Unidad 1

Desarrollo de Software

Entornos de desarrollo

1

Índice

1.1. Programas y aplicaciones informáticos
1.1.1. Concepto de programa informático
1.1.2. Concepto de aplicación informática
1.1.3. Software a medida y software estándar

1.2. Lenguajes de programación
1.2.1. Tipos de lenguajes de programación
1.2.2. Características de los lenguajes más difundidos

1.3. El proceso detraducción/compilación

1.4. Desarrollo de una aplicación
1.4.1. Fases del desarrollo de una aplicación
1.4.2. La documentación
1.4.3. Roles o figuras que forman parte del proceso de desarrollo de software

Introducción

Nos referimos a software como el conjunto de los elementos de un sistema informático que no son tangibles, un equivalente al soporte lógico del conjunto completo. Está compuesto por todos los componentes lógicos del sistema informático y se diseña para que realice una tarea determinada en el mismo sistema.

El software, además, tiene la tarea siempre de comunicarse con la parte física de nuestro sistema informático, en este caso el hardware, es decir, traduce a lenguaje comprensible por el mismo nuestras órdenes.

El estándar 729 de IEEE define software de la siguiente manera:

"El conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación"

Además, veremos cómo se crea el software en base a unas características básicas que posee:

- El software es lógico, es decir, es intangible.
- El software se desarrolla. A las personas que se dedican a crear software los llamamos desarrolladores de software.
- En algunas ocasiones, se creará el software a medida. Además del software a medida, existe el llamado software enlatado.

Al finalizar esta unidad

- Se habrán comprendido los principios básicos del desarrollo de software como lenguajes de programación y sus características, así como las fases del desarrollo de una aplicación.
- Seremos capaces de diferenciar entre compiladores, intérpretes y máquinas virtuales.

1.1. Programas y aplicaciones informáticos

1.1.1. Concepto de programa informático

Nos referimos a programa informático como la serie de instrucciones que ordenadas de manera concreta llevarán a cabo una acción determinada.

```
#include <stdio.h>

int main (void)
{
    printf ("Hello, World!\n");
    return 0;
}
```

Imagen 1. Ejemplo de programa "Hello World".

En la Imagen 1 podemos ver un ejemplo de un programa en C que nos mostrará las palabras Hello World por pantalla.

1.1.2. Concepto de aplicación informática

De manera general, una aplicación informática se encuentra formada por varios programas con sus propias librerías que funcionando al unísono y compenetrándose realizan alguna tarea o finalidad concreta. Una aplicación informática puede que conste de un solo programa informático. Además, hay ocasiones en las que un conjunto de programas se ejecuta de manera independiente unos a otros, aunque tienen características en común, este conjunto se suele llamar suite o paquete integrado, un ejemplo de esto es Microsoft Office. Hay que recalcar que estas suites suelen tener interconectados todos los programas mediante librerías, almacén o datos, además, todos son compatibles entre ellos al 100%.

También tenemos que destacar que una aplicación informática en sí no se comunica de manera directa con el hardware, lo hace a través del sistema operativo.

Ejemplos de aplicaciones informáticas que nos ayudan a la hora de automatizar o realizar algunas tareas son:

- Programas de contabilidad.
- Programas de diseño gráfico.
- Programas de facturación.
- Hojas de cálculo.
- Herramientas de correo electrónico.

1.1.3. Software a medida y software estándar

Nos referimos al software a medida cuando hablamos de un software específico que se ha desarrollado a petición de una empresa u organización y que cuenta con unas características acordes a las necesidades de la empresa, su organización y su manera de efectuar el trabajo.

Algunas características de este tipo de software son:

- Como cualquier tipo de software, necesita ser desarrollado, y esto lleva su tiempo.
- En la gran mayoría de ocasiones no será transferible a otras empresas porque se rige por las necesidades específicas de la empresa. Hay ocasiones en las que la empresa desarrolladora de software tiene ya modelos estructurales hechos y adapta el código a cada empresa.
- Es común que se encuentren errores durante su uso, ya que no abarca todas las posibilidades muchas veces. Además, suele llevar un mantenimiento para revisar y mejorar este software.
- Suele ser más costoso que otros tipos de software porque necesita ser desarrollado para una sola organización, por lo que esta asume los costes totalmente.

El otro tipo de software, el software estándar o enlatado es un software genérico, es decir, n mismo programa o aplicación que usarán varias aplicaciones o clientes. DE manera común, grandes softwares como por ejemplo SAP se pueden parametrizar y configurar para que se ajuste a las necesidades de la empresa u organización. Es cuando no conseguimos adaptarnos a nuestras necesidades con este software cuando tendríamos que solicitar un software a medida.

Las características más generales del software estándar son:

- Este software se compra ya hecho. No es necesario que se desarrolle para un cliente.
- Cuenta con multitud de errores menos que en el software a medida porque se ha probado en muchos ámbitos y se han recorrido casi todas las posibilidades.
- Al compartir el gasto entre varios clientes, resulta por lo general más económico.
- En la gran mayoría de ocasiones, la empresa no usará todas sus funciones, así como necesitará de otras que no se contemplen en este software. Muchas veces esta carencia se subsana con alguna otra aplicación secundaria y no es necesario recurrir al software a medida.

1.2. Lenguajes de programación

En la actualidad, todos los programas informáticos se desarrollan con un lenguaje de programación, ya que es casi imposible programar directamente en código máquina debido la dificultad extrema de entender su lenguaje.

Podemos decir entonces que los lenguajes de programación son lenguajes artificiales que se han creado con la intención de traducir a código máquinas las instrucciones que se van dando al programa.

Todos los lenguajes de programación siguen una normativa y sintaxis para hacer que la traducción de las instrucciones sea lo más acertada posible.

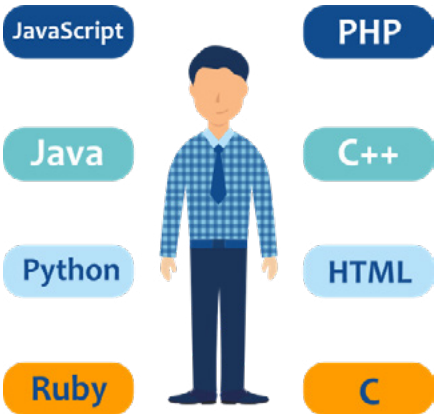

Imagen 2. Distintos lenguajes de programación.

1.2.1. Tipos de lenguajes de programación

El lenguaje de programación nació como algo muy complicado de entender y que solo algunas personas muy cualificadas podían manejar, pero esto ha cambiado. A lo largo de los años han ido surgiendo más lenguajes y adaptando los antiguos, haciendo que hoy en día sea un lenguaje user friendly, es decir, de fácil entendimiento para el usuario. Esto no quiere decir que cualquier persona pueda programar, pues, aunque sean intuitivos, se necesitará una serie de conocimientos para realizar un programa eficiente.

A veces, el programar algo rápido y sencillo puede que se traduzca en algo poco eficiente y que ocupe más espacio, pues no se ha depurado bien.

En la actualidad existen tres "tipos" de lenguajes de programación:

Lenguaje máquina:

Se trata de un lenguaje en el que sus instrucciones son composiciones de 1 y 0, lo que las hace ininteligibles.

Se entiende directamente con la máquina, por lo que es el único lenguaje que no necesita ser traducido por otro.

Es el lenguaje que nació primero y se necesitaba un profundo dominio del hardware para poder emplearlo.

Dependiendo del tipo de procesador, tendrá unas instrucciones u otras.

Salvando alguna excepción, se trata de un lenguaje que ya nadie usa.

Lenguaje de medio nivel o ensamblador:

Surge a raíz de la dificultad de programar en código máquina, ayudando en la labor de programación.

Aunque todavía se trata de un lenguaje cercano al hardware, su uso ya no se basa en unos y ceros, sino en mnemotécnicos, lo que lo hace más inteligible.

Necesita de compilación y traducción a código máquina para que se pueda ejecutar.

Su uso consiste en trabajar directamente con los registros y direcciones físicas del sistema.

Sigue siendo un lenguaje complicado todavía a la hora de leerlo y comprenderlo.

Lenguajes de alto nivel:

La gran mayoría de lenguajes de programación que se usan hoy en día se engloban en este grupo.

Es mucho más sencillo e intuitivo programar en este tipo de lenguaje que en los dos anteriores.

Mucho más cercano al lenguaje humano al incorporar expresiones propias de algunos idiomas como por ejemplo FOR.

Se desarrollan junto con el lenguaje una serie de funciones y librerías que solventan problemas que se presentan durante la programación.

Muchas veces vienen dotados de frameworks para hacer más rápida y eficiente la programación.

En este término surge la programación orientada a objetos que nos ayuda a la reutilización y encapsulamiento de componentes.

Además, si queremos también podemos clasificar los lenguajes de programación dependiendo de cómo se ejecutan:

- **Lenguajes compilados.** El compilador es un programa que tiene como función traducir el código fuente a código máquina. Se caracterizan por ejecutarse más rápido que los dos tipos de lenguaje que vamos a ver a continuación. Adicional al compilador, se necesita de un programa que permite que se una en código objeto del programa con el código objeto de las librerías, este es llamado enlazador o linker.
- **Lenguajes interpretados.** En estos lenguajes no se genera un código objeto. Las instrucciones generadas se traducen con un programa cargado en memoria que trabaja de interprete leyendo cada una de estas y ejecutándolas. Este proceso se hace con las instrucciones a medida que se van ejecutando, es decir, una a una, por lo que, si paramos el programa, no se habrá traducido todo lo demás aún.
- **Lenguajes virtuales.** Se trata de un lenguaje "portable" debido a que, una vez compilado el código, se genera un bytecode que puede ser también interpretado por una máquina virtual que se haya instalado en cualquier equipo. Este tipo de lenguaje es algo más lento, pero se usan mucho debido que es el más versátil.

1.2.2. Características de los lenguajes más difundidos

Java

Java surgió como una evolución a C++ y en su momento álgido fue considerado el lenguaje de la web. Sus principales características son:

- Se trata de una simplificación de C++ porque no está permitida la sobrecarga de operadores ni la herencia múltiple.
- A la hora de manejar los strings o cadena de caracteres, Java resulta mucho sencillo y funciona más eficiente que cuando se manejaban en C o C++.
- Es un lenguaje orientado a objetos.
- Una de las funciones para las que se desarrolla este lenguaje es para trabajar con las redes y sus distintos protocolos como TCP/IP, FTP, etc.
- Java es un lenguaje interpretado al mismo tiempo que es un lenguaje virtual.
- Se puede ejecutar en cualquier plataforma, esto lo hace un lenguaje portable.
- Es uno de los lenguajes de programación más seguros, de hecho, en las versiones modernas se está intentando conseguir la encriptación del código fuente.
- Nos da la posibilidad de ejecutar varios hilos en un mismo programa, es decir, ejecutar varias acciones al mismo tiempo.

```
public class EntornosdeDesarrollo {
    public static void main(String[] args) {
    System.out.ptrinln("Entornos de Desarrollo");
    }
}
```

Imagen 3. Ejemplo de código en Java.

Python

Sus características son:

- Se trata de un lenguaje de alto nivel ya que es de los más portables existentes. Exceptuando algunas librerías que puedan ser particulares de cada sistema, un programa en Python se puede ejecutar de manera eficiente y completa en cualquier sistema.
- También se trata de un lenguaje interpretado.
- Como casi todos los lenguajes de programación modernos, es un lenguaje orientado a objetos.
- Tiene la peculiaridad de que podemos escribir partes de código en C y se combina con las partes en Python, haciendo que el programa sea más rápido de ejecutar.
- Se puede hacer también a la inversa.
- Es de los lenguajes más sencillos para aprender a programar.

```
# Entornos de Desarrollo en Python
print ("Entornos de Desarrollo")
```

Imagen 4. Ejemplo de código en Python.

C y C++

Posiblemente los dos lenguajes más usados a lo largo de la historia de la informática debido a su gran potencia y versatilidad. Hoy en día se sigue usando en varios entornos de desarrollo. Sus principales características son:

- La entrada y salida de información se realiza a través de funciones.
- Mientras que C estuvo ligado a Unix, cuando surgió C++ se intentó desligar de este sistema.
- Se trata de un lenguaje muy ligado a funciones, así como estructurado.
- Tanto C como C++ incluyen el elemento puntero. Este término hace referencia a una variable que contiene la dirección física de memoria de otra variable. Aunque es un elemento que nos puede ayudar a desarrollar un programa más flexible, es una instrucción compleja a la hora de manejarla.
- Aunque casi todo es de alto nivel, se pueden dar trozos de código a bajo nivel.
- Los programas son muy rápidos y eficientes.
- Una vez compilados, sus programas son pequeños.
- Son relativamente portables porque con poquitos cambios que se realicen (a veces ninguno), se puede recompilar en casi cualquier sistema.

```
using namespace std;
int main(int argc, char *argv[] {
    std::cout <<"Entornos de Desarrollo" <<endl;
    return 0;
}
```

Imagen 5. Ejemplo de código en C++.

JavaScript

Se trata del lenguaje más usado en entornos Web, pero no de manera directa. Se usan sobre todo en los frameworks que se basan en JavaScript como Angular o React. Sus características principales son:

- Se parece en muchas cosas a Java, C y C++, es por esto por lo que resulta cómodo de trabajar para usuarios expertos en estos lenguajes.
- Se trata de un lenguaje que funciona a base de scripting.
- Es un lenguaje front end, es decir, del lado del cliente.
- Es muy seguro y fiable.
- Al interpretarse en el lado del cliente, cualquiera puede ver su código.
- Existen multitud de librerías basadas en JavaScript, las más conocidas son:
- AngularJS
- ReactJS
- MeteorJS
- JQuery
- Foundation JS
- BackBone.js

```
<script type="text/javascript">
    alert("Entornos de Desarrollo");
</script>
```

Imagen 6. Ejemplo de código en JavaScript.

PHP

El lenguaje de la web moderna por excelencia suele estar detrás de muchos CMS como Joomla o WordPress. Sus características más destacadas son:

- Es un lenguaje que se puede instalar en prácticamente cualquier sistema, los llamados multiplataforma.
- Siempre ha estado orientado a las webs dinámicas.
- Tiene una integración casi total con el lenguaje SQL y aplicaciones como MySQL.
- También se permite la orientación a objetos.
- Backend, es decir, se ejecuta en el lado del servidor.
- No hace falta definir variables, es un lenguaje interpretado.
- Hay multitud de frameworks que se han basado en PHP y por eso nos permiten trabajar los patrones del diseño modelo-vista-controlador (MVC).

```php
<?php echo '<p>Entornos de Desarrollo</p>'; ?>
```

Imagen 7. Ejemplo de código en PHP.

VB.NET

Visual Basic sufrió una actualización por parte de Microsoft tan grande que dio lugar a este nuevo lenguaje. Sus principales características son:

- Está basado en Visual Basic, un referente en cuanto a los lenguajes de programación se habla.
- Su predecesor Visual Basic sigue siendo el lenguaje que se usa para la programación de macros en los programas de Office.
- Si se implementa el framework .NET, se trata de un lenguaje orientado a objetos.
- Se suele programar el código fuente en la herramienta Microsoft Visual Studio.
- Hay otros entornos de desarrollo en .NET que son libres, no propiedad de Microsoft, pero no son tan potentes.

```
Module VBModule
Sub Main()
Console.WirteLine("Entornos de Desarrollo")
End Sub
End Module
```

Imagen 7. Ejemplo de código en VB.NET.

1.3. El proceso de traducción/compilación

Como hemos comentado antes, hay dos maneras de traducir los lenguajes de alto nivel a código máquina o ensamblador: interpretando o compilando.

La manera de trabajar de un intérprete es la siguiente: va traduciendo el código fuente línea a línea.

Vamos a explicar bien esto, el intérprete traduce la primera línea, detiene la traducción, y después ejecuta la orden que se haya dad, esto de manera sucesiva hasta que se llegue al final del programa. Como se ejecuta línea a línea, tanto el código fuente como el intérprete deben de estar cargados en memoria durante toda la ejecución del programa.

Por otro lado, el compilador traduce de una todo el código fuente a código máquina. El proceso de compilación tiene una característica, y es que el código que se genere solo será válido para el mismo hardware y software donde se han generado.

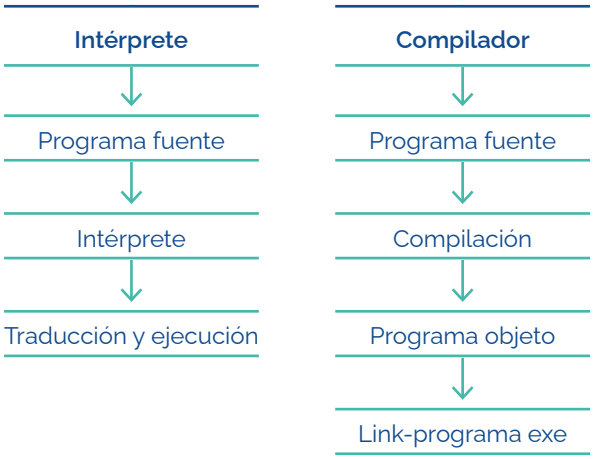

Imagen 1. Fases de un intérprete y un compilador.

Vamos a explicar más detalladamente cómo funciona un compilador. Los pasos son los siguientes:

1. En primera instancia realizará un preprocesado de nuestro código fuente en el que todos los comandos de este proceso se irán ejecutando y traduciendo.
2. Los archivos del código fuente son compilados y se genera un código intermedio.
3. Volvemos a compilar el código intermedio y lo traducimos o bien a lenguaje ensamblador o bien a código objeto.
4. Se enlaza con el linker el código objeto a las librerías con el fin de traducirlo a código máquina y tener un programa listo para ejecutar.

Para resumir, debemos tener claro que el proceso de compilación se divide en dos subfases:

- En una primera fase se realiza un análisis léxico del programa leyendo de manera secuencial el código. Con esto intentamos detectar todas las instrucciones que van reflejadas en el código.
- En la segunda fase nos dedicamos al análisis sintáctico y semántico del mismo, comprobando que todo está en la forma y estructura correcta.

DIFERENCIAS DE CÓDIGO

Hay que saber diferenciar claramente:

- **Código fuente.** Código escrito directamente en lenguaje de programación
- **Código objeto.** Resultado de la compilación del código fuente.
- **Código ejecutable.** Código resultante después del proceso de compilado y enlazado.

1.4. Desarrollo de una aplicación

1.4.1. Fases del desarrollo de una aplicación

Para desarrollar una aplicación o programa informático hay que seguir los siguientes pasos:

1. Fase inicial
2. Análisis
3. Diseño
4. Codificación
5. Pruebas
6. Explotación
7. Mantenimiento

Todas estas fases vamos a detallarlas ahora.

Fase inicial

Se trata del punto de partida del proyecto. En esta fase se van a definir las bases del desarrollo y para eso se ha de contar con personal cualificado. En un desarrollo informático es común que haya un jefe de proyecto para dirigir estos y organizarlo.

Resumiendo, aquí detallaremos toda la planificación de nuestro proyecto: recursos humanos y económicos, consistencia de la aplicación, posibles errores y soluciones, código en el que se va a desarrollar.

Se elabora en esta fase una documentación relativa al proyecto que es de alto nivel para consensuarla con la dirección de la empresa o los jefes de proyecto, ya que, en la gran mayoría de ocasiones, las decisiones tomadas ahora nos afectarán a lo largo del desarrollo.

Análisis

Procedemos ahora a analizar los requisitos específicos de la aplicación. En esta fase se harán distintas reuniones con el cliente para analizar cada una de las necesidades de la aplicación y formalizarlas en un documento,

Es recomendable firmar este documento con el cliente para que ambas partes se comprometan el desarrollador a cumplir con una serie de requisitos, y el cliente a que no se podrá modificar durante la primera fase de desarrollo ninguno de esos. Esto se realizar como hemos dicho para que el desarrollador no falle en su cumplimiento y para que el cliente no pueda quitar y agregar funciones conforme le plazca.

Diseño

Es ahora cuando procedemos a detallar las funciones específicas y las distintas opciones de la aplicación. Se definen los procesos específicos que va a seguir cada una de las distintas fases de ejecución de la aplicación y se comienza después con la programación inicial.

En esta fase se suelen crear dos documentos distintos, uno genérico y otro detallado. Estos los generan los analistas con el apoyo del jefe de proyecto.

Codificación o implementación

La fase más complicada y amplía, es el momento de programar.

En esta fase empezamos a desarrollar el software que se haya solicitado en un lenguaje de programación elegido y se implementa todo lo decidido durante la fase de diseño.

Aunque es común que durante el desarrollo del código se vayan añadiendo comentarios que expliquen las funciones que se van realizando, en esta fase también se realiza una documentación muy detallada en la que incluiremos trozos de código en la medida de lo necesario para aclaraciones como quien está desarrollando en ese momento o porque se usa una cosa y no otra que también podría,

Para terminar, hay que tener en cuenta que es ahora cuando tenemos que cuidar hasta el más mínimo detalle, porque esta es la base de nuestro futuro software.

Pruebas

Básicamente, en esta fase, procedemos a comprobar que el software ofrece todo lo necesario y que funciona de manera eficiente antes de pasarlo a entornos de producción.

Con respecto a la documentación que generaremos en este punto, tenemos dos tipos:

- **Funcionales.** Cuando comencemos a realizar las pruebas de funcionamiento de la aplicación según los requisitos del cliente, este debe de estar presente para que certifique las pruebas. En cuanto a la documentación, se realizará un documento con todos los fallos encontrados y modificaciones solicitadas o realizadas para que se firme con el cliente y seguir implementando detalles.
- **Estructurales.** Nos referimos aquí a las pruebas que son comunes para todos los softwares, es decir, las pruebas técnicas como que pasa si abro muchas ventanas, si tengo poca red, etc. Estas pruebas también se deben de reflejar en algún documento, pero ya no es necesaria la presencia del cliente.

Explotación

Es el momento de pasar nuestra aplicación a un entorno de producción. Se han terminado todas las pruebas y subsanado los errores y ya se puede proceder con el uso de la aplicación de manera cotidiana. Es el proceso más largo puesto que puede durar años usando la misma aplicación.

Durante su uso surgirán incidencias o errores (bugs) y demás necesidades que se irán implementando previo acuerdo con el cliente.

Es recomendable detallar en un documento todos los cambios que se vayan realizando para si en un futuro se ha de volver a lo anterior o modificar algo, que sepamos en qué condiciones sucedieron los cambios.

Mantenimiento

El mantenimiento consiste en el cuidado de la aplicación durante el proceso de explotación.

Se aplicarán los correctivos o actualizaciones necesarias del código, se adaptarán contenidos o procesos y se eliminarán otros.

Es muy importante haber elaborado previamente una documentación técnica válida porque si no, el mantenimiento sería muy complicado de llevar al no saber todas las especificaciones que en el código haya y que realiza cada expresión escrita.

También hay que ir documentando todos los cambios de mantenimiento para así, poder facilitar el próximo trabajo.

> Cabe destacar que todas estas fases de desarrollo de una aplicación no son estructurales. Las distintas fases se pueden solapar o cambiar el orden dependiendo de las necesidades de la aplicación, y es cuando están todas finitas (excepto explotación y mantenimiento), cuando podemos decir que tenemos una aplicación.

1.4.2. La documentación

A parte de la documentación que se ha ido describiendo anteriormente en cada proceso, hay una serie de documentación común que hay que generar en toda aplicación informática.

Toda esta documentación debe estar adaptada a que el usuario final la pueda comprender, y los tres siguientes documentos son indispensables:

- **Manual de usuario.** Como indica su nombre, se refiere a una descripción del funcionamiento de la aplicación que se realiza para que el usuario final pueda usar el programa sin necesidad de consultar al desarrollador.
- **Manual técnico.** Igual que el anterior, pero en este caso para el personal técnico cualificado que pueda entender el lenguaje de programación en el que se ha implementado.
- **Manual de instalación.** En este manual se detallarán los requisitos y pasos necesarios para instalar la aplicación en un equipo, servidor, entorno etc.

1.4.3. Roles o figuras que forman parte del proceso de desarrollo de software

En este punto vamos a hablar de las personas de las que se compone un equipo de desarrollo de software. Estas personas adoptan roles para que se defina su papel en el desarrollo de la aplicación.

Hay que aclarar que esto no es estricto, es decir, puede que haya roles que existan y otros que no, igual que en un equipo puede haber un rol que aquí no se comente.

Arquitecto de Software

Suele ser una persona con un conocimiento muy amplío en cuanto a las tecnologías de programación y su tarea es la de decidir sobre cómo se realizará el proyecto y su cohesión.

Lógicamente, también es el encargado de decidir que lenguaje se va a usar y con que recursos se va a contar.

Jefe de proyecto

Este rol, puede ser ocupado por una persona que ocupe también otro, generalmente el arquitecto de software.

Es el encargado de dirigir el proyecto y asegurarse de su correcto desarrollo, gestionando la parte humana de manera principal y los acuerdos con el cliente.

Analista de sistemas

La persona que desarrolla este rol es la encargada de analizar todos los requisitos que se han solicitado mediante un estudio exhaustivo y diseñará el sistema que después se pretenderá desarrollar. En este puesto la experiencia es fundamental.

Programador

Es el encargado de codificar en lenguaje de programación todo lo que se solicita por el analista. Es muy común que el analista también sea el programador, y cuando esto sucede se le suele asignar el rol de analista-programador.

www.universae.com