UNIDAD 1 - DESARROLLO DE SOFTWARE

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

1. Software y tipos

- **Software.** Conjunto de programas y datos que coexisten en el ordenador.
- **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, etc.

1.1 Tipos de Software según licencia

- **Software libre.** Implica 4 libertades:
  - Ejecutar el programa.
  - Estudiar el funcionamiento y su adaptación según necesidades.
  - Redistribuir copias.
  - Mejorar el programa, poniendo sus mejoras a disposición del público para beneficio de la comunidad.
- **Software de código abierto.** Es cualquier programa cuyo código fuente se pone a disposición para su uso o modificación por parte de usuarios o desarrolladores:
  - Debe ser redistribuido sin restricciones.
  - El código fuente debe estar disponible (para mejoras o modificaciones).
  - La licencia puede que las versiones mejoradas lleven un nombre o versión distinta del software original.
- **Software con copyleft.** No se pueden agregar restricciones adicionales al producto original o a su modificación al redistribuirlo, es decir, la versión modificada debe tener las mismas restricciones que la original.
- **Freeware.** Se usa para programas que permiten la redistribución pero no la modificación. Su código fuente no está disponible en ningún caso.
- **Shareware.** Software con autorización para redistribuir copias, pero debe pagarse cargo por licencia de uso.
- **Software privativo.** Aquel cuyo uso, redistribución o modificación están prohibidos o necesitan autorización.
- **Software comercial.** Desarrollado por una empresa que pretende ganar dinero por su uso.

2. Relación hardware-software

Existe una relación entre hardware y software: los dispositivos hardware necesitan estar instalados y configurados con el software correspondiente para que funcionen correctamente. Existen dos puntos de vista:

- **Punto de vista del SO.** El SO es el intermediario entre el usuario y el hardware. Todas las apps necesitan recursos hardware durante su ejecución, y será siempre el SO el encargado de controlar todos estos aspectos de manera transparente para las apps y el usuario.
- **Punto de vista de las apps.** Una app no es otra cosa que un conjunto de programas escritos en algún lenguaje de programación (generalmente de alto nivel) que el hardware debe interpretar y ejecutar. Asimismo, existe un proceso de traducción de código para que el ordenador ejecute las instrucciones del programa y "entienda" un lenguaje que no es el suyo.

3. Desarrollo de software

Se trata de un 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. Consta de una serie de pasos de obligado cumplimiento, garantizando que sean eficientes, fiables, seguros y que respondan a las necesidades de los usuarios finales.

3.1 Modelos de Ciclo de vida del software

3.1.1. Modelos en Cascada

- **Modelo clásico en Cascada.** Para empezar una etapa es necesario finalizar la anterior, sin posibilidad de vuelta atrás.
- **Modelo en Cascada con Realimentación.** Extensión del modelo anterior, que incluye una realimentación entre etapas, pudiendo volver atrás en cualquier momento para corregir, modificar o depurar algún aspecto. Se usa para proyectos rígidos.

3.1.2. Características de los modelos en cascada

Sus **ventajas** son las siguientes:

- Fácil de comprender, planificar y seguir.
- Alta calidad del producto.
- Permite personal poco cualificado.

Sus principales **inconvenientes** son:

- Necesidad de tener todos los requisitos definidos desde el principio.
- Difícil volver atrás si se cometen errores.
- Producto no disponible hasta que esté completamente terminado.

Está **recomendado** su uso en:

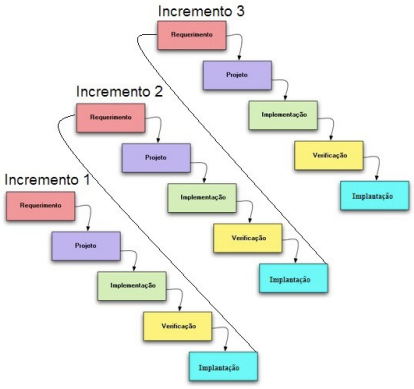
- Proyectos similares a otros ya realizados anteriormente con éxito.
- Requisitos estables y bien comprendidos.

- Cuando el cliente no precisa versiones intermedias, solo el producto final.

3.3.1. Modelos evolutivos

- **Modelo iterativo incremental**. Basado en el modelo en cascada con realimentación, donde las fases se repiten y refinan, propagando su mejora a las fases siguientes. Se entrega el proyecto en partes pequeñas pero utilizables llamadas "incrementos", construido sobre el proyecto entregado anteriormente.

  Sus **ventajas** son:

  - No se necesitan conocer todos los requisitos al comienzo.
  - Permite entregar rápidamente partes operativas del producto final.

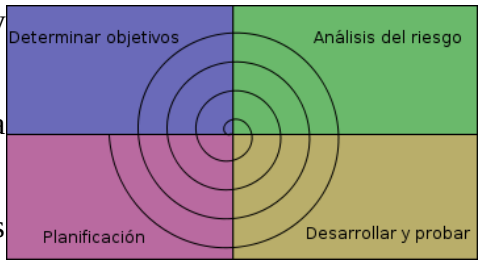
  Sus **inconvenientes** son:

  - No se puede estimar el coste y esfuerzo final total.
  - Riesgo de no acabar nunca.
  - No recomendable para sistemas en tiempo real o de alto nivel de seguridad.

  Recomendado cuando:

  - Los requisitos no están definidos totalmente y es posible que haya grandes cambios.
  - Se prueban o introducen nuevas tecnologías.

- **Modelo en espiral**. Es una combinación del modelo iterativo con el modelo en cascada, donde cada ciclo además tiene en cuenta el análisis de riesgos. El software se va construyendo repetidamente en forma de versiones que son cada vez mejores, debido a que incrementan funcionalidades. Se trata de un modelo complejo.

  - Determinación de objetivos: Se identifican objetivos y restricciones.
  - Análisis de riesgo: Se identifican los riesgos y la manera de resolverlos.
  - Desarrollo y prueba: Se verifica que la solución es aceptable.
  - Planificación: Se revisa y evalúa todo lo hecho y se decide si se continúa. En caso afirmativo, se vuelve al punto 1, planificando fases del ciclo siguiente.

Sus principales **ventajas** son:

- No requiere definición completa de requisitos para empezar a funcionar.
- Análisis de riesgo en todas las etapas.

Los inconvenientes son:

- El coste del proyecto aumenta a medida que la espiral crece.
- Difícil evaluar riesgos y, por tanto, el éxito del proyecto depende de la fase de análisis de riesgos.

Está recomendado cuando:

- Proyectos de gran tamaño con constantes cambios.

4. Herramientas de ayuda al desarrollo

4.1 Herramientas de ayuda al desarrollo

Las herramientas CASE son un conjunto de aplicaciones usadas para reducir costes y tiempo del proceso, mejorando la productividad. Estas herramientas tratan de automatizar fases del desarrollo para mejorar la calidad del proceso y su resultado. Permiten:

- Mejorar la planificación.
- Mejorar y agilizar el proceso de desarrollo.
- Reutilizar software en futuros proyectos.
- Hacer que las apps respondan a estándares.
- Mejorar el mantenimiento de programas.

4.2. Clasificación de herramientas CASE

Se clasifican en función de las fases del ciclo de vida del software en la que ayudan:

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

Algunas herramientas son ArgoUML, ObjectBuilder, etc.

5. Lenguajes de programación

Un lenguaje de programación es un lenguaje formal, formado por símbolos y reglas sintácticas y semánticas, diseñado para realizar un código que, tras ser traducido, puede ser entendido y ejecutado por ordenadores.

5.1. Evolución de los lenguajes de programación

- **Lenguaje máquina.** Sus instrucciones son código binario, el único que entiende el ordenador. No necesita traducción, y es único para cada procesador, es decir, no es portable.
- **Lenguaje ensamblador.** Se programa usando mnemotécnicos (instrucciones complejas). Necesita traducción al lenguaje máquina para poder ejecutarse. Sus instrucciones son sentencias que hacen referencia a distintos registros del ordenador.

- **Lenguaje de alto nivel basado en código.** En lugar de mnemotécnicos, se utilizan sentencias y órdenes derivadas del inglés (que después 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, generándose luego el código automáticamente. Necesitan traducción al lenguaje máquina y son completamente portables.

5.2. Clasificación de los lenguajes de programación

Clasificación en base a lo cerca que está del lenguaje humano:

- **Lenguajes de Programación de alto nivel**: más próximos al razonamiento humano.
- Lenguajes de Programación de **bajo nivel**: más próximos al funcionamiento interno del ordenador, como son el lenguaje ensamblador y máquina.

Clasificación en base a la técnica de programación utilizada:

- Lenguajes de programación **Estructurados:** Pascal, C, etc.
- Lenguajes de Programación **Orientados a Objetos:** C++, Java, Ada.
- Lenguajes de programación **Visuales:** Visual Basic.Net, Borland Delphi.

5.3. Programación estructurada:

Sus principales ventajas son:

- Programas fáciles de leer, sencillos y rápidos.
- Mantenimiento de programas sencillo.
- Estructura de programa sencilla y clara.

Sus principales inconvenientes son:

- Todo el programa se concentra en un único bloque, haciéndolo difícil de manejar a medida que crece.
- Menor funcionalidad que la programación modular, donde se codifica por módulos y bloques.

5.4. Programación orientada a objetos

Se trata a los programas como un conjunto de objetos independientes entre sí y reutilizables, que colaboran entre ellos para realizar acciones. Sus características más notorias son:

- Los objetos tendrán una serie de atributos diferenciadores.
- Se define clase como una colección de objetos con similares características.
- Mediante los métodos, los objetos se comunican con otros, pudiendo producir cambios en su estado.
- Los objetos son unidades individuales e indivisibles que forman la base de esta programación.

Sus principales ventajas son:

- El código es completamente reutilizable.
- En caso de error, es más fácil de localizar y depurar.

Sus principales inconvenientes son:

- Es necesaria una extensa documentación.
- Los objetos, al ser abstractos, no tienen por qué coincidir entre distintos programadores.

6. Fases del ciclo de vida del software

A la hora de desarrollar un proyecto, hay una serie de etapas que debemos seguir, el denominado ciclo de vida:

6.1. Análisis

Es la fase más difícil y de mayor importancia del proyecto, ya que todo lo demás dependerá de lo detallada que esté. Se especifican y analizan los requisitos:

- **Funcionales.** Qué funciones tendrá que realizar la aplicación, qué respuesta dará ante todas las entradas y cómo se comportará ante situaciones inesperadas.
- **No funcionales.** 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 todos los aspectos a tener en cuenta en el análisis, como:

- La planificación de reuniones.
- Relación de los objetivos del usuario cliente y del sistema.
- Relación de los requisitos funcionales y no funcionales.
- Relación de objetivos prioritarios y temporización.
- Reconocimiento de requisitos mal planteados o que conllevan contradicciones.

6.2. Diseño

En esta fase se debe dividir el sistema en partes, se establece qué relaciones habrá entre ellas y se decide qué hará exactamente cada una. Se fijarán:

- Entidades y relaciones
- Selección del Sistema Gestor de Base de Datos
- Selección del lenguaje de programación.

6.3. Codificación y obtención del ejecutable

Durante la codificación se realiza el proceso de programación, el cual debe cumplir con todos los datos impuestos en el análisis y en el diseño. Las características deseables del código son:

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

- **Corrección:** que haga lo que se pide realmente.
- **Legible:** cualquier programador en un futuro deberá entenderlo fácilmente, lo que implica una buena documentación.
- **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:

- **Código Fuente:** escrito por los programadores en un código de programación alto.
- **Código Objeto:** código binario resultado de compilar o interpretar el código fuente. La compilación es la traducción de una sola vez del programa, realizada usando 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, se traduce y ejecuta en un solo paso.
- **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 si es directamente entendible por el PC.

Código Fuente

Un aspecto muy importante en esta fase es la elaboración previa de un algoritmo, definido 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.

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

Código objeto

Es el resultado de traducir código fuente a un código equivalente formado por unos y ceros que aún no puede ser ejecutado directamente por el ordenador. Es decir, es un código binario generado a partir de un código fuente libre de errores sintácticos y semánticos 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:** La traducción se realiza sobre todo el código fuente, en un solo paso. Se crea código objeto que habrá de 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 la detección de errores es más detallada.

Código ejecutable

Es el resultado de enlazar los archivos de código objeto mediante un software linker o enlazador y obtener un único archivo ejecutable directamente por el ordenador.

6.4. Pruebas

Una vez obtenido el software, la siguiente fase consiste en la realización de pruebas. Realizadas sobre un conjunto de datos de prueba, que consisten en un conjunto de datos seleccionados y predefinidos que pretenden llevar al límite al software.

Este paso es esencial para asegurar la validación y verificación del software construido. Se dividen en:

- **Pruebas unitarias.** Prueba una a una las diferentes partes del software y comprueban su funcionamiento por separado y de manera independiente. Un ejemplo de Java es Junit.
- **Pruebas de integración.** Se realizan una vez se han realizado las pruebas unitarias y consisten en la comprobación del sistema completo.

La prueba final se denomina Beta Test, realizada sobre el entorno de producción donde el software va a ser usado 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 realicen con diseño modular.

Se distinguen tres tipos de documentos en el desarrollo de software:

- **Guía técnica.** Refleja el diseño de la aplicación, la codificación de programas y las pruebas realizadas. Va destinada al personal técnico informático para facilitar un correcto desarrollo, correcciones y mantenimiento.
- **Guía de uso.** Describe la funcionalidad de la aplicación, cómo se ejecuta, ejemplos de uso, requerimientos de la aplicación y solución a posibles problemas. Va destinada a los usuarios que van a usar la aplicación para facilitar toda la información necesaria para su uso.
- **Guía de instalación.** Tiene toda la información necesaria para su puesta en marcha, explotación y seguridad del sistema. Va destinada al personal responsable de su instalación y su intención es dar toda la información necesaria para su implantación segura, confiable y precisa.

La **explotación** es la fase en la que los usuarios finales conocen la aplicación y comienzan a usarla. Se trata de la instalación, puesta a punto y funcionamiento de la aplicación en el equipo del cliente.

Primero se instalan los programas creados en el equipo del cliente, se configura y se verifica 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:** Mejoran la funcionalidad del software.
- **Evolutivos:** Nuevas necesidades del cliente.
- **Correctivos:** Corrección de errores 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 de hardware en el mercado y afrontar nuevas situaciones 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 tendrá un coste temporal y económico.

7. Máquinas virtuales

Cuando el código fuente se compila, se obtiene el código objeto intermedio. Para ejecutarlo en cualquier máquina, es necesaria una independencia respecto al hardware concreto que se vaya a usar. 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 de los componentes físicos instalados, lo que garantiza la portabilidad de aplicaciones.

Éstas ofrecen:

- Portabilidad de las aplicaciones.
- Eficiencia de memoria.
- Comunicación con el sistema donde se instala para controlar el hardware implicado.
- Cumplimiento de las normas de seguridad de las aplicaciones.

7.1. Frameworks

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

Sus principales ventajas son:

- Desarrollo rápido de software.
- Reutilización de partes de código para otras aplicaciones. Diseño uniforme de software.
- Portabilidad de aplicaciones de un ordenador a otro.

Sus principales inconvenientes son:

- Gran dependencia del código respecto al framework usado.
- Los recursos del sistema usados en la instalación e implementación del framework.

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 del sistema.
- Enlazar los archivos con las bibliotecas y subprogramas creados.
- Depurar los programas, comprobando la existencia de errores semánticos.

Un entorno de ejecución está formado por la máquina virtual y los APIs (bibliotecas de clase estándar, necesarias para que la aplicación pueda ser ejecutada).