MEDAC
Instituto Oficial de Formación Profesional

ENTORNOS DE DESARROLLO

Introducción a los entornos de desarrollo

4

ÍNDICE

OBJETIVOS

Conocer la evolución de los entornos de desarrollo.

Saber qué es un IDE y sus funcionalidades.

Identificar las características básicas de los IDEs.

Conocer los diferentes tipos de IDEs.

/ 1. Introducción y contextualización práctica

El desarrollo de software se considera una tarea de cierta complejidad que requiere de herramientas que faciliten la tarea a los programadores y a su vez que permitan aumentar su productividad de manera sencilla.

En la actualidad existe un gran número de entornos de desarrollo que se pueden agrupar por la tecnología a la que va destinada, por ejemplo, entornos de escritorio, dispositivos móviles, videojuegos, etc.

Los entornos de desarrollo no siempre han existido, pues tienen una gran dependencia del sistema operativo sobre el que se van a ejecutar.

Para que un IDE sea realmente útil (desde el punto de vista de la productividad) es necesario que dispongan de una interfaz gráfica que permita que los programadores puedan acceder a las funcionalidades de una forma sencilla.

Como su nombre indica, los entornos de desarrollo normalmente incluyen todas las herramientas que los desarrolladores puedan requerir, por ejemplo, para poder editar el código rápidamente y poder realizar la compilación de una manera sencilla.

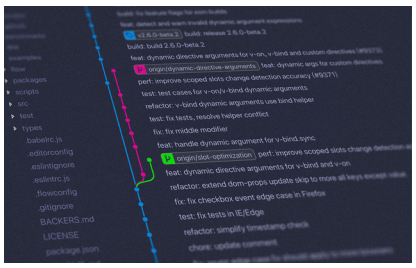

Fig 1. Características de los IDEs para poder integrar gestión de versiones

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

Audio intro. "Creación de una aplicación de control de acceso".
https://bit.ly/3alQ8dF

/ 2. Qué es un IDE

Un **IDE** (Integrated Development Environment) es un tipo de aplicación cuyo objetivo es facilitar el desarrollo del software. Normalmente, el software que desarrollamos adquiere unas dimensiones que requiere de un entorno de desarrollo que facilite las tareas que tengamos que realizar como depuración, compilación, edición etc.

Típicamente, un IDE incluye un editor que suele ser muy potente y ayuda al desarrollador a escribir el código de una manera más rápida a través del autocompletado y autoformato. Además, puede ser posible que un IDE incluya soporte para más de un lenguaje, siendo su editor muy potente para poder cubrir sus tareas de autoformato y autocompletado para diferentes plataformas.

Con el fin de facilitar el desarrollo, muchos IDEs incluyen una interfaz gráfica muy potente capaz de:

- Incluir muchas funcionalidades al alcance de los desarrolladores a través de una interfaz gráfica o GUI.
- Incluir herramientas para generar una representación visual del código. En muchas ocasiones el código crece tan rápido que puede ser imprescindible tener su representación visual para poder tener una visión global de éste.

Con el objetivo de aumentar la potencia de los IDEs, muchos se diseñan de una forma modular, permitiendo añadir pequeñas funcionalidades al núcleo original a través de plugins.

Aunque se trata de una funcionalidad secundaria, algunos IDEs son realmente conocidos por la capacidad de poder incorporar gran potencia adicional a través de los plugins. Por ejemplo, algunos desarrolladores pueden compartir sus propias abreviaturas de código para mejorar el autocompletado.

Fig. 2. Ejemplo de visualización de código del IDE Vistual Studio

Video 1. "Instalación de un IDE de desarrollo ".
https:/bit.ly/32g1E5k

2.1. Evolución histórica

Tendemos a pensar que los IDEs han existido siempre, sin embargo, en los inicios de la programación no se utilizaban archivos de texto sino tarjetas perforadas. Para poder desarrollar dichas tarjetas se requería de las herramientas adecuadas que permitieran realizar la perforación requerida.

En la computación moderna, el desarrollo del software se ha basado en la utilización de archivos de texto plano que pueden ser traducidos a código ensamblador a través de un proceso de compilación. La evolución de los IDEs ha estado marcada por la evolución tanto de los sistemas operativos como de las interfaces gráficas. Normalmente, para que un IDE sea potente, debe disponer de una interfaz gráfica o secuencia de comandos, hasta el punto que, si el sistema operativo no ofrece una GUI interactiva, no será posible crear un IDE potente.

El lenguaje Dartmouth Basic fue el primer lenguaje en ser soportado por un IDE sin interfaz gráfica, pues se centraba en utilizar la línea de comandos a través de atajos y comandos propios del IDE. A pesar de disponer de un IDE para su desarrollo, éste no se parecía a los IDEs visuales que tenemos en la actualidad. Al no disponer de interfaz gráfica, la productividad de los desarrolladores no aumentaba como lo hace con los IDEs actuales.

Como hemos dichos anteriormente, los IDEs son diseñados modularmente para poder añadir funcionalidades extras a través de plugins. Sin embargo, esta funcionalidad no ha estado siempre al alcance del desarrollo. Uno de los primeros IDEs en soportar esta funcionalidad fue Softbench.

En definitiva, podemos decir que gracias a la utilización de los IDEs y a su evolución, disponemos hoy en día de un software en menor tiempo y de mejor calidad. Al tener un entorno totalmente integrado, los desarrolladores se pueden centrar en el desarrollo concreto de la aplicación en la que están trabajando además de poder acceder a un amplio conjunto de herramientas para lanzar el producto a tiempo.

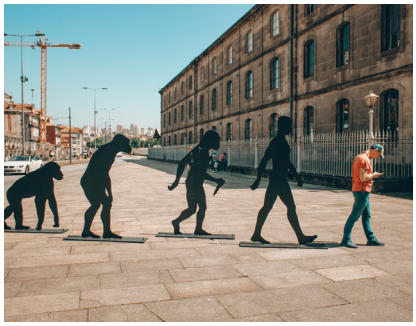

Fig. 3. Evolución histórica del desarrollo

Audio 1. "Desarrollo de aplicaciones de manera sencilla".
https://bit.ly/30OUSpe

/ 3. Caso práctico 1: "Control de acceso a sala"

Planteamiento. Uno de nuestros principales clientes nos pide que desarrollemos una solución software para la gestión del acceso a una ubicación segura en la que se debe tener en cuenta diferentes aspectos como acceso a bases de datos, lectura de dispositivos externos, etc.

Para ello, nos dice que todo debe ser realizado mediante archivos de texto plano, pues dado que se trata de un entorno que debe garantizar la seguridad, no es posible utilizar ningún entorno de desarrollo. Todo el proceso de compilación y resolución de dependencias debe realizarse a mano.

Otro requisito, es que el tiempo de entrega es de 7 días.

Nudo. ¿Crees que lo que nos pide el cliente tiene sentido?

Desenlace. Según hemos visto, realizar la programación de un entorno con dicha complejidad sin la utilización de un IDE puede ser realmente complejo.

La justificación de que se trata de un entorno seguro no es válida, pues el compilador utilizado por el IDE o sin él, será el mismo, y de una forma u otra, el propio compilador generará el código ensamblador que se va a ejecutar en la solución final, se use IDE o no.

Desde el punto de vista de la productividad, al no disponer de un IDE de desarrollo, el tiempo para tener la solución final será mayor, pues no tendremos acceso a todas las ventajas que nos ofrece un IDE como la detección temprana de errores y sugerencias para su arreglo.

Fig. 4. Debemos analizar en detalle lo que nos plantean los clientes

/ 4. Funciones de un entorno de desarrollo

Como podemos deducir, un IDE está diseñado y desarrollado en muchas ocasiones principalmente por desarrolladores que saben perfectamente las necesidades del producto para el resto de la comunidad. De esta forma, un IDE está preparado para proporcionar un alto grado de productividad a los programadores.

Desde el punto de vista de una empresa, interesa que los programadores sean más efectivos, pues en menos tiempo son capaces de realizar más tareas. Por ello, las empresas requieren que sus desarrolladores conozcan algún IDE. De esta forma, un IDE es un programa que permite realizar prácticamente todas las tareas que un desarrollador requiera como la edición de código.

Los IDEs ofrecen una interfaz para crear y modificar el código existente a través de su editor inteligente. Desde el punto de vista de la compilación, un IDE permite facilitar el proceso de tal forma que, generalmente, los desarrolladores desconocen lo que está pasando realmente 'por detrás'.

Como podemos observar, los IDEs permiten reducir la configuración previa que debe realizar el desarrollador para comenzar a ser productivo. De nuevo, al requerir menos tiempo de configuración, el programador podrá aumentar su productividad.

Una de las características más relevantes de los IDEs es la posibilidad de poder informar al programador de los errores cometidos durante el propio desarrollo sin necesidad de compilar. De esta forma, el editor que ofrecen los IDEs son inteligentes, pues pueden ir realizando un precompilado e interpretación del código para indicar al programador los posibles errores léxicos, sintácticos, componentes no encontrados, etc. Además, una vez que detecta un error, el propio editor es capaz de ofrecer diferentes alternativas para poder solucionar dichos problemas.

Fig. 5. Características de los IDE de desarrollo

/ 5. Componentes y tipos de entornos de desarrollo

Algunos IDEs están especializados en un lenguaje de programación en concreto, permitiendo que el programador pueda realizar todas las tareas necesarias en dicho lenguaje. Un problema que tiene este enfoque es que los desarrolladores deberán conocer diferentes IDEs para diferentes lenguajes. Como el número de lenguajes aumenta constantemente, los programadores deberán realizar un doble aprendizaje, el lenguaje más, el IDE.

Por ello, existe otro enfoque en el que los IDEs pueden trabajar con más de un lenguaje, como es el caso de Eclipse, Netbeans, XCode, etc., y con numerosos componentes cómo los plugings o bibliotecas de terceros, que les hacen adquirir la versatilidad adecuada.

a. Tipos de entornos de desarrollo

Existen diferentes tipos de IDEs dependiendo de la dimensión que apliquemos. De esta forma, se pueden utilizar una o varias de las siguientes dimensiones para diferenciar el tipo de IDE:

- **Número de lenguajes soportados**. Si realizamos un desarrollo con multilenguaje, por ejemplo, en entornos web, es recomendable que el IDE soporte todos ellos.
- **Sistemas operativos soportados**. Muchas veces, un IDE solo se encuentra disponible para un sistema operativo en concreto, por ejemplo, Visual Studio .NET o XCode.
- **Características de automatización.** Se trata de una característica que puede influir directamente en la productividad del entorno.
- **Impacto en el rendimiento del sistema.** Muchos de los IDEs actuales requieren de sistemas potentes, si no disponemos de ellos puede que el rendimiento de la máquina decaiga significativamente.
- **Plugins y extensiones.** Se trata de una de las características determinantes para muchos programadores para poder personalizar el entorno de trabajo.

- **Pago por uso.** Cada vez surgen nuevos IDEs que tienen una versión gratuita y otra de pago con nuevas funcionalidades.

Video 2. "Tipos de entornos de desarrollo".
https://bit.ly/2MMhn8r

/ 6. Caso práctico 2: "Selección del IDE"

Planteamiento. En nuestro equipo de desarrollo tomamos la decisión de utilizar el IDE de desarrollo "ProgramaMás" porque es el que más lenguajes soporta actualmente en el mercado. Como resultado, se convierte en el IDE oficial de la empresa para poder desarrollar cualquier problema software que nos llegue desde los clientes.

Nudo. ¿Crees que hemos realizado una elección correcta? ¿Deberíamos tener en cuenta otros criterios para la elección del IDE?

Desenlace. Según hemos visto en el tema, existen diferentes criterios para poder realizar la elección de un IDE de una manera eficiente. En el caso expuesto, se utiliza una única dimensión para poder realizar la elección del IDE: el número de lenguajes de programación soportados.

Sin embargo, nos estamos olvidando de otros aspectos muy relevantes como la existencia de herramientas de compilación, depuración, autocompletado, etc. Todas estas herramientas hacen que el IDE sea realmente una herramienta productiva para poder mejorar el rendimiento de los desarrolladores.

De poco servirá un IDE que destaque en una única dimensión si ignora el resto de ellas. Por ello, es importante que cuando se realice una elección de un IDE se acote su contexto de uso, por ejemplo, para lenguajes de propósito general orientados a objetos.

En este caso práctico, tendríamos realizar una elección más balanceada, en la que el IDE utilizado sea capaz trabajar con 1 ó 2 lenguajes, pero incluya herramientas externas para compilar, enlazar, depurar y un buen editor de código con autocompletado.

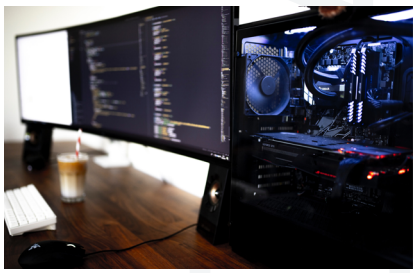

Fig. 6. Entorno de desarrollo con una máquina potente

/ 7. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema hemos presentado las principales herramientas de desarrollo actuales como son los IDEs. Como hemos visto, se trata de unos entornos de desarrollo integrados que ofrecen todas las funcionalidades que un desarrollador necesita para poder crear nuevas funcionalidades en diferentes lenguajes.

También hemos podido comprobar cómo los IDEs permiten aumentar la productividad de los desarrolladores al brindar la posibilidad de tener un entorno completamente configurado con unos pocos clics. Desde el punto de vista de las empresas, interesa que los programadores conozcan los IDEs para aumentar su rentabilidad. Para que un IDE sea realmente útil es necesario que disponga de determinadas características como la autogeneración de código, depuración, compilación, etc.

Una de las características que más se demanda en la actualidad en los IDEs es la posibilidad de incluir nuevas funcionalidades a través de plugins. A través de ellos, los programadores pueden personalizar su entorno y compartirlo con el resto de la comunidad de programadores.

Resolución Del Caso Práctico De La Unidad

Para poder desarrollar aplicaciones en un entorno móvil, es necesario conocer la tecnología sobre la que vamos a implementar nuestro desarrollo.

En la actualidad existen dos grandes entornos de desarrollo móvil que son Android y iOS.

Para poder desarrollar en cada uno de estos entornos es necesario conocer la SDK y el entorno de desarrollo propio para cada tipo de dispositivo, Android Studio para Android y XCode para iOS.

Una alternativa sería crear una aplicación híbrida en la que usemos tecnología web además de tecnología móvil. Nótese, que cada plataforma de desarrollo está vinculada a un lenguaje en concreto.

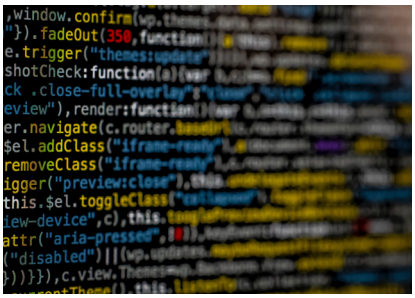

Fig. 7. Soluciones híbridas de tecnología

/ 8. Bibliografía

Boudreau, T. (2003). NetBeans: the definitive guide. Beijing Cambridge Mass: O'Reilly.

Burnette, E. (2005). Eclipse IDE: pocket guide. Sebastopol, CA: O'Reilly.

Hagos, T. (2019). Android Studio IDE Quick Reference: a Pocket Guide to Android Studio Development. Berkeley, CA: Apress Imprint Apress.