1. **TEMA 1**
   1. **Software**

Un sistema informático se compone de un hardware y un software. El hardware son los componentes físicos que componen el sistema, aquellos, que pueden “verse y tocarse”. En cambio, el software lo compone la parte lógica, el conjunto de instrucciones que debe cumplir o ejecutar el hardware para realizar las tareas que tiene encomendadas.

Según el diccionario de la RAE, se define software como “conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora”.

Según el estándar 729 del IEEE, se define software como “el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación”.

Podríamos precisar, que el software es todo aquello que se refiere a los programas y datos almacenados en un ordenador: programas encargados de dar instrucciones para realizar tareas con el hardware o para comunicarnos con otro software, y los datos necesarios para la ejecución de dichos programas.

En función de la tarea que realiza un software, podemos realizar una clasificación como:

* **Software de sistema**, aquel que permite que el hardware funcione. Lo forman los programas que permiten la administración de la parte física o los recursos del ordenador, y es el que interacciona entre el usuario y los componentes hardware del ordenador.
* **Software de aplicación**, aquel software que nos ayuda a realizar tareas específicas en cualquier campo susceptible de ser automatizado o asistido. Este software hace que el ordenador sea una herramienta útil para el usuario.
* **Software de programación o desarrollo**, aquel software que proporciona al usuario (programador) las herramientas necesarias para ayudar o asistir en la creación de nuevo software, según diferentes lenguajes de programación.

En función del método de distribución de software, se puede hablar de dos grandes grupos:

* **Software libre**, aquel software que se distribuye con la principal premisa de que sea libre (lo cual no conlleva necesariamente que la distribución sea gratuita), para que cualquiera pueda distribuirlo, utilizarlo o incluso modificarlo para mejorarlo o adaptarlo a sus propósitos particulares.
* **Software propietario**, aquel software desarrollado por empresas que hacen del desarrollo de software su modelo de negocio y se distribuye mediante un pago por uso o utilización del mismo.

Hay otras modalidades de distribución con ciertas similitudes y diferencias con los modelos anteriores.

* **Shareware**, se trata de una modalidad de distribución, en la cual se proporciona de forma, habitualmente, gratuita el producto, por un tiempo determinado para su evaluación, terminado ese tiempo, el usuario deberá adquirir una licencia de uso, si bien, en algunas ocasiones se permite continuar usando el producto, pero con ciertas limitaciones que desaparecen al adquirir la licencia.
* **Freeware**, se trata de software que se distribuye sin cargo. Habitualmente, este software, aunque se distribuye gratuitamente suele tener ciertas restricciones, como su modificación, venta o distribución no autorizada.
* **Open Source**, se trata de software distribuido con su código fuente (no necesariamente se distribuye el ejecutable), tiene grandes similitudes con el software libre, aunque no es exactamente lo mismo, mientras que el software libre se centra en las cuestiones morales y éticas del modelo de distribución, el Open Source se centra en las ventajas de compartir el código fuente como metodología para conseguir productos de mayor calidad. Por otra parte, mientras que el software libre obliga a que sus derivados sigan siendo software libre, el Open Source no.
* **Dominio público**, se trata de aquel software que carece de licencia, bien porque se desconoce el autor, bien porque se han abandonado sus derechos o bien por expiración del plazo de la misma.
  1. **Ciclo de vida del software.**

El estándar ISO/IEC 12207-01 define ciclo de vida del software como “un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de requisitos hasta la finalización de su uso”.

Se consideran las siguientes etapas en el ciclo de vida del software:

* Análisis, es en esta etapa en la que se estudia y comprende el problema a resolver.
* Diseño, una vez estudiado el problema, es necesario especificar el como vamos a abordar su resolución, las estructuras necesarias en el software, el interfaz de usuario, etc.
* Codificación, a partir del diseño, se requiere su traducción al lenguaje del computador para su resolución mediante el uso del hardware.
* Pruebas, se debe verificar que el software realiza su tarea con ciertos requisitos de calidad.
* Mantenimiento, tras la entrega del software pueden ser necesarias labores posteriores para su posible adaptación a los cambios obligados por errores no encontrados en la fase de pruebas, o bien por requerimiento del cliente en función de mejoras demandadas por el mismo.
  1. **Modelos de ciclo de vida**
     1. **Modelo en cascada**

Este modelo de ciclo de desarrollo consta de las siguientes etapas:

* Análisis del sistema:
  + Identificación de las necesidades del cliente.
  + División del sistema en subsistemas de información.
  + Evaluación de la viabilidad técnica y económica.
  + Establecimiento de restricciones de coste y plazo de entrega.
* Análisis conceptual: Diseñar las abstracciones operacionales y de datos del sistema a informatizar.
* Diseño lógico:
  + Diseño lógico de datos.
  + Diseño arquitectónico.
  + Diseño lógico de los procedimientos.
  + Diseño lógico de la interfaz hombre-máquina
* Diseño físico o codificación: Traducir el diseño lógico de forma que el resultado obtenido sea ejecutable por la máquina.
* Pruebas y puesta a punto.
* Lanzamiento y mantenimiento.

Análisis del sistema

Análisis conceptual

Diseño lógico

Codificación

Pruebas

Mantenimiento

En este modelo de desarrollo de software cada fase empieza donde termina la anterior

Para pasar de una a otra se deben haber conseguir todos los objetivos de la etapa previa. Es el tipo de ciclo de vida y desarrollo de software más antiguo y más usado.

Tiene como ventajas:

* Fácil de comprender y planificar.
* El producto resultante de alta calidad.
* Permite trabajar con personal poco cualificado.

Y como desventajas:

* Requiere tener definidos todos los requisitos desde el principio.
* Hasta no terminar una etapa no se pasa a la siguiente.
* El sistema en funcionamiento no está disponible hasta el final.

Se recomienda cuando:

* Se han elaborado proyectos similares con éxito.
* Cuando el cliente no necesita versiones intermedias.

<http://es.wikipedia.org/wiki/Desarrollo_en_cascada>

* + 1. **Modelo iterativo incremental**

En este caso se van añadiendo componentes funcionales al sistema (incrementos).

Cada versión o refinamiento parte de una versión previa a la que se añaden nuevas funciones y/o requisitos. Su principal ventaja es que permite ampliar los requisitos.

Análisis

Diseño

Codificación

Pruebas

Análisis

Diseño

Codificación

Pruebas

Análisis

Diseño

Codificación

Pruebas

Entrega del incremento 1

Entrega del incremento 2

Entrega del incremento 3

Sus ventajas:

* No necesita el conocimiento de todos los requisitos desde el comienzo.
* Permite la entrega temprana de versiones operativas, no finales, del producto.
* Las entregas tempranas permiten realimentación para futuras evoluciones.

Desventajas:

* El coste final es difícil de evaluar.
* No es recomendable para sistemas de alta disponibilidad o altos requisitos de calidad.

Se recomienda para:

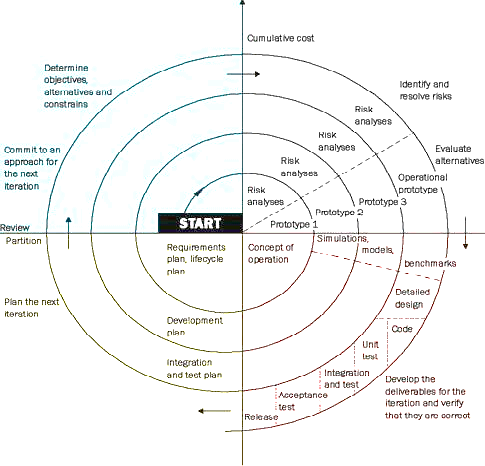
* Casos en los que los requisitos no están completamente definidos desde el comienzo.
* Para nuevas tecnologías o casos de innovación.

<http://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente>

* + 1. **Modelo en espiral**

En el modelo Espiral el software se construye en una serie de versiones incrementales. En las primeras iteraciones la versión incremental podría ser un modelo en papel o bien un prototipo. En las últimas iteraciones se producen versiones cada vez más completas del sistema diseñado.

En cada nivel sucesivo de refinamiento se lleva a cabo el análisis crítico del producto conseguido decidiendo si seguir con su desarrollo bajando a mayores niveles de detalle o bien abandonarlo por motivos técnicos o económicos.



Sus ventajas:

* No necesita el conocimiento de todos los requisitos desde el comienzo.
* Incluye análisis de riesgos en cada ciclo.
* Reduce riesgos e incorpora objetivos de calidad.

Desventajas:

* El coste final es difícil de evaluar.
* El coste aumenta en cada ciclo.
* Depende enormemente de la fase de análisis de riesgos.

Se recomienda para:

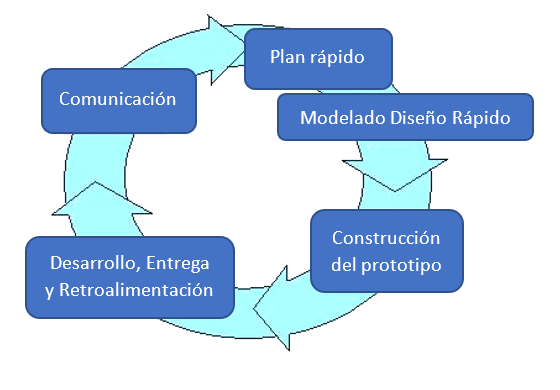
* Proyectos de gran tamaño y con constantes cambios.
* Proyectos en los que sea importante el factor riesgo.

<http://es.wikipedia.org/wiki/Software>

* + 1. **Prototipado**

Este modelo de desarrollo de software consta de las siguientes etapas:

* Recolección de requisitos.
* Elaboración de un prototipo.
* Evaluación del prototipo por el usuario.
* Refinamiento de requisitos.
* las etapas anteriores se vuelven a repetir hasta que se cumple con todos los requisitos deseados por el usuario.



Ventajas:

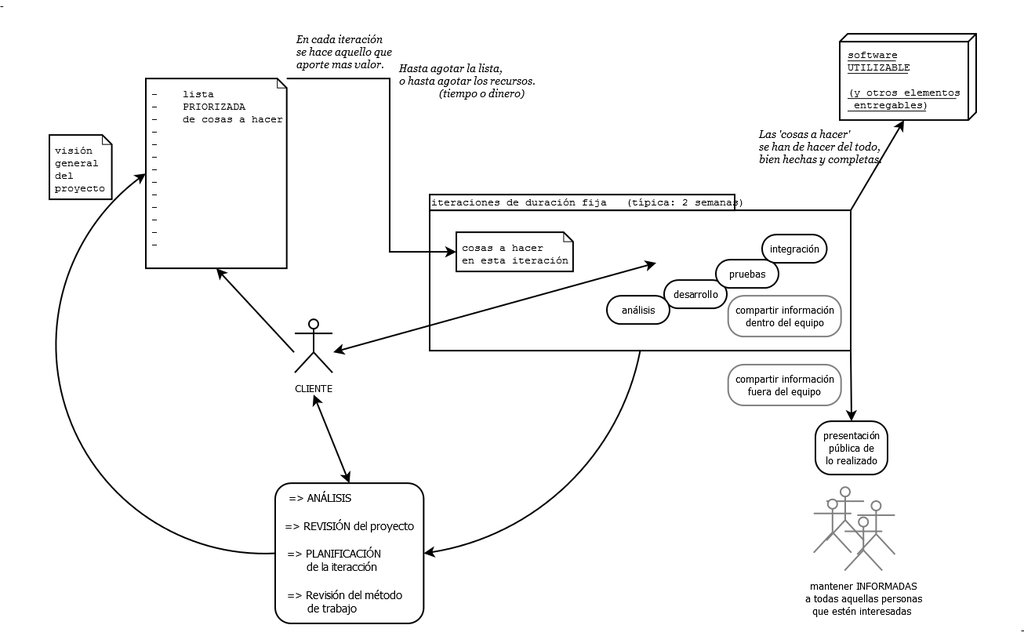
* Este modelo es útil cuando el cliente conoce los objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida.
* También ofrece un mejor enfoque cuando el responsable del desarrollo del software está inseguro de la eficacia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debería tomar la interacción con el usuario.
* Se puede reutilizar el código

Desventajas:

* El usuario tiende a crearse unas expectativas cuando ve el prototipo de cara al sistema final.
* La calidad y el mantenimiento a largo plazo pasan a segundo plano frente al desarrollo rápido de prototipos.
* La falta de documentación y el desarrollo rápido puede provocar pérdida en la transferencia de conocimiento necesaria para la continuidad del proyecto.

<http://es.wikipedia.org/wiki/Modelo_de_prototipos>

* + 1. **Desarrollo ágil de software**

Está basado en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. El trabajo es realizado mediante la colaboración de equipos auto-organizados y multidisciplinarios, inmersos en un proceso de toma de decisiones a corto plazo compartido. Los métodos ágiles enfatizan las comunicaciones cara a cara en vez de la documentación. La mayoría de los equipos ágiles están localizados en una simple oficina abierta, a veces llamadas "plataformas de lanzamiento" (bullpen en inglés). La oficina debe incluir revisores, escritores de documentación y ayuda, diseñadores de iteración y directores de proyecto. Los métodos ágiles también enfatizan que el software funcional es la primera medida del progreso. Combinado con la preferencia por las comunicaciones cara a cara, generalmente los métodos ágiles son criticados y tratados como "indisciplinados" por la falta de documentación técnica.

<https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software>

<http://www.proyectosagiles.org/>

<http://www.proyectosagiles.org/que-es-scrum>

<http://blog.gfi.es/gestion-agil-de-proyectos-con-la-metodologia-scrum/>

* 1. **Fases de desarrollo de software.**
     1. **Análisis**

En todo desarrollo de software es necesario un análisis inicial, análisis en el cual se debe entender y comprender el problema, se deben especificar claramente los requisitos y para ello, es primordial una buena comunicación con el cliente.

Para la comunicación con el cliente, se usan varias técnicas:

* **Entrevistas**, es la técnica más tradicional y no por ello menos importante, las entrevistas con el cliente y partes implicadas nos permiten ahondar en el problema para poder ofrecer la solución buscada con mayor acierto.
* **Desarrollo conjunto de aplicaciones (JAD)**, mediante dinámica de grupos se realiza una planificación en la que toma parte todas las partes implicadas en su desarrollo.
* **Planificación conjunta de requisitos (JRP)**, es un tipo de JAD enfocada hacia la dirección, buscando resolver los requisitos de alto nivel o estratégicos que puedan tener que ver con el desarrollo.
* **Brainstorming**, reuniones de grupo que pretenden obtener ideas solución desde diferentes puntos de vista, los cambios de marco de referencia pueden ser muy útiles a la hora de encontrar posibles problemas o soluciones que de otra forma no se plantearían.
* **Prototipos**, mediante versiones intermedias se puede tener un feedback muy útil, ya que se obtiene de una forma práctica y más directa.
* **Casos de uso**, técnica definida en UML y que nos permite abordar el problema desde el punto de vista de las distintas situaciones que se pueden producir.

Tipos de requisitos, hay dos tipos de requisitos:

* **Requisitos funcionales**, tienen que ver con las funcionalidades que va a poseer el desarrollo y como se van a abordar.
* **Requisitos no funcionales**, tienen que ver con el entorno de operación del desarrollo, por ejemplo el hardware sobre el cual va a correr o el usuario que lo va a utilizar.

Se pueden representar los requisitos mediante el uso de varios tipos de diagramas.

* **Diagramas de flujo de datos (DFD)**, se representan mediante los mismos, el flujo de datos entre diferentes procesos, entidades externas y almacenes del sistema.

Entidad Externa

Entidad Externa

Entidad Externa

Almacenamiento

Información de entrada

Información de entrada

Información de salida

Datos intermedios

Datos intermedios

Datos intermedios

Entrada de Almacenamiento

Salida de Almacenamiento

* **Diagramas de transición de estados (DTE)**, representa como se comporta el sistema como consecuencia de sucesos externos.

En reposo

Esperando contraseña

Esperando selección

Se introduce una tarjeta

Contraseña correcta

Se pulsa cancelar

* **Diagramas de flujo de control (DFC)**, similares a los DFD pero mostrando el flujo de control en lugar del flujo de datos.

Tenemos los ingredientes

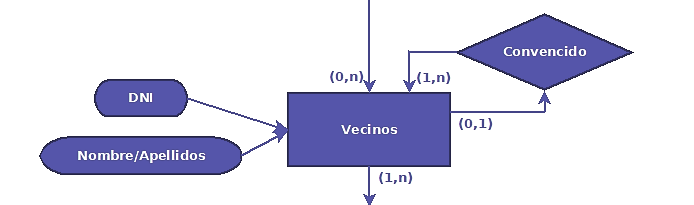
Compramos ingrediente

Cocinamos según receta

SI

NO

* **Diagrama Entidad/Relación (DER)**, se utiliza para representar los datos y como se relacionan entre ellos.



* **Diccionario de Datos (DD)**, se trata de una descripción detallada de los datos utilizados por el sistema que gráficamente se encuentra representado por los flujos de datos y los almacenes presentes sobre el conjunto del DFD.

Todos los pasos de la fase de Análisis deberán ser reflejados en el documento “Especificación de Requisitos de Software”, documento que no debe tener ambigüedades y debe ser completo, consistente, fácil de verificar y modificar, fácil de utilizar en la fase de explotación y mantenimiento, fácil de identificar el origen y las consecuencias de los requisitos. En el estándar IEEE 830 se puede ver la propuesta de estructura de este documento[[1]](#footnote-1).

* + 1. **Diseño**

Una vez realizado el análisis, se pasa a la parte de diseño, principalmente se puede hablar de dos tipos de diseño:

**Diseño estructurado o clásico**, se fundamenta en el desarrollo de cuatro fases que se apoyan unas en otras:

Diseño de datos

Diseño arquitectónico

Diseño de interface

Diseño procedimental

* **Diseño de datos**, transforma el modelo de dominio de la información creado durante el análisis en las estructuras de datos que se utilizarán para implementar el software. Este diseño se apoya en el DER y DD.
* **Diseño arquitectónico**, se centra en la representación de la estructura de los componentes del software, sus propiedades e interacciones. Para ello los diagramas DFD pueden ser de gran ayuda.
* **Diseño de la interface**, describe como se comunica el software consigo mismo, con los sistemas con los que interacciona y con el usuario.
* **Diseño a nivel de componentes o procedimental**, a través de este diseño se transforma los elementos estructurales de la arquitectura del software en una descripción procedimental de los componentes del software. Para ello se utiliza la **programación estructurada** que se apoya en tres tipos de construcciones:
  + **Secuencial**. En cualquier algoritmo se suceden pasos y funciones de una forma secuencial.

Instrucción 3

Instrucción 2

Instrucción 1

* + **Condicional**. Es necesario incorporar estructuras que permitan realizar una toma de decisiones a partir de condiciones lógicas. Ejemplos: if, if…else, if…elseif, switch…case,…

Condición

Instrucción 1

NO

SI

Instrucción 2

* + **Repetitiva**. Frecuentemente es necesario realizar bucles condicionados a cierta toma de decisiones. Ejemplos: for, while, do…while,…

Condición

Instrucción

SI

NO

Como ayuda en la **programación estructurada** se utilizan distintos recursos:

* + **Diagramas de flujo**. Es quizá el tipo de diagrama más conocido (las estructuras anteriores se han ilustrado con este tipo de diagramas
  + **Diagramas de caja**. Es una alternativa a los diagramas de flujo, surgió como propuesta de representación que no permitiera la violación de las construcciones estructuradas.
  + **Tablas de decisión**. Cuando en un algoritmo es necesario evaluar una combinación compleja de condiciones las tablas de decisión ayudan a organizar dicha toma de decisiones.
  + **Pseudocódigo**. Permite, utilizando texto descriptivo, realizar el diseño de un algoritmo que luego podrá ser portado a diferentes lenguajes de programación.

**Diseño orientado a objetos.** El diseño orientado a objetos posee una complejidad mayor que la programación estructurada clásica, por ello se dedicará un capítulo a su descripción metodológica, en particular mediante el uso de **UML** (Unified Modeling Language), un lenguaje de modelado basado en diagramas que permiten representar los modelos y que se ha convertido en el estándar de facto para la mayoría de metodologías orientadas a objetos.

En el diseño orientado a objetos, primeramente, se requiere un análisis orientado a objetos, en el cual se definen las clases, sus atributos y métodos, así como las relaciones entre las clases.

A continuación, se procede con el diseño, que en esta ocasión consta de 4 capas de diseño:

* Subsistema, que implementan las funciones principales del sistema.
* Clases y Objetos, requeridos para implementar el sistema.
* Mensajes, de colaboración entre objetos.
* Responsabilidades, identifica operaciones y atributos que caracterizan cada clase.
  + 1. **Codificación.**

Lenguaje interpretado y lenguaje compilado

Una vez realizado el diseño, se procede a la codificación, el programador, transfiere las especificaciones del diseño a un programa mediante un conjunto de instrucciones según la semántica propia de un determinado lenguaje de. Dicho conjunto de instrucciones se denomina **código fuente** del programa.

El código fuente se debe escribir según una serie de normas de escritura que facilite su lectura y mantenimiento posterior, incluso por aquellos que no han desarrollado el código original. En el caso particular del lenguaje de programación **java** tenéis en el documento “**Normas de codificación.pdf**” una recopilación de las mismas.

Mediante un programa compilador (o interprete en el caso de lenguajes interpretados) se traduce dicho conjunto de instrucciones a otro conjunto codificado en lenguaje máquina que puede ser entendido por la CPU, se denomina esto como **código objeto**.

Sin embargo, el código objeto, todavía no es ejecutable, se necesita otro programa denominado enlazador o montador, que toma el código objeto y lo conecta a una serie de librerías para construir el denominado **código ejecutable**.

Una vez obtenido el código ejecutable, se hace necesario probarlo para comprobar y verificar que cumple con las especificaciones de diseño, para ello será de utilidad los manuales técnicos y de referencia que deben ser escritos a la vez que se desarrolla el código correspondiente.

* + 1. **Pruebas**

Una vez obtenido el software, es preciso probar el software para asegurar su adecuación a las especificaciones (pruebas de verificación y validación), así como minimizar al máximo la posibilidad de presencia de algún tipo de errores.

Se pueden realizar las pruebas según dos procedimientos diferenciados:

* **Pruebas de caja negra**, se trata de pruebas que se centran en validar los requisitos funcionales sin fijarse en el funcionamiento interno del programa.
* **Pruebas de caja blanca**, son aquellas pruebas que se realizan conociendo la estructura interna del programa y suelen centrarse en los detalles procedimentales del código.

El flujo natural en un proceso de pruebas de software se puede resumir en los siguientes pasos:

* Generación del **plan de pruebas** a partir de la documentación del proyecto y del software.
* **Diseño de pruebas** a partir del plan de pruebas, así como identificación de las técnicas necesarias.
* Generación de los **casos de pruebas** según las técnicas identificadas. En este punto es preciso identificar los casos de pruebas que requieren pruebas de caja blanca y cuales requieren pruebas de caja negra.
* Definición de los **procedimientos de pruebas**: como se llevará a cabo, quien lo realizará, cuando, etc.
* **Ejecución de pruebas**, aplicando los casos de prueba generados anteriormente.
* **Evaluación**, comparando los resultados obtenidos con los esperados. Es preciso en este punto realizar un **informe de pruebas**.
* Localizar los posibles errores para, mediante un proceso de **depuración**, lograr corregirlos.
* Realizar un **análisis de errores** como medio para predecir la fiabilidad del software y mejorar los procesos de desarrollo.
  + 1. **Documentación**

Todas las etapas del desarrollo deben quedar documentadas, una vez validado el producto tras la etapa de pruebas, será necesario recopilar toda la documentación generada para clasificarla según su objetivo y nivel técnico.

Hay dos clases fundamentales de documentación:

* **Documentación del proceso**. Son los documentos que registran el proceso de desarrollo y mantenimiento.
* **Documentación del producto**. Son aquellos documentos que describen el producto desarrollado y a su vez se puede distinguir entre dos tipos de documentación:
  + **Documentación de usuario**, ofrece la descripción del producto orientada a los usuarios que utilizarán el sistema. Hay distintos niveles de usuarios del sistema y se pueden distinguir diferentes tipos de documentación de usuario en función de dichos niveles:
    - Descripción funcional del sistema.
    - Documento de instalación del sistema.
    - Manual introductorio.
    - Manual de referencia del sistema.
    - Guía del administrador del sistema.
    - Tarjeta de referencia rápida.
  + **Documentación del sistema**, son documentos que describen el sistema. Estos documentos abarcan desde la especificación de requisitos hasta las pruebas de aceptación.
    - Fundamentos del sistema.
    - Análisis y especificación de requisitos.
    - Diseño
    - Implementación.
    - Plan de pruebas del sistema.
    - Plan de pruebas de aceptación.
    - Diccionarios de datos.

**Estructura del documento**. Todos los documentos generados deberán seguir una estructura ordenada según ciertas pautas mínimas. En el estándar IEEE 2001 para la documentación de usuario se propone una estructura de documento.

* + 1. **Explotación.**

En esta etapa se lleva a cabo la instalación y puesta en marcha del producto software en el entorno de trabajo del cliente. Se pueden distinguir varias tareas:

**Estrategia de implementación del proceso**, se trata de recibir, registrar, solucionar y hacer seguimiento de los problemas del producto en el entorno de trabajo

**Pruebas de operación**, pruebas de funcionamiento en cada reléase del producto que si son superadas finalizan en la liberación del producto para su uso operativo.

**Uso operacional del sistema**, el producto se utiliza en el entorno previsto de acuerdo con la documentación de usuario.

**Soporte al usuario**, comprende la asistencia y asesoría al usuario cuando lo solicite.

* + 1. **Mantenimiento.**

El mantenimiento del software se regula según el estándar IEEE 1219. Hay diversos tipos de mantenimiento:

* **Mantenimiento adaptativo**, tiene como objetivo adaptar el software a los cambios en el hardware y software del entorno en el que se ejecuta. Se trata del tipo de mantenimiento más habitual, debido a los rápidos cambios que se suceden en el campo de la tecnología informática, que, de hecho, pueden hacer obsoleto un producto de software recién desarrollado.
* **Mantenimiento correctivo**, después de la entrega del producto, no es raro que el cliente encuentre errores o defectos que no han sido localizados en las etapas de pruebas y verificaciones del producto.
* **Mantenimiento perfectivo**, el cliente puede descubrir funciones adicionales que pueden aportar beneficios y por tanto puede solicitar su implementación.
* **Mantenimiento preventivo**, son las modificaciones que se realizan en el software sin alterar las especificaciones del mismo, con el fin de mejorar y facilitar futuras tareas de mantenimiento.

En la fase de mantenimiento se deberán realizar las siguientes tareas:

* Implementación del proceso, se debe preparar, documentar los planes y procedimientos necesarios para llevar a cabo las tareas de mantenimiento.
* Análisis de problemas y modificaciones, se debe analizar el informe del problema o la petición de modificación.
* Implementación de las modificaciones, se debe determinar que unidades de software y que documentaciones son preciso modificar.
* Revisión/aceptación del mantenimiento, se debe autorizar las modificaciones, de forma que se determine la integridad del sistema modificado.
* Migración, es preciso preparar, documentar y ejecutar un plan de migración.
* Retirada del software, se debe preparar y documentar un plan de retirada para el cese del soporte activo por parte de las organizaciones de operación y mantenimiento.
  1. **Programa**

Un programa informático es un conjunto de instrucciones escritas en un lenguaje de programación que, aplicadas sobre un conjunto de datos, resuelven un problema o parte del mismo.

Para ejecutar un programa se necesitan los recursos hardware de un ordenador o computadora. Los equipos computacionales actuales utilizan, en su casi totalidad, la arquitectura **Von Neumann**.

En clase se describió el proceso de ejecución de una instrucción según dicha arquitectura, que consta siempre de dos fases:

* **Fase de búsqueda**, en la cual se localiza la instrucción a ejecutar dentro de la memoria principal y se lleva a la Unidad de Control para procesarla.
* **Fase de ejecución**, se realizan las instrucciones asociadas a la instrucción leída.

Se proporcionan, igualmente, dos documentos descargados que describen de forma parecida, procesos análogos al visto en clase: “**C02-VonNeumannYGeneral.pdf**” y “**Ejecución de una instrucción en arquitectura Von Neumann.pdf**”.

* 1. **Lenguajes de programación**

Un lenguaje de programación consta de los siguientes elementos:

* **Un alfabeto o vocabulario (léxico)**: formado por el conjunto de símbolos permitidos.
* **Una sintaxis**: las reglas que indican cómo realizar las construcciones con los símbolos del lenguaje.
* **Una semántica**: las reglas que determinan el significado de cualquier construcción del lenguaje.

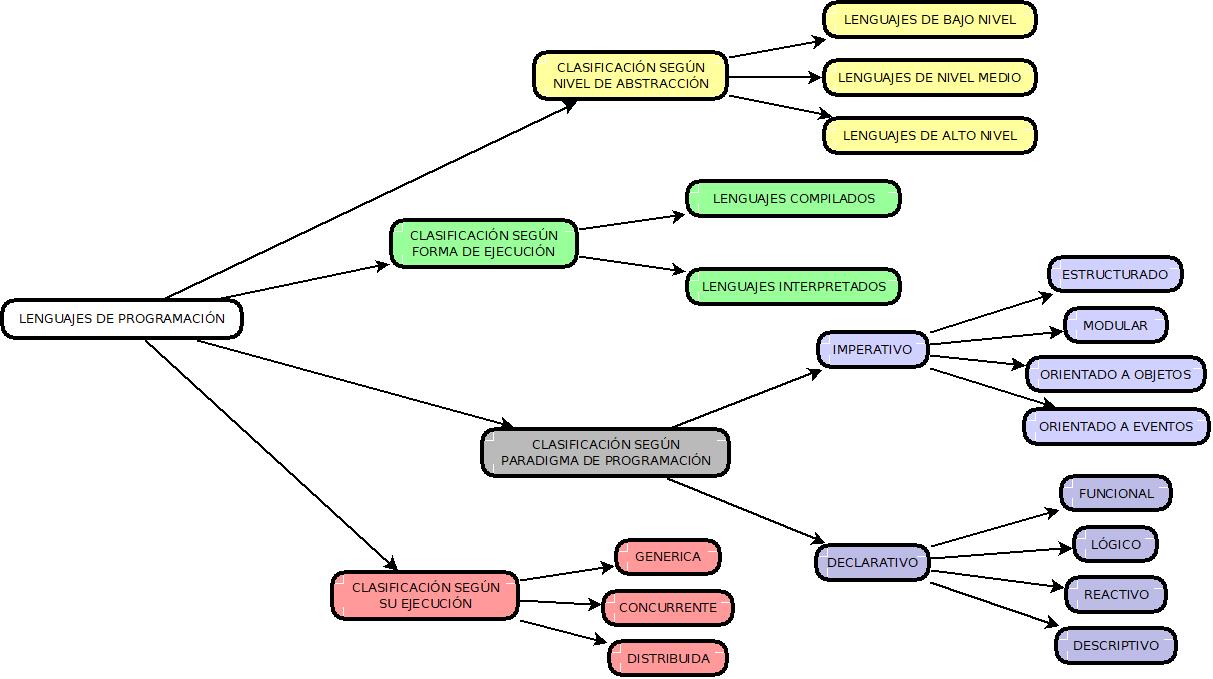
Podemos clasificar los lenguajes de programación atendiendo a distintos criterios:

Según su nivel de abstracción podemos hablar de:

* **Lenguajes de bajo nivel**, se acercan al funcionamiento del ordenado, el lenguaje de más bajo nivel es el lenguaje máquina que entiende directamente la máquina y está constituido por cadenas de ceros y unos que representan funciones operables por un hardware determinado, una característica de los lenguajes de bajo nivel es que son particulares de cada arquitectura de hardware. El siguiente nivel sería el lenguaje ensamblador, que abstrae las cadenas de ceros y unos que entiende la CPU por nombres nemotécnicos que aluden a su funcionalidad.
* **Lenguajes de nivel medio**, son lenguajes que poseen características que los acercan a los lenguajes de bajo nivel, pero a su vez también tienen características de los lenguajes de alto nivel. El lenguaje más representativo de este tipo es el C.
* **Lenguajes de alto nivel**, se trata de lenguajes más fáciles de aprender y cercanos al lenguaje natural. Los lenguajes de alto nivel son independientes de la arquitectura del hardware, aunque requieren de un software de compilación específico para cada arquitectura de hardware. Otra característica es que su cercanía al lenguaje natural hace más fácil elaborar la solución al problema planteado. Algunos de los lenguajes más representativos de alto nivel son por ejemplo: Java, C++ y PHP.

Según la forma de ejecución:

* **Lenguajes compilados**, requieren un compilador que puede leer un programa escrito en un lenguaje de alto nivel y convertirlo a un programa escrito en el código máquina que entiende el hardware.
* **Lenguajes interpretados**, son lenguajes que en lugar de traducir los programas al código máquina, nos da la impresión de ejecutar directamente las operaciones especificadas en el programa fuente mediante un intérprete. Como ventajas, se elimina la necesidad de traducir el código cada vez que se modifica y como desventajas que el rendimiento es menor, debido a la necesidad de realizar la traducción con cada instrucción.

Otros criterios pueden ser: según el paradigma de programación o según su ejecución.

* 1. **Obtención del ejecutable**

Tipos de código, como ya hemos visto anteriormente, en un programa compilado tenemos tres tipos de código: el código fuente, el código objeto y el código ejecutable. Para pasar de uno a otro, es necesario realizar un proceso de compilación.

La compilación a código objeto pasa por varias etapas:

* **Análisis léxico**, se lee secuencialmente todo el código obteniendo las unidades o bloques significativos, denominados tokens.
* **Análisis sintáctico**, se recibe el código fuente en forma de tokens y se realiza un análisis sintáctico que determina la estructura del programa. Es decir, se comprueba si las construcciones de tokens cumplen las reglas de sintaxis definidas en el lenguaje de programación.
* **Análisis semántico**, se comprueba que las declaraciones son correctas, se verifican los tipos de todas las expresiones, si las operaciones pueden realizarse sobre dichos tipos, si los array tiene el tamaño adecuado, etc.
* **Generación de código intermedio**, se genera una representación intermedia similar al código máquina.
* **Optimización de código**, se procede a mejorar el código intermedio, de forma que el resultante sea más fácil y rápido de interpretar por la máquina.
* **Generación de código**, se finaliza con la generación del código objeto del programa.

Por último, la generación de código ejecutable por el enlazador consiste en recuperar las funciones utilizadas por las librerías enlazadas por el código objeto e incorporarlas al código existente para obtener un código ejecutable resultante.

* 1. **Máquinas virtuales**

Se trata de un software que es capaz de ejecutar programas en un entorno como si se tratase de una máquina real.

**Máquinas virtuales de sistema**, permiten ejecutar en la misma maquina física varias máquinas virtuales en un entorno que es visto por la máquina virtual como una maquina real independiente. Algunos ejemplos son VMWare o VirtualBox.

**Máquinas virtuales de proceso**, permite ejecutar un proceso dentro de un sistema operativo proporcionando un entorno de ejecución independiente de la plataforma hardware y del sistema operativo. El ejemplo más extendido es la máquina virtual de Java.

* 1. **Herramientas de creación de código**

Diagramas de flujo, nos permiten modelizar la estructura procedimental de un programa, para realizar un diagrama de flujo, se requieren varios elementos, los más destacados son:

* Inicio y final del programa, se expresan con cajas redondeadas:
* Entrada y salida de datos, todo proceso de I/O se muestra con un rectángulo inclinado:
* Procesos o cálculos internos, se muestran con un rectángulo:
* Toma de decisiones, cuando el resultado de una operación puede condicionar las siguientes operaciones necesarias, se muestra la posible bifurcación mediante un rombo:
* Líneas de flujo, los distintos elementos anteriores, se unen mediante líneas de flujo, que representan el orden de interpretación de los mismos, dichas líneas indican su dirección con una punta de flecha en uno de sus extremos.

Vamos a ejemplificar esto:

*Supongamos que un grupo de amigos realizan un programa que logran vender mediante dos canales de distribución, cada canal proporciona unos determinados ingresos.*

*Se decide que, a la hora de repartir los beneficios, se repartirán entre los componentes del grupo mediante cantidades enteras, de forma que si tras el reparto sobra una cierta cantidad entera que no puede repartirse sin fraccionarla, se dejará de bote.*

Podemos modelizar un programa que resuelve esta situación mediante el siguiente diagrama de flujo:

SI

¿El resto es diferente de cero?

Los ingresos menos el resto → total2=resta de total y resto

total2=total

El reparto entre los miembros → reparto=total2 entre equipo

Inicio

¿Cuantos amigos forman el grupo? → equipo

¿Cuáles son los ingresos por el canal A? → ingreso\_a

¿Cuáles son los ingresos por el canal B? → ingreso\_b

Ingresos totales → total=suma de ingreso\_a e ingreso\_b

Cantidad que sobra al dividir los ingresos entre los

miembros → resto=resto de dividir total entre equipo

NO

Total ingresado → mostrar total

Dinero que recibe cada miembro → mostrar reparto

Cantidad que queda de bote → mostrar resto

Fin

Pseudocódigo, mediante el pseudocódigo podemos describir procedimentalmente un programa usando lenguaje común e instrucciones clave que representan sintaxis habituales en los lenguajes de programación procedimentales. El programa ilustrado en el diagrama de flujo anterior, se podría transcribir en formato de pseudocódigo como sigue:

**Inicio**

**Declarar** reparto=0

**Visualizar** “¿Cuántos amigos forman el grupo?”

**Leer** equipo

**Visualizar** “¿Cuáles son los ingresos por el canal A?

**Leer** ingreso\_a

**Visualizar** “¿Cuáles son los ingresos por el canal B?

**Leer** ingreso\_b

**Declarar** total=ingreso\_a+ingreso\_b

**Declarar** resto=total%equipo

**Si** resto no es igual a 0 **Entonces**

**Declarar** total2=total-resto

**Si no**

**Declarar** total2=total

**Fin si**

**Declarar** reparto=total2/equipo;

**Visualizar** “Total ingresado: ” total

**Visualizar** “Dinero que recibe cada miembro: ” reparto

**Visualizar** “Cantidad que queda de bote: ” resto

**Fin**

En el anterior pseudocódigo vemos varios tipos de instrucciones procedimentales. Podemos resumir, en la siguiente tabla, las más importantes:

|  |  |
| --- | --- |
| Condicional | **Si** condición **Entonces**  Instrucciones  **Fin si** |
| **Si** condición **Entonces**  Instrucciones 1  **Si no**  Instrucciones 2  **Fin si** |
| Condicional múltiple | **Según** sea variable **Hacer**  **Caso** valor 1  Instrucciones 1  **Caso** valor 2  Instrucciones 2  **Caso** valor 3  Instrucciones 3  **Otro caso**  Instrucciones n  **Fin Según** |
| Bucle indeterminado | **Hacer**  Instrucciones  **Hasta** que se cumpla condición |
| **Mientras** que se cumpla condición **Hacer**  Instrucciones  **Fin Mientras** |
| Bucle determinado | **Para** valor inicial **Hasta** valor final **Incrementar** valor y **Hacer**  Instrucciones  **Fin Para** |

1. Se puede ver una traducción del documento en <http://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf> [↑](#footnote-ref-1)