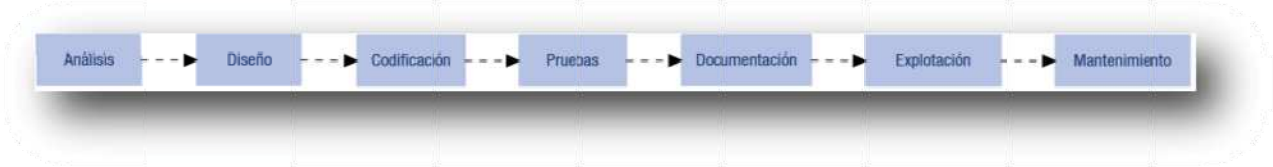
3.- Desarrollo de software.

Caso práctico
En BK programación ya están manos a la obra. Ada reúne a toda su plantilla para desarrollar el nuevo proyecto.
Ella sabe mejor que nadie que no será sencillo y que habrá que pasar por una serie de etapas. Ana no quiere perderse la reunión, quiere descubrir por qué hay que tomar tantas anotaciones y tantas molestias antes incluso de empezar.

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, pues sólo así podremos garantizar que los programas creados son eficientes, fiables, seguros y responden a las necesidades de los usuarios finales (aquellos que van a utilizar el programa).

Como veremos con más detenimiento a lo largo de la unidad, el desarrollo de software es un proceso que conlleva una serie de pasos. Genéricamente, estos pasos son los siguientes:

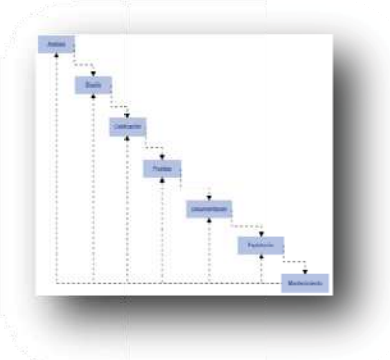
Etapas en el desarrollo de software:

Como vamos a ver en el siguiente punto, según el orden y la forma en que se lleven a cabo las etapas hablaremos de diferentes ciclos de vida del software.

La construcción de software es un proceso que puede llegar a ser muy complejo y que exige gran coordinación y disciplina del grupo de trabajo que lo desarrolle.

Reflexiona
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 es la falta de una buena planificación de las etapas y mala gestión de los pasos a seguir. ¿Por qué el porcentaje de fracaso es tan grande? ¿Por qué piensas que estas causas son tan determinantes?

3.1.- Ciclos de vida del software.



Ya hemos visto que la serie de pasos a seguir para desarrollar un programa es lo que se conoce como Ciclo de Vida del Software.

Cada etapa vendrá explicada con más detalle en el punto de la presente unidad dedicado a las fases del desarrollo y ejecución del software.

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

Siempre se debe aplicar un modelo de ciclo de vida al desarrollo de cualquier proyecto software.

1. **Modelo en Cascada**

   Es el modelo de vida clásico del software.

   Es prácticamente imposible que se pueda utilizar, ya que requiere conocer de antemano todos los requisitos del sistema. Sólo es aplicable a pequeños desarrollos, ya que las etapas pasan de una a otra sin retorno posible. (se presupone que no habrá errores ni variaciones del software).

2. **Modelo en Cascada con Realimentación**

   Es uno de los modelos más utilizados. Proviene del modelo anterior, pero se introduce una realimentación entre etapas, de forma que podamos volver atrás en cualquier momento para corregir, modificar o depurar algún aspecto. No obstante, si se prevén muchos cambios durante el desarrollo no es el modelo más idóneo.

   Es el modelo perfecto si el proyecto es rígido (pocos cambios, poco evolutivo) y los requisitos están claros.

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 con realimentación, 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.

Autoevaluación
Si queremos construir una aplicación pequeña, y se prevé que no sufrirá grandes cambios durante su vida, ¿sería el modelo de ciclo de vida en espiral el más recomendable?

- ○ Sí.
- ◉ No.

3.2.- Herramientas de apoyo al desarrollo del software.

En la práctica, para llevar a cabo varias de las etapas vistas en el punto anterior contamos con herramientas informáticas, cuya finalidad principal es automatizar las tareas y ganar fiabilidad y tiempo.

Esto nos va a permitir centrarnos en los requerimientos del sistema y el análisis del mismo, que son las causas principales de los fallos del software.

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.

¿En qué fases del proceso nos pueden ayudar?

En el diseño del proyecto, en la codificación de nuestro diseño a partir de su apariencia visual, detección de errores…

El desarrollo rápido de aplicaciones o **RAD** es un proceso de desarrollo de software que comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE. Hoy en día se suele utilizar para referirnos al desarrollo rápido de interfaces gráficas de usuario o entornos de desarrollo integrado completos.

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, al permitir visualizar las fases de forma gráfica.

CLASIFICACIÓN

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…

Para saber más

En el siguiente enlace se presenta una ampliación de los tipos y ayudas concretas de la herramientas CASE.

http://temariotic.wikidot.com/tema-58-boe-13-02-1996

4.- Lenguajes de programación.

Caso práctico
Una de los aspectos del proyecto que más preocupa a Ana es la elección del lenguaje de programación a utilizar.
Necesita tener muy claros los requerimientos del cliente para enfocar correctamente la elección, pues según sean éstos unos lenguajes serán más efectivos que otros.

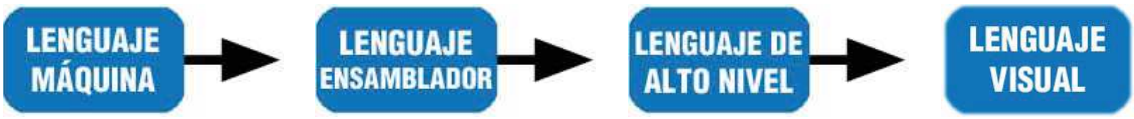
Ya dijimos anteriormente que los programas informáticos están escritos usando algún lenguaje de programación. Por tanto, podemos definir un Lenguaje de Programación como un idioma creado de forma artificial, formado por un conjunto de símbolos y normas que se aplican sobre un alfabeto para obtener un 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.

En otras palabras, es muy importante tener muy clara la función de los lenguajes de programación. 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. (No necesita traducción).
  - 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 ala ubicación física de los archivos en el equipo.
  - Es difícil de utilizar.

- **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.
- **Lenguajes visuales**:
  - Están sustituyendo a los lenguajes de alto nivel basados en código.
  - 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.
  - Son completamente portables de un equipo a otro.

Para saber más

En el siguiente enlace, verás la evolución entre los distintos tipos de Lenguajes de Programación en la historia.

http://www.monografias.com/trabajos38/tipos-lenguajes-programacion/tipos-lenguajes-programacion.shtml

4.1.- Concepto y características.

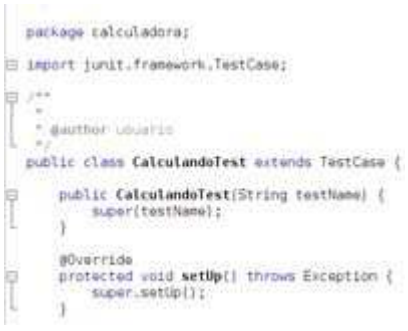
Ya sabemos que los lenguajes de programación han evolucionado, y siguen haciéndolo, siempre hacia la mayor usabilidad de los mismos (que el mayor número posible de usuarios lo utilicen y exploten).

La elección del lenguaje de programación para codificar un programa dependerá de las características del problema a resolver.

CONCEPTO

Un lenguaje de programación es el conjunto de:

- **Alfabeto**: conjunto de símbolos permitidos.
- **Sintaxis**: normas de construcción permitidas de los símbolos del lenguaje.
- **Semántica**: significado de las construcciones para hacer acciones válidas.



CARACTERÍSTICAS

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

- **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.
- **Según la técnica de programación utilizad**a:
  - 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, Ada, Delphi, etc.
  - Lenguajes de Programación Visuales: Basados en las técnicas anteriores, permiten programar gráficamente, siendo el código correspondiente generado de forma automática. Ejemplos: Visual Basic.Net, Borland Delphi, etc.

A pesar de la inmensa cantidad de lenguajes de programación existentes, Java, C, C++, PHP y Visual Basic concentran alrededor del 60% del interés de la comunidad informática mundial.

Para saber más
En la página web siguiente encontrarás un resumen de las características de los Lenguajes de Programación más utilizados en la actualidad.
http://www.larevistainformatica.com/LENGUAJES-DE-PROGRAMACION-listado.html

4.2.- Lenguajes de programación estructurados.

Aunque los requerimientos actuales de software son bastante más complejos de lo que la técnica de programación estructurada es capaz, es necesario por lo menos conocer las bases de los Lenguajes de Programación estructurados, ya que a partir de ellos se evolucionó hasta otros lenguajes y técnicas más completas (orientada a eventos u objetos) que son las que se usan actualmente.

La programación estructurada se define como una técnica para escribir lenguajes de programación que permite sólo el uso de tres tipos de sentencias o estructuras de control:

- Sentencias secuenciales.
- Sentencias selectivas (condicionales).
- Sentencias repetitivas (iteraciones o bucles).

Los lenguajes de programación que se basan en la programación estructurada reciben el nombre de lenguajes de programación estructurados.

Para saber más
En el siguiente enlace encontrarás un breve documento donde se explica para qué sirve cada sentencia de control con unos sencillos ejemplos escritos usando el lenguaje C.
http://www.juntadeandalucia.es/educacion/adistancia/cursos/file.php/420/ED01/ED01_Web/index.html#anexo_i_sentencias_de_control_de_la_programacin_estructurada.html

La programación estructurada fue de gran éxito por su sencillez a la hora de construir y leer programas. Fue sustituida por la programación modular, que permitía dividir los programas grandes en trozos más pequeños (siguiendo la conocida técnica "divide y vencerás"). A su vez, luego triunfaron los lenguajes orientados a objetos y de ahí a la programación visual (siempre es más sencillo programar gráficamente que en código, ¿no crees?).

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

- 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.

Ejemplos de lenguajes estructurados: *Pascal, C, Fortran*.

La Programación estructurada evolucionó hacia la Programación modular, que divide el programa en trozos de código llamados módulos con una funcionalidad concreta, que podrán ser reutilizables.

4.3.- Lenguajes de programación orientados a objetos.

Después de comprender que la programación estructurada no es útil cuando los programas se hacen muy largos, es necesaria otra técnica de programación que solucione este inconveniente. Nace así la Programación Orientada a Objetos (en adelante, P.O.O.).

Los lenguajes de programación orientados a objetos tratan a los programas no como un conjunto ordenado de instrucciones (tal como sucedía 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.

Su primera desventaja es clara: no es una programación tan intuitiva como la estructurada.

A pesar de eso, alrededor del 55% del software que producen las empresas se hace usando esta técnica.

Razones:

- El código es reutilizable.
- Si hay algún error, es más fácil de localizar y depurar en un objeto que en un programa entero.

Características:

- 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.

Principales lenguajes orientados a objetos: *Ada, C++, VB.NET, Delphi, Java, PowerBuilder, etc.*

Para saber más

En el siguiente enlace hay un documento muy interesante de introducción a la programación orientada a objetos, en concreto, del lenguaje C++.

http://mat21.etsii.upm.es/ayudainf/aprendainf/Cpp/manualcpp.pdf