-CASE:** ofrece ayuda en análisis y diseño.

• **L-CASE:** ayuda en la programación del software, detección de errores del código, depuración de programas y pruebas y en la generación de la documentación del proyecto.

Ejemplos de herramientas CASE libres son: ArgoUML, Use Case Maker, ObjectBuilder…

5. Lenguajes de programación. Tipos.

Lenguaje formal formado por símbolos y reglas sintácticas y semánticas, diseñado para realizar un código que posteriormente traducido puede ser entendido y ejecutado por computadoras.

5.1. Evolución de los 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). Único para cada procesador (no es portable).

• **Lenguaje ensamblador**: 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 distintos registros de la computadora.

• **Lenguaje de alto nivel basados en código**: En lugar de mnemotécnicos, se utilizan sentencias y órdenes derivadas del idioma inglés. (Necesita traducción al lenguaje máquina).

• **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. Completamente portables.

5.2 Clasificación de los lenguajes de programación

Podemos clasificar los distintos tipos de Lenguajes de Programación en base a distintas características:

5.2.1. Según lo cerca que esté del lenguaje humano

• Lenguajes de programación de alto nivel: Por su esencia, 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.

5.2.2. Según la técnica de programación utilizada

5.2.2.1. Lenguajes de programación estructurados

Ventajas de la programación estructurada:

• Los programas son fáciles de leer, sencillos y rápidos.

• El mantenimiento de los programas es sencillo.

• La estructura del programa es sencilla y clara.

Inconvenientes de la programación estructurada:

• Todo el programa se concentra en un único bloque (si se hace demasiado grande es difícil manejarlo).

• No permite reutilización eficaz de código, ya que todo va "en uno".

• Es por esto que a la programación estructurada le sustituyó la programación modular, donde los programas se codifican por módulos y bloques permitiendo mayor funcionalidad.

5.2.2.2. Lenguajes de Programación Orientados a Objetos:

Los lenguajes de programación orientados a objetos tratan a los programas no como un conjunto ordenado de instrucciones (como en la programación estructurada) sino como un conjunto de objetos que colaboran entre ellos para realizar acciones.

En la P.O.O. los programas se componen de objetos independientes entre sí que colaboran para realizar acciones. Los objetos son reutilizables para proyectos futuros.

Ventajas de la programación orientada a objetos:

• El código es completamente reutilizable.

• Si hay algún error, es más fácil de localizar y depurar en un objeto que en un programa entero.

Inconvenientes de la programación orientada a objetos:

• Los objetos requieren una extensa documentación.

• Los objetos al ser abstractos pueden no coincidir en la visión de un programador a otro.

Características de la POO:

• Los objetos del programa 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, pues, como unidades individuales e indivisibles que forman la base de este tipo de programación.

5.2.2.3. Lenguajes de programación visuales

Permiten programar gráficamente, siendo el código correspondiente generado de forma automática. Ejemplos: Visual Basic.Net, Borland Delphi

6. Fases del ciclo de vida del software

A la hora de desarrollar un proyecto, independientemente del modelo elegido, siempre hay una serie de etapas que debemos seguir (ciclo de vida) para construir software fiable y de calidad:

• Análisis

• Diseño

• Codificación y Obtención código ejecutable

• Ejecución y Pruebas

• Documentación

• Explotación

• Mantenimiento

6.1 - Análisis

Aunque no lo parezca, es la fase de mayor importancia y más complicada en el desarrollo del proyecto y todo lo demás dependerá de lo bien detallada que esté, dependiendo en gran medida del analista que la realice.

En esta fase se especifican y analizan los siguientes requisitos:

• **Funcionales**: 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 del sistema**: Tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.

La culminación de esta fase es el documento ERS (Especificación de Requisitos Software), en el que quedan especificados entre otros:

• 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, etc.

6.2 Diseño

En esta fase debemos dividir el sistema en partes, establecer qué relaciones habrá entre ellas y decidir qué hará exactamente cada parte.

En este punto, se establecerán:

• Entidades y relaciones

• Selección del Sistema Gestor de Base de Datos.

• Selección del lenguaje de programación a utilizar.

6.3 Codificación y obtención del ejecutable

Durante la codificación se realiza el proceso de programación; es realizada por el programador y tiene que cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.

Las características deseables de todo código son:

• **Modularidad**: que esté dividido en trozos más pequeños.

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

• **Legible**: cualquiera en un futuro pueda entender fácilmente el código, para ello debe estar bien documentado.

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

• **Portabilidad**: que se pueda implementar en cualquier equipo.

• **Fácil de mantener**: cualquier programador futuro que detecte un error, debe poder arreglarlo.

Durante esta fase, el código pasa por diferentes estados:

6.3.1. Código Fuente

Es el escrito por los programadores en algún lenguaje de programación de alto nivel.

Un aspecto muy importante en esta fase es la elaboración previa de un algoritmo, que lo definimos como un conjunto de pasos a seguir para obtener la solución del problema. El algoritmo lo diseñamos en pseudocódigo y facilitará la codificación posterior a algún lenguaje de programación

Para obtener el código fuente de una aplicación informática:

• Se debe partir de las etapas anteriores de análisis y diseño.

• Se diseñará un algoritmo que simbolice los pasos a seguir para la resolución del problema.

• Se elegirá una Lenguajes de Programación de alto nivel apropiado para las características del software que se quiere codificar.

• Se procederá a la codificación del algoritmo antes diseñado.

La culminación de la obtención de código fuente es un documento con la codificación de todos los módulos, funciones, bibliotecas y procedimientos.

6.3.2. Código Objeto

Es el código binario resultado de compilar o interpretar 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 es un código intermedio entre el código fuente y el ejecutable y sólo existe si el programa se compila, ya que si se interpreta (traducción línea a línea del código) se traduce y se ejecuta en un solo paso.

Es el resultado de traducir código fuente a un código equivalente formado porunos y ceros que aún no puede ser ejecutado directamente por la computadora. Es decir es un código binario generado a partir de un código fuente libre de errores sintácticos y semánticos que está distribuido en varios archivos, cada uno de los cuales corresponde a cada programa fuente compilado.

El proceso de traducción de código fuente a código objeto puede realizarse de dos formas:

• Compilación: El proceso de traducción se realiza sobre todo el código fuente, en un solo paso. Se crea código objeto que habrá que enlazar. El software responsable se llama compilador.

• Interpretación: El proceso de traducción del código fuente se realiza línea a línea y se ejecuta simultáneamente. No existe código objeto intermedio. El software responsable se llama intérprete. El proceso de traducción es más lento que en el caso de la compilación, pero es más fácil para un programador inexperto, ya que da la detección de errores es más detallada.

6.3.3. Código Ejecutable

Es el código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias. También es conocido como código máquina y ya sí es directamente entendible por la computadora. El código ejecutable es el resultado de enlazar los archivos de código objeto mediante un software llamado linker (enlazador) y obtener un único archivo ejecutable directamente en la computadora.

6.4 Pruebas

Una vez obtenido el software, la siguiente fase del ciclo de vida es la realización de pruebas. Normalmente, éstas 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. La realización de pruebas es imprescindible para asegurar la validación y verificación del software construido.

Entre todas las pruebas que se efectúan sobre el software podemos distinguir básicamente:

• PRUEBAS UNITARIAS: Consisten en probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente). JUnit es el entorno de pruebas para Java.

• PRUEBAS DE INTEGRACIÓN: Se realizan una vez que se han realizado con éxito las pruebas unitarias y consistirán en comprobar el funcionamiento del sistema completo: con todas sus partes interrelacionadas. La prueba final se denomina comúnmente Beta Test, ésta se realiza sobre el entorno de producción donde el software va a ser utilizado por el cliente.

6.5 Documentación, explotación y mantenimiento

Es importante documentar todas las fases para dar toda la información a los usuarios de nuestro software y poder acometer futuras revisiones del proyecto.

Una correcta documentación permitirá la reutilización de parte de los programas en otras aplicaciones, siempre y cuando se desarrollen con diseño modular.

Distinguimos tres grandes documentos en el desarrollo de software:

• Guía técnica

• Guía de uso

• Guía de instalación

A continuación se mostrará una tabla que explica a fondo estos documentos:

	GUÍA TÉCNICA	GUÍA DE USO	GUÍA DE INSTALACIÓN
Quedan reflejados:	• El diseño de la aplicación • La codificación de los programas • Las pruebas realizadas.	• Descripción de la funcionalidad de la aplicación • Forma de comenzar a ejecutar la aplicación • Ejemplos de uso del programación • Requerimientos software de la aplicación • Solución de los posibles problemas que se pueden presentar.	Toda la información necesaria para: • Puesta en marcha. • Explotación. • Seguridad del sistema.
¿A quién va dirigido?	Al personal técnico en informática (analistas y programadores)	A los usuarios que van a usar la aplicación (clientes).	Al personal informático responsable de la instalación, en colaboración a los usuarios que van a usar la aplicación (clientes).
¿Cual es su objetivo?	Facilitar un correcto desarrollo, realizar correciones en los programas y permitir un mantenimiento futuro.	Dar a los usuarios finales toda la información necesaria para utilizar la aplicación.	Dar toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa

La explotación es la fase en que los usuarios finales conocen la aplicación y comienzan a utilizarla. Por tanto, es la instalación, puesta a punto y funcionamiento de la aplicación en el equipo final del cliente.

Primero se instalan los programas creados en el ordendador/es del cliente, después se configuran (generalmente por el usuario final) y finalmente se verifica (por ejemplo con la beta test) que todo esté correcto.

El mantenimiento se define como el proceso de control, mejora y optimización del software. Su duración es la mayor en todo el ciclo de vida del software, ya que también comprende las actualizaciones y evoluciones futuras del mismo. Los tipos de cambios que hacen necesario el mantenimiento del software son los siguientes:

• Perfectivos: Para mejorar la funcionalidad del software.

• Evolutivos: El cliente tendrá en el futuro nuevas necesidades.

• Correctivos: La aplicación tendrá errores en el futuro que habrá que corregir.

La etapa de mantenimiento es la más larga de todo el ciclo de vida del software. Por su naturaleza, el software es cambiante y deberá actualizarse y evolucionar con el tiempo. Deberá ir adaptándose de forma paralela a las mejoras del hardware en el mercado y afrontar situaciones nuevas que no existían cuando el software se construyó. Además, siempre surgen errores que habrá que ir corrigiendo y nuevas versiones del producto mejores que las anteriores. Por todo ello, debe pactarse con el cliente un servicio de mantenimiento de la aplicación (que también tendrá un coste temporal y económico).

7. Máquinas virtuales.

Cuando el código fuente se compila se obtiene código objeto intermedio. Para ejecutarlo en cualquier máquina se requiere tener independencia respecto al hardware concreto que se vaya a utilizar. Para ello, la máquina virtual aísla la aplicación de los detalles físicos del equipo en cuestión. Una máquina virtual es un tipo especial de software cuya misión es separar el funcionamiento del ordenador de los componentes hardware instalados.

Con el uso de máquinas virtuales podremos desarrollar y ejecutar una aplicación sobre cualquier equipo, independientemente de las características concretas de los componentes físicos instalados, lo que garantiza la portabilidad de las aplicaciones.

Las funciones principales de una máquina virtual son las siguientes:

• Conseguir que las aplicaciones sean portables.

• Reservar memoria para los objetos que se crean y liberar la memoria no utilizada.

• 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.

7.1 Frameworks

Un framework es una plataforma software donde están definidos programas soporte, bibliotecas, lenguaje interpretado, etc., que ayuda a desarrollar y unir los diferentes módulos o partes de un proyecto.

Ventajas:

• Desarrollo rápido de software.

• Reutilización de partes de código para otras aplicaciones. Diseño uniforme del software.

• Portabilidad de aplicaciones de un computador a otro, ya que los bytecodes que se generan a partir del lenguaje fuente podrán ser ejecutados sobre cualquier máquina virtual.

Inconvenientes:

• Gran dependencia del código respecto al framework utilizado (si cambiamos de framework, habrá que reescribir gran parte de la aplicación).

• La instalación e implementación del framework en nuestro equipo consume bastantes recursos del sistema.

7.2 Entornos de ejecución

Un entorno de ejecución es un servicio de máquina virtual que sirve como base software para la ejecución de programas. Es decir, es un conjunto de utilidades que permiten la ejecución de programas.

Durante la ejecución, los entornos se encargarán de:

• Configurar la memoria principal disponible en el sistema.

• Enlazar los archivos del programa con las bibliotecas existentes y con los subprogramas creados.

• Depurar los programas, comprobando la existencia (o no existencia) de errores semánticos (los sintácticos se detectan durante la compilación).

Funcionamiento del entorno de ejecución:

El Entorno de Ejecución está formado por la máquina virtual y los API's (bibliotecas de clases estándar, necesarias para que la aplicación, escrita en algún Lenguaje de Programación pueda ser ejecutada).