2.1 CARACTERÍSTICAS

Definición de entorno de desarrollo: "*Un entorno de desarrollo integrado o IDE (Integrated Development Environment) es un programa informático que tiene el objetivo de asistir al programador en la tarea de diseñar y codificar un software mediante la inclusión de múltiples herramientas destinadas para dicha tarea*".

Cada entorno de desarrollo (IDE a partir de ahora) tiene unas características y funcionalidades específicas que lo definen. No obstante, todos mantienen unos componentes comunes.

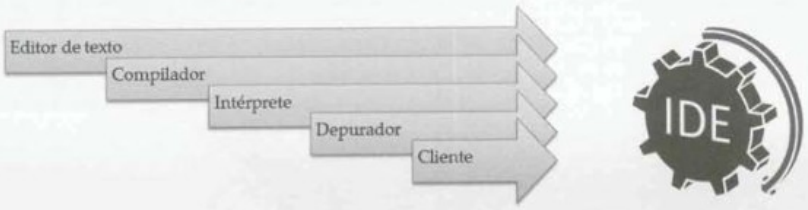

Figura 2.1. *Componentes básicos de un entorno de desarrollo*

2.1.1 EXTENSIONES Y HERRAMIENTAS

Dentro de un IDE tenemos a nuestra disposición un sinfín de herramientas con las que trabajar. Podríamos utilizar simplemente un editor de texto y un compilador (en caso de que se tratase de un lenguaje no interpretado), pero nos perderíamos muchas de las facilidades y herramientas que nos facilita el IDE.

Hoy en día, existen una gran cantidad de editores de texto que incluyen una de las funciones más triviales pero increíblemente útiles de un entorno de desarrollo: el coloreado de sintaxis. Se identifican las palabras reservadas y elementos clave del lenguaje coloreándolos para tener una mejor visión del código.

EJEMPLO 2.1a

CON COLOREADO DE SINTAXIS

```
double getVelocidad(){                                          (color azul)
// Hola! Soy un comentario!                                     (color verde)
  switch (_tipo){                                               (color azul)
    case EUROPEA:                                               (color azul)
          return getBVelocidadBase();
      case NORUEGO_AZUL:                                        (color azul)
          return (_esMoteado) ? 0 : getVelocidadBase(_voltaje);
   }
   throw new RuntimeException("Debería ser inalcanzable");      (colores azul y rojo)
}
```

EJEMPLO 2.1b

SIN COLOREADO DE SINTAXIS

```
double getVelocidad(){
// Hola! Soy un comentario!
  switch (_tipo){
    case EUROPEA:
           return getBVelocidadBase();
      case NORUEGO_AZUL:
           return (_esMoteado) ? 0 : getVelocidadBase(_voltaje);
   }
   throw new RuntimeException("Debería ser inalcanzable");
}
```

Como podemos apreciar, esta característica tan sencilla nos mejora la visión y el entendimiento del código con un simple vistazo. Obviamente, ésta no es la funcionalidad más importante que nos ofrece el IDE. Realmente no existe "la mejor herramienta del IDE", es una cuestión más subjetiva que depende del programador y de su forma de codificar un software.

Pese a ello, una de las características más importantes para cualquier programador es el autocompletado de código. Cada vez que comenzamos a escribir una palabra reservada, aparece un listado de sugerencias por el que podemos navegar para elegir la que queremos o seguir escribiendo mientras se filtra el listado. También resulta útil al mostrar los métodos y propiedades de un objeto o clase cuando escribimos el "." para acceder a ellos.

Figura 2.2. *Autocompletar / IntelliSense*

También nos puede crear las estructuras de clases o instrucciones de manera automática mediante *snippets*, por ejemplo, en Visual Studio si se escribe "for" y se presiona dos veces tabulador, se obtendrá la estructura de la instrucción *for*.

EJEMPLO 2.2

INSTRUCCIÓN FOR

```
for (int i = 0; i < length; i++)
{

}
```

Nos ofrecen herramientas de refactorización, una ejecución en depuración y muchas opciones más con el objetivo de facilitar y acortar el tiempo dedicado al desarrollo y codificación de software.

No todas las herramientas están integradas en el IDE, muchas de ellas las podemos incluir mediante extensiones que añaden, modifican y mejoran el IDE.

2.1.2 PERSONALIZACIÓN Y CONFIGURACIÓN

Los IDE son altamente configurables, ya que las necesidades de cada programador o grupo de trabajo pueden ser diferentes, el objetivo es ofrecer al usuario una aplicación amigable con la que trabajar, por lo que poder personalizar y configurar la herramienta es un aspecto verdaderamente importante.

La configuración del IDE permite entre otras cosas añadir y modificar las barras de herramientas, pudiendo crear comandos personalizados y atajos de teclado para cada una de ellas. Estableciendo el posicionamiento de las ventanas y barras conjuntamente con los atajos de teclado podremos mejorar sumamente nuestro rendimiento y aprovechar con mayor comodidad todas las funciones del IDE.

Las configuraciones de depuración y compilación de proyectos son una práctica recurrente en el desarrollo de aplicaciones, por lo que resulta extremadamente efectivo poder configurarlas al gusto, obteniendo mejores resultados al poder manejar las interrupciones de las excepciones o desenredar la pila de llamadas.

2.2 CRITERIOS DE ELECCIÓN DE UN IDE

Ya sabemos qué es un entorno de desarrollo integrado y qué particularidades y funcionalidades nos puede llegar a ofrecer, ahora solo necesitamos saber cuál elegir. Para poder elegir correctamente el IDE con el que vamos a trabajar, necesitamos saber las características que buscamos en él, si satisface nuestras necesidades y si nosotros mismos cumplimos los requisitos del IDE más allá de los requerimientos técnicos de hardware.

2.2.1 SISTEMA OPERATIVO

Sin lugar a dudas, uno de los criterios más restrictivos a la hora de seleccionar nuestro entorno de trabajo es saber en qué sistema operativo vamos a trabajar y, más importante aún, para qué sistema operativo vamos a desarrollar nuestro software.

Aunque hoy en día con los equipos modernos no es muy complicado virtualizar o emular el sistema operativo o el IDE que queremos usar, siempre es aconsejable no utilizar esos métodos si no es como último recurso. Siempre y cuando nuestro software no vaya a ser ejecutado mediante una máquina virtual, estaremos desarrollando para un sistema operativo concreto, por lo que si estamos desarrollando aplicaciones para Linux, resultaría bastante inusual desarrollar la aplicación en Windows para ello. Esto realmente no se debe a una restricción inherente al IDE, sino al compilador que tiene el IDE integrado. Si recordamos el proceso de obtención de código ejecutable del capítulo anterior, veremos que el compilador se encarga de traducir nuestro código fuente en código objeto, que es el que ejecutará el sistema. Este problema es fácilmente salvable compilando nuestro código fuente en un compilador de otro sistema operativo (siempre y cuando exista), por lo que dependiendo de nuestras necesidades pudiera ser un problema menor.

2.2.2 LENGUAJE DE PROGRAMACIÓN Y FRAMEWORK

Como comentamos anteriormente, un IDE puede soportar uno o varios lenguajes de programación, por lo que saber en qué lenguaje de programación vamos a codificar nuestro software y qué lenguajes nos ofrecen los distintos IDE es una información valiosa que hay que tener en cuenta.

Este criterio va de la mano con el sistema operativo, ya que si quisiéramos desarrollar en Visual Basic bajo un sistema operativo Linux no sería Visual Studio nuestra opción, sino que tendríamos que utilizar Gambas.

Lo mismo ocurre con las plataformas de trabajo, también llamadas *framework*, no solo depende de la plataforma de trabajo sobre la que vayamos a trabajar, también necesitamos saber bajo qué sistema operativo vamos a desarrollarla o ejecutarla. Siguiendo con el ejemplo anterior, si fuésemos a desarrollar con Visual Basic con la plataforma .NET bajo Linux, tendríamos que usar Mono Develop en lugar de Visual Studio.

2.2.3 HERRAMIENTAS Y DISPONIBILIDAD

Las diferentes herramientas de las que disponen los IDE son el último criterio de selección, seguramente nos encontremos con varios IDE que cumplen los requisitos de lenguaje y sistema operativo, pero no todos tienen las mismas funciones, por lo que saber cuáles son esas herramientas es un dato sumamente importante en nuestra decisión.

En ocasiones pueden ser restrictivos ya no solo por tus propias preferencias, sino por trabajar de manera colaborativa y el modo de utilizar e interpretar diferentes códigos entre diferentes IDE; por ejemplo, si nuestros compañeros de trabajo están utilizando el Team Server Foundation (TFS) como sistema de control de versiones, nosotros deberemos usarlo también, y el único modo de hacerlo es mediante Visual Studio; por otro lado, si nuestros compañeros están trabajando en Java con un sistema de control de versiones Subversion (SVN), podríamos usar indistintamente Netbeans, Eclipse o InteliJ DEA entre otros, ya que SVN está disponible para todos los IDE.

Fuera del marco de trabajo colaborativo, también tendremos nuestras preferencias y necesidades, por lo que, si necesitamos tener una funcionalidad para crear archivos de ayuda y documentación, deberíamos buscar un IDE que tuviese dicha funcionalidad o, en su defecto, que hubiese una extensión o *plugin* para ese IDE que aporte la funcionalidad deseada.

Podríamos también ir más allá de las meras funcionalidades añadidas e irnos a un ámbito mucho más centrado en el propio software, como podría ser la interfaz de usuario, los IDE pueden incluir sus propios y específicos controles que mejoran la interfaz y aportan mayor funcionalidad y usabilidad a nuestros formularios y aplicaciones. También podríamos ver este criterio desde un punto de vista más destinado a la codificación y pensar en qué refactorizaciones automáticas nos podemos encontrar entre los IDE a la hora de elegirlo.

El mayor problema que nos podemos encontrar no es ya elegir incorrectamente, sino no saber qué funcionalidades podría estar otorgando un IDE u otro, ya que podríamos no conocer todas las funcionalidades que puede llegar a ofrecer y, por tanto, deberemos invertir una gran cantidad de tiempo investigando y documentándonos a fondo sobre los IDE potenciales que podemos usar o invertir ese tiempo en probarlos de manera empírica.

A simple vista, parecen demasiados factores a tener en cuenta y una operación larga, tediosa y complicada, no debemos agobiarnos por esta cuestión, ya que, al igual que con los lenguajes de programación, no podemos pretender empezar sabiendo todas las posibilidades que nos ofrecen y tenemos que ir aprendiéndolas de modo incremental en función de nuestra experiencia.

Para que se vea de un modo más claro, observemos una comparativa sencilla sobre los tres IDE más populares para Java.

Figura 2.3. *Comparación IDE Java*

Aún nos quedaría además tratar el asunto de la disponibilidad, el cual, una vez comprobados todos los criterios de selección mencionados, puede desbaratar nuestra toma de decisiones. El aspecto más restrictivo de la disponibilidad reside en el precio de la aplicación, dependiendo de nuestro presupuesto podremos acceder a diferentes IDE.

Aunque nuestro presupuesto sea escaso, existen soluciones muy económicas e incluso gratuitas, por lo que, aunque no podamos tener el IDE que queremos, tendremos alternativas más que suficientes a nuestra disposición que cumplirán sin duda al menos la mayoría de los criterios que necesitamos.

2.3 USO BÁSICO DE UN IDE

Los entornos de desarrollo integrado son por definición una herramienta de desarrollo de software, por lo que la respuesta a la pregunta de cuál sería su uso o funcionalidad básica resulta bastante obvia: desarrollar software. No obstante, la tarea de un IDE no se queda ahí, ya que nos permite realizar una infinidad de operaciones que no podríamos realizar de otro modo. Además, si la tarea de un IDE solo fuese desarrollar aplicaciones, entonces un editor de texto y un compilador harían la misma función. La particularidad, como hemos visto a lo largo del capítulo, de un IDE reside en la cantidad y calidad de las funcionalidades añadidas que ofrece al desarrollador, por medio de herramientas integradas o externas.

Muchas de las herramientas más habituales que solemos usar conjuntamente con los IDE también se encuentran disponibles fuera de ellos, como podría ser el caso de una aplicación para crear modelados y diagramas o aplicaciones para automatizar las pruebas unitarias. En esencia, todas esas aplicaciones pueden o podrían estar disponibles sin necesidad de un entorno de desarrollo, o mejor dicho, sin que fuese un entorno de desarrollo integrado, ya que si utilizamos un editor de texto, un compilador, un programa de refactorización, un cliente de trabajo colaborativo con acceso a repositorios y un analizador de código, entonces todas esas aplicaciones que utilizamos por separado serían en concepto un entorno de desarrollo, aunque no estuviesen empaquetadas en una misma aplicación.

2.3.1 EDICIÓN DE PROGRAMAS Y GENERACIÓN DE EJECUTABLES

La necesidad básica que todo IDE debe cubrir es la creación o edición de programas y convertir ese código fuente en código ejecutable. Inicialmente, sin un IDE la operación tampoco sería tan complicada, tal y como hemos visto en el capítulo 1, compilaríamos nuestro código fuente y utilizaríamos un enlazador para combinar ese código objeto con nuestras librerías, obteniendo como resultado nuestro programa ejecutable. Un IDE realiza esa operación de manera conjunta y compacta. Gracias a un gestor de proyectos podemos ajustar las dependencias de cada sección de nuestro programa y todas las necesidades y opciones de compilación que queramos.

Los IDE, además, suelen ofrecer una funcionalidad añadida, ya que permiten ejecutar de manera virtual el programa que se está codificando en cualquier momento (siempre y cuando no se produzcan errores durante la compilación), de ese modo permiten comprobar la funcionalidad del programa sin tener que crear una publicación o despliegue de la aplicación para probar cualquier cambio o añadido.

2.3.2 DESARROLLO COLABORATIVO

El desarrollo colaborativo hace referencia al proceso de desarrollo de un software de manera descentralizada y distribuida, donde los desarrolladores no necesitan conocerse, tener el mismo (o algún) jefe, ni hablar el mismo idioma, pero trabajan en el mismo proyecto. Este modelo de desarrollo es muy habitual en proyectos de software libre, aunque no es restrictivo, cualquier empresa o grupo de desarrollo puede utilizar el desarrollo colaborativo o las herramientas que se utilizan en él. Para poder llevar un control del código realizado desde tantas y diversas fuentes, se utilizan los controles de versiones.

Los programas de control de versiones son aplicaciones que constan de servidor y cliente, donde en la parte del servidor se crean repositorios para que los clientes puedan descargar y subir código. Son herramientas asíncronas que permiten controlar y gestionar las fuentes y versiones del código del repositorio.

Podemos observar el funcionamiento y uso de un control de versiones con el siguiente diagrama:

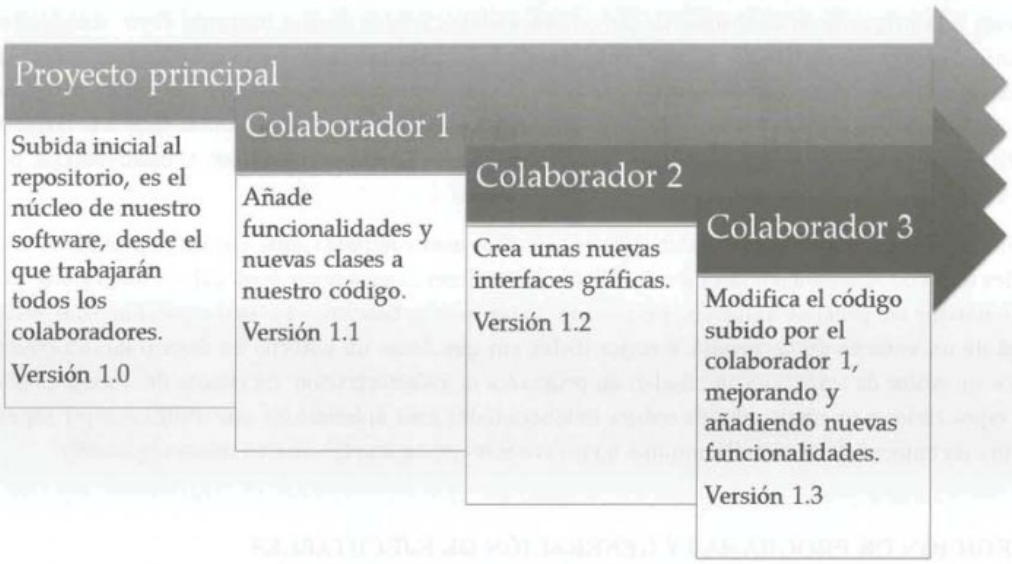

Figura 2.4. *Desarrollo colaborativo*

Como vemos en el diagrama, el proceso es continuo y acumulativo, es decir, el proyecto parte del mismo núcleo y se va modificando con el paso del tiempo, los siguientes desarrolladores no trabajan sobre el proyecto principal inicial, sino sobre el trabajo realizado por los anteriores colaboradores, es decir, el colaborador 2 trabaja sobre el código resultante de la intervención del colaborador 1, y así consecutivamente.

Podemos además elegir sobre qué versión trabajar, es decir, qué versión descargarnos en nuestro PC para trabajar sobre ella; por ejemplo, el colaborador 3 podría haber elegido trabajar sobre la versión 1.2 en vez de sobre la 1.1.

Todas estas operaciones se pueden realizar desde aplicaciones cliente externas o integradas, pero es en las herramientas integradas donde realmente está la magia, ya que, gracias a las herramientas del IDE, podemos hacer un uso mucho más rápido y avanzado de los controles de versiones, pudiendo elegir qué archivos actualizar en cualquier lado de nuestra conexión (servidor o cliente), omitir cambios para no pisar nuestro trabajo con el de otros, y viceversa, y una gran cantidad de operaciones de la misma índole. Un uso apropiado y sincronizado de los IDE y el control de versiones permiten trabajar de manera paralela sobre el mismo proyecto sin entorpecerse en el trabajo.

La sincronización de un IDE con el repositorio del proyecto permite además saber qué archivos han cambiado y por ende tener un control de versiones más allá del servidor, tenerlo directamente sobre el código que estamos actualmente desarrollando, viendo qué archivos han cambiado, borrado o incluido desde la última actualización al repositorio.

Figura 2.5. *Funcionamiento básico de un control de versiones*

Es muy habitual que en los centros de trabajo se utilice el control de versiones como una ayuda directa al desarrollador del proyecto en cuestión. Aunque no sea el propósito inicial de la herramienta, no es tampoco una idea descabellada, nos permite tener un control de versiones a modo de copia de seguridad selectiva. Cualquier error o fallo garrafal en una aplicación nos podría permitir volver a una posición anterior donde la aplicación era perfectamente estable.

En definitiva, la integración del control de versiones en un IDE nos ofrece una multitud de opciones tanto para el desarrollo diario como para realizar trabajos y proyectos de manera colaborativa, una opción única e indispensable a la que todo buen desarrollador debería prestar atención.

2.4 NUESTRA ELECCIÓN VISUAL STUDIO

Siguiendo los criterios que hemos aprendido en este capítulo, hemos elegido utilizar el Visual Studio Ultimate 2010 como entorno de desarrollo. Visual Studio es uno de los entornos de desarrollo más pulidos, profesionales y completos que podemos encontrar en el mercado, además, tiene a sus espaldas una excelente plataforma de trabajo: el *framework* .NET.

Hemos escogido el Visual Studio por ser un entorno de desarrollo muy completo con una gran cantidad de características integradas y la solidez del programa en sí mismo. Las herramientas de las que hablaremos durante el resto del libro serán específicas para Visual Studio, aunque se realizan las menciones que se consideren oportunas a herramientas de funcionalidad semejante en otros entornos de desarrollo.

El código que utilizaremos para los ejemplos será principalmente de C#, aunque será recurrente utilizar código en Visual Basic o Java para los ejemplos. También se incluirá código y uso de características propias y específicas de la plataforma .NET, obteniendo con ello una visión más global que si nos centrásemos de manera exclusiva en un único entorno de desarrollo y en un único lenguaje de programación.

2.4.1 INSTALACIÓN

Una vez que tenemos el producto en nuestras manos, procederemos a instalarlo, la instalación es igual de sencilla que la de cualquier otro programa.

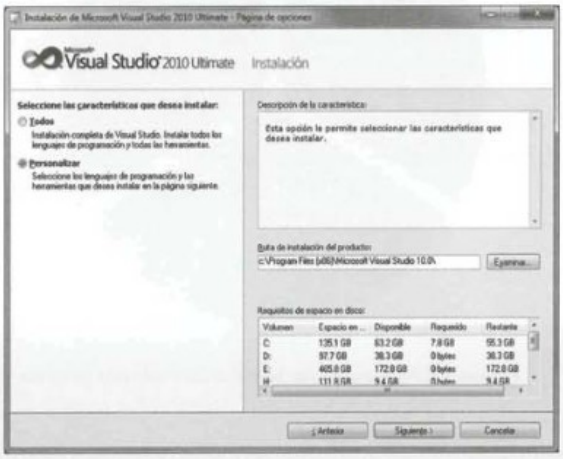

Figura 2.6. *Selección de ubicación y tipo de instalación VS2010 Ultimate*

Si queremos personalizar las características que instalaremos con Visual Studio, así como los lenguajes que vamos a utilizar, elegiremos la instalación personalizada. Y seleccionaremos las opciones deseadas.

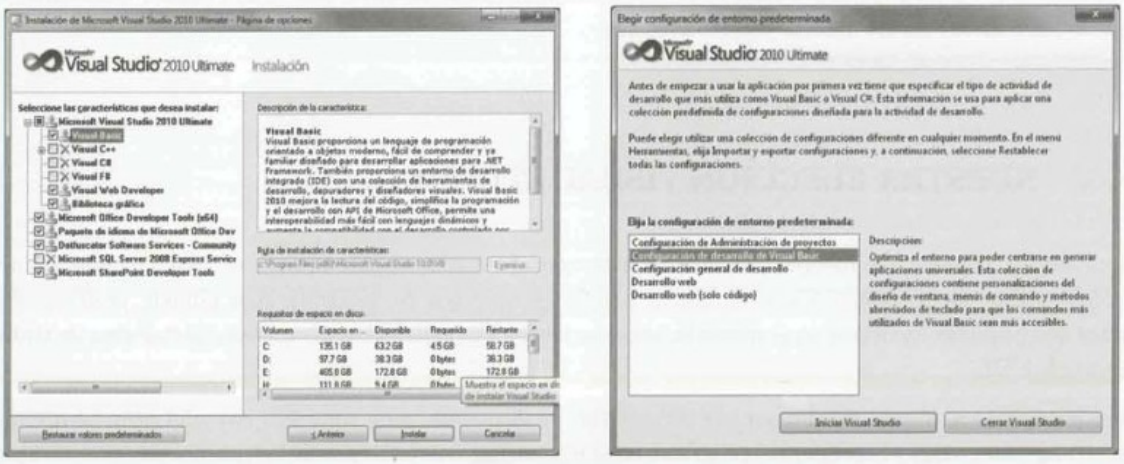

Figura 2.7. *Selección de características y configuración de entorno predeterminada*

Una vez terminado el proceso de instalación, pasamos a la primera ejecución del IDE. Nos aparecerá un cuadro de diálogo para escoger nuestra configuración de entorno. Visual Studio 2010 Ultimate tiene unas configuraciones predeterminadas dependiendo del uso que se le vaya a dar, o dependiendo del lenguaje de programación que se vaya a utilizar.

2.4.2 RECORRIDO POR LAS VENTANAS Y PALETAS PRINCIPALES

Como todo IDE que se precie, una vez que lo tengamos arrancado en nuestro equipo, visualizaremos una serie de ventanas acopladas y diversas barras de herramientas. Realizaremos un breve paseo general por las ventanas para familiarizarnos con el entorno.

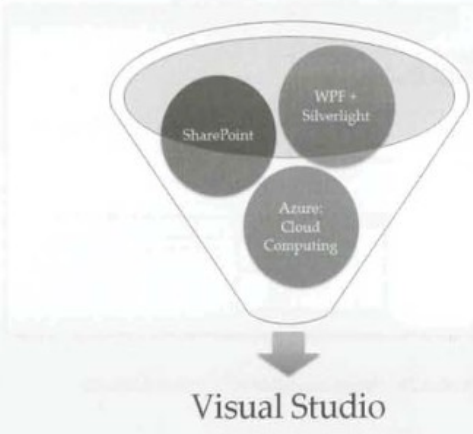

Figura 2.8. *Tecnologías y frameworks incluidas en Visual Studio 2010*

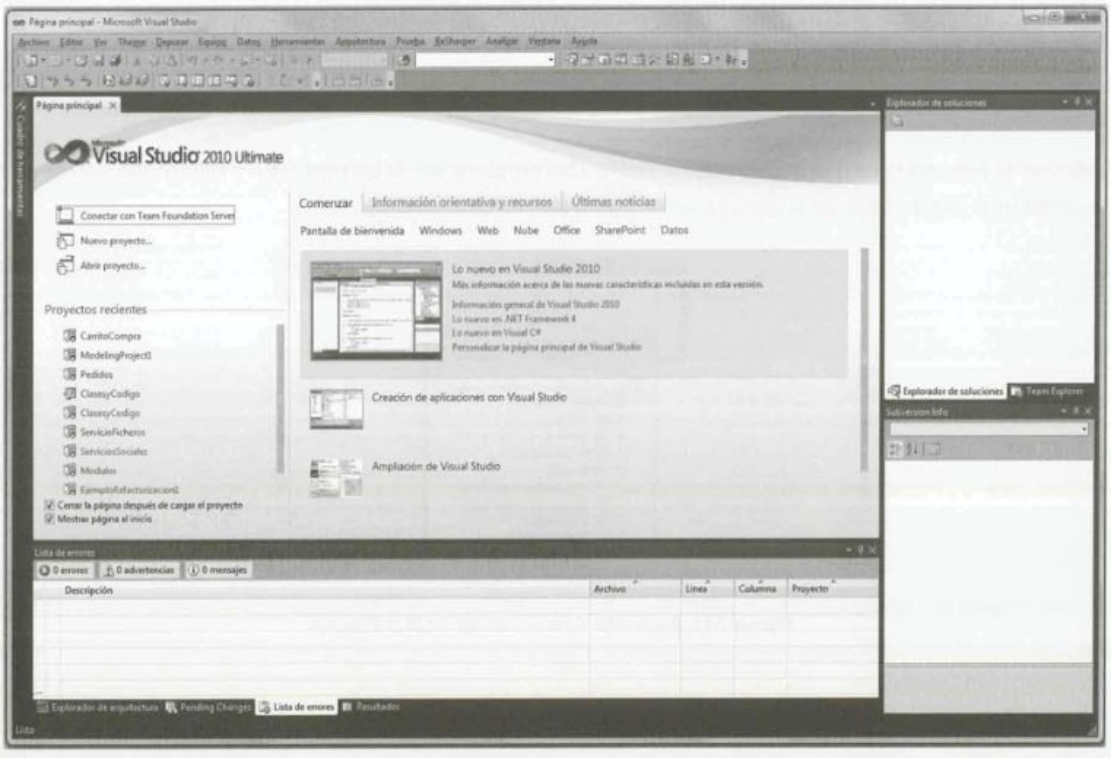

Figura 2.9. *Vista general del VS2010 Ultimate*

Página principal personalizada

Disponemos en primera plana de una página principal con información de actividad reciente que se puede personalizar para mostrar últimas noticias, tutoriales y guías, o simplemente información adicional sobre el IDE.

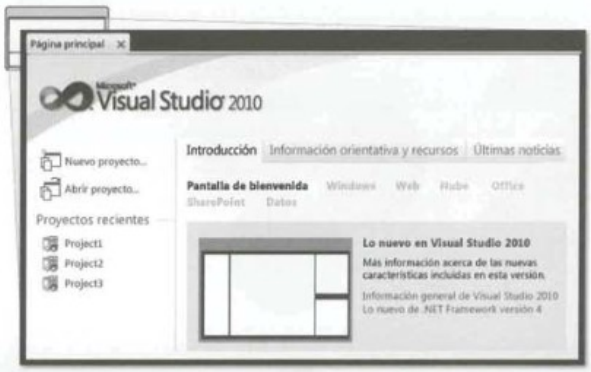

Figura 2.10. *Página principal del VS2010 Ultimate*

Explorador de soluciones

En la parte superior derecha del IDE tenemos el explorador de soluciones, en el que podemos ver la jerarquía de carpetas y proyectos de nuestras aplicaciones.

Nos permite además ver las referencias, conexiones de datos y dependencias de los diferentes proyectos, pudiendo establecer propiedades adicionales a las mismas.

Es dentro de esta ventana donde podremos acceder a las propiedades de los proyectos y soluciones, así como añadir nuevos elementos dentro de la jerarquía, ya sean clases, proyectos o ficheros varios.

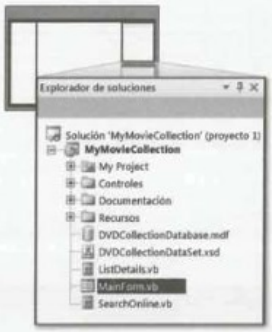

Figura 2.11. *Explorador de soluciones del VS2010 Ultimate*

Editor de diseño

Nuestras aplicaciones más básicas se basarán en el uso de la información, utilizando para ello distintos formularios, ya sean una aplicación de escritorio o una aplicación web.

Figura 2.12. *Editor de diseño del VS2010 Ultimate*

Con el editor de diseño, podremos crear y posicionar los controles que nuestra interfaz necesita.

Editor de código

Como es lógico, las clases gráficas o formularios también necesitarán de una lógica que esté detrás de su comportamiento, para ello, podemos utilizar la vista de código del editor de formularios.

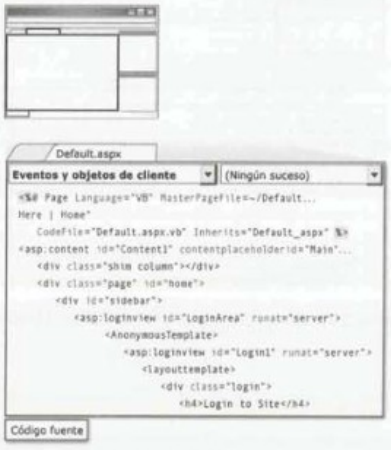

Figura 2.13. *Editor de código del VS2010 Ultimate*

Mediante los campos de selección situados en la parte superior del editor de código, podemos movernos por los diferentes controles y eventos para acceder a ellos de una manera más rápida y directa, facilitando y acortando el trabajo de codificación.

Editor de vista compartida

En ocasiones nos será de suma utilidad poder visualizar tanto el código como la interfaz de usuario relacionada, para ello, tenemos la vista compartida, que nos permite partir la pantalla de forma horizontal o vertical y modificar la posición de la línea divisoria para que se ajuste a nuestras necesidades.

Figura 2.14. Vista compartida en VS2010 Ultimate

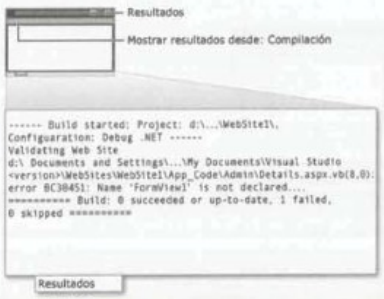

Figura 2.15. Consola compartida del VS2010 Ultimate

Consola de compilación

Además de la consola de pantalla que incluyen todos los entornos de desarrollo, también disponemos de una consola de compilación donde revisar los procesos y errores ocurridos durante dicho proceso.

Ventanas de depuración

Durante el proceso de depuración, podremos establecer puntos de observación e inspecciones para facilitar y mejorar la experiencia de depuración. Para ello, el Visual Studio nos ofrece una serie de pantallas específicas, visibles durante la depuración del programa con el objetivo de ofrecer toda la información necesaria de la manera más compacta mientras dura la ejecución de depuración.

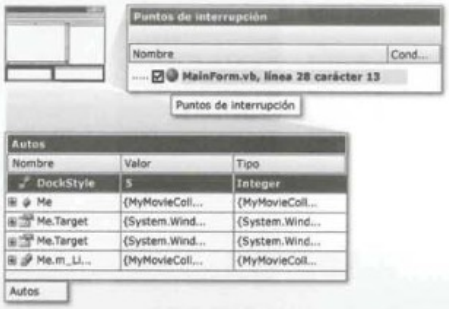

Figura 2.16. *Ventanas de depuración del VS2010 Ultimate*

2.4.3 PERSONALIZACIÓN Y CONFIGURACIÓN

Todo entorno de desarrollo debe permitir personalizar y añadir controles, y Visual Studio no es una excepción. La personalización del entorno de desarrollo es algo que no se debe infravalorar, recordemos que el objetivo principal de un entorno de desarrollo es facilitar al programador su tarea, por lo que poder redimensionar, reposicionar y modificar los elementos del entorno de desarrollo es una parte muy importante en la funcionalidad de un IDE.

Obviamente, la personalización no acaba en una configuración gráfica, sino que va más allá, estableciendo valores por defecto, y configuraciones específicas del comportamiento del IDE para ajustarlo a nuestras necesidades.

Ventanas

En Visual Studio Ultimate 2010, podemos añadir, quitar, acoplar y mover las ventanas de la forma que queramos de una manera muy sencilla. Como hemos visto antes, la configuración de ventanas inicial se compone de 4 partes: central, inferior, lateral izquierdo y lateral derecho. Un dato importante que hay que tener en cuenta es que tenemos dos tipos de ventanas: ventanas de documentos y ventanas de herramientas. Las ventanas de documentos siempre se encuentran en el bloque central del entorno y las de herramientas se encuentran en cualquier posición alrededor de ellas, por ello, vamos a diferenciar la personalización de posicionamiento y tamaño según el tipo de ventana.

Ventanas: Documentos

Las ventanas de documentos siempre se posicionan en el bloque central del entorno, aunque se pueden hacer flotantes arrastrándolas hacia cualquier punto de la pantalla. Se pueden colocar como una ventana independiente, como una parte de la organización de fichas (como si fuesen pestañas) o como parte de un grupo dividido de ventanas o grupo dividido de fichas. El modo más fácil y sencillo para lograrlo es arrastrando la ventana, una vez que la ventana

está siendo arrastrada, aparece un menú a modo de pequeñas imágenes para escoger la posición en la que queremos colocar la ventana.

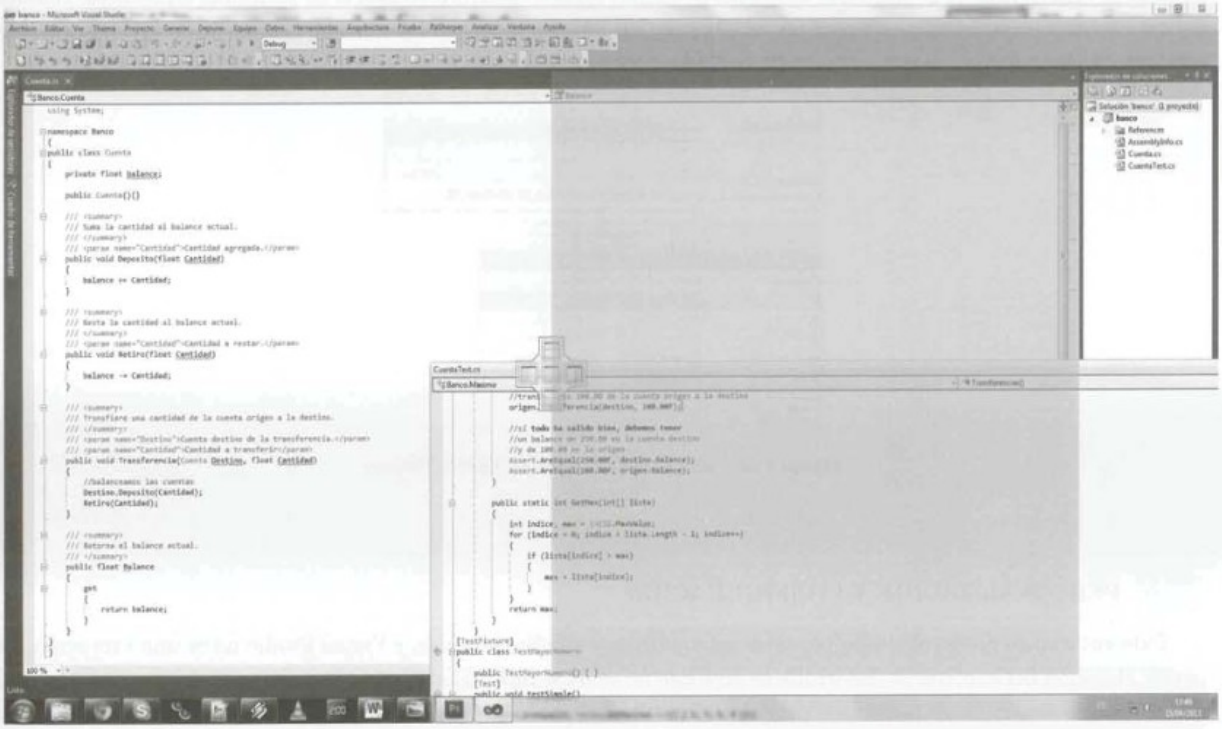

Figura 2.17. *Redistribución de ventanas en VS2010 Ultimate*

Ventanas: Herramientas

Las ventanas de herramientas tienen una mayor versatilidad de configuración, pueden colocarse en cualquier bloque o subbloque del entorno de desarrollo, incluyendo un bloque que viene desocupado por defecto, el superior, o en el bloque central.

Cualquier ventana de herramientas puede posicionarse en cualquier posición posible, incluyendo las pestañas o subbloques de los bloques de ventanas. Mediante el menú contextual del botón derecho delratón (o a través de la flecha en la parte superior derecha de la ventana) podemos elegir que esté acoplada, acoplada como ficha (como las fichas del bloque central) o que se oculte automáticamente, ocupando una porción ínfima y desplegándose al pasar el ratón por encima.

Por último, si queremos añadir una ventana al entorno de desarrollo, no tenemos más que ir al menú **Ver > Otras Ventanas** y escoger la que queramos.

Barras de herramientas

Como es habitual en muchas aplicaciones de escritorio, tenemos diversas barras de herramientas a nuestra disposición en los entornos de desarrollo. Se pueden añadir, quitar, mover y modificar.

El método más rápido para añadir barras de herramientas predefinidas es elegirla mediante el menú contextual que aparece al hacer clic con el botón derecho en la zona prefijada para las barras de herramientas, el bloque inmediatamente inferior al menú.

Para personalizar o crear nuevas barras de herramientas, utilizaremos la opción del menú **Herramientas > Personalizar**, y dentro de la ventana **Personalizar**, en **Comandos**. Dentro de esa ventana podremos modificar, reorganizar y añadir o eliminar elementos.

Opciones del entorno

Vamos a centrarnos más en la configuración del entorno de desarrollo que en la personalización de la interfaz del entorno. Para ello, nos vamos a ir a **Herramientas > Opciones** para modificar lo que necesitemos.

En el cuadro de diálogo que nos aparecerá, seleccionamos la opción **Mostrar todas las configuraciones**, que aparece en la parte inferior izquierda de la ventana.

Ahora, ya podemos ver todas las opciones básicas que nos ofrece el entorno de desarrollo, tendremos que tener cuidado con las modificaciones que hagamos, ya que muchas de ellas cambian sensiblemente la funcionalidad del entorno.

Dentro del apartado entorno, podemos ver una de las características de los entornos de desarrollo que resaltábamos al principio de este capítulo: el coloreado de código. Dentro de esta sección podremos elegir la fuente y especificar uno a uno cómo queremos que nos coloree los diferentes fragmentos de código para visualizarlos de la manera que nos parezca más clara o estemos más acostumbrados.

En el apartado de depuración tenemos una de las opciones más útiles: **Editar y continuar**, si lo activamos, podremos modificar el código en tiempo de ejecución mientras estamos depurando, lo cual nos permite resolver o hacer pequeños cambios para pruebas de una manera mucho más rápida que tener que parar la depuración, realizar la modificación y volver a depurar, sobre todo si nuestro proyecto es grande y tarda algunos minutos en compilar.

Dependiendo de la naturaleza y sistema objetivo de la aplicación, las configuraciones de rendimiento pueden ser claves para observar problemas de rendimiento al poder visualizar el tiempo en función de los ciclos de proceso y no de los milisegundos que ha tardado en realizarlo; los milisegundos dependen de la frecuencia y velocidad de proceso, los ciclos son más objetivos y, lo más importante, constantes, tal y como vimos en el tema 1, una operación siempre tarda los mismos ciclos (microinstrucciones) independientemente del procesador que lo ejecute.

Como podéis observar en un vista preliminar, hay una gran cantidad de opciones ofrecidas por el IDE para su configuración y personalización, siempre es útil invertir algo de tiempo en leer todas las opciones para comprobar si alguna opción nos resultaría útil modificarla, añadirla o incluso retirarla.

En caso de duda, se recomienda documentarse al respecto en la MSDN de Microsoft, la cual es muy completa y viene avalada por una gran cantidad de ejemplos e ilustraciones que facilitan la comprensión y ayudan a entender mejor las opciones y necesidades que podemos suplir con la configuración del IDE.

Aún no hemos añadido extensiones ni herramientas adicionales al entorno de desarrollo, por lo que todas ellas hacen referencia a las funcionalidades internas del IDE. Dependiendo de las extensiones instaladas, podremos encontrar opciones de configuración en esta misma ventana de diálogo, relacionadas con la extensión.

Opciones del proyecto

Las diferentes opciones y configuraciones que nos ofrece el entorno de desarrollo no se aplican solamente al entorno en sí mismo, sino a los proyectos que podemos crear.

Dependiendo del tipo de proyecto, también nos podremos encontrar con diferentes opciones dentro de las propiedades del proyecto, por ejemplo en un proyecto de aplicación web, tenemos las configuraciones del servidor de desarrollo o la ruta del directorio virtual del IIS, opciones inexistentes en una aplicación de escritorio.

Para acceder a las propiedades del proyecto, deberemos hacer clic con el botón derecho en el proyecto y seleccionar **Propiedades** en el menú contextual emergente que nos aparece. Una vez en dicha pantalla (por defecto se coloca como una ficha en nuestra ventana de documentos), tenemos un menú con todas las categorías de opciones disponibles en la parte izquierda.

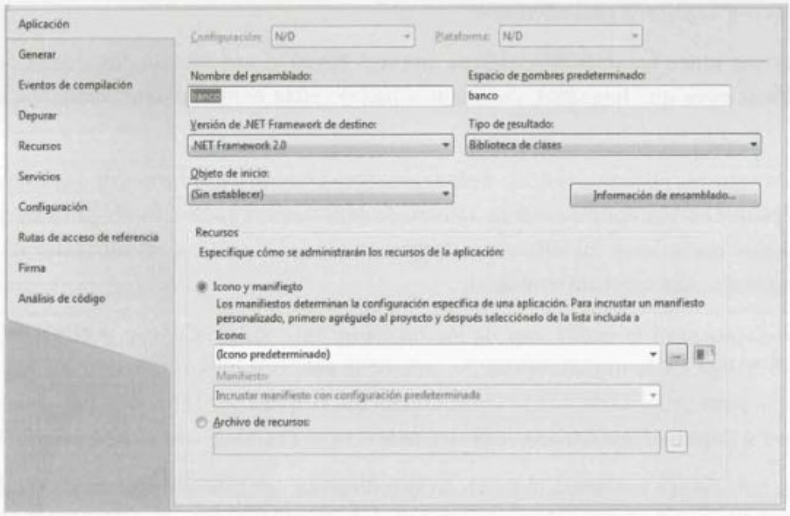

Figura 2.18. *Opciones de proyectos en VS2010 Ultimate*

Dentro de la sección Aplicación, nos encontramos con varias opciones interesantes, una de ellas de suma importancia, la que define cuál será el objeto de inicio, pudiendo usar para ello un formulario, o un método "Sub Main" como primer bloque de código que hay que ejecutar al arrancar la aplicación.

Habilitar marco de trabajo de la aplicación también resulta una opción importante que hay que tener en cuenta, nos permite, entre otras cosas, establecer que la aplicación sea de instancia única (no puedes tener dos procesos de la aplicación abiertos) o utilizar el método de identificación de Windows para registrar ese usuario en la aplicación. Si tu aplicación tendrá una autentificación de usuarios mediante bases de datos, Windows o directorio activo (LDAP) es una opción importante para elegir la configuración adecuada y evitar así problemas derivados de una mala propiedad de proyecto.

Como es obvio, las opciones del proyecto no solo dependen del tipo de proyecto, sino del lenguaje utilizado en el proyecto, si estuviésemos con una aplicación de escritorio Windows Forms con Visual Basic, veríamos unas opciones dentro de la sección Compilar que no veríamos con un proyecto de C#, como por ejemplo las opciones de compilación *optión explicit*, *option strict*, *option compare* u *option infer*.

En la sección Publicar podremos especificar la versión de publicación y la posibilidad de que se incremente automáticamente cada vez que publiquemos la aplicación. Dentro de esta sección, podemos también realizar la acción de publicar nuestro proyecto directamente.

El análisis de código establece las reglas para avisar de posibles errores durante el codificado y compilación del proyecto. Las reglas se pueden especificar dentro del propio entorno de desarrollo, para todos los proyectos, y no tener que configurarlo proyecto por proyecto; no obstante, en ocasiones podremos querer crear o modificar reglas para un proyecto concreto debido a sus características.

RESUMEN DEL CAPÍTULO

Una vez establecida una base en el mundo del desarrollo del software, ya hemos entrado en la materia que nos interesa: los entornos de desarrollo.

En este tema hemos aprendido las características que debe tener un IDE y en cuáles nos tenemos que fijar a la hora de elegir nuestro IDE. Hemos realizado un recorrido global por las diferentes funcionalidades de los IDE, consiguiendo con ello saber qué debemos esperar de un IDE, sean cuales sean sus propósitos.

El alumno debería ser capaz de evaluar y conocer sus necesidades, esto le permitirá escoger el IDE apropiado mediante un breve período de evaluación y documentación. El alumno, además, ha tenido una primera toma de contacto con el entorno de desarrollo Visual Studio Ultimate 2010, aprendiendo cómo instalarlo y configurarlo, así como a familiarizarse con los diferentes bloques de herramientas necesarias para su utilización.

Este capítulo introductorio a los IDE y Visual Studio carece de actividades, ya que el propio contenido del capítulo establece una larga actividad en el apartado 2.4. Por ello, se recomienda estudiar dicho apartado teniendo delante el entorno de desarrollo, esto permitirá una mayor comprensión de dicho bloque temático.

En este capítulo, hemos escogido el entorno de desarrollo sobre el cual se va a centrar este libro, por lo que las herramientas y uso que se explicarán de aquí en adelante serán específicos para Visual Studio 2010 Ultimate.