TEMA I

DESARROLLO DEL SOFTWARE

1.- Software y programa. Tipos de software.

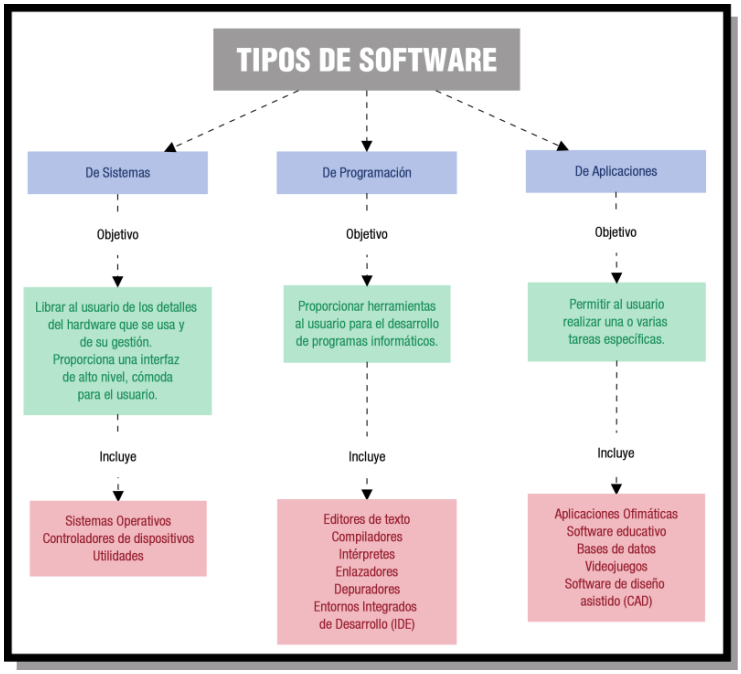
Es de sobra conocido que el ordenador se compone de dos partes bien diferenciadas: ***hardware*** y ***software***. El software es el conjunto de programas informáticos que actúan sobre el hardware para ejecutar lo que el usuario desee.

Según su función se distinguen tres tipos de software: sistema operativo, software de programación y aplicaciones.

- El sistema operativo es el software base que ha de estar instalado y configurado en nuestro ordenador para que las aplicaciones puedan ejecutarse y funcionar. Son ejemplos de sistemas operativos: Windows, Linux, Mac OS X …
- El software de programación es el conjunto de herramientas que nos permiten desarrollar programas informáticos.
- Las aplicaciones informáticas son un conjunto de programas que tienen una finalidad más o menos concreta. Son ejemplos de aplicaciones: un procesador de textos, una hoja de cálculo, el software para reproducir música, un videojuego, etc.

A su vez, un programa es un conjunto de instrucciones escritas en un lenguaje de programación.

En definitiva, distinguimos los siguientes tipos de software:



En este tema, nuestro interés se centra en las aplicaciones informáticas: cómo se desarrollan y cuáles son las fases por las que necesariamente han de pasar. A lo largo de esta primera unidad vas a aprender los conceptos fundamentales de software y las fases del llamado ciclo de vida de una aplicación informática.
También aprenderás a distinguir los diferentes lenguajes de programación y los procesos que ocurren hasta que el programa funciona y realiza la acción deseada.

2.- Relación hardware-software.

Existe una relación indisoluble entre el hardware y el software, ya que necesitan estar instalados y configurados correctamente para que el equipo funcione. El software se ejecutará sobre los dispositivos físicos. Ya hemos dicho que una aplicación no es otra cosa que un conjunto de programas, y que éstos están escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar.

Hay multitud de lenguajes de programación diferentes (como ya veremos en su momento). Sin embargo, todos tienen algo en común: estar escritos con sentencias de un idioma que el ser humano puede aprender y usar fácilmente. Por otra parte, el hardware de un ordenador sólo es capaz de interpretar señales eléctricas (ausencias o presencias de tensión) que, en informática, se traducen en secuencias de 0 y 1 (código binario).

Esto nos hace plantearnos una cuestión: ¿Cómo será capaz el ordenador de "entender" algo escrito en un lenguaje que no es el suyo?
Como veremos a lo largo de esta unidad, tendrá que pasar algo (un proceso de traducción de código) para que el ordenador ejecute las instrucciones escritas en un lenguaje de programación.

3.- Desarrollo del software.

Entendemos por Desarrollo de Software todo el proceso que ocurre desde que se concibe una idea hasta que un programa está implementado en el ordenador y funcionando

El proceso de desarrollo, que en un principio puede parecer una tarea simple, consta de una serie de pasos “de obligado cumplimiento”, sólo así podremos garantizar que los programas creados son eficientes, fiables, seguros y responden a las necesidades de los usuarios finales.

A esos pasos o etapas se las denomina **Ciclo de vida del software**. Según el orden y la forma en que se lleven a cabo estos pasos hablaremos de diferentes ciclos de vida del software.

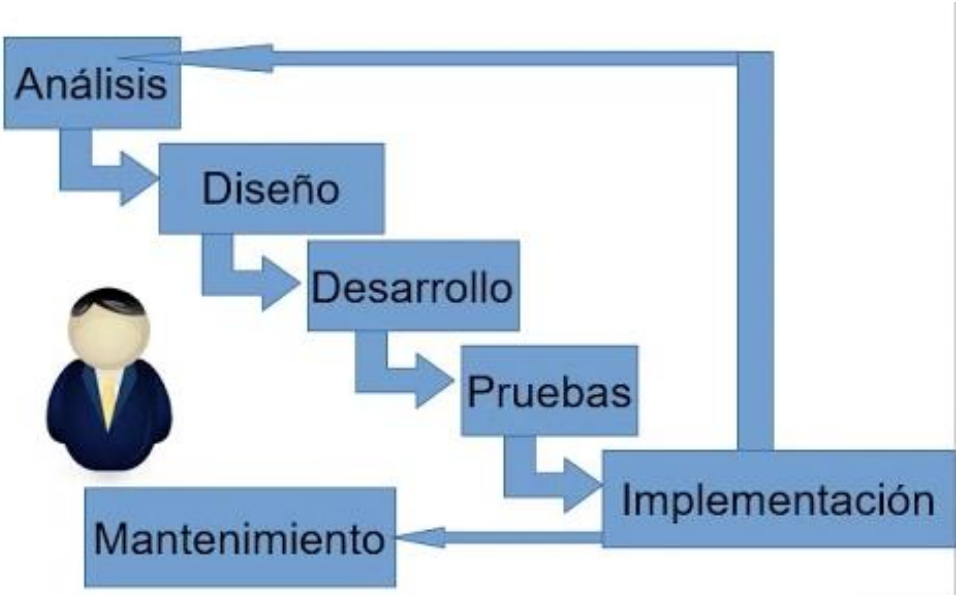
Según estimaciones, el 26% de los grandes proyectos de software fracasan, el 48% deben modificarse drásticamente y sólo el 26% tienen rotundo éxito. La principal causa del fracaso de un proyecto o es la falta de una buena planificación de las etapas y mala gestión de los pasos a seguir.

3.1.- Ciclos de vida del software.

Diversos autores han planteado distintos modelos de ciclos de vida, pero los más conocidos y utilizados son los que aparecen a continuación:

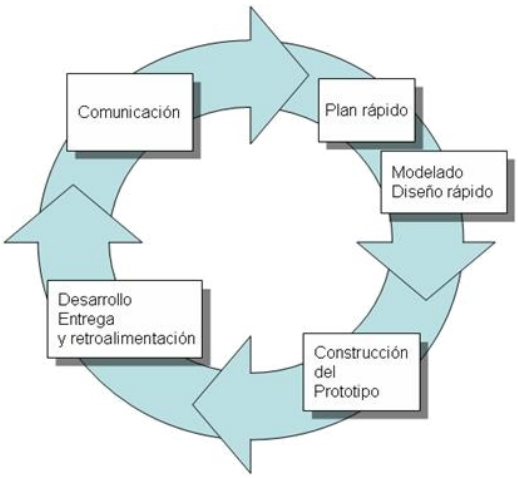
1. Modelos Clásicos (En cascada o en V)

Es el modelo de vida clásico del software. Requiere conocer de antemano todos los requisitos del sistema. Las etapas pasan de una a otra sin retorno posible (se presupone que no habrá errores ni variaciones del software).



2. Prototipos

Se define el problema**.** Se elabora un prototipo inicial de la aplicación, el cliente lo evalúa y se amplía con las aportaciones del cliente, así sucesivamente hasta obtener la solución definitiva. Es útil cuando el cliente interactúa mucho con el sistema.



3. Modelos Evolutivos

Son más modernos que los anteriores. Tienen en cuenta la naturaleza cambiante y evolutiva del software. Distinguimos dos variantes:

1. Modelo Iterativo Incremental. Está basado en el modelo en cascada, donde las fases se repiten y refinan, y van propagando su mejora a las fases siguientes.

2. Modelo en Espiral. Es una combinación del modelo anterior con el modelo en cascada. En él, 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. Metodologías ágiles

Son métodos de ingeniería del software basados en el desarrollo iterativo e incremental. Los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. El trabajo es realizado mediante la colaboración de equipos auto-organizados, inmersos en un proceso compartido de toma de decisiones a corto plazo.

Por definición, las **metodologías ágiles** son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno.

Las ventajas que nos brinda la gestión ágil de proyectos:

- Mejora de la calidad del producto: Estas metodologías fomentan el enfoque proactivo de los miembros del equipo. Además, la integración, comprobación y mejora continua de las propiedades del producto mejora considerablemente el resultado final.
- Mayor satisfacción del cliente: El cliente está más satisfecho al verse involucrado y comprometido a lo largo de todo el proceso de desarrollo. Mediante varias demostraciones y entregas, el cliente vive a tiempo real las mejoras introducidas en el proceso.
- Mayor motivación de los trabajadores: Los equipos de trabajo autogestionados, facilitan el desarrollo de la capacidad creativa y de innovación entre sus miembros.
- Trabajo colaborativo: La división del trabajo por distintos equipos y roles junto al desarrollo de reuniones frecuentes, permite una mejor organización del trabajo.
- Uso de métricas más relevantes: Las métricas utilizadas para estimar parámetros como tiempo, coste, rendimiento, etc. son normalmente más reales en proyectos ágiles que en

los tradicionales. Gracias a la división en pequeños equipos y fases podemos ser más conscientes de lo que está sucediendo.

- Mayor control y capacidad de predicción: La oportunidad de revisar y adaptar el producto a lo largo del proceso ágil, permite a todos los miembros del proyecto ejercer un mayor control sobre su trabajo, cosa que permite mejorar la capacidad de predicción en tiempo y costes.
- Reducción de costes: La gestión ágil del proyecto elimina prácticamente la posibilidad de fracaso absoluto en el proyecto, porque los errores se van identificando a lo largo del desarrollo en lugar de esperar a que el producto esté acabado y toda la inversión realizada.

https://www.youtube.com/watch?v=uxlOPJC3NzY

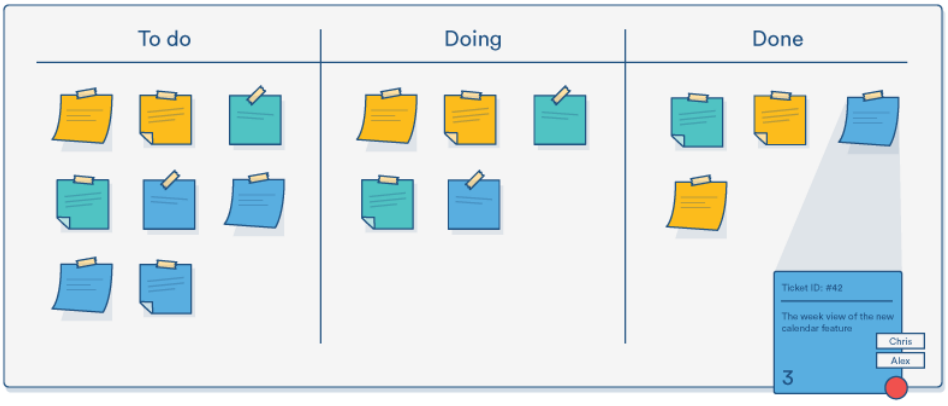
Algunas de las metodologías más conocidas son:

- Kanban (Trello)
- Scrum
- XP (eXtreme Programming)

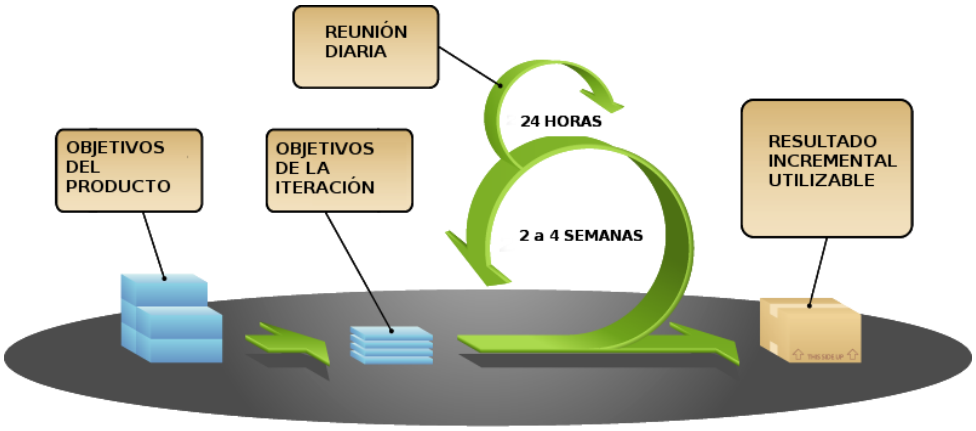
https://www.youtube.com/watch?v=DT2NEBCAPHw&t=24s

Kanban: También se denomina "sistema de tarjetas". Desarrollado inicialmente por Toyota para la industria de fabricación de productos. Controla por demanda la fabricación de los productos necesarios en la cantidad y tiempo necesarios. Enfocado a entregar el máximo valor para los clientes, utilizando los recursos justos.

Pizarra kanban

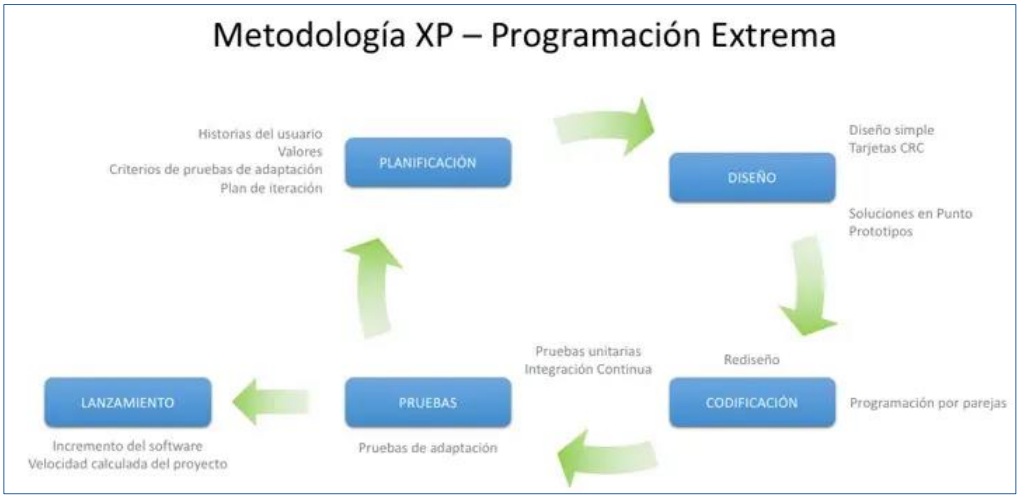


Scrum: Modelo de desarrollo incremental. Iteraciones (**sprint**) regulares cada 2 a 4 semanas. Al principio de cada iteración se establecen sus **objetivos priorizados** (**sprint acklog**). Al finalizar cada iteración se obtiene una **entrega parcial utilizable por el cliente**. Existen reuniones diarias para tratar la marcha del sprint.



XP (Programación extrema)

Extreme Programming o XP Programming es un marco de desarrollo de software ágil que tiene como objetivo producir un software de mayor calidad para mejorar la eficiencia del equipo de desarrollo. ... A pesar de ello, se trata de una de las metodologías ágiles de desarrollo de software más exitosas. Sus principales características: **Diseño sencillo, pequeñas mejoras continuas, pruebas y refactorización, Integración continua, e**l cliente se integra en el equipo de desarrollo….

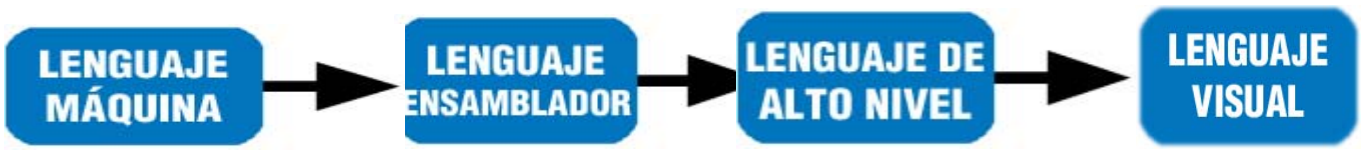


4.- Lenguajes de programación.

Ya dijimos anteriormente que los programas informáticos están escritos usando algún lenguaje de programación. Por tanto, podemos definir un Lenguaje de Programación como un idioma creado de forma artificial, formado por un conjunto de símbolos y normas que se aplican para obtener un código que el hardware de la computadora pueda entender y ejecutar.
Los lenguajes de programación son los que nos permiten comunicarnos con el hardware del ordenador. Son los instrumentos que tenemos para que el ordenador realice las tareas que necesitamos.

Hay multitud de lenguajes de programación, cada uno con unos símbolos y unas estructuras diferentes. Además, cada lenguaje está enfocado a la programación de tareas o áreas determinadas. Por ello, la elección del lenguaje a utilizar en un proyecto es una cuestión de extrema importancia.

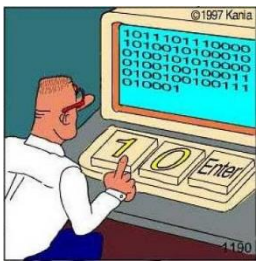
Los lenguajes de programación han sufrido su propia evolución, como se puede apreciar en la figura siguiente:



Características de los Lenguajes de Programación

Lenguaje máquina:

- Sus instrucciones son combinaciones de unos y ceros.
- Es el único lenguaje que entiende directamente el ordenador.
- Fue el primer lenguaje utilizado.
- Es único para cada procesador (no es portable de un equipo a otro).
- Hoy día nadie programa en este lenguaje.



Lenguaje ensamblador:

- Sustituyó al lenguaje máquina para facilitar la labor de programación.
- 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 la ubicación física de
- los archivos en el equipo.
- Es difícil de utilizar.

```
00001A1E 4D 4B       LDR             R3, =(stdout_ptr - 0xC000)
00001A20 E3 58       LDR             R3, [R4,R3] ; stdout
00001A22 1B 68       LDR             R3, [R3]
00001A24 18 46       MOV             R0, R3  ; stream
00001A26 FF F7 52 EA BLX             fileno
00001A2A 03 46       MOV             R3, R0
00001A2C 18 46       MOV             R0, R3
00001A2E 4A 4B       LDR             R3, =(a1ChangeDisplay - 0x1
00001A30 7B 44       ADD             R3, PC  ; "1 ) Change displ
00001A32 19 46       MOV             R1, R3
00001A34 00 F0 34 FB BL              print
00001A38 48 4B       LDR             R3, =(stdin_ptr - 0xC000)
00001A3A E3 58       LDR             R3, [R4,R3] ; stdin
00001A3C 1B 68       LDR             R3, [R3]
00001A3E 18 46       MOV             R0, R3  ; stream
00001A40 FF F7 44 EA BLX             fileno
00001A44 02 46       MOV             R2, R0
00001A46 07 F5 43 63 ADD.W           R3, R7, #0xC30
00001A4A 10 46       MOV             R0, R2
00001A4C 19 46       MOV             R1, R3
00001A4E 4F F0 02 02 MOV.W           R2, #2
00001A52 4F F0 0A 03 MOV.W           R3, #0xA
00001A56 00 F0 77 FB BL              read
00001A5A 03 46       MOV             R3, R0
00001A5C 00 2B       CMP             R3, #0
```

Lenguaje de alto nivel basados en código:

- Sustituyeron al lenguaje ensamblador para facilitar más la labor de
- programación.
- En lugar de mnemotécnicos, se utilizan sentencias y órdenes derivadas del
- idioma inglés. (Necesita traducción al lenguaje máquina).
- Son más cercanos al razonamiento humano.
- Son utilizados hoy día, aunque la tendencia es que cada vez menos.

También podemos clasificarlos según la técnica de programación utilizada:

- Lenguajes de Programación Estructurados: Usan la técnica de programación estructurada. Ejemplos: Pascal, C, etc.
- Lenguajes de Programación Orientados a Objetos: Usan la técnica de programación orientada a objetos. Ejemplos: C++, Java, C#.
- Lenguajes de Programación Visuales: Basados en las técnicas anteriores, permiten programar gráficamente, siendo el código correspondiente generado de forma automática. Ejemplos: .Net...

5.- Fases en el desarrollo y ejecución del software.

Ya hemos visto en puntos anteriores que debemos elegir un modelo de ciclo de vida para el desarrollo de nuestro software. Independientemente del modelo elegido, siempre hay una serie de etapas que debemos seguir para construir software fiable y de calidad.

Estas etapas son:

1. ANÁLISIS DE REQUISITOS.

Se especifican los requisitos funcionales y no funcionales del sistema.

2. DISEÑO.

Se divide el sistema en partes y se determina la función de cada una.

3. CODIFICACIÓN.

Se elige un Lenguajes de Programación y se codifican los programas.

4. PRUEBAS.

Se prueban los programas para detectar errores y se depuran.

5. DOCUMENTACIÓN.

De todas las etapas, se documenta y guarda toda la información.

6. EXPLOTACIÓN.

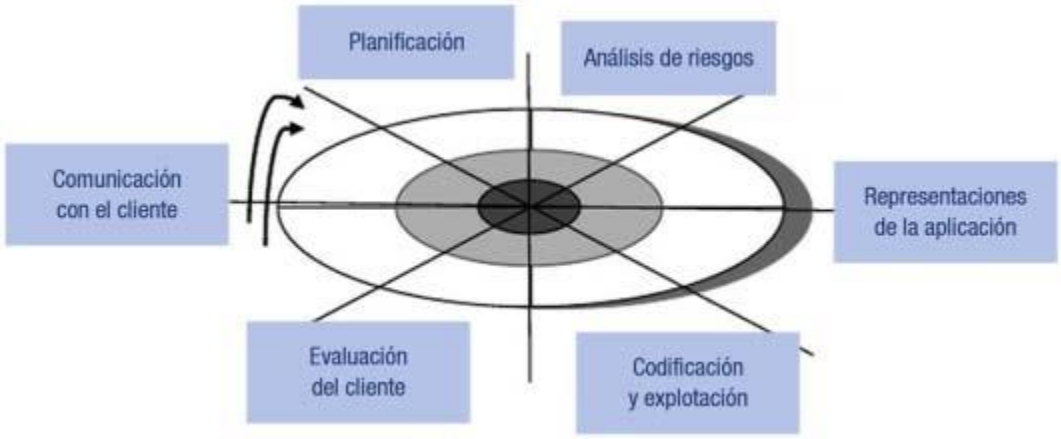
Instalamos, configuramos y probamos la aplicación en los equipos del cliente.

7. MANTENIMIENTO.

Se mantiene el contacto con el cliente para actualizar y modificar la aplicación el futuro.

5.1.- Análisis.

Esta es la primera fase del proyecto. Es la fase de mayor importancia en el desarrollo del proyecto y todo lo demás dependerá de lo bien detallada que esté. También es la más complicada, ya que no está automatizada y depende en gran medida del analista que la realice.



En esta fase se especifican y analizan los requisitos funcionales y no funcionales del sistema.

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

Lo fundamental es la buena comunicación entre el analista y el cliente para que la aplicación que se va a desarrollar cumpla con sus expectativas.

La culminación de esta fase es el documento ERS (Especificación de Requisitos Software). En este documento quedan especificados:

- 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.

"Todo aquello que no se detecte, o resulte mal entendido en la etapa inicial provocará un fuerte impacto negativo en los requisitos, propagando esta corriente degradante a lo largo de todo el proceso de desarrollo e incrementando su perjuicio cuanto más tardía sea su detección (Bell y Thayer 1976)(Davis 1993)."

5.2.- Diseño.

Durante esta fase, donde ya sabemos lo que hay que hacer, el siguiente paso es ¿Cómo hacerlo? Se debe dividir el sistema en partes y establecer qué relaciones habrá entre ellas. Decidir qué hará exactamente cada parte.

En definitiva, debemos crear un modelo funcional-estructural de los requerimientos del sistema global, para poder dividirlo y afrontar las partes por separado. En este punto, se deben tomar decisiones importantes, tales como:

- Entidades y relaciones de las bases de datos.
- Selección del lenguaje de programación que s e va a utilizar.
- Selección del Sistema Gestor de Base de Datos.

Design is not just what it looks like and feels like. Design is how it works.Steve Jobs ("El diseño no es sólo lo que parece y cómo parece. Diseño es cómo se trabaja").

5.3.- Codificación. Tipos de código.

Durante la fase de codificación se realiza el proceso de programación. Consiste en elegir un determinado lenguaje de programación, codificar toda la información anterior y llevarlo a código fuente.

Esta tarea la realiza el programador y tiene que cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.

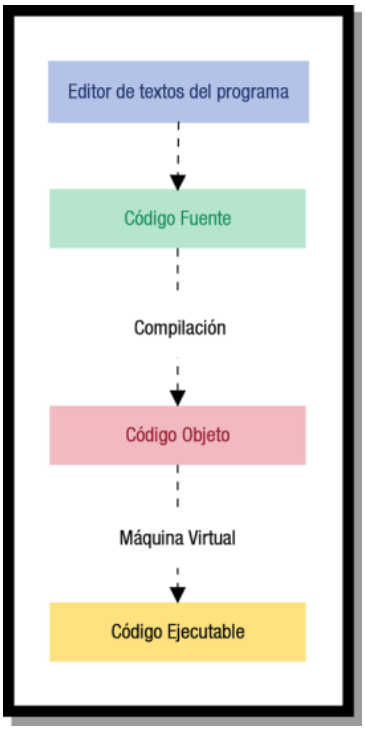
Las características deseables de todo código son:

- Modularidad: que esté dividido en trozos más pequeños.
- Corrección: que haga lo que se le pide realmente.
- Fácil de leer: para facilitar su desarrollo y mantenimiento futuro.
- Eficiencia: que haga un buen uso de los recursos.
- Portabilidad: que se pueda implementar en cualquier equipo.

Durante esta fase, el código pasa por diferentes estados:

- Código Fuente: es el escrito por los programadores en algún editor de texto. Se escribe usando algún lenguaje de programación de alto nivel y contiene el conjunto de instrucciones necesarias.
- Código Objeto: es el código binario resultado de compilar 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 no es directamente inteligible por el ser humano, pero tampoco por la computadora. 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.
- Código Ejecutable: Es el código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias. El sistema operativo será el encargado de cargar el código ejecutable en memoria RAM y proceder a ejecutarlo. También es conocido como código máquina y ya sí es directamente inteligible por la computadora.

Los programas interpretados no producen código objeto. El paso de fuente a ejecutable es directo.



A partir de un editor, escribimos el lenguaje fuente con algún Lenguaje de programación. A continuación, el código fuente se compila obteniendo código objeto o bytecode. Ese bytecode, pasa a código máquina, ya directamente ejecutable por la computadora.

Entornos de desarrollo

IDE (Entorno de desarrollo integrado)

El IDE es el software en el cual escribes tu código y que te ayudará revisando la sintaxis, auto-acompletando, corriendo tu código, y lo más importante hacer debug de tu aplicación de una manera sencilla. Los más usados son Eclipse e IntelliJ

Framework

Un framework (marco de trabajo del desarrollo rápido de aplicaciones) es una estructura de ayuda al programador, en base a la cual podemos desarrollar proyectos sin partir desde cero. Se trata de una plataforma software donde están definidos programas soporte, librerías que te

permitirán reusarlas y reducir el código que tendrás que desarrollar. En Java uno de los más conocidos es Spring, hay frameworks para testing como Junit incluso grandes empresas tienen sus propios frameworks

Ventajas de utilizar un framework:

- 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 (sin 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.

Ejemplos de Frameworks:

- .NET es un framework para desarrollar aplicaciones sobre Windows. Ofrece el "Visual Studio .net" que nos da facilidades para construir aplicaciones y su motor es el ".Net framework" qu e permite ejecutar dichas aplicaciones. Es un componente que se instala sobre el sistema operativo.
- Spring de Java. Son conjuntos de bibliotecas (API's) para el desarrollo y ejecución de aplicaciones.

Entornos de ejecución.

Un entorno de ejecución es un servicio de máquina virtual que sirve como base software para la ejecución de programas. En ocasiones pertenece al propio sistema operativo, pero también se puede instalar como software independiente que funcionará por debajo de la aplicación. Es decir, es un conjunto de utilidades que permiten la ejecución de programas.

Se denomina runtime al tiempo que tarda un programa en ejecutarse en la computadora.

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). Estos dos componentes se suelen distribuir conjuntamente, porque necesitan ser compatibles entre sí.

El entorno funciona como intermediario entre el lenguaje fuente y el sistema operativo, y consigue ejecutar aplicaciones. Sin embargo, si lo que queremos es desarrollar nuevas aplicaciones, no es suficiente con el entorno de ejecución. Adelantándonos a lo que veremos en la próxima unidad, para desarrollar aplicaciones necesitamos algo más. Ese "algo más" se llama entorno de desarrollo.

Java runtime environment.

Se denomina JRE al Java Runtime Environment (entorno en tiempo de ejecución Java). El JRE se compone de un conjunto de utilidades que permitirá la ejecución de programas java sobre cualquier tipo de plataforma.

JRE está formado por:

- Una Máquina virtual Java (JMV o JVM si consideramos las siglas en inglés), que es el programa que interpreta el código de la aplicación escrito en Java.
- Bibliotecas de clase estándar que implementan el API de Java.

- Las dos: JMV y API de Java son consistentes entre sí, por ello son distribuidas conjuntamente.

5.4.- 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 framework 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 (a ser posible, en los equipos del cliente y bajo un funcionamiento normal de su empresa). El período de prueba será normalmente el pactado con el cliente.

5.5.- Documentación.

Todas las etapas en el desarrollo de software deben quedar perfectamente documentadas.

Tenemos que ir documentando el proyecto en todas las fases de este, para pasar de una otra de forma clara y definida. 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:

Documentos a elaborar en el proceso de desarrollo de software			
	GUÍA TÉCNICA	**GUÍA DE USO**	**GUÍA DE INSTALACIÓN**
Quedan reflejados:	• El diseño de la aplicación. • La codifica-ción de los programas. • Las pruebas realizadas.	• Descripción de la funciona-lidad de la aplicación. • Forma de comenzar a ejecu-tar la aplicación. • Ejemplos de uso del progra-ma. • 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 técni-co en informática (analistas y pro-gramadores).	A los usuarios que van a usar la aplicación (clientes).	Al personal informático res-ponsable de la instalación, en colaboración con los usuarios que van a usar la aplicación (clientes).
¿Cuál es su objetivo?	Facilitar un co-rrecto desarrollo, realizar correc-ciones en los pro-gramas y permitir un mantenimiento futuro.	Dar a los usuarios finales toda la información necesaria para utilizar la aplicación.	Dar toda la información nece-saria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa.

5.6.- Explotación.

Después de todas las fases anteriores, una vez que las pruebas nos demuestran que el software es fiable, carece de errores y hemos documentado todas las fases, el siguiente paso es la explotación. Aunque diversos autores consideran la explotación y el mantenimiento como la misma etapa, nosotros vamos a diferenciarlas en base al momento en que se realizan.

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

En el proceso de instalación, los programas son transferidos al computador del usuario cliente y posteriormente configurados y verificados. Es recomendable que los futuros clientes estén presentes en este momento e irles comentando cómo se va planteando la instalación.

En este momento, se suelen llevan a cabo las Beta Test, que son las últimas pruebas que se realizan en los propios equipos del cliente y bajo cargas normales de trabajo.

Una vez instalada, pasamos a la fase de configuración. En ella, asignamos los parámetros de funcionamiento normal de la empresa y probamos que la aplicación es operativa. También puede ocurrir que la configuración la realicen los propios usuarios finales, siempre y cuando les hayamos dado previamente la guía de instalación.

Si la aplicación es más sencilla, podemos programar la configuración de manera que se realice automáticamente tras instalarla.

Una vez se ha configurado, el siguiente y último paso es la fase de producción normal. La aplicación pasa a manos de los usuarios finales y se da comienzo a la explotación del software.

Es muy importante tenerlo todo preparado antes de presentarle el producto al cliente: **será el momento crítico del proyecto.**

5.7.- Mantenimiento.

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, se pacta con el cliente un servicio de mantenimiento de la aplicación (que también tendrá un coste temporal y económico).

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. Por tanto, serán necesarias modificaciones, expansiones o eliminaciones de código.
- Adaptativos: Modificaciones, actualizaciones... para adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, etc.
- Correctivos: La aplicación tendrá errores en el futuro (sería utópico pensar lo contrario).