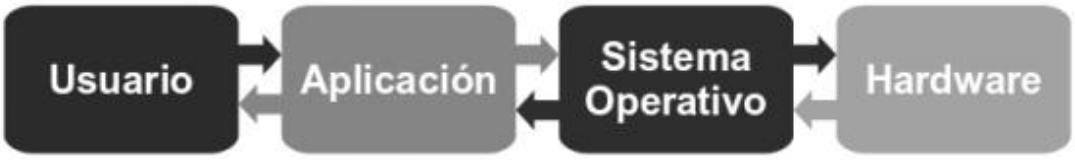
INTRODUCCION A LA PROGRAMACION

"El software se está comiendo el mundo".

(Marc Andreessen- Fundador de Netscape)

SISTEMAS, COMPUTADORAS, SOFTWARE y HARDWARE

Un sistema es un conjunto de elementos que interactúan en un dominio para realizar un propósito específico. Un sistema Informático puede dividirse en cuatro componentes: el hardware, los programas de aplicación, los programas de sistemas y los usuarios. Una computadora es un dispositivo electrónico utilizado para procesar datos y obtener resultados. Los datos y la información se pueden introducir en la computadora como entrada (input) y a continuación se procesan para producir una salida (output). Los componentes físicos que constituyen la computadora, junto con los dispositivos que realizan las tareas de entrada y salida, se conocen con el término de Hardware. El conjunto de instrucciones que hace funcionar a la computadora se denomina programa, software o aplicación; a la persona que escribe programas se la denomina programador o desarrollador. Una aplicación sirve para dar servicio a las necesidades de alguna actividad en concreto como ser, un comercio, un banco, una universidad, un laboratorio químico o casi cualquier proceso del mundo real que se pueda imaginar. El Software se puede clasificar en Software de Aplicación (Los ejemplos de aplicaciones incluyen: procesadores de texto, programas de bases de datos, planillas de cálculo, navegadores Web, herramientas de automatización industrial, entre otros) y en Software de Sistema que se encarga de administrar el hardware del equipo y ofrecer una plataforma para ejecutar los distintos software de aplicación (Sistemas Operativos, los lenguajes de programación, controladores de dispositivos, utilitarios y editores de textos).



¿QUÉ ES UN LENGUAJE DE PROGRAMACIÓN?

Los lenguajes de programación son un conjunto de reglas, herramientas y condiciones que nos permiten crear programas o aplicaciones dentro de una computadora. Estos programas son los que nos permitirán ordenar distintas acciones a la computadora en un "idioma" comprensible por ella.
Como su nombre lo indica, un lenguaje tiene una parte sintáctica y una parte semántica. ¿Qué quiere decir esto? La sintaxis es la forma en que se combinan las palabras para formar sentencias válidas y la semántica es el significado, el sentido de las mismas.
Por ejemplo, las siguientes dos frases son válidas desde el punto sintáctico, pero según se use o no la coma puede tener distintos significados:
¡No tenga compasión!
¡No, tenga compasión!

Nota

Un error de sintaxis en nuestro programa puede suceder cuando se escribe código de una forma no admitida por las reglas del lenguaje (el olvido de un punto y coma, por ejemplo). Un error semántico es cuando la sintaxis es correcta, pero el resultado no es lo que se pretendía (Un cuelgue del programa, por ejemplo).

¿POR QUE EL LENGUAJE IMPORTA?

En el principio, cuando nació la computación, no había lenguajes de programación. Los programas escritos lucían algo así:

```
00110001 00000000 00000000
00110001 00000001 00000001
00110011 00000001 00000010
01010001 00001011 00000010
00100010 00000010 00001000
01000011 00000001 00000000
01000001 00000001 00000001
00010000 00000010 00000000
01100010 00000000 00000000
```

Este sistema de codificación es conocido como lenguaje máquina que es el lenguaje nativo de una computadora. Este programa de computadora simplemente suma los números del 1 al 10 e imprime el resultado: 1 + 2 + ... + 10 = 55. Escribir estos programas es sumamente complicado y propenso a errores. Como ventaja podemos mencionar que el hardware entiende directamente este código sin ninguna interpretación o traducción.
En un lenguaje de programación como PHP el programa podría haberse escrito de la siguiente manera:

```
<?php
$total = 0;
$contar = 1;
while ($contar <= 10)
{
        $total += $contar;
        $contar += 1;
}
echo $total;
?>
```

Como podemos ver este programa es más sencillo de leer y es más parecido al lenguaje humano. Sin embargo, las computadoras sólo entienden las instrucciones en lenguaje máquina, por lo que será preciso traducir los programas resultantes a lenguajes de máquina antes de que puedan ser ejecutadas por ellas.

PSEUDOCODIGO

En el aprendizaje de la programación de computadoras se suele comenzar también con el uso de los lenguajes algorítmicos, similares a los lenguajes naturales, mediante instrucciones escritas en pseudocódigo que son palabras o abreviaturas de palabras escritas en inglés o español. Posteriormente se realiza la conversión al lenguaje de programación que se vaya a utilizar realmente en la computadora, tal como C, Java o PHP en nuestro caso. La ventaja del pseudocódigo es que en su uso el programador se puede concentrar en la lógica y en las estructuras de control y no preocuparse de las reglas de un lenguaje específico. Es también fácil modificar el pseudocódigo si se descubren errores o anomalías en la lógica del programa, mientras que en muchas ocasiones suele ser difícil el cambio en la lógica, una vez que está codificado en un lenguaje de programación.
A continuación, mostraremos como sería el pseudocódigo para el ejemplo de la suma de los primeros 10 números:

```
Inicio
Establecer total a 0
Establecer contar a 1
Mientras (contar <= 10) hacer
        Sumar contar a total, guardarlo en total
        Incrementar contar en 1
FinMientras
imprimir total
Fin
```

ETAPAS EN EL DESARROLLO DE UN SISTEMA

La ingeniería de software es la responsable de controlar y dirigir el proceso de desarrollo del software. Durante el ciclo de vida de un proyecto determina que tareas deben realizarse desde la captura de requisitos hasta las pruebas e implantación.
El que mostraremos a continuación es el ciclo de vida más clásico que podemos encontrar, está conformado en su versión ampliada por siete etapas que se pueden representar mediante un modelo en cascada o waterfall (Recibe este nombre porque tal como el agua fluye hacia abajo; esta se caracteriza por un flujo irreversible de trabajo, etapa por etapa, hasta finalizar cada una de ellas):

1) Definición de necesidades
2) Análisis
3) Diseño
4) Codificación
5) Pruebas
6) Validación
7) Mantenimiento y Evolución

Figura 2.1 Etapas en el desarrollo de un sistema.

- DEFINICION DE NECESIDADES: En primera instancia tenemos el pedido del cliente. En esta etapa se detecta un problema o una necesidad que para su solución y/o satisfacción es necesario realizar un desarrollo de software.
- ANÁLISIS: En esta etapa se debe entender y comprender de forma detallada cual es la problemática a resolver, verificando el entorno en el cual se encuentra dicho problema, de tal manera que se obtenga la información necesaria mediante una recolección para afrontar su respectiva solución. En simple palabras y básicamente, durante esta fase, se adquieren, reúnen y especifican los requerimientos funcionales y no funcionales que deberá cumplir el futuro programa o sistema a desarrollar. Esta etapa es conocida como la del QUÉ se va a solucionar.
- DISEÑO: Una vez que se tiene la suficiente información del problema a solucionar, es importante determinar la estrategia que se va a utilizar para resolver el problema. Esta etapa es conocida bajo el CÓMO y define como los requisitos obtenidos en la etapa de análisis se cumplirán. El resultado de esta etapa es un prototipo que luego se codificara. Al finalizar esta etapa estaremos en condiciones de mostrarle a nuestro cliente una propuesta teórica acerca de cómo funcionaría el proyecto.
- CODIFICACION: Partiendo del análisis y diseño de la solución, en esta etapa se procede a desarrollar el correspondiente programa que solucione el problema mediante el uso de un lenguaje de programación.
- PRUEBAS: Los errores humanos dentro de la programación son muchos y aumentan considerablemente con la complejidad del problema.
Cuando se termina de escribir un programa, es necesario realizar las debidas pruebas que garanticen el correcto funcionamiento de dicho programa bajo el mayor número de situaciones posibles a las que se pueda enfrentar. Lo más importante que tenemos que probar son los requisitos funcionales que definen el alcance del proyecto. Otras pruebas consisten en lo que denominamos prueba de cargas, que es un conjunto seleccionado de datos típicos a los que puede verse sometido el sistema.

También es importante involucrar a los usuarios claves que han participado en la definición de los requisitos de la aplicación. Mientras se prueba pueden obtenerse importantes feedbacks.
- VALIDACION: Garantiza que el software cumple con las especificaciones definidas en la etapa de análisis.
- MANTENIMIENTO y EVOLUCION: Una vez puesto en marcha la aplicación para realizar la solución del problema previamente planteado o satisfacer una determinada necesidad, es importante mantener una estructura de actualización, verificación y validación que permitan a dicho programa ser útil y mantenerse actualizado según las necesidades o requerimientos planteados durante su vida útil.

¿QUE ES UN ALGORITMO?

Las tres primeras fases conducen a un diseño detallado escrito en forma de algoritmo. Un algoritmo es un conjunto de operaciones que busca resolver un problema determinado a través de secuencias lógicas. Entendiendo al algoritmo como un medio, podemos decir que para operar como tal debe ser:

- **Preciso:** Cada paso y su orden de realización deben ser claros y concretos.
- **Definido:** Se deben obtener resultados delimitados a las órdenes y estos siempre deben ser los mismos.
- **Finito:** Su diseño debe tener un número limitado de pasos.
- **Ordenado:** La secuencia de pasos debe seguir un orden que no puede ser alterado.

Cada vez que una persona realiza una búsqueda en Internet, hace reservas de vuelos o realiza una transacción en un home banking está utilizando un algoritmo. Más aun, hoy en día hacen tareas que hasta hace algunos años se consideran casi imposibles como recomendar bandas de música, reconocer imágenes o conducir un auto. En cierta medida, esto también es posible gracias a la aparición de una nueva generación de algoritmos que no requieren que se le indique al detalle los pasos a ejecutar: aprenden solos a partir de ejemplos. Esos también son algoritmos, muchas veces llamados “aprendizaje automático” o “inteligencia artificial”.

Nota

Para comprender más esto tómenos el ejemplo del buscador de Google, al momento de entregarnos un resultado, Google, tiene en cuenta más de cincuenta elementos que definen su criterio de búsqueda; por ejemplo, que tipo y versión de sistema operativo utilizamos para conectarnos, también analiza nuestra ubicación, de que país procede la búsqueda y que navegador web utilizamos, etc. Esto demuestra que no hay un tipo de búsqueda estándar, sino que cada una de las búsquedas ofrece resultados únicos, hechos a medida.

Para llegar a la realización de un programa es necesario el diseño previo de un algoritmo, de modo que sin algoritmo no puede existir un programa. Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y ejecutarse en una computadora distinta; sin embargo, el algoritmo será siempre el mismo.

Así, por ejemplo, en una analogía con la vida diaria, una receta de un plato de cocina se puede expresar en español, inglés o francés, pero cualquiera que sea el lenguaje, los pasos para la elaboración del plato se realizarán sin importar el idioma del cocinero.
Los algoritmos se pueden expresar por fórmulas, diagramas de flujo o pseudocódigos. Esta última representación es tal vez la más utilizada.

Figura 2.2 Ejemplo de Algoritmo para lavarse correctamente las manos.

INTELIGENCIA ARTIFICIAL, BIG DATA, APRENDIZAJE AUTOMATICO

A falta de una definición estándar de Inteligencia Artificial (AI, sus siglas en inglés), está comúnmente aceptado que es la disciplina que desarrolla métodos que permiten a las máquinas actuar como personas, y entre sus grandes áreas de conocimiento están la Robótica, el Razonamiento Lógico, los Sistemas MultiAgente o el Aprendizaje Automático. El Aprendizaje Automático (Machine Learning) se aprovecha de la experiencia para poder proveer soluciones a problemas. En este sentido, un algoritmo de Machine Learning desarrolla la capacidad de modificarse a sí mismo con el fin de adaptarse a los datos que está procesando. El objetivo de ello es poder resolver un problema computacional por su propia cuenta. Por ejemplo, el caso más emblemático es el de Netflix y sus recomendaciones de películas y series. Una vez aclarada la diferencia entre Machine Learning e Inteligencia Artificial, es necesario entender el rol del Big Data. Al igual que con la IA, el Big Data no tiene una definición fija, sin embargo, podemos definirlo como el proceso de recopilar y analizar grandes volúmenes de datos cuyo procesamiento resulta imposible para los sistemas informáticos convencionales.

Su principal relación con el Machine Learning es por los datos en sí mismos: Machine Learning necesita de ejemplos datificados para poder encontrar sus soluciones.

Nota

La producción de Netflix, House of Cards, se filmó en base al comportamiento de los espectadores. La mayoría detenía las películas a los 20 minutos de comenzar (presumiblemente para comer algo o ir al baño) ante esto, los autores armaron para el minuto 19 escenas de tal intensidad para que nadie pueda abandonar el sillón.

ABSTRACCIÓN

Cuando construimos un algoritmo estamos modelando una solución a un problema de la realidad y, por tratarse de un modelo, vamos a tener que tomar solo algunos puntos que sean representativos para el problema que queremos resolver mientras se abandonan los irrelevantes. Ningún modelo representa exactamente la realidad porque es demasiada compleja. Es decir, solo tomamos algunos aspectos de esa realidad, esto es lo que se denomina Abstracción.

DIFERENCIAS CON OTRAS PROFESIONES

Desde sus orígenes, la ingeniería del software intentó perseverantemente emular a las ingenierías clásicas. En ingeniería software, a diferencia de la arquitectura u otras ingenierías tradicionales, no se puede separar tan claramente la fase de diseño y de la de construcción. Las ingenierías clásicas precisan mucho de un diseño previo a la construcción, por ejemplo, el disponer de los planos del arquitecto siempre antes de empezar el edificio. Los planos para construir son precisos. Y los realizan personas ajenas a la fase de construcción (los arquitectos). La construcción tiene poco componente intelectual y mucho manual. Y todo esto hace que existan dos actividades claramente diferenciadas: el diseño y la construcción. Ciclos de vida como el de cascada puro (donde cada fase se hace una sola vez y no se pasa a la siguiente fase hasta que se termina la previa) nacieron con este objetivo. Pero en el software está demostrado que es muy difícil especificar en la una única y primera fase de diseño todas las cuestiones a tratar en la programación. Por la complejidad de muchos de los requerimientos que automatizamos cuando construimos software es muy difícil saber qué software se quiere hasta que se trabaja en su implementación. De ahí que en software diseño y construcción se solapen, y que por ello que muchas veces se recomiende otro tipo de metodología de construcción de software como el uso de las metodologías agiles.

Nota

Cuando se diseña un auto se crean una serie de planos donde se concreta de forma exacta la ubicación de cada uno de las piezas del auto, se realiza un prototipo sobre el que se hacen pruebas y después se fabrica en cadena, con un alto nivel de productividad y donde se conoce a priori el tiempo de producción, ensamblaje, distribución, etc.

Por otro lado, tenemos los costos. El software, por su naturaleza (y si se construye mínimamente bien), es más fácil de modificar. Cambiar líneas de código tiene menos impacto que por ejemplo cambiar los pilares de un edificio ya construido. De ahí que existan numerosas propuestas que recomiendan construir rápido una versión software y modificarla evolutivamente. En software no existe esa división tan clara entre los costes del diseño y los de la construcción.

Nota

Las metodologías agiles se enfocan en realizar entregas parciales pero funcionales del producto comprometiendo e involucrando en todo el proyecto al cliente. Gracias a estas dos características se eliminan características innecesarias del producto.

REQUERIMIENTOS DE SOFTWARE

Una etapa fundamental en proyectos de ingeniería de software, es la identificación y documentación de los requerimientos del futuro software, los mismos expresan las necesidades y restricciones que deben satisfacer un producto para contribuir a la solución del problema. Si no se identifican de manera correcta, el software no proporcionará al usuario la funcionalidad esperada; además si no se determinan de manera completa y clara no se conocerá el alcance ni será posible estimar la dimensión real del proyecto. La forma más sencilla de clasificarlos es mediante su funcionalidad.

Requerimientos funcionales: Describen lo que el sistema debe hacer, es decir especifican acciones que el sistema debe ser capaz de realizar, sin considerar restricciones físicas. Ejemplos de ellos podrían ser: El usuario debe tener la posibilidad de buscar los productos por código o descripción. El listado de productos deberá exportarse a html o pdf. etc.
Requerimientos no funcionales: Describen únicamente atributos del sistema o atributos del ambiente del sistema y pueden ser por ejemplo: requerimientos de interfaz, de diseño, de implementación, legales, físicos, de costo, de tiempo, de calidad, de seguridad, de construcción, de operación, entre otros.

Nota

Conforme el proyecto avanza, los requerimientos se pueden ir refinando y validando y hasta es posible que surjan algunos nuevos.

ROLES PROFESIONALES

Dentro de cada proyecto de software hay varios roles profesionales involucrados:

.

LIDER DE PROYECTO (PROJECT LEADER): El jefe de proyecto es el que decide qué se hace. Gestiona y asigna recursos humanos, coordina las interacciones con los clientes y los usuarios finales. Planifica reuniones. Define prioridades. Etc. En definitiva, es responsable de todo el ciclo de desarrollo del producto.

ARQUITECTO DE SOFTWARE: Al igual que decíamos que el jefe de proyecto se encarga de qué se hace, el arquitecto se encarga de cómo se realiza. Es quien tiene a su cargo la labor técnica, para ello tiene la potestad de elegir la forma de abordar cada proyecto, definir las tecnologías a emplear, permitiendo a todo el conjunto personas del equipo compartir una misma línea de trabajo.

ANALISTA FUNCIONAL: Es el responsable de realizar tareas de Relevamiento, Análisis y diseño de los sistemas informáticos. Entre sus competencias está, la de entender el negocio del cliente y descubrir sus necesidades de información.

DESARROLLADOR: Convierte la especificación del sistema en código desarrollado por uno o más lenguajes de programación. Dentro de este perfil encontramos tres momentos profesionales: programador junior, semisenior o intermedio y senior. En principio la diferencia se mide en años de experiencia previa sin embargo es mucho más que eso, es un conjunto de características que determinan que tipo de desarrollador uno es.

El programador junior es un programador con muy poco o ninguna experiencia en el rubro que solo se dedica a plasmar los requisitos en código fuente en general necesita que alguien lo guie en su trabajo. Por otro lado, un programador senior ya trabaja con mayor independencia, toma decisiones, se anticipa a posibles errores dentro del proyecto, guía a otros programadores, sabe delegar y es especialista en lenguajes y técnicas de desarrollo.

TESTER: Es responsable por realizar el Control de Calidad del Producto de Software (Quality Control).

SOPORTE AL USUARIO: El soporte es el servicio de asistencia al usuario una vez que la aplicación se encuentra en uso.

MERCADO LABORAL DE UN DESARROLLADOR

La industria del software (también denominada la Industria del Conocimiento) es una de las actividades económicas más jóvenes y de mayor crecimiento en el mundo, para ello se ha creado el concepto de software Factory (fábrica de software) que son industrias que tienen como objetivo cubrir los requerimientos específicos de clientes a través del desarrollo a medida de software. En nuestro país el software ya se posiciona como el tercer rubro exportador detrás del sector agropecuario y el sector automotor. Según la Cámara de la Industria Argentina del Software (CESSI), en el año 2018 el sector creció un 26% siendo uno de los mayores crecimientos de los sectores argentino. El puesto más solicitado en este rubro es el de desarrollador de aplicaciones web.

Nota

En el sitio http://cessi.org.ar/perfilesit/ puede verse los perfiles ocupacionales actuales dentro de la industria it en Argentina.

Nota

Openqube (https://openqube.io) es una plataforma colaborativa en la que los usuarios pueden detallar los "pros" y "contras" de trabajar en un empresa de software en Argentina.

HABILIDADES Y COMPETENCIAS

En nuestra disciplina intervienen un conjunto de habilidades y competencias intelectuales que sirven para toda la vida y son aplicables a todos los demás campos de estudios. Constituyen una forma de pensar que tiene características propias y diferentes a la de otras ciencias donde se destaca:

Modelización y Formalización: Permite visualizar el sistema a construir representando la realidad en términos de otros elementos conceptuales.
Descomposición de subproblemas: La descomposición, es la capacidad de dividir un problema complejo en partes más pequeñas y fácilmente tratables.
El reconocimiento de patrones: Un patrón es una solución ya probada que puede ser usado en contextos similares. No siempre debe comenzar de cero para crear una solución
Uso de patrones y la abstracción: Es la capacidad de tomar los patrones encontrados previamente y utilizarlos para resolver problemas de aplicación en diferentes contextos. Los patrones proporcionan abstracción ya que ignoran detalles no esenciales para resolver el problema.
Trabajo en Equipo: Es la creación de grupos de personas que se reúnen, colaboran e interactúan de forma específica para un proyecto determinado. Algunos de los mejores proyectos creativos se hacen gracias a la colaboración de personas inspiradas en trabajar en el mismo objetivo.
Autoaprendizaje: Es un proceso donde el individuo adquiere conocimientos, actitudes y valores por cuenta propia, puede ser dado mediante estudios o experiencia.
Creatividad: Crear algo de la nada o a partir de algo muy pequeño es, posiblemente, el mejor ejemplo de ser creativo.

Nota

Los Meetup son una gran manera de conocer a otros desarrolladores, permiten a sus miembros reunirse en la vida real bajo un mismo interés (tecnología, lenguaje de programación, etc.) en estas charlas uno puede compartir experiencias e ir aprendiendo. En argentina podemos encontrar una lista de eventos en el sitio http://meetupjs.com.ar/calendario.html

Nota

Es tal la importancia del desarrollo de software que cada vez más países consideran al desarrollo de software como una ciencia básica en la educación de todos los niveles. En Estonia, país donde se creó Skype por ejemplo, se enseña programación básica a partir de los seis años ya que les permite a los alumnos a desarrollar su creatividad y el pensamiento lógico y a resolver problemas.

Nota

A pesar de que hay mucho contenido de calidad en español, no es un secreto que el mejor material de programación (libros, tutoriales, videos ,foros, etc.) se encuentra en inglés. Afortunadamente podemos encontrar varias herramientas para aprender el idioma inglés. Una de las mejores es Duolingo (https://www.duolingo.com) una plataforma gratuita que actúa como una red social para aprender inglés y otros idiomas.

TRADUCTORES DE LENGUAJE

El desarrollo de un programa se realiza escribiendo una serie de órdenes o instrucciones que siguen las sintaxis de un lenguaje de programación. Estas instrucciones las escribimos en archivos de texto, utilizando algún editor de textos o IDE (Entorno de Desarrollo). A estos archivos de texto se los llama fuente (source). Sin embargo, sabemos que la computadora sólo entiende su propio lenguaje, que normalmente es extraordinariamente sencillo comparado el lenguaje de programación que estamos aprendiendo. El lenguaje de la computadora es el código máquina (machine code), un conjunto de ceros y uno.
Como vimos anteriormente los humanos expresamos la dinámica de un programa mediante un lenguaje de los llamados de "alto nivel". Estos son lenguajes como C#, Java, PHP, Python, etc.

Nota

Rachel Potvin, Ingeniera de Google, estima que todas las líneas de código necesarias para correr los servicios de Internet de Google, como el motor de búsqueda, el cliente de correo Gmail o Google Maps, está constituida por 2 mil millones de instrucciones de código.

El proceso de traducción de un programa fuente (programa desarrollado por el programador) a un lenguaje máquina comprensible por la computadora (formado por unos y ceros), se realiza mediante programas llamados “traductores”. Los traductores se dividen en compiladores e intérpretes. Básicamente un lenguaje interpretado es aquel en el cual sus instrucciones o más bien el código fuente, escrito por el programador en un lenguaje de alto nivel, es traducido por el intérprete a un lenguaje entendible para la máquina paso a paso, instrucción por instrucción hasta terminar el programa. El proceso se repite cada vez que se ejecuta el programa el código en cuestión. Entre los lenguajes de programación que necesitan un intérprete podemos mencionar a Ruby, Python, PHP, JavaScript y otros como Perl y Smalltalk.

Los lenguajes interpretados tienen por ventaja una gran independencia de la plataforma donde se ejecutan, como desventaja de estos traductores es el tiempo que necesitan para ser interpretados. Al tener que ser traducido a lenguaje máquina con cada ejecución, este proceso es más lento que en los lenguajes compilados, sin embargo, algunos lenguajes poseen una máquina virtual que hace una traducción a lenguaje intermedio con lo cual el traducirlo a lenguaje de bajo nivel toma menos tiempo.
Un lenguaje compilado es aquel cuyo código fuente, escrito en un lenguaje de alto nivel, es traducido por un compilador a un archivo ejecutable o binario entendible para la máquina en determinada plataforma. Con ese archivo se puede ejecutar el programa cuantas veces sea necesario sin tener que repetir el proceso por lo que el tiempo de espera entre ejecución y ejecución es ínfimo. Dentro de los lenguajes de programación que son compilados tenemos la familia C que incluye a C++, Objective C, C# y también otros como Fortran, Pascal, Haskell y Visual Basic.
Principalmente vemos los lenguajes interpretados en el desarrollo de aplicaciones o sitios web. A los lenguajes compilados los vemos más en software de escritorio ya que requieren de mayores recursos y de acceso a archivos determinados.
Hemos visto que los programas interpretados o compilados tienen distintas ventajas e inconvenientes. En un intento de combinar lo mejor de ambos mundos, durante la década de los 90 surge con fuerza el enfoque de máquina virtual (virtual machine).
Los principales lenguajes abanderados de esta tecnología son, por un lado, el lenguaje Java de Oracle, y por otro, los lenguajes de la plataforma .NET de Microsoft (Visual Basic.NET y C#). La filosofía de la máquina virtual es la siguiente: el código fuente se compila, detectando los errores sintácticos, y se genera una especie de ejecutable, con un código máquina dirigido a una máquina imaginaria, con una CPU imaginaria. A esta especie de código máquina se le denomina código intermedio, o a veces también lenguaje intermedio, p-code, o byte-code).
Como esa máquina imaginaria no existe, para poder ejecutar ese ejecutable, se construye un intérprete. Este intérprete es capaz de leer cada una de las instrucciones de código máquina imaginario y ejecutarlas en la plataforma real. A este intérprete se le denomina el intérprete de la máquina virtual.
Con esto logramos algunas ventajas: El código es portable independiente de la plataforma a ejecutarse, el mismo código puede ejecutarse en cada plataforma (Hardware, sistema operativo) usando solo el intérprete de ese sistema haciendo también más rápida la ejecución ya que no comprueba la sintaxis del código.

TIPOS DE APLICACIONES

En el mercado informático actual, nos encontramos con diferentes soportes de hardware que albergan variados tipos de aplicaciones, ya sean exclusivas de Internet, del sistema operativo o de un aplicativo en general. Podríamos llegar a diferenciar los distintos tipos de aplicaciones en cuatros grandes grupos: las de escritorio, las Webs, las móviles y las hibridas (mezcla de las anteriores).

APLICACIONES DE ESCRITORIO

Las aplicaciones de escritorio o desktop apps son aplicaciones creadas para ejecutarse en una computadora sobre un sistema operativo y su rendimiento depende de diversas configuraciones de hardware como memoria RAM, disco rígido, memoria de video, etc. Se puede desarrollar este tipo de aplicaciones conociendo lenguajes de programación con C#, Visual Basic, C++ etc. Algunos ejemplos de estos tipos de aplicaciones pueden ser Adobe Reader o Microsoft Word.

APLICACIONES WEB

Por otro lado, las aplicaciones Web o Web apps son ejecutadas sobre navegadores Webs. Con este tipo de aplicaciones, el usuario ingresa la dirección de ubicación, conocida como URL, y comienza a interactuar con ella, tal cual como si se tratara de una aplicación de escritorio. Algunos ejemplos de estos tipos de aplicaciones pueden ser: Amazon, Gmail, Wikipedia, Facebook etc.
La base de programación de las aplicaciones web es el HTML5, conjuntamente con JavaScript y CSS.
Habitualmente las aplicaciones webs ofrecen menos funcionalidades que las aplicaciones de escritorio. Esto se debe a que las funcionalidades que se pueden realizar desde un navegador son más limitadas que las que se pueden realizar desde el sistema operativo. A su vez las aplicaciones webs tienen como ventaja que no necesitan ser descargadas e instaladas y que sus actualizaciones son automáticas, cuando nos conectamos a dicho sitio siempre estaremos utilizando la última versión desarrollada.

APLICACIONES MOVILES

Las aplicaciones webs no permiten aprovechar al máximo la potencia de los diferentes componentes del hardware del teléfono por ejemplo no puedo acceder a mi listado de contactos. Las aplicaciones móviles son aquellas que han sido desarrolladas con el software que ofrece cada sistema operativo a los programadores, llamado genéricamente Software Development Kit o SDK. Así, Android, iOS tienen uno diferente y las aplicaciones nativas se diseñan y programan específicamente para cada plataforma, en el lenguaje utilizado por el SDK por ejemplo Java en el caso de Android y Objective-C o Swift en el caso de iOS.
Este tipo de aplicaciones se descargan e instalan desde las tiendas de aplicaciones (Store o Market). Las aplicaciones móviles se actualizan frecuentemente y en esos casos, el usuario debe volver a descargarlas para obtener la última versión, que a veces corrige errores o añade mejoras.
Algunos ejemplos de este tipo de aplicaciones son: WhatsApp, y Office Lens. La desventaja de este tipo de aplicaciones es que el desarrollo de una aplicación es muy distinto para cada sistema operativo.

APLICACIONES HIBRIDAS

Las aplicaciones hibridas o Hybrid Apps son una mezcla de distintos tipos de aplicaciones, esto permite trabajar con lo mejor de un mundo y del otro. El mejor caso se da entre el mundo web y del mundo móvil donde podremos usar los estándares de desarrollo web (HTML5 y JavaScript) y aprovechar las funcionalidades del dispositivo móvil tales como la cámara, el GPS o los contactos. En el mercado podemos encontrar algunas herramientas para desarrollar este tipo de aplicaciones: Apache Cordova, Ionic y React Native son algunas de las más populares. Las desventajas de estas aplicaciones es el rendimiento ligeramente inferior a una aplicación móvil nativa.
Inclusive, el desarrollo híbrido no es exclusivo del ecosistema móvil. Existen aplicaciones de escritorio híbridas que se usa con frecuencia como WhatsApp, Visual Studio Code, y Slack. ya que permiten ejecutarse en distintos Sistemas Operativos (Windows,Mac y Linux) usando Electron.