ASIGNATURA

Fundamentos de Programación I

GRADO EN INGENIERÍA INFORMÁTICA

TEMA 2

Herramientas de Programación

Autor: Francisco José Palacios Burgos

Unidad didáctica.

Tema 2: Herramientas de Programación

INDICE

I. PRESENTACION

El campo del desarrollo de software, al igual que ocurre con otras disciplinas, hace uso de una serie de técnicas y herramientas que son necesarias para conseguir el producto final: una aplicación o programa que pueda ejecutarse en un sistema informático. En este tema vamos a hacer un recorrido por las herramientas más habituales que se usan para ese fin.

II. OBJETIVOS.

Mediante el estudio de este tema se adquirirá la capacidad de:

- Conocer cuáles son las principales herramientas implicadas en la fase de programación o implementación de aplicaciones informáticas
- Tener un criterio para valorar cuál de dos herramientas equivalentes es la mejor para situaciones concretas que ocurren en etapa de desarrollo
- Saber reconocer los principales mensajes que estas herramientas pueden darnos en su uso diario

III. ESQUEMA

IV. CONTENIDO

1. Herramientas de desarrollo

La etapa de implementación, en la que se plasma un algoritmo en un código fuente que en última instancia se ejecutará en una computadora, requiere de una serie de herramientas que enumeramos a continuación:

- Editor de programación. Nos permitirá escribir el código fuente del programa
- Compilador. En caso de que estemos escribiendo código en un lenguaje de tipo compilado, esta herramienta transformará el código fuente en código objeto o, con la ayuda de un enlazador, en un ejecutable para una plataforma concreta
- Intérprete. Si el lenguaje de programación escogido es de tipo interpretado, esta herramienta analizará el código fuente e irá ejecutando las instrucciones una por una
- Depurador. Esta herramienta nos permitirá encontrar errores en el código del programa. Especialmente, aquellos que se producen en tiempo de ejecución, indicándonos el lugar (línea de código) en que se ha producido y los valores de las variables en ese momento.

Adicionalmente a estas herramientas esenciales, existe otro conjunto de ellas muy útiles en la etapa de implementación que favorecen la productividad y minimizan los errores y las situaciones problemáticas, especialmente en proyectos grandes.

- Entornos de desarrollo integrados (IDE)
- Sistemas de control de versiones.
- Sistemas de generación automática de documentación

2. Editor de Programación

Esta herramienta permite escribir los ficheros de código fuente de los que consta un programa. A la hora de elegir un editor de programación podremos escoger entre un amplio abanico de ellos. Los hay que son generalistas y por ello soportan una gran cantidad de lenguajes y los hay especializados en un lenguaje de programación concreto.

En cualquiera de los casos, lo importante es que entre los lenguajes de programación soportados esté el que vayamos a emplear en el desarrollo del programa. Esto significará, como mínimo, que cuente con la característica de resaltado de sintaxis (Syntax highlighting) para ese lenguaje.

El **resaltado de sintaxis** es la característica mediante la cual, el editor hace uso de un esquema de colores o énfasis para mostrar de forma diferente las palabras reservadas en ese lenguaje, los nombres de las variables, las constantes o las cadenas de caracteres, por ejemplo.

A continuación se muestra un ejemplo de esta característica:

```c
/* Programa ejemplo */

#include <stdio.h>

#define CONSTANT 100

int main(int argc, char* argv[])
{
    int i = 0;
    int suma = 0;

    for(i=1; i<=CONSTANT; i++) {
        suma = suma + i;
    }

    printf("La suma de los %d primeros números es %d", CONSTANT, suma);

    return 0;
}
```

Figura 1. Ejemplo de editor con resaltado de sintaxis.

Otras características que suelen estar presentes en los editores de programación:

Numeración de líneas y navegación. Cada línea es numerada consecutivamente, de manera que a la hora de encontrar errores es mucho más sencillo. En estos casos suele haber un menú o comando para situarse en una línea concreta.

Multiedición con pestañas. Se pueden editar varios ficheros a la vez. En esos casos se suele poder navegar por ellos a través de una serie de pestañas.

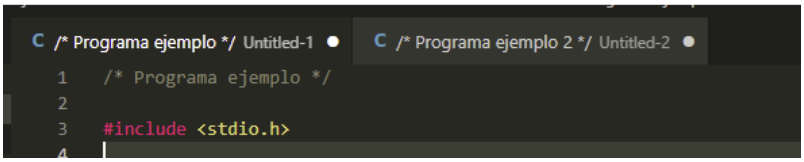


Figura 2. Modo de edición con pestañas.

Autocompletación de código. Ésta es una propiedad por la cual el editor rellena código automáticamente ante ciertas acciones de usuario. Por ejemplo es muy típico que en muchos lenguajes los bloques de código se delimiten por un par de llaves { }. En estos casos, al escribir el usuario la primera llave, el editor automáticamente pone la segunda.

Ayuda contextual. Es una propiedad por la cual al escribir código, el editor muestra una cierta ayuda sobre todo en funciones o características de las librerías estándar del lenguaje.

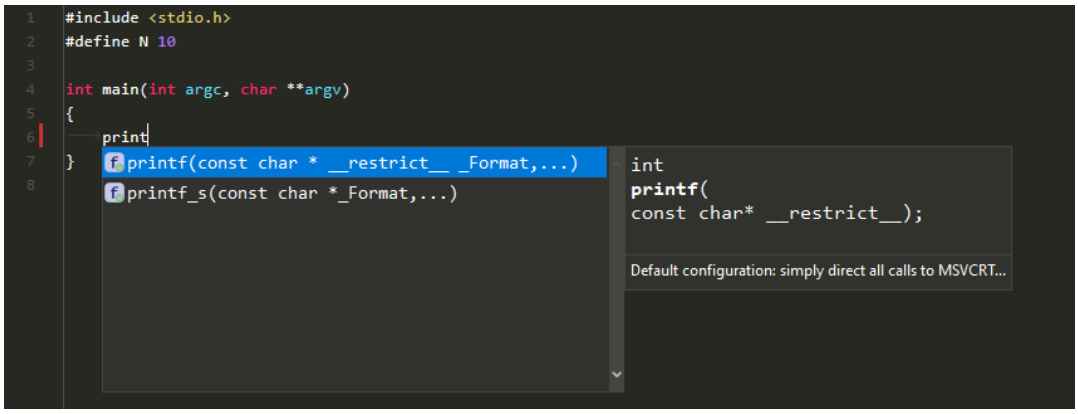


Figura 3. Ejemplo de ayuda contextual

Formateado de código. El editor es capaz de dar formato al documento, introduciendo por ejemplo espacios. sangrías y saltos de línea que hacen la lectura del código más fácil.

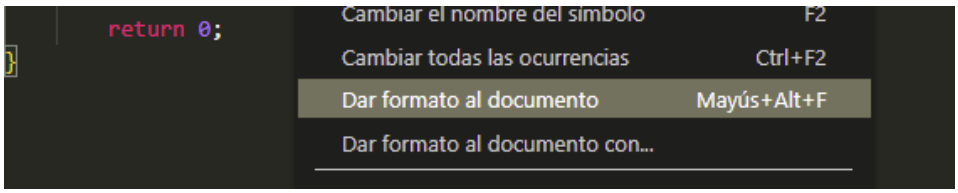


Figura 4. Formateado del código

Comentar/descomentar bloque de código. El editor es capaz de tomar una serie de líneas o una porción de una línea y rodearla con símbolos de comentario, de manera que ese código ya no se toma en cuenta en la compilación o la ejecución.

Refactorización de código. Esta es una operación mediante la cual se cambia masivamente el identificador (nombre) de una constante, una variable o función en todos aquellos sitios en los que aparezca. Se distingue de una operación más simple de Búsqueda y reemplazo en que analiza el contexto para saber si se trata de ese elemento o no.

3. Compilador

Esta herramienta es típica de los lenguajes de tipo compilado. Ejemplos de estos lenguajes podrían ser C, C++, Fortran, Pascal, etc.

En estos lenguajes el código fuente se emplea para generar un programa ejecutable para una plataforma concreta. En el proceso de generación interviene el compilador.

Un compilador es un programa que realiza un análisis léxico, sintáctico y/o semántico para generar después un archivo de código binario (código objeto) con instrucciones de bajo nivel, similares a las de ensamblador o código máquina.

Es por ello, que el proceso de compilación es técnicamente complejo y estructurado en etapas, aunque desde el punto de vista del usuario (el desarrollador), solamente se percibe el resultado final, que es el ejecutable.

Figura 5. Etapas de compilación

Como vemos, los compiladores suelen ser herramientas muy complejas, que funcionan en modo línea de comandos y que cuentan con un número muy amplio de opciones o switches que regular o alteran el proceso de compilación.

Entre las opciones que deberemos conocer del compilador elegido estarían:

- Cambio del nombre del ejecutable resultante. Por defecto, muchos compiladores generan el ejecutable con un nombre predefinido, pero mediante una opción de la línea de comando podemos especificar nosotros el nombre final
- Inclusión de ficheros de cabecera o ficheros de definiciones. Muchos lenguajes tienen la característica de poder indicar en el código fuente que se incluya ficheros externos que suelen contener definiciones de tipos, constantes, etc. En estos casos, se puede indicar al compilador una serie de rutas donde buscar estos ficheros.
- Inclusión de librerías. En este caso, se hace uso en el código fuente de funciones o procedimientos que se encuentran en un fichero de librería. El compilador entonces nos permite especificar las rutas donde encontrar dichos ficheros.

Una librería es un fichero de código binario (y por tanto ejecutable) que contiene funciones, procedimientos, definiciones de variables, etc. Este fichero no está pensado para ser ejecutado directamente en la shell de un sistema operativo, sino como parte de un programa (librerías estáticas) o para ser cargado en memoria cuando un programa lo solicite (librerías de carga dinámica)

- Optimización de código. Muchos compiladores cuentan con la característica de poder aplicar diversos grados de optimización al código analizado para producir ejecutables que sean más pequeños y eficientes. Algunas de estas optimizaciones son genéricas y están basadas simplemente en reorganizaciones del código fuente o sustituciones en el mismo. Otras en cambio está basadas en hacer uso de las características subyacentes del hardware en el que se va a ejecutar el programa.

Los compiladores pueden tener una serie de características adicionales entre las que podrían estar:

- Crosscompiling (Compilación cruzada). Esta es una característica que permite al compilador generar ejecutables para plataformas distintas a las que está corriendo. Por ejemplo, podríamos estar programando en MS Windows y compilamos el mismo programa para GNU/Linux
- Compilación a lenguaje intermedio. En este caso, no se genera un ejecutable para una plataforma concreta, sino que se genera código ensamblador o código objeto para una especie de plataforma intermedia como paso previo a la generación del código final. Aquí tendríamos dos casos distintos.

  Podría ser que esa plataforma intermedia sea de tipo ensamblador, pero no ligada a una plataforma real concreta. En ese caso, haría falta después un segundo proceso que tradujese ese código ensamblador intermedio a código ensamblador específico de plataforma y finalmente se produjese un ejecutable. El compilador LLVM es un ejemplo de ello.

  Por otra parte, podría ser que el código generado fuese código objeto de una especie de máquina virtual. En este caso, ese código objeto no se puede ejecutar sin más, sino que requiere de una infraestructura de runtime que es la que en última instancia traducirá el código de máquina virtual a código específico de la plataforma donde se está ejecutando el runtime. Ejemplos de este tipo son los compiladores de Java o .NET

- Compilación de una pasada o de varias pasadas. Si es de una pasada, el proceso de compilación analiza el código fuente una sola vez y a continuación se produce el código objeto. Si por el contrario el compilador es de varias pasadas, el código fuente es analizado varias veces antes de producir el código objeto.

4. Intérprete

El intérprete es un aplicativo software que es capaz de tomar un código fuente e ir ejecutando línea por línea las instrucciones que en él se contienen. No necesita por tanto producir un ejecutable para una determinada plataforma.

Debido a esa ejecución línea a línea, en la que se va traduciendo el código fuente a instrucciones nativas de la plataforma, los programas interpretados suelen partir de una situación de desventaja con respecto a los programas compilados ofreciendo un peor rendimiento de ejecución. Si bien lo anterior es cierto, hay muchos intérpretes que han aplicado técnicas de optimización mejorando en gran medida sus rendimientos.

Una técnica bastante común en los intérpretes modernos es la de transformar el código fuente de forma previa a una representación intermedia que sea luego más eficiente para su traducción al código máquina nativo de la plataforma.

Otra técnica que se emplea es la compilación Just In Time (JIT). Mediante esta técnica el código (bien sea el de partida o el de representación intermedia) o partes del código son compiladas internamente a código nativo de la plataforma, que es ejecutado luego de forma más rápida. Si bien esto tiene un cierto coste en el consumo de memoria de todo el proceso o en el tiempo inicial de arranque del proceso de interpretación, las mejoras obtenidas compensan ampliamente esto.

Hoy en día es también bastante frecuente que los intérpretes no se ejecuten solamente en la shell de un sistema operativo, sino que puede haber intérpretes que se ejecuten dentro de otros programas. Un ejemplo de eso es Javascript, que puede ejecutarse y realizar su tarea dentro de un nevegador de internet.

5. Depurador

El depurador es una herramienta que nos va a ser útil en la etapa de implementación y sobre todo en la etapa de pruebas. Su utilidad reside en que nos va a permitir ejecutar el programa resultante de la ejecución y detectar todos aquellos errores que se dan en tiempo real de ejecución. Estos errores pueden deberse a multitud de causas: datos incorrectos en las variables que provocan operaciones no permitidas (ej: divisiones por cero), accesos a posiciones no válidas de memoria, etc.

También nos va a permitir hacer un seguimiento de todos los valores que van tomando las diferentes variables que tenemos en nuestro programa (control del estado del programa).

Las principales operaciones que permite hacer un depurador suelen ser las siguientes:

- Ejecución paso a paso por instrucciones. En este caso el depurador va ejecutando el código línea a línea. Si se encuentra un salto en el código debido al estar presente una llamada a una función, entra en la función y empieza a ejecutar línea a línea.

- Ejecución paso a paso por procedimientos. En este caso, si se encuentra una función la ejecuta de golpe sin entrar en ella. El resultado es tenido en cuenta en la siguiente línea en la que se encuentre la llamada.
- Establecer un punto de interrupción/ruptura (breakpoint). Con esta operación indicamos al depurador que queremos pausar la ejecución del código en una línea concreta. Suele marcarse en el editor mediante un resaltado en otro color o un icono en la barra lateral de numeración

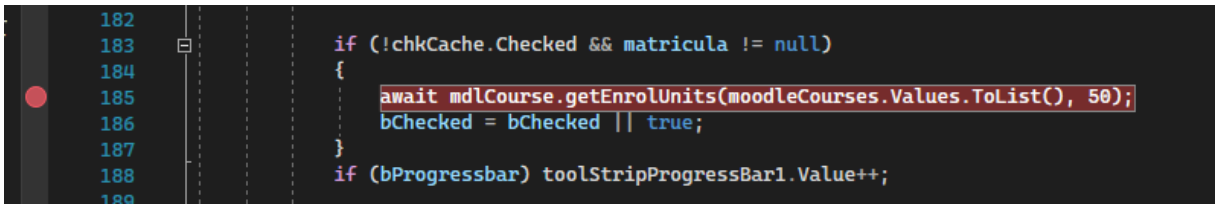


Figura 6. Punto de interrupción establecido en una cierta línea de código. Se resalta la línea y se pone un icono en la barra lateral.

Los puntos de ruptura pueden incondicionales (se producen siempre que se alcance esa línea de código) o condicionales (se exige además que se cumpla una determinada condición)

- Agregar inspección de variables. Permite en un punto de la ejecución ver los valores que tienen en ese momento un cierto conjunto de variables. Normalmente dependerá de la visibilidad de las mismas en los bloques.

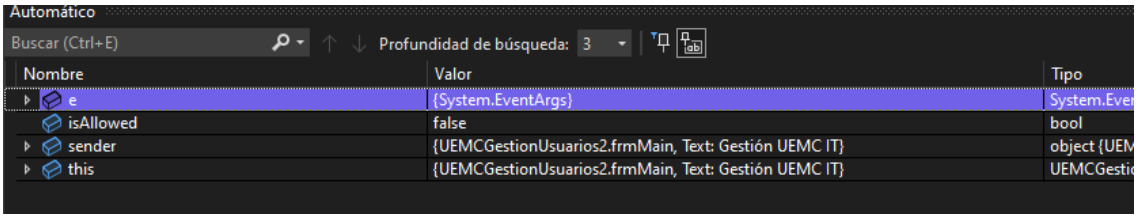


Figura 7. Panel de inspección de variables.

- Examinar el contenido de los registros del procesador y la pila del programa. Ésta es información de más bajo nivel pero que a veces ayuda a encontrar errores.
- Mostrar la traza de la pila de llamadas que han conducido al punto actual de ejecución en el programa. Esta característica es muy útil para saber que camino ha seguido el programa, ya que en ocasiones esto no es tan evidente.

Los depuradores también suelen categorizarse en dos tipos: los que realizan depuración local y los que admiten depuración remota. En este último caso, funcionaría en uan especie de modo servidor y permitiría que un cliente de depuración se conectase desde otra ubicación para ejecutar un programa.

6. Entornos de Desarrollo Integrado (IDE)

Estos sistemas suelen ser proyectos software que engloban o permiten interactuar con las otras herramientas que intervienen en el proceso de desarrollo de programas.

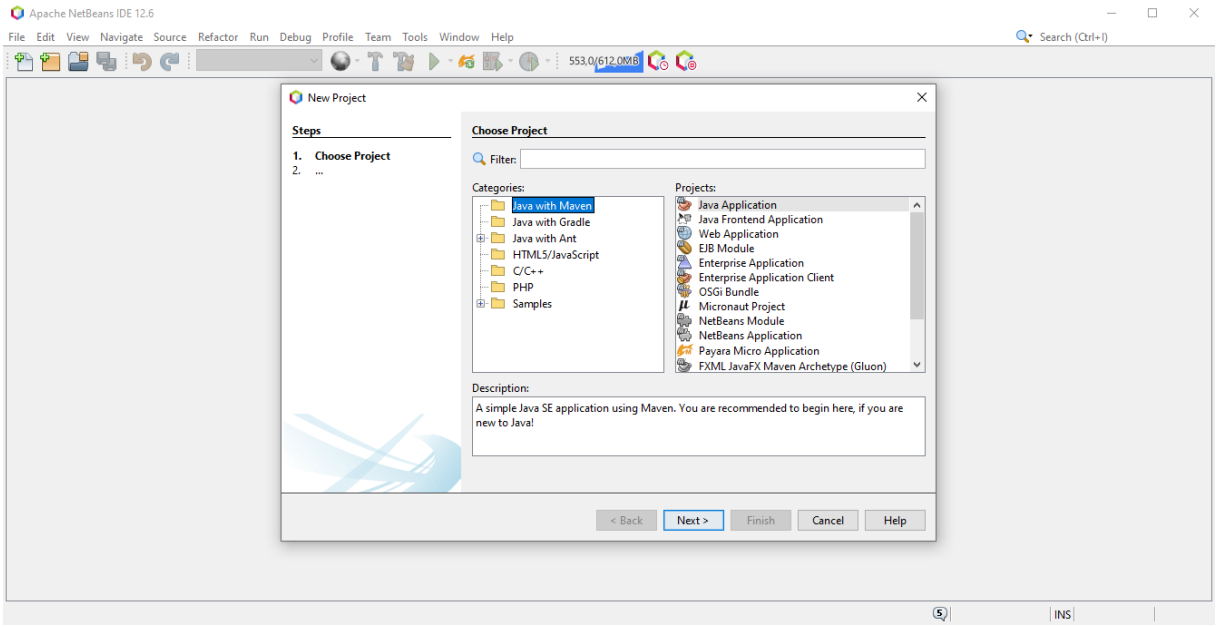


Figura 8. Creación de un proyecto en un IDE.

Suelen estar enfocados al trabajo por proyectos y es también una carácterística muy común que se puedan crear diferentes tipos de proyectos. La creación de proyectos suele hacerse mediantes asistentes, que nos guían por varios pasos en los que se suelen configurar varios aspectos del proyecto:

- Nombre del mismo
- Tipo de proyecto
- Ubicación. Estos IDE's suelen trabajar con workspaces. Por defecto crearán el proyecto dentro de un workspace
- Opciones de configuración varias (versión del runtime a emplear, librerías extras o frameworks, soporte para sistemas de control de versiones, etc.)

La interfaz de estos IDE's suele ser muy parecida de unos a otros. Es fácil identificar varios paneles o zonas que se repiten en ellos:

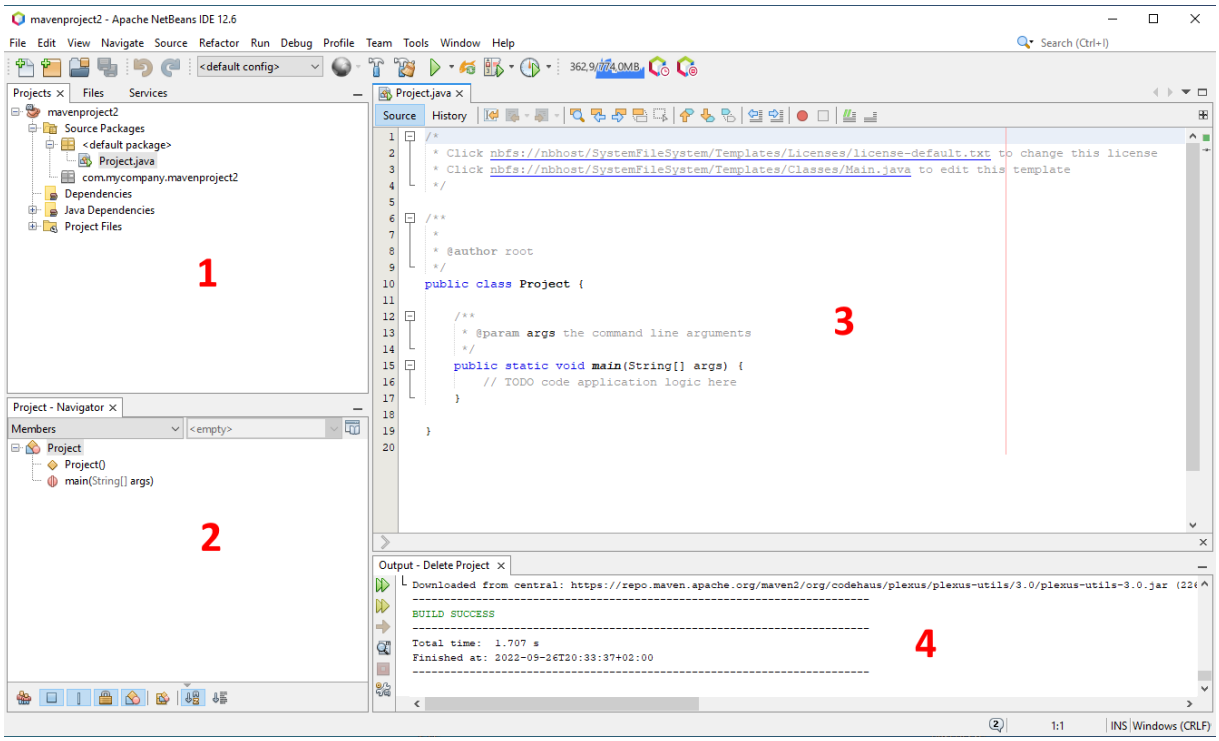


Figura 9. Aspecto de la interfaz de un IDE.

Por ejemplo, es habitual un navegador/gestor de proyecto (1). En este tipo de paneles se suele mostrar el conjunto de ficheros que forman parte del proyecto. Es usual mostrar esa colección de ficheros en forma de árbol. Además, ese árbol suele corresponderse con una cierta estructura de directorios en el disco duro. El árbol puede ser más o menos completo, pudiéndose mostrar también por ejemplo las dependencias o librerías adicionales.

Otro panel que suele aparecer es el navegador de funciones o clases (2). Ahí podremos ver de forma rápida (también en forma de árbol o listado) el conjunto de clases (si estamos en un lenguaje de Programación Orientada a Objetos) o el conjunto de funciones/métodos que se hayan presente en el archivo actual de edición.

En la parte central siempre suele estar el editor (3), que suele presentarse con las características habituales de estos programas (resaltado de sintaxis, multiedición con pestañas, code folding, etc.)

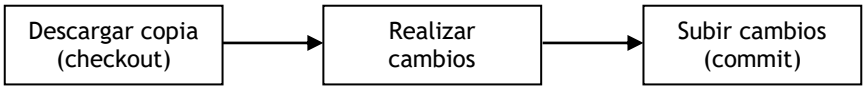
Un panel o zona también muy habitual es el panel de estado (4) que normalmente suele mostrar información muy valiosa como la salida (output) del compilador o programa (cuando se ejecuta) o los errores que se producen en tiempo de compilación.

Además, todos estos entornos cuentan con una barra de menús muy poblada que da lugar a toda la funcionalidad que despliegan estos entornos. Entre esos menús conviene tener siempre localizados (y si es posible conocer sus atajos de teclado) los de compilación/construcción del proyecto, los de depuración y los de edición/refactorización de código.

7. Sistemas de control de versiones

Estas herramientas van a ser muy útiles en proyectos grandes formados por multitud de ficheros de código fuente, en proyectos de cualquier tamaño en los que se produzcan frecuentes cambios a lo largo del tiempo y en aquellos proyectos en los que el equipo de desarrollo esté formado por varios integrantes.

El principio de funcionamiento de estos sistemas pasa por almacenar en una ubicación centralizada el código fuente, junto con todos los cambios o revisiones que se van produciendo. Así, casi todas estas herramientas funcionan con una arquitectura cliente-servidor. Los clientes, que en este caso son los equipos de los desarrolladores, suelen descargarse del servidor la última versión del código fuente, trabajan con la copia local para hacer modificaciones y cuando éstas están listas entonces se suben al servidor.



Algunas consideraciones sobre este proceso:

- En el servidor de control de versiones se suele almacenar el código base de una versión y luego las diferencias o cambios que se van produciendo en los archivos del mismo.

- La copia local que los desarrolladores se descargan lleva metainformación sobre la versión de código

- A la hora de subir cambios se produce un proceso de merging, calculándose los cambios necesarios sobre la versión actual en el servidor. Si hubiera conflictos (por ejemplo, porque dos desarrolladores han hecho modificaciones sobre el mismo fichero), el sistema resuelve automáticamente en la mayoría de los casos.

- Estos sistemas también suelen contar con capacidades de ramificación, y así mantener en paralelo el desarrollo de varias líneas del código fuente, que pueden corresponder a despliegues distintos o versiones diferentes de una aplicación.

A continuación vamos a ver cuáles son los principales sistemas de control de versiones utilizados por los desarrolladores:

7.1. Subversion (SVN).

Sistema de control de versiones con licencia libre (Apache) que se basa en almacenar en un repositorio centralizado el código fuente y las diferencias (deltas) que se van produciendo en él. Tiene capacidad para trabajar con ramas, siendo la estructura más habitual de un repositorio la siguiente:

- Trunk – Esta sería la rama principal de desarrollo
- Branches – Bajo esta localización se podrían crear ramas de desarrollo paralelo a trunk. Suele emplearse para revisiones de mantenimiento (corrección de errores) o para trabajar en nuevas versiones del producto.
- Tags (también releases) – Bajo esta localización estarían ramas cerradas, correspondientes muchas veces a versiones pasadas. No se suelen emplear para subir cambios.

Subversión gestiona eficientemente tanto ficheros de texto (calcula las diferencias mediante un algoritmo de tipo diff) y también archivos binarios. En el caso de los archivos de texto, lo que se trasmite a través de la red siempre son esas diferencias, nunca el fichero completo, haciendo que sea un sistema bastante eficiente.

Los protocolos más habituales que usa subversión son svn (para repositorios locales), ssh y el http/https (utiliza webdav como puente).

Los comandos más habituales para trabajar con subversión son:

- svn import – Realiza una importación inicial de un proyecto al sistema de control de versiones
- svn checkout – Descarga el codigo fuente de un repositorio para crear una copia local de trabajo sobre la que operar
- svn update – Actualiza la copia de trabajo con los últimos cambios que se hayan producido en el repositorio
- svn commit – Envía los cambios realizados sobre la copia de trabajo al servidor
- svn log – Informa sobre los cambios que se han producido en una rama
- svn revert – Revierte cambios hechos en la copia local, que no se han subido aún al servidor

- svn add – Añade ficheros nuevos al control de versiones en la copia local
- svn delete – Elimina ficheros de la copia local y los marca para ser también eliminados en el repositorio
- svn export – Obtiene el código fuente de una rama, pero limpio de toda información de control de metadatos

Actualmente se pueden obtener clientes svn (también servidores) para la mayor parte de los sistemas operativos (Windows, Linux, MacOS). Estas herramientas de cliente se encuentran disponibles:

- En modo de utilidades de comandos
- En modo gráfico
- Integradas en IDE's

7.2. Git

Git es una sistema de control de versiones de código abierto (Licencia GNU) pensado para trabajar con proyectos que cuenten con un número grande de ficheros de código fuente. Originalmente diseñado por Linus Torvalds actualmente es mantenido por la comunidad y ha tenido un gran crecimiento, de manera que en la actualidad multitud de proyectos de software lo utilizan para su desarrollo.

Algunas de las carecterísticas más relevantes de Git:

- Es un sistema de control de versiones con carácter eminentemente distribuido. Estaba inicialmente muy pensado para comunidades de desarrolladores de software libre. Cada desarrollador recibe en un copia local un historial completo de los cambios producidos en el repositorio. Los cambios se propagan cuando se producen a los repositorios locales.
- Los ficheros se almacenan internamente con un nombre codificado y con metainformación. Operaciones como el renombrado de archivos se simplifican enormemente.
- Gestión muy eficiente del cálculo de diferencias entre archivos. Esto hace que tenga un fuerte rendimiento en proyectos muy grandes.
- Trabaja con multitud de protocolos (ssh, http/https, rsync, …) Cuenta con pasarelas para integrarse con otros sistemas de control de versiones (CVS, git-svn, …)

Los comandos más habituales cuando se trabaja con Git son los siguientes:

- git init – Inicializa un nuevo repositorio
- git fetch – Descarga los cambios que se hayan producido en el repositorio
- git merge – Mezcla los cambios descargados por fetch en la rama actual de trabajo
- git pull – Es la mezcla de git fetch y git merge
- git commit – Confirma los cambios producidos para ser incorporados a la rama de desarrollo
- git push – Sincroniza la rama local de trabajo con su equivalente en el repositorio (sube los cambios)
- git status – Muestra los cambios producidos sobre la rama
- git reset – Descarta cambios que se hayan producido y que no se hayan fusionado aún con commit
- git add – Añade un nuevo fichero a la rama de trabajo
- git rm – Elimina un fichero de la rama de trabajo

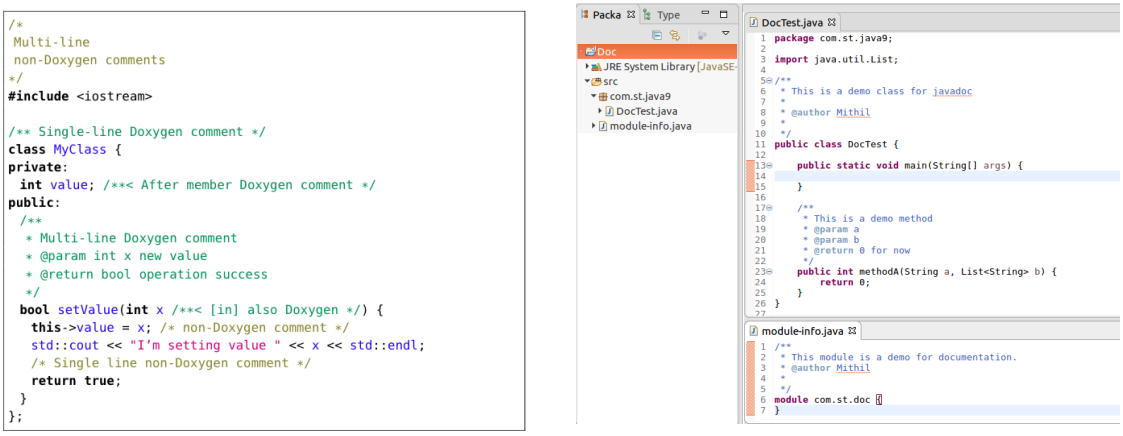
Actualmente existen versiones de Git para todas las plataformas habituales (Windows, Linux, MacOS). Todas las herramientas de cliente pueden encontrase en modo comando, con interfaz gráfica y por suspuesto integradas en los IDE's de programación.

Existen también interesantes y muy exitosos proyectos para albergar repositorios basados en esta tecnología en la nube (GitHub, GitLab, …).

8. Sistemas de documentación automática

Estos sistemas surgen de la necesidad de generar documentación de tipo técnico sobre las características de los módulos, funciones y código fuente de los proyectos software. Por lo tanto, generan documentación cuya audiencia natural son los desarrolladores del proyecto u otros programadores que por alguna razón tuvieran que entender las interioridades del código fuente.

Casi todos suelen funcionar de la misma forma: se introducen comentarios en el código fuente, pero con una sintaxis especial. Para los compiladores/intérpretes estos son simples comentarios, pero para las herramientas de generación de documentación son intrucciones para la creación de la misma:



```
/*
 Multi-line
 non-Doxygen comments
*/
#include <iostream>

/** Single-line Doxygen comment */
class MyClass {
private:
  int value; /**< After member Doxygen comment */
public:
  /**
   * Multi-line Doxygen comment
   * @param int x new value
   * @return bool operation success
   */
  bool setValue(int x /**< [in] also Doxygen */) {
    this->value = x; /* non-Doxygen comment */
    std::cout << "I'm setting value " << x << std::endl;
    /* Single line non-Doxygen comment */
    return true;
  }
};
```

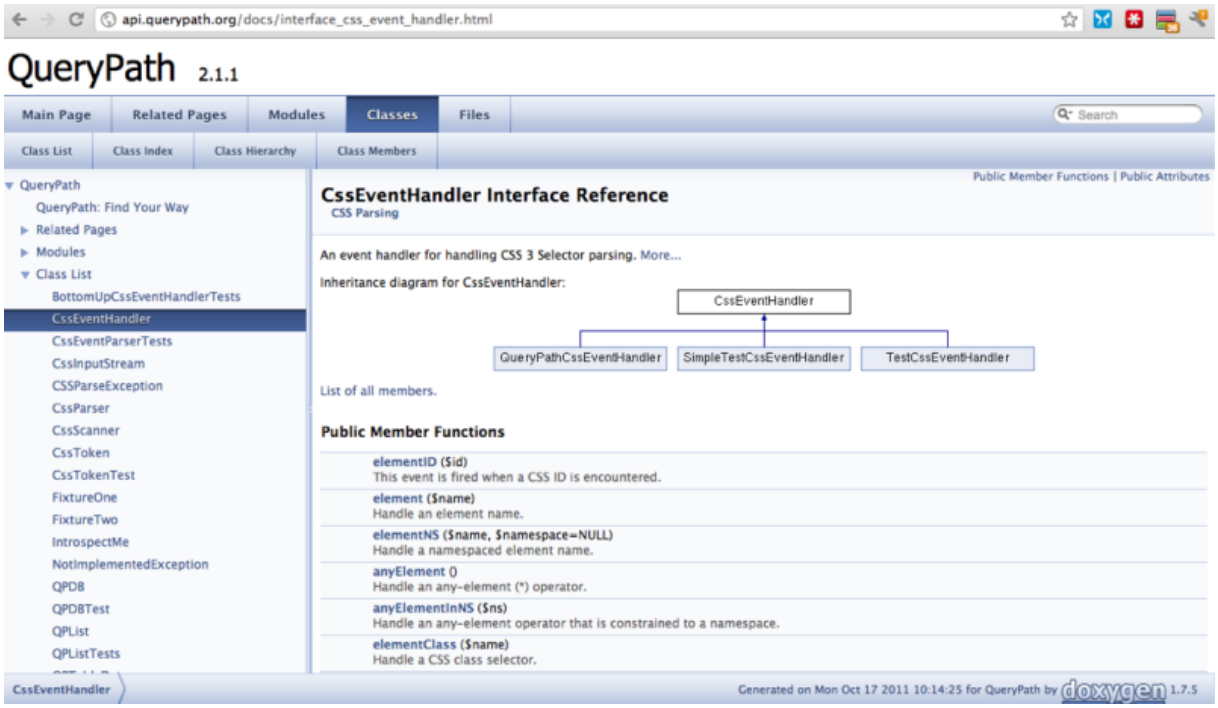
```
package com.st.java9;

import java.util.List;

/**
 * This is a demo class for javadoc
 *
 * @author Mithil
 *
 */
public class DocTest {

    public static void main(String[] args) {

    }

    /**
     * This is a demo method
     * @param a
     * @param b
     * @return 0 for now
     */
    public int methodA(String a, List<String> b) {
        return 0;
    }
}
```

```
/**
 * This module is a demo for documentation.
 * @author Mithil
 *
 */
module com.st.doc {
}
```

Figura 10. Ejemplos de anotaciones para documentación

Estos comentarios suelen ir estratégicamente situados antes de las definiciones de clases, de variables (o atributos en clases) o antes de las funciones existentes. En el caso de las funciones, se centran en describir su finalidad de forma resumida así como cuáles son los parámetros de entrada y que valor o valores devuelve la función.

Una vez documentado un cierto código se ejecuta una herramienta externa que tras analizar estos comentarios genera una documentación de tipo API, en formato HTML o imprimible (PDF)



V. RESUMEN

En este tema hemos visto cuáles son las principales herramientas que intervienen en la etapa de implementación. En función de la naturaleza del proyecto y del lenguaje de programación escogido se hará necesario un dominio efectivo de las mismas.

Mención especial requiere el Entorno de Desarrollo Integrado (IDE) que normalmente hará de interfaz para todas las demás herramientas (compiladores, depuradores, sistemas de control de versiones).

VI. GLOSARIO

Volvemos a repasar algunos de los principales conceptos de esta unidad:

Editor de Programación	Es un editor de texto que cuenta con capacidades muy relacionadas con la programación en un determinado lenguaje (resaltado de sintaxis, autocompletación de código, …)
Compilador	Herramienta de programación que a partir de un código fuente genera archivos binarios de código objeto para una determinada plataforma o archivos de código binario para una infraestructura de máquina virtual
Intérprete	Herramienta de programación que a analiza un código fuente y va ejecutando sus instrucciones dentro de una plataforma concreta de tipo sistema operativo o en el interior de otro programa
Depurador	Herramienta que permite detectar errores de tiempo de ejecución. Para ello ejecuta un cierto programa al cuál se le puede haber añadido información de depuración. Permite inspeccionar variables y pausar la ejecución en puntos del código.
Entorno de desarrollo Integrado	Herramienta gráfica que integra en una sola interfaz el acceso a diversas herramientas

Sistemas de control de versiones	Sistemas que ayudan en la gestión de cambios en el código fuente mediante el almacenamiento centralizado de los códigos y sus diferencias
Sistemas de documentación automática	Sistemas que generan documentación de tipo técnico sobre la arquitectura de funciones de un proyecto software