UNIDAD 1: ELEMENTOS DEL DESARROLLO DEL SOFTWARE.

Módulo Profesional: Entornos de Desarrollo

Índice

RESUMEN INTRODUCTORIO

En la presente unidad se estudiarán conceptos relacionados con los elementos de desarrollo del software. En este sentido, se pretenden analizar los diferentes tipos de lenguajes de programación, la función que desempeñan tanto los compiladores como los depuradores, las fases de desarrollo de una aplicación (análisis, diseño, codificación, pruebas, explotación y mantenimiento) así como la importancia de la documentación durante estas fases del desarrollo.

INTRODUCCIÓN

Los lenguajes de programación son parte esencial del software, siendo el conjunto de sus líneas de código la esencia de su función. La implementación de aplicaciones informáticas forma parte del día a día de multitud de empresas de programación, conocidas como factorías de software, donde los empleados trabajan para dar soluciones a diferentes clientes como el sector energético, el transporte, servicios financieros, las telecomunicaciones o administraciones públicas.

La realidad que involucra a este desarrollo no es trivial. Se deben aplicar ciertas metodologías que permitan seguir este proceso de desarrollo basado, normalmente en etapas. Una de las metodologías más ampliamente aceptadas es la formada por las siguientes 5 fases: análisis, diseño, implementación, pruebas y mantenimiento. Todos los profesionales involucrados deben conocer en profundidad cuál es la función que desarrolla en cada una de ellas.

De forma transversal a todo el desarrollo, la documentación será un pilar fundamentar que debe tenerse siempre presente. Es imprescindible documentar cada proceso, fase o etapa en detalle para disponer de una buena gestión del conocimiento de todo el desarrollo de software realizado y acudir a él siempre que sea necesario.

CASO INTRODUCTORIO

En una de las factorías de software de Indra trabaja Juan, que como jefe de proyecto dirige su propio equipo de desarrollo. Un día como hoy recibe una petición de un cliente externo para crear una aplicación que gestione una biblioteca. Tanto Juan como los miembros de su equipo conocen la importancia de aplicar una metodología de desarrollo que ayude a organizar el trabajo. Para ello, la primera tarea a realizar es llevar a cabo una reunión entre Juan y sus analistas con el cliente, para así concretar los requisitos de la aplicación. Posteriormente, y atendiendo a esta lista de requisitos, los diseñadores realizarán los diagramas oportunos que recojan las características técnicas de cada situación. Una vez realizado todo el diseño, se procederá a implementar (desarrollar) la aplicación en el o en los lenguajes de programación acordados. A medida que se avanza en esta fase, los tester del equipo de Juan realizarán las pruebas oportunas. Para finalizar el proceso, dos o tres miembros del equipo de Juan realizarán la instalación del programa en los propios equipos del cliente, es decir, en los terminales de la biblioteca, realizando pruebas oportunas de todos los requisitos acordados. Por último, se establecerán unas pautas para llevar a cabo el mantenimiento de este software de forma periódica.

Al finalizar la unidad el alumnado:

- ✓ Conocer los diferentes tipos de lenguajes de programación.
- ✓ Conocer los procesos de compilación y depuración de los programas.
- ✓ Comprender y aplicar una metodología de desarrollo basada en las siguientes etapas: análisis, diseño, implementación, pruebas y mantenimiento.
- ✓ Será capaz de comprender la importancia de la documentación en el desarrollo de software.
- ✓ Conocer los tipos de pruebas como las de caja blanca y negra.

1. INTRODUCCIÓN A LOS LENGUAJES DE PROGRAMACIÓN

Se entiende por **programa informático** a una serie de comandos ejecutados por el equipo que permiten obtener, modificar y mostrar información. En otras palabras, gestionar la información.

Un programa, o una parte de un programa, estará formado por es un simple archivo de texto (escrito a través de procesador o editor de texto), llamado archivo fuente o código fuente. Los ordenadores, a través de su CPU, sólo son capaces de procesar código binario, es decir, una serie de 0s y 1s, por tanto, necesita de un lenguaje de programación para escribir de manera legible, esto quiere decir, escribir a través de comandos, que facilite la compresión de que lo se necesita ejecutar.

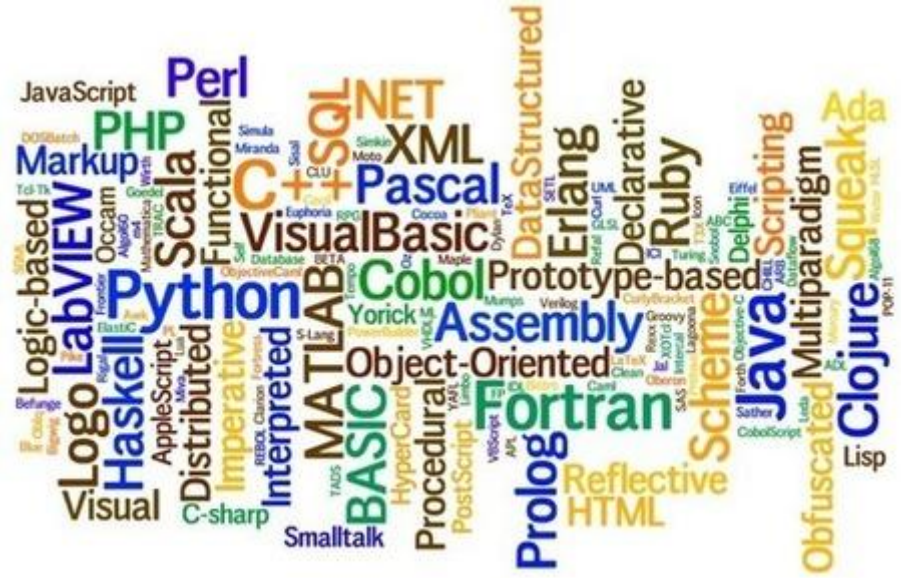


Imagen: Collage de diferentes lenguajes de programación
Fuente: https://www.hackingpublico.net/lenguajes-de-programacion-i/

A través de un lenguaje de programación se realiza la escritura de un programa, y está formado por conjunto de símbolos y palabras reservadas que permitirán al programador aportar instrucciones a un ordenador para que sean ejecutadas. El primer paso a realizar, por tanto, es escribir el programa en un lenguaje de programación. Se recomienda que este lenguaje seleccionado sea bien conocido por el programador.

Una vez realizado, el programa escrito se va a traducir al lenguaje máquina. Este proceso es al que se conoce como **compilación**. Cada lenguaje de

programación tiene su propio compilador (excepto los lenguajes interpretados).

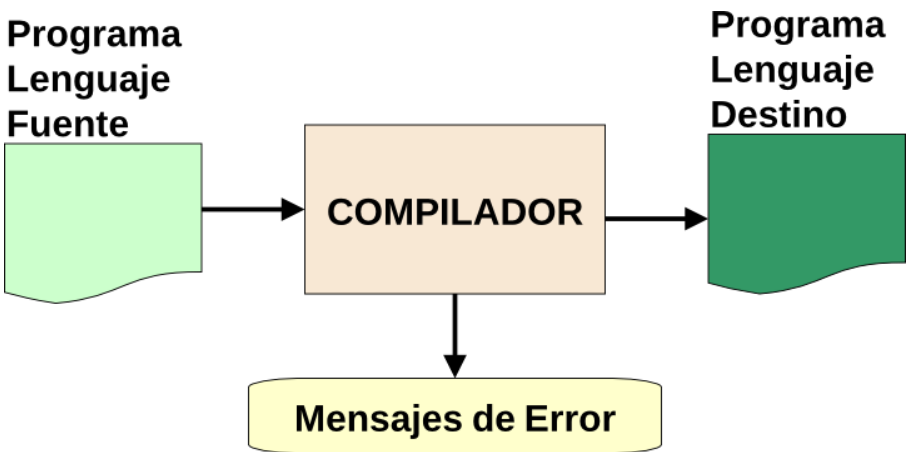


Imagen: Función del Compilador

El archivo que contiene el código fuente incluye el código del programa que ha escrito el programador. Este archivo, debe compilarse una vez se ha completado. La compilación se realiza en dos pasos:

- El compilador transforma el código fuente en código objeto y lo guarda en un archivo objeto, es decir, traduce el archivo fuente a lenguaje máquina. Algunos compiladores crean un archivo en ensamblador, se trata de un lenguaje similar al lenguaje máquina ya que posee las funciones básicas, con la diferencia de que puede ser leído por el ser humano.

- Seguidamente, el compilador llama a un editor de vínculos (enlazador o ensamblador) que permite insertar los elementos adicionales (funciones y bibliotecas) a los que hace referencia el programa dentro del archivo final.

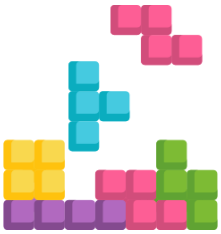
Imagen: Compilación

El resultado será un archivo ejecutable, que será el punto inicial donde arrancará la aplicación.

PARA SABER MÁS

Para ampliar información sobre el proceso de compilación se recomienda descargar el siguiente material:

http://www.uhu.es/04004/material/Transparencias3.pdf



EJEMPLO PRÁCTICO

Se procede a implementar un programa Java que permite saber si un número introducido por pantalla es o no número primo. Una vez escrito el programa, ¿cuál es la primera tarea a realizar?

Solución:

Se deberá proceder a la compilación del mismo para conocer si existe algún tipo de error y poder subsanarlo. Si no se encuentran errores, se procede a su ejecución.

2. TIPOS DE LENGUAJES DE PROGRAMACIÓN

A lo largo de este apartado veremos los tipos de lenguaje de programación dividiéndolos por un lado según los niveles y por otro en función del paradigma de programación.

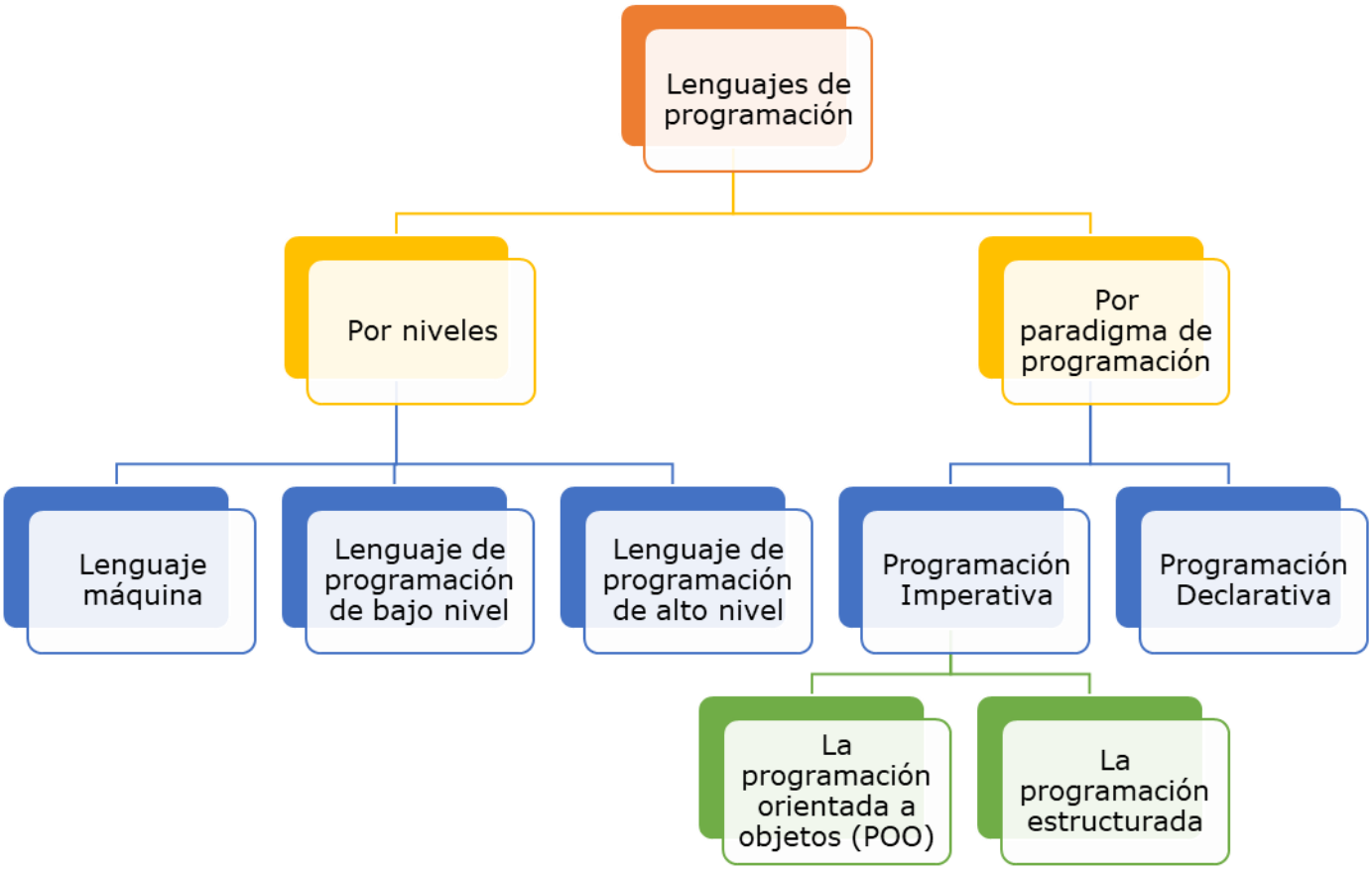


Imagen: Tipos de lenguajes de programación

2.1 Por niveles

En esta clasificación, los lenguajes se diferencian principalmente por el nivel al que pertenecen, donde en el nivel más bajo se encuentran los lenguajes que son interpretados por la computadora. Los lenguajes de programación de mayor nivel necesitarán de intérpretes o compiladores.

- **El lenguaje máquina**: es el lenguaje de programación que entiende directamente la computadora o máquina. Este lenguaje de programación utiliza el alfabeto binario, es decir, el 0 y el 1. Con estos dos únicos dígitos, se forman las cadenas binarias

(combinaciones de ceros y unos) formando las instrucciones entendibles por el microprocesador.

- **Lenguajes de programación de bajo nivel**: Son mucho más fáciles de utilizar que el lenguaje máquina, pero dependen mucho de la máquina o computadora. El lenguaje ensamblador fue el primer lenguaje de programación que trató de sustituir el lenguaje máquina por otro mucho más entendible para los seres humanos. Los lenguajes de bajo nivel consiguen crear programas muy rápidos, pero son muy difíciles de aprender y tienen el inconveniente de ser específicos de cada procesador, con lo que si se ejecuta en otro procesador se deberá reescribir.

- **Lenguajes de programación de alto nivel**: Este tipo de lenguajes de programación son, por regla general, independientes de la máquina, y se puede usar en cualquier otro computador con pocas modificaciones. Los lenguajes de programación de alto nivel necesitan de un programa intérprete, como se ha comentado anteriormente, llamado compilador, que traduzca este lenguaje de programación de alto nivel a uno de bajo nivel (lenguaje máquina) que la computadora pueda entender. De forma general, los lenguajes de programación de alto nivel son fáciles de aprender y suelen usan palabras del lenguaje natural (casi siempre palabras en inglés como FOR, WHILE, IF, etc).

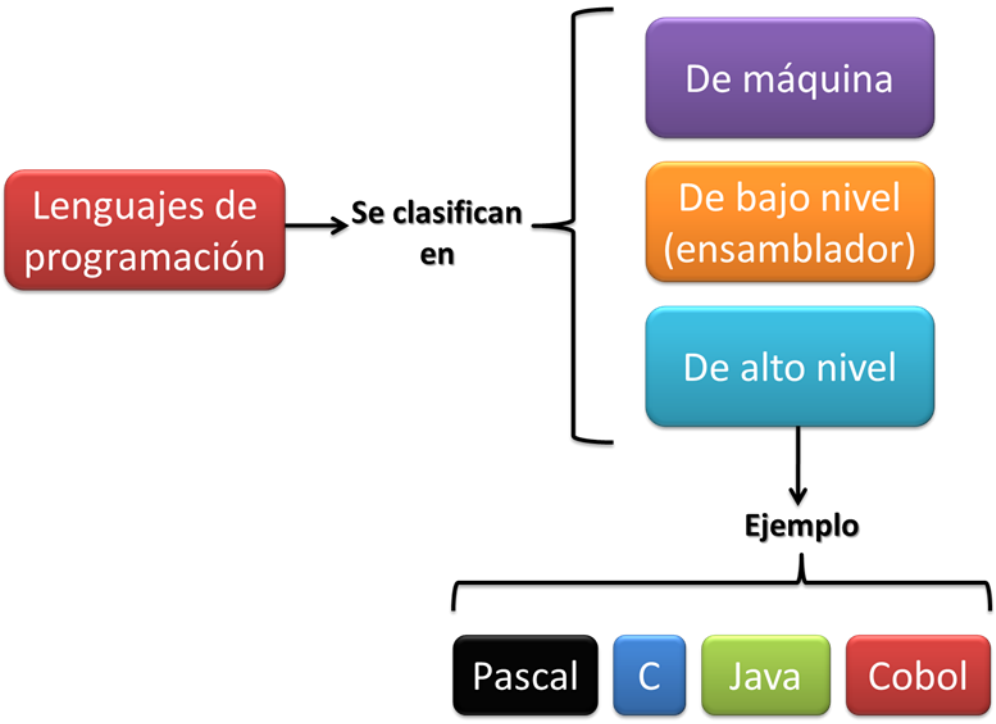


Imagen: Clasificación de los lenguajes de programación

2.2 Paradigmas de programación

Otra clasificación de los lenguajes de programación es según su paradigma. Los paradigmas de programación son propuestas tecnológicas y se enfocan a resolver problemas definidos y delimitados.

- **Programación Imperativa**: Con este paradigma se resuelven los problemas especificando una secuencia de acciones a realizar a través de uno o varios procedimientos denominados funciones o subrutinas. Las sentencias modifican el estado del programa (el estado de un programa viene definido por la instrucción que se está ejecutando y el valor de los datos que está tratando en un momento dado).

- **La Programación Orientada a Objetos** y **la Programación Estructurada** forman parte de la programación imperativa, ya que, al programar con algunos de estos paradigmas, se describe la secuencia que debe seguir el programa para resolver un problema dado.

- **Programación Declarativa**: Con la programación declarativa, la solución es alcanzada a través de mecanismos internos de control, no se especifica exactamente cómo llegar a ella. En la programación declarativa, una expresión puede ser sustituida por el resultado de ser evaluada. Dentro de la programación declarativa se engloban la programación funcional, cuyo lenguaje más utilizado es Haskell y la programación lógica con Prolog como principal lenguaje utilizado.

PARA SABER MÁS

Para ampliar información sobre los lenguajes de programación:

http://recursostic.educacion.es/observatorio/web/ca/componente/content/article/502-monografico-lenguajes-de-programacion?showall=1

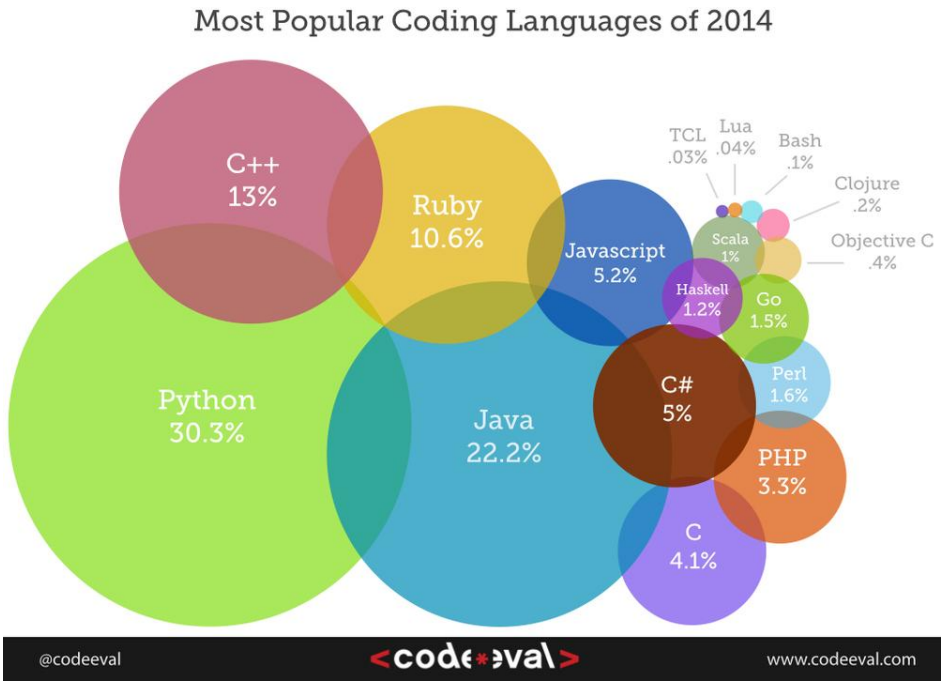


Imagen: Lenguajes de programación más populares en el año 2014
Fuente: http://blog.codingforinterviews.com/best-programming-language-jobs/

3. COMPILADORES Y DEPURADORES

Un **compilador** es un programa que traduce un software escrito en un lenguaje de alto nivel o lenguaje de cuarta generación, denominado programa fuente, en otro denominado programa objeto.

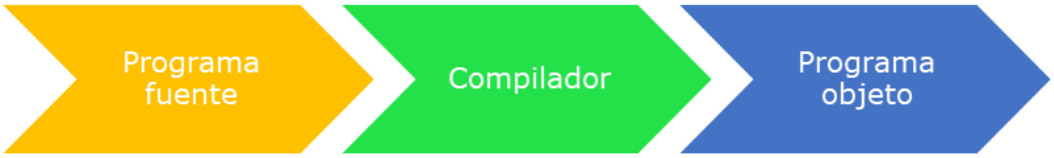


Imagen: Esquema del Compilador

El programa objeto puede almacenarse en memoria auxiliar para ser ejecutado posteriormente sin necesidad de volver a realizar la traducción. La traducción de un compilador consta de 2 etapas: la etapa de **análisis** y la etapa de **síntesis**.

La compilación es un proceso complejo que consume a veces un tiempo superior a la ejecución del propio programa. En cualquiera de las fases, el compilador puede generar mensajes de error detectados en el programa fuente, cancelando en ocasiones la compilación, siendo necesario que el

programador realice las correcciones correspondientes en el programa fuente.

En la fase de análisis se ejecutan tres fases fundamentales: el análisis léxico, el análisis sintáctico y el análisis semántico. Cada una de estas fases la realiza un analizador determinado. La síntesis del programa objeto conduce a la generación de código intermedio, su optimización y la generación del código final.

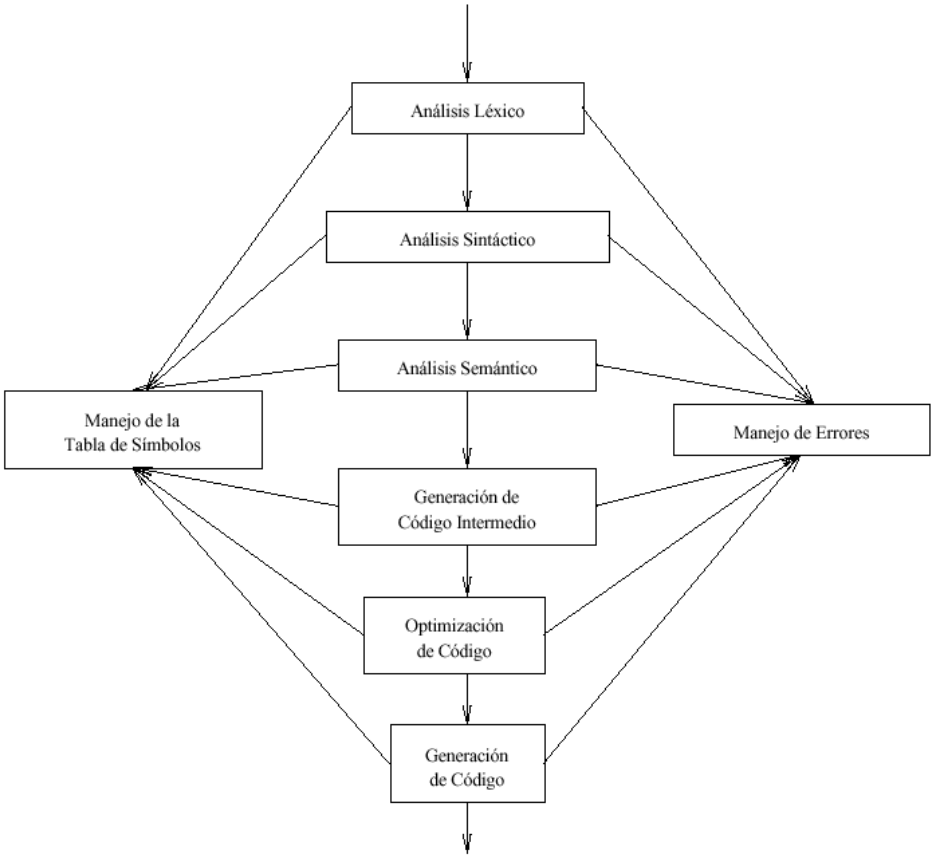


Imagen: Fases de la compilación
Fuente: http://robots-men.blogspot.com/2010/05/compiladores.html

3.1 Análisis

Analizador léxico: Examina el programa fuente para localizar las unidades básicas de información pertenecientes al lenguaje, denominadas **tokens**.

Los tokens son caracteres o secuencias de caracteres que tienen un significado especial en el programa fuente (palabras reservadas, identificadores, etc.). El analizador léxico pasa por alto los comentarios, aísla los tokens, identifica el tipo y los almacena en unas tablas denominadas tablas de símbolos. Se puede considerar que, como resultado

del análisis léxico, se obtiene una representación del programa formada por la descripción de símbolos en las tablas y una secuencia de símbolos junto con la referencia a la ubicación del mismo en la tabla. Esta representación contiene la misma información que el programa fuente, pero en una forma más compacta, no presentando ya el código como una secuencia de caracteres, sino de símbolos.

La información almacenada en las tablas de símbolos se completa y utiliza en las fases posteriores de la traducción. La tabla de símbolos es la estructura que almacena toda la información relativa a identificadores, variables, constantes, estructuras de datos y otros elementos pertenecientes al programa que se está compilando. Esta información suele incluir el tipo de cada elemento, sus dimensiones y otras características. La tabla de símbolos está relacionada con todas las fases del proceso de compilación y está presente en todas ellas, siendo utilizada por cada uno de los módulos. Algunas de las fases se ocupan de completar esta tabla mientras que otras utilizan la información en ella contenida.

Analizador sintáctico: La sintaxis de un lenguaje de programación especifica cómo deben escribirse los programas, mediante un conjunto de reglas de sintaxis o gramática del lenguaje. Un programa es sintácticamente correcto cuando sus estructuras están escritas en un orden correcto. Por ejemplo, las asignaciones normalmente en la mayoría de los lenguajes requieren que a la izquierda del operador de asignación aparezca un identificador y a la derecha del mismo aparezca una expresión, la cual deberá cumplir también un esquema especificado por un conjunto de reglas para las expresiones. De esta forma el analizador sintáctico recibe la cadena de tokens obtenida por el analizador léxico y busca en ella los posibles errores sintácticos que aparezcan. Entre los posibles errores que se pueden presentar están la duplicidad de identificadores de variables o las instrucciones escritas incorrectamente.

Un analizador sintáctico es un programa que reconoce si una o varias cadenas de caracteres forman parte de un determinado lenguaje. Se han definido varios sistemas para definir la sintaxis de los lenguajes de programación.

Entre ellos, cabe destacar la notación BNF (Backus–Naur Form) y los diagramas sintácticos. Entre los métodos para realizar el análisis sintáctico, uno bastante sencillo es realizar un procedimiento para reconocer cada uno de los símbolos terminales del lenguaje. Con este esquema de análisis el analizador puede generar mensajes de error cuando se encuentre un símbolo no esperado. Los lenguajes habitualmente reconocidos por los analizadores sintácticos son los lenguajes libres de contexto.

Cabe notar que existe una justificación formal que establece que los lenguajes libres de contexto son aquellos reconocibles por un autómata de pila, de modo que todo analizador sintáctico que reconozca un lenguaje libre de contexto es equivalente en capacidad computacional a un autómata de pila. Los analizadores sintácticos fueron extensivamente estudiados durante los años 70 del siglo XX, detectándose numerosos patrones de funcionamiento en ellos, lo que permitió la creación de programas generadores de analizadores sintácticos a partir de una especificación de la sintaxis del lenguaje.

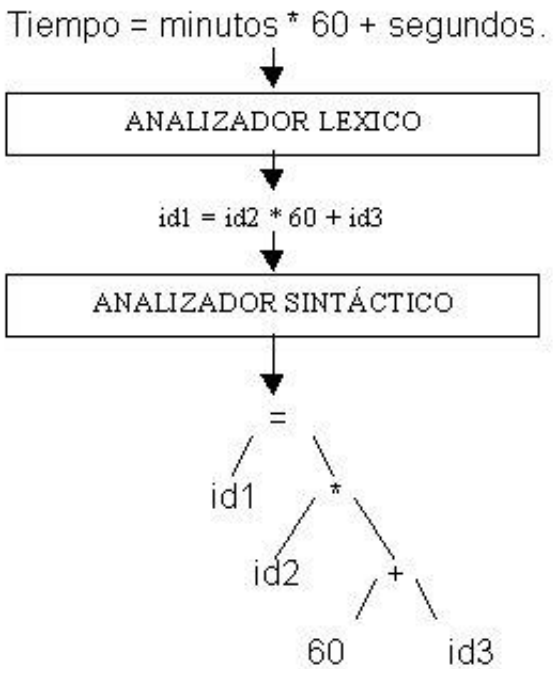


Imagen: Ejemplo analizador léxico y sintáctico
Fuente: https://www.ecured.cu/Analizador_sint%C3%A1ctico

Analizador semántico: La fase de análisis semántico revisa el programa fuente para tratar de encontrar errores semánticos y reúne la información sobre los tipos para la fase posterior de generación de código. En el análisis semántico se realizan, entre otras funciones, la verificación de los tipos de las variables declaradas, comprobando si cada operador tiene operandos permitidos por la especificación del lenguaje. Al producirse un error en cualquiera de las fases de análisis, éste se comunica al programador mediante un mensaje indicativo del error producido y el lugar en que éste ocurrió. En algunos casos, ciertos errores no afectan gravemente al proceso de ejecución, e incluso permiten la ejecución del programa. Este tipo de errores se denominan advertencias o warnings.

3.2 Síntesis

En la fase de síntesis se lleva a cabo la generación de código intermedio, la optimización de código y la generación del código final.

Generación de código intermedio: Si no se han producido errores en alguna de las etapas anteriores, este módulo realiza la traducción a un código interno propio del compilador, denominado código intermedio, a fin de permitir la transportabilidad del lenguaje a otros ordenadores. Para un determinado lenguaje de alto nivel se hace común todo el proceso de análisis y generación de código intermedio y es la generación de código objeto la que se particulariza para cada tipo de microprocesador.

Optimización de código: La misión del optimizador de código consiste en recibir el código intermedio y optimizarlo atendiendo a determinados factores, tales como la velocidad de ejecución o el tamaño del programa objeto. Se realiza una optimización, analizándose el programa objeto globalmente. Existen compiladores que permiten al usuario omitir o reducir las fases de optimización, disminuyéndose así el tiempo global de la compilación. Tras la optimización del código intermedio se lleva a cabo la generación del código final del programa. Esta fase se basa en una traducción del código intermedio optimizado.

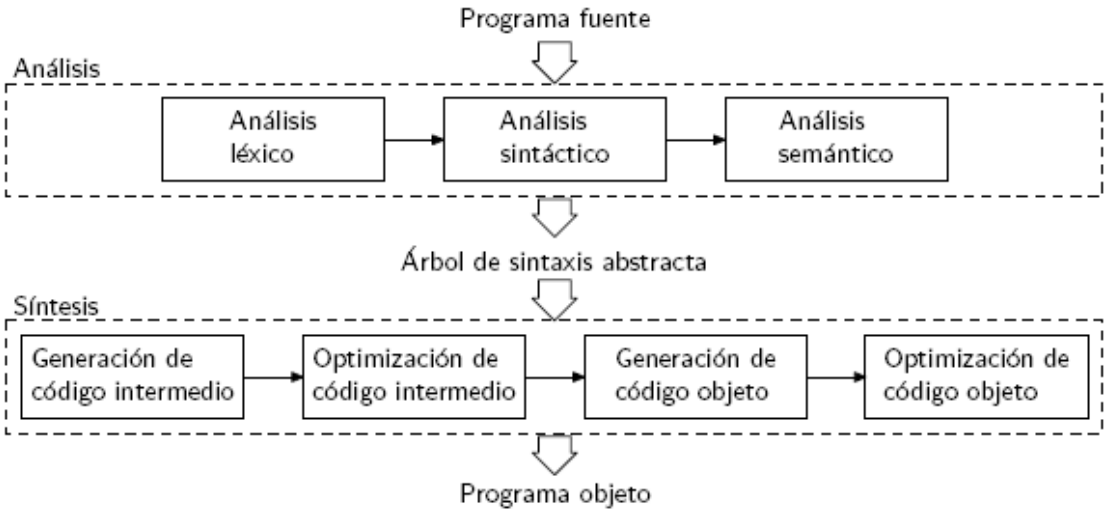


Imagen: Fases de análisis y de síntesis
Fuente: https://loscompiladores.wordpress.com/category/procesos-postcompilacion/

3.3 Depuradores

Los **depuradores** facilitan la detección y, en muchos casos, la recuperación de los errores producidos en las diferentes fases de la compilación. El compilador, cuando detecta un error, trata de buscar su localización exacta y su posible causa para presentar al programador un mensaje de diagnóstico, que será incluido en el listado de compilación.

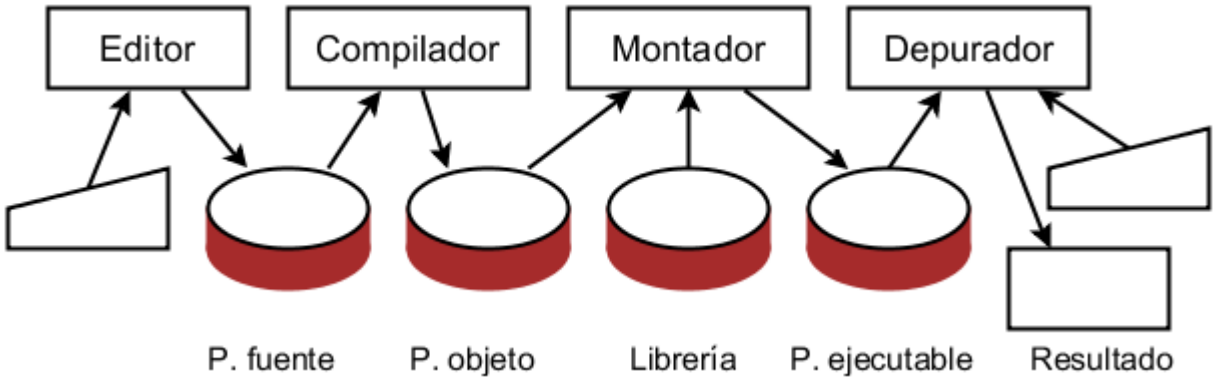


Imagen: Diferentes procesos por los que pasa una aplicación software
Fuente: http://lml.ls.fi.upm.es/ep/entornos.html

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Las 5 **fases o etapas** del desarrollo de una aplicación software son:

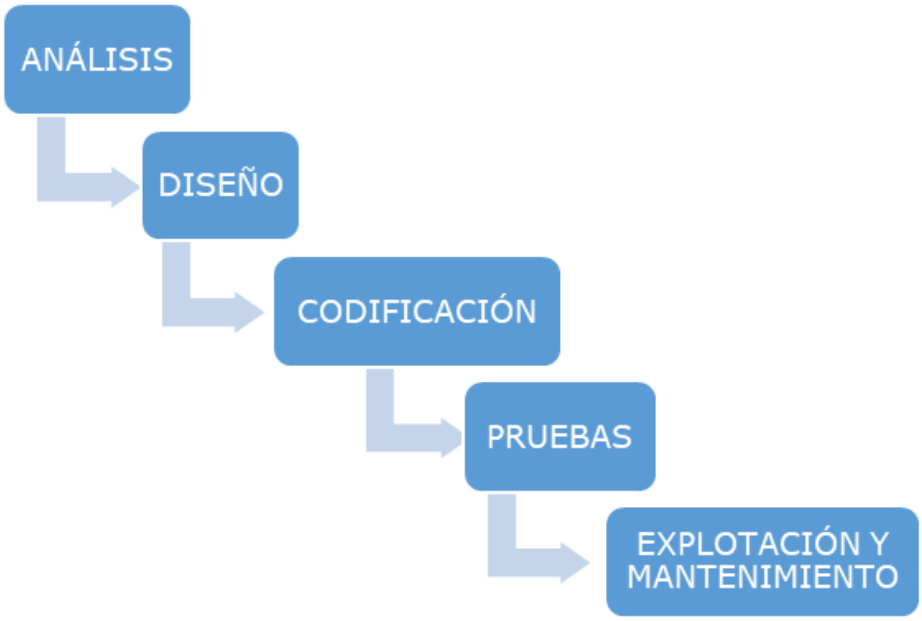


Imagen: Fases del desarrollo de una aplicación

4.1 Análisis

En esta fase se establece el producto a desarrollar, siendo necesario especificar los procesos y la estructura de datos, en definitiva, la funcionalidad del software que se van a desarrollar en la **toma de requisitos**.

En esta fase debe existir una clara comunicación entre el cliente y el analista para poder conocer todas las necesidades que precisa la aplicación. En el caso de falta de información ya sea por falta de comunicación o conocimientos por parte del cliente, se puede recurrir al desarrollo de prototipos (software de pruebas que irán definiendo la aplicación final) para saber con más precisión sus requerimientos y funcionalidades.

CITA

El principio de Pareto establece que "el 80% del tiempo de elaboración de un programa se debe destinarlo a la planificación y el 20% restante a la codificación".

En la fase de análisis se pueden definir las siguientes etapas:

Diagramas de flujo de datos

- Sirven para conocer el comportamiento del sistema mediante representaciones gráficas.

Modelos de datos

- Sirven para conocer las estructuras de datos y sus características. (Entidad relación y formas normales)

Diccionario de datos

- Sirven para describir todos los objetos utilizados en los gráficos, así como las estructuras de datos

Definición de los interfaces de usuario

- Sirven para determinar la información de entrada y salida de datos.

Imagen: Etapas de la fase de análisis

Al final de esta fase de han de tener claro las especificaciones, es decir, las funcionalidades y requerimientos a desarrollar en la aplicación.

4.2 Diseño

En esta fase se alcanza con mayor precisión una solución óptima de la aplicación, es decir, se deben tener en cuenta los recursos físicos del sistema (tipo de equipos, servidores, periféricos, comunicaciones, etc.) y los recursos lógicos (sistemas operativos, programas de utilidad, lenguaje/s de programación, bases de datos, etc.) donde se ejecutará la aplicación. En el diseño estructurado se definen las siguientes etapas:

Imagen: Etapas de la fase de diseño

4.3 Codificación

Una vez se tienen claras las funcionalidades y los requerimientos y se ha diseñado como será la aplicación a nivel de módulos, clases, etc. en la etapa de diseño, se procede a traducir y crear el código fuente en el lenguaje de programación que se decidió en la etapa de diseño, además del entorno u entornos de desarrollo y las tecnologías a utilizar. La implementación de código implica realizar pruebas de funcionamiento para comprobar si el software realiza las funciones programadas. No obstante, en la siguiente fase se realizan pruebas más exhaustivas y de mayor precisión.

4.4 Pruebas

Tras la codificación de la aplicación se deben de realizar las pruebas necesarias para comprobar la calidad y estabilidad del programa. Existen dos métodos generales en el diseño de pruebas de programas:

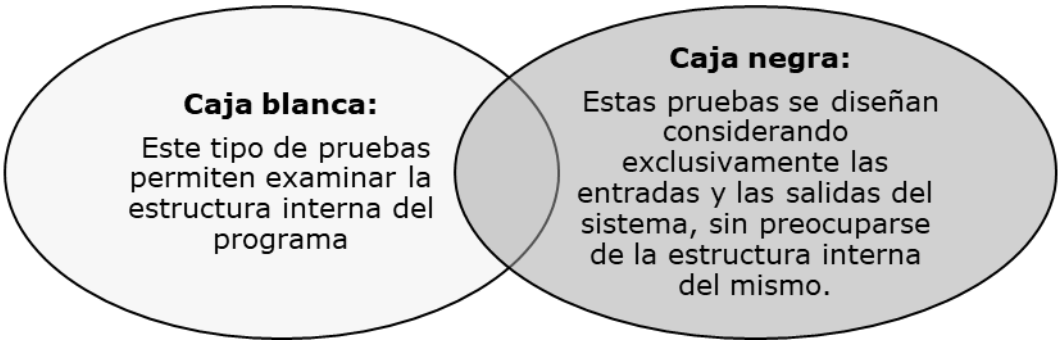


Imagen: Pruebas de caja blanca y de caja negra

Se pueden distinguir 3 fases en el proceso de pruebas:

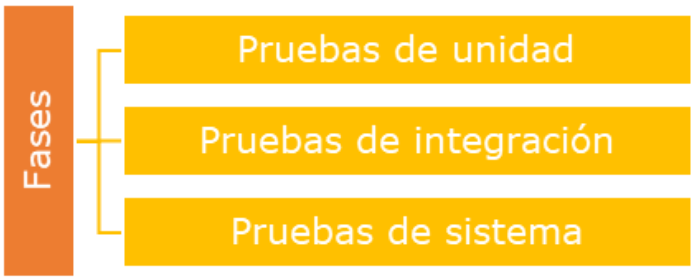


Imagen: Fases del proceso de pruebas

Pruebas de unidad: Están destinadas a la prueba de cada una de las unidades del sistema, esto es, se centran en la verificación de la menor unidad en el diseño del software (el módulo). Se verifican por tanto las estructuras de datos, la interfaz, las estructuras de control, el control de errores, las condiciones límite de funcionamiento y la eficiencia del código. Dentro de cada una de éstas, las pruebas se centrarán en unos u otros aspectos.

Pruebas de integración: Una vez que se ha comprobado la corrección de errores en cada uno de los módulos por separado con las pruebas de unidad, la siguiente fase de pruebas va dirigida a probar la correcta interconexión de los módulos, ya que esto no queda garantizado con las pruebas de unidad. Algunos de los problemas que pueden surgir son la

modificación no deseada de variables globales (causando posibles efectos laterales) y el acarreo de imprecisiones por parte de varias funciones involucradas en el mismo cálculo. La integración de los módulos ha de realizarse de manera incremental, es decir, el programa se irá construyendo añadiendo los módulos poco a poco, de manera que los posibles errores sean fácilmente localizables y corregibles en cada uno de los ellos. Según la estrategia de integración elegida, se pueden distinguir entre una integración entre módulos **ascendente** o bien una integración **descendente**.

- La integración **ascendente** comienza con la construcción y prueba de los módulos de más bajo nivel y va realizando la integración de los mismos en módulos mayores, es decir, de manera ascendente. Los programas probados siguiendo esta estrategia no existen hasta que se han añadido y probado todos los módulos. Además, se consume mucho tiempo realizando pruebas para cada módulo que se genera, no siendo útiles muchos de los programas realizados para realizar dichas pruebas.

- En la integración **descendente** se construye primero el programa principal, y a éste se le van incorporando los módulos conforme se vayan implementando. Esta estrategia permite que las interfaces de nivel más alto sean las primeras en probarse. Pueden surgir problemas cuando se requiere el procesamiento de algún módulo de nivel inferior para poder probar adecuadamente los de nivel superior. Puede ser difícil encontrar datos de entrada del nivel más alto que sirvan para probar correctamente un módulo de nivel inferior.

La selección de una u otra estrategia depende de las características del software que se esté desarrollando. No obstante, muchas veces se opta por una solución mixta que emplea ambas estrategias. En estas ocasiones se suele utilizar una integración descendente para los módulos superiores y una ascendente para los módulos de más bajo nivel.

Pruebas de sistema: En esta fase, el sistema construido debe quedar en perfecto estado de funcionamiento. Para ello se realizan diversas tareas como:

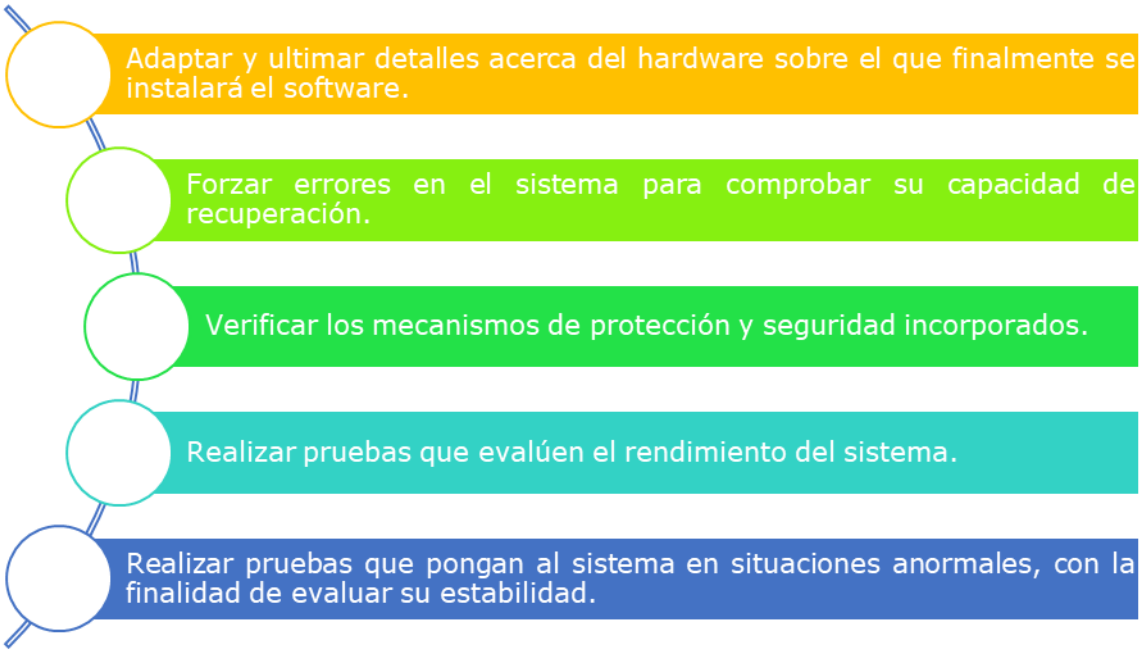


Imagen: Tareas de las pruebas de sistema

4.5 Explotación y mantenimiento

La **explotación** consiste en realizar la implantación de la aplicación en el sistema o sistemas físicos donde van a funcionar habitualmente (entorno de producción) y su puesta en marcha para comprobar el buen funcionamiento:

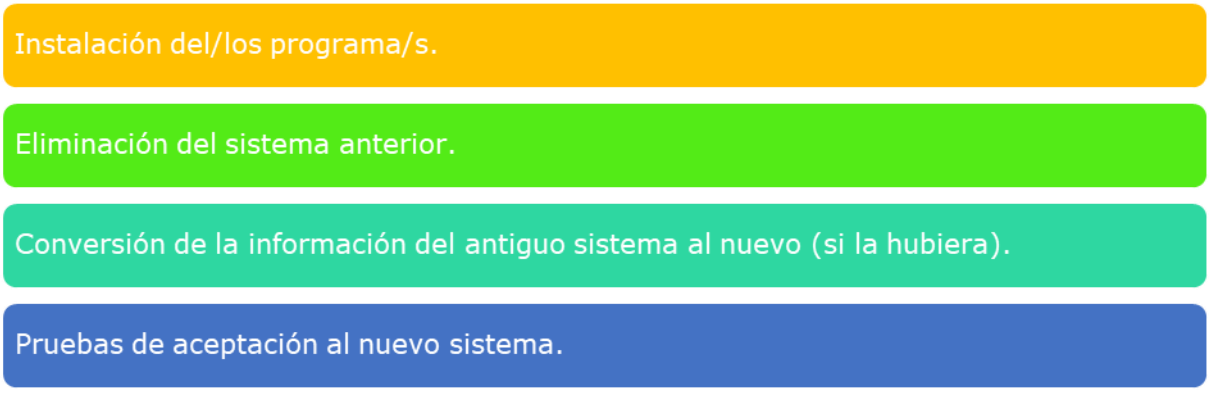


Imagen: Actividades en la fase de explotación

Al final de la explotación se le facilita al cliente toda la documentación necesaria para la explotación del sistema (manuales de ayuda, de uso, guía de la aplicación, información sobre el soporte, etc.).

En cuanto al **mantenimiento**, es la fase que completa el ciclo de vida y en ella se deben solventar los posibles errores o deficiencias de la aplicación. Es posible que se necesiten aclarar nuevas especificaciones que el cliente no informó de forma correcta, lo cual implicaría reiniciar el ciclo de vida, siendo esta la opción menos deseable de todas. Existen fundamentalmente 3 tipos de mantenimiento:

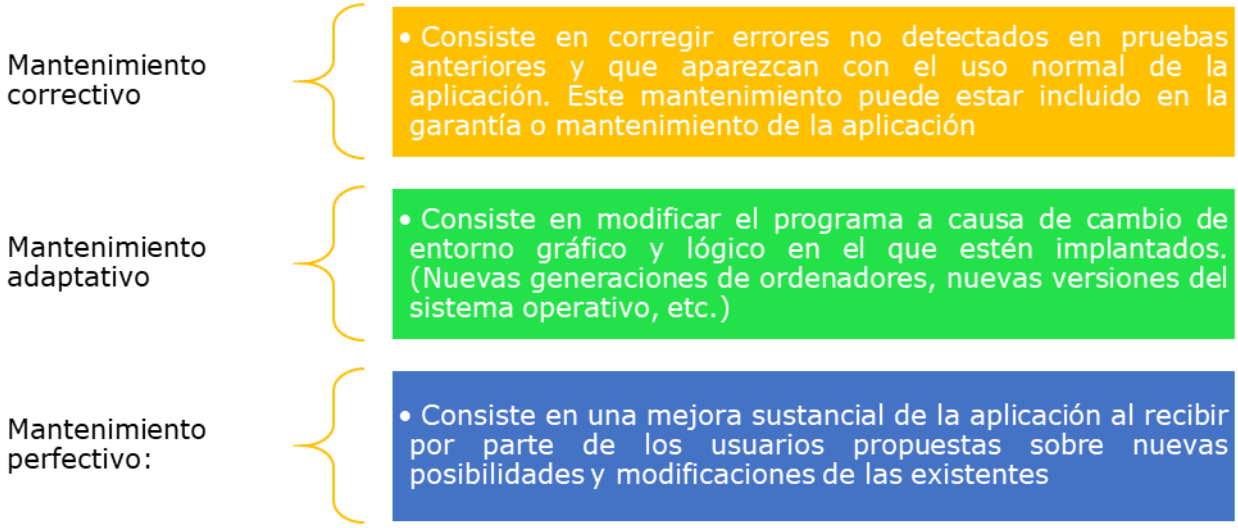


Imagen: Tipos de mantenimiento

ENLACE DE INTERÉS

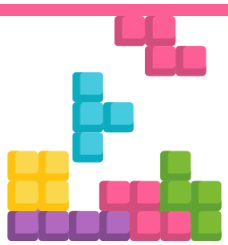
En la siguiente web se recoge mucha información sobre el desarrollo software:

http://www.desarrollo-software.com/

Los tipos de mantenimiento adaptativo y perfectivo reinician el ciclo de vida, debiendo proceder de nuevo al desarrollo de cada una de sus fases para obtener un nuevo producto.

Durante todas las fases del desarrollo del software es muy importante ir documentando todo, utilizando tanto documentación externa como interna. Son muchos los programadores que aún no documentan el código internamente con los problemas que esto conlleva. Existen ocasiones en las que las empresas no disponen de informáticos especializados en la plantilla por lo que es de vital importancia dejarles un buen manual de instalación y uso de la aplicación. Además, por si en el futuro fuera necesario, disponer de todo el código documentado para que sea más entendible para

posteriores cambios o mejoras por su parte o la de otro compañero es una práctica imprescindible.



EJEMPLO PRÁCTICO

Se observa que un software realizado por el equipo de desarrollo tiene cierta funcionalidad que parece ralentizar el equipo donde se encuentra instalado. Las diferentes opciones y acciones cumplen los requisitos del cliente, sin embargo, parece que el sistema operativo se encuentra en un estado de máximo consumo de recursos. ¿Cuáles son las pruebas que se han de realizar? ¿Cómo se puede proceder a resolver este tipo de inconvenientes?

Solución:

En primer lugar, es necesario realizar un chequeo del equipo informático donde se están realizado las ejecuciones. Es probable que el equipo necesite algún tipo de actualización software o revisión del hardware por si estuviera obsoleto.
Si el equipo se encuentra en un estado óptimo, sería recomendable realizar las mismas pruebas en un equipo diferente, y comprobar su funcionalidad. En función de ello, es posible obtener 3 resultados: O bien encontrar la misma lentitud que en el primer equipo, aumentar esa lentitud o reducirla. Si la lentitud es similar o incluso mayor, se hace necesario revisar el código de la aplicación software e ir analizando cada funcionalidad, ya que es posible que exista algún cuello de botella (acceso a una base de datos o servidor web, recursos hardware bajo mínimos, etc.). En caso de que la lentitud desaparezca, es conveniente realizar una comparativa entre los equipos de prueba, para intentar observar qué está ocurriendo en el primero de ellos y actuar en consecuencia.

Es importante tener en cuenta que existen aplicaciones software en las que va a haber un acceso concurrente importante (aplicaciones bancarias, consultas, etc.). Para ello es importante realizar un tipo de pruebas llamadas pruebas de rendimiento, capaces de detectar anomalías como las del ejemplo anterior.

COMPRUEBA LO QUE SABES

¿Es lo mismo una prueba de estrés que una prueba de rendimiento? Analiza tu respuesta.

5. DOCUMENTACIÓN DURANTE LAS FASES DEL DESARROLLO

El ciclo de vida de una aplicación o las diferentes fases por las que pasa son: análisis, diseño, codificación, pruebas, explotación y mantenimiento.

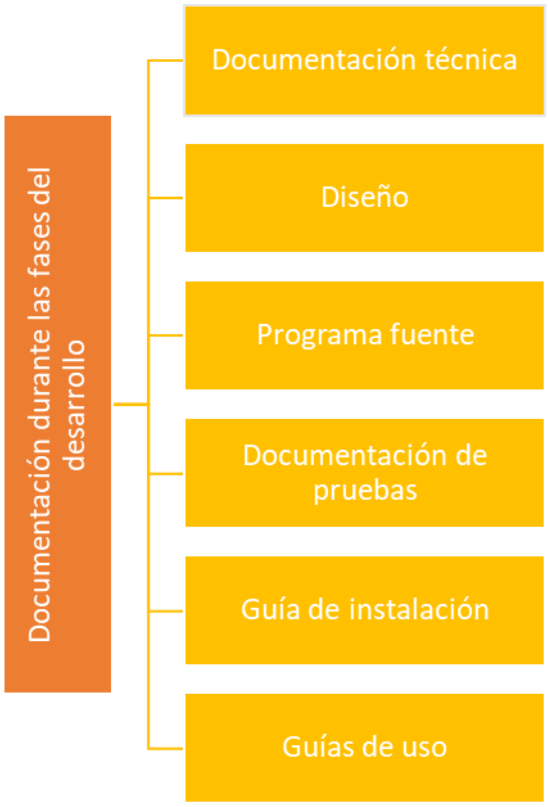


Imagen: Documentación durante las fases del desarrollo

CITA

"If your program isn't worth documenting, it probably isn't worth running" J. Nagler. 1995, Coding Style and Good Computing Practices.

5.1 Documentación técnica

La guía técnica o manual técnico es el documento donde queda reflejado el diseño del proyecto o aplicación, la codificación de los programas y las pruebas realizadas para su correcto funcionamiento. Este documento está

destinado al personal técnico en informática (analistas y programadores). El principal objetivo de este documento es el de facilitar el desarrollo, la corrección y el futuro mantenimiento de los programas de forma rápida y precisa.

5.2 Diseño

Es el documento donde queda reflejada la solución de la aplicación, basándose en las necesidades del usuario y en el análisis efectuado previamente por los analistas con el fin de obtener los requerimientos de la aplicación. Este documento está destinado a los programadores que van a desarrollar los programas de la aplicación, con objeto de que la codificación sea efectuada con la mayor calidad y rapidez posible. Este documento debe estar realizado de tal forma que permita la división del trabajo de programación, para que varios programadores puedan trabajar de forma independiente, aunque coordinados por uno de los programadores o por un analista. En el cuaderno de carga hay que ajustarse a las características del sistema físico donde se va implantar dicha aplicación, así como al tipo de sistema operativo y lenguaje que se va a utilizar. El diseño de la aplicación se puede dividir en las siguientes etapas:

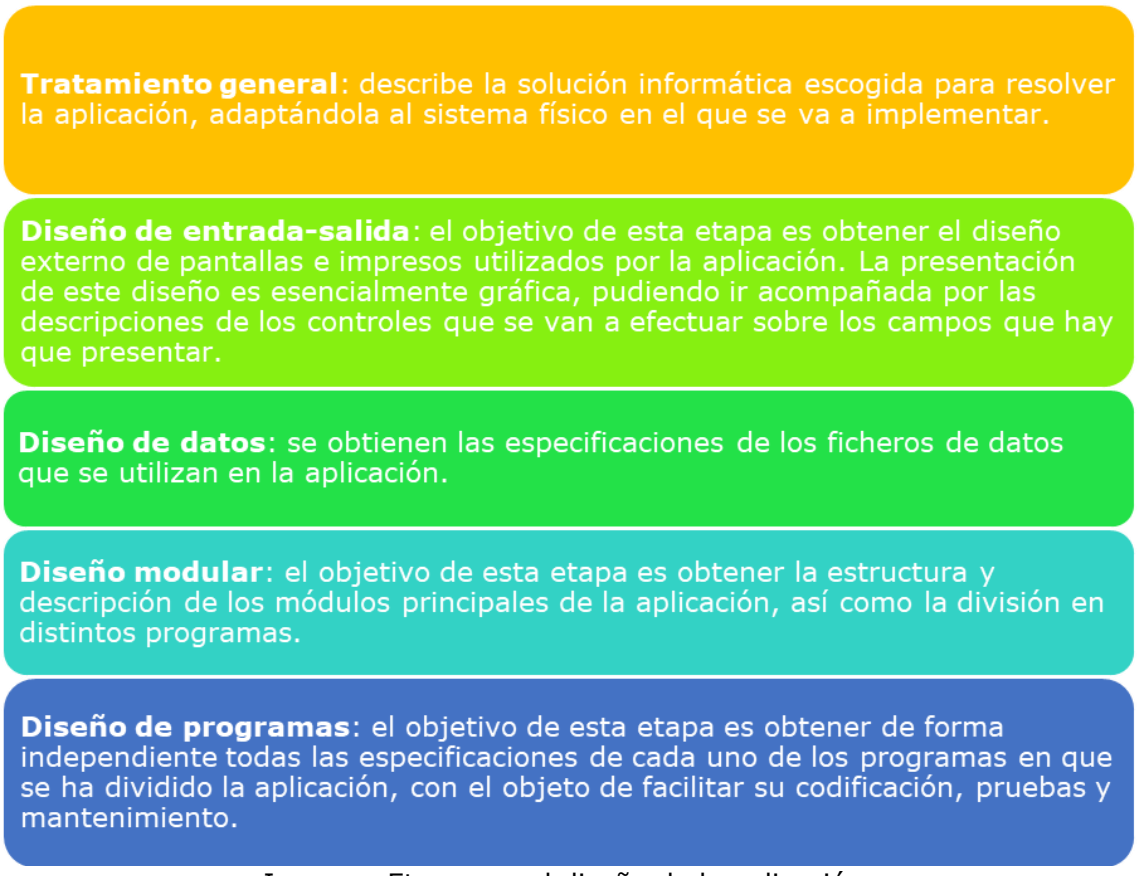


Imagen: Etapas en el diseño de la aplicación

5.3 Programa fuente

En el código fuente escrito en los programas se debe cuidar que éste sea claro, legible y bien documentado. Para ello se deben seguir una serie de normas referentes al estilo de escritura, las zonas en las que ubicar ciertas secciones del código y no escatimar en colocar comentarios en el código fuente. En el estilo de escritura se deben tener en cuenta las siguientes normas:

- Utilizar suficientes espacios y líneas en blanco para separar bien las expresiones y los bloques de código cuya funcionalidad sea diferente a la del código adyacente en el mismo fichero fuente.

- Utilizar siempre tabuladores para delimitar el ámbito de las estructuras de control, respetando el sangrado (espacios en blanco introducidos delante de una línea) de las estructuras anidadas.

- Utilizar paréntesis, ya que, en expresiones complicadas, resulta más rápido desglosar mentalmente la expresión si ésta tiene los paréntesis apropiados.

- Utilizar separadores visuales, como guiones entre comentarios, para separar los grandes bloques del programa, por ejemplo, los módulos entre sí y del programa principal. En algunos lenguajes se pueden colocar módulos diferentes en ficheros diferentes, ayudando a diferenciar completamente el código de unos y otros.

- Utilizar palabras representativas para los nombres de identificadores (variables y constantes), evitando nombres muy cortos (poca legibilidad) o muy largos (incómodos a la hora de programar, a la vez que alargan mucho la línea de código).

- Intentar simplificar el código, evitando expresiones complicadas en la medida de lo posible.

- Evitar las negaciones en las expresiones comparativas, la afirmación es normalmente más clara.

- Evitar grandes anidamientos si no son necesarios.

- Diferenciar claramente las zonas de inclusiones e importaciones, declaraciones de identificadores, procedimientos y funciones. Aunque algunos lenguajes no obligan a que las declaraciones ocupen un lugar específico, es buena costumbre establecerse un criterio estándar en

éste orden, de esta forma los atributos serán fácilmente localizables y modificables. Si el diseño presenta alguna estructura compleja, se deben usar comentarios explicativos que la aclaren, preferiblemente antes de dicho código.

En relación a los comentarios, se debe tener en cuenta lo siguiente:

- Clarifican la estructura de cada módulo. Se incluirán comentarios que clarifiquen la estructura de cada módulo, así como la función realizada por cada bloque de instrucciones y las variables y tipos.

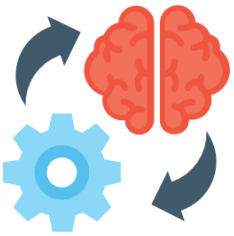
- Son recomendables al principio de cada procedimiento o función, aclarando la funcionalidad del mismo, además es recomendable indicar en éstos también el significado de los argumentos tomados y el valor devuelto. En cualquier lugar del código en que se vaya a realizar una tarea complicada de comprender es recomendable la aclaración mediante un comentario explicativo.

- Son muy recomendables principalmente por dos razones:

    - El propio programador puede requerir la reutilización de un código escrito tiempo atrás, habiendo ya olvidado en la fecha presente el código que se escribió.
    - La programación no es tarea de una única persona, y al tener que compartir el código con otras personas debe aclarar de la mejor manera posible la forma en que ha escrito dicho código, para facilitar la tarea al resto del grupo.

- Se pueden distinguir dos tipos de comentarios, comentarios de prólogo y descriptivos:

    - Los comentarios de prólogo aparecen al principio de cada módulo y consisten en un resumen de la función que realiza, nombre del autor, versión del módulo, fechas de modificaciones, y la información adicional que se considere necesaria.
    - Los comentarios descriptivos se insertan dentro del código y describen en lenguaje natural que hace cada bloque de código.



RECUERDA

Cuando se realiza un proyecto software es muy importante documentarlo porque en ese momento se recuerda el porqué de todas las sentencias de código, pero ¿qué pasa si pasados unos meses se deben realizar algunos cambios en la codificación?

5.4 Documentación de pruebas

Las pruebas se pueden realizar en cualquiera de las fases de la aplicación, pudiéndose establecer tres tipos de pruebas:

Pruebas unitarias: Para cada módulo dentro de un programa

Pruebas de integración: Para cada uno de los programas con todos sus módulos

Pruebas de sistemas: Para todos los programas que componen la aplicación

Imagen: Tipos de pruebas

En el documento de juego de pruebas se pueden especificar el tipo de prueba, módulos y programas para los que se efectúan dichas pruebas, datos de entrada, datos previstos de salida y los datos reales de salida obtenidos en las pruebas.

La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. El hecho de realizar pruebas del software proporciona una garantía de calidad del software. Además, representa una revisión final de las especificaciones, del diseño y de la codificación. La prueba no puede asegurar la ausencia de errores en el programa, solo puede indicar su existencia. Un buen caso de prueba será aquel que muestre un error no descubierto hasta entonces. Se deben seguir los siguientes principios para la realización de pruebas:

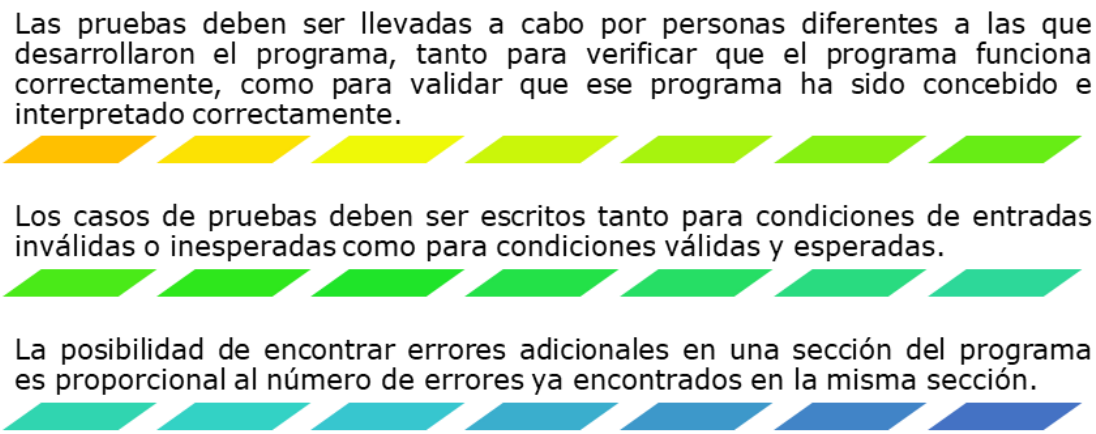


Imagen: Principios en la realización de las pruebas

PARA SABER MÁS

Las pruebas de software son una fase muy importante para garantizar calidad en los programas. La calidad del software se puede medir y evaluar, de tal manera que un software pueda ser evaluado atendiendo al nivel de calidad que ofrece. Con esta premisa, se recomiendan las siguientes lecturas:

- http://www.javiergarzas.com/2012/10/iso-9126-iso-25000-1.html
- http://www.javiergarzas.com/2012/10/iso-9126-iso-25000-2.html

5.5 Guía de instalación

La guía de instalación o manual de explotación es el documento que contiene la información necesaria para poner en marcha el sistema diseñado y para establecer las normas de explotación.

Estas normas harán precisiones sobre las siguientes tareas:

- Pruebas de implantación de los programas en el sistema físico donde van a funcionar en adelante, con la colaboración entre usuarios y desarrolladores.

- Forma en que se van a capturar los datos existentes en el sistema anterior al que se va a implantar. Esto implicará un trasvase de información que puede estar en soportes de uso manual o en soportes informáticos que exijan una conversión del formato de la información y la realización de programas de captura de datos.

- Pruebas del nuevo sistema con toda la información capturada, lo que equivale a tener funcionando en paralelo los dos sistemas, corrigiendo lo posibles errores de funcionamiento. Cuando estás pruebas sean satisfactorias, el nuevo sistema sustituye al anterior y entra en vigor la nueva documentación.

BIBLIOGRAFÍA RECOMENDADA

Calero, C., Piattini, M. y Moraga, M.A. (2010). Calidad del producto y proceso software. Madrid, España: RaMa. Y en concreto al capítulo: **La relevancia de las técnicas y tecnologías para la documentación del software.**

5.6 Guía de uso

La guía de uso o manual de usuario es el documento que contiene la información precisa y necesaria para que los usuarios utilicen correctamente los programas de la aplicación. La información contenida en la guía de usuario proviene de la guía técnica de la aplicación, pero se presenta de forma que el usuario la comprenda con toda claridad, prescindiendo de toda la parte técnica de desarrollo y centrándose en los aspectos de la entrada y salida de la información que maneja la aplicación.

La guía de uso no debe nacer referencia a ningún apartado de la guía técnica, pues normalmente la guía técnica queda en poder de los desarrolladores y al usuario solamente se le proporciona las guías de uso y de instalación.

La guía de uso debe estar redactada con un estilo claro y en el caso de que se haga necesario el uso de terminología informática no conocida por los usuarios, debe ir acompañada por un glosario de términos informáticos. Este documento puede ser utilizado para completar la formación que los usuarios de la aplicación deben recibir para el correcto manejo de la misma. La guía de uso puede contener la siguiente información:

Imagen: Información de la guía de uso

COMPRUEBA LO QUE SABES

¿Cuáles son las diferencias entre la guía de uso y la guía de instalación? ¿Podrían unirse en un único documento?

RESUMEN FINAL

En esta unidad se han estudiado los tipos de lenguajes de programación, clasificados según diferentes criterios. Además, se ha aprendido la importante labor de los compiladores, de los intérpretes y de los depuradores, a la hora de implementar código. Por su parte, se han visto en detalle las diferentes fases de desarrollo de una aplicación software (análisis, diseño, implementación, pruebas, explotación y mantenimiento) y cómo la documentación juega un papel fundamental, de forma transversal a todas ellas.