INSTITUTO de ENSEÑANZAS a DISTANCIA de ANDALUCÍA

2º de Bachillerato

Tecnologías de la Información y Comunicación

Contenidos

Ciclo de vida:
Depuración. Herramientas CASE

Imagen en Flickr de Kyle McDonald con Algunos derechos reservados

A estas alturas ya tienes una idea bastante aproximada del proceso de creación y desarrollo del software, es decir, de su ciclo de vida. Ya va tomando forma en tu mente y puedes apreciar todo el proceso en su conjunto, o casi todo... En este tema verás dos aspectos más del proceso: la depuración y el uso de herramienta CASE.

Ya has podido observar, y comprobar con las prácticas, que **cuando se desarrolla software, a medida que se va probando, nos vamos encontrando errores**, bien de sintaxis o de funcionamiento. Los errores de sintaxis, como ya sabes, son los relativos a la forma en la que se escriben las distintas instrucciones que componen el programa, y la solución a los mismos no es otra que la de comprobar meticulosamente cómo, el lenguaje utilizado, nos especifica la manera de expresar cada orden. Una vez corregidos todos los errores de sintaxis nos encontraremos los errores de funcionamiento, que pueden ocasionar la interrupción de la ejecución del programa o no. Las causas de estos errores son más difíciles de detectar, sobre todo cuando no provocan la interrupción de la ejecución. **Aquí entra en juego la depuración**, que hará más leve el trabajo de detección de los mismos al programador. En los siguientes apartados del tema verás en qué consiste, las posibilidades que aporta y cómo se utiliza.

En la segunda parte del tema, **estudiarás un tipo de software que te permitirá abordar prácticamente todas las fases del ciclo de vida del software. Nos referimos a las herramientas CASE**. Podrás apreciar cómo trabajan estas herramientas para conseguir que nuestro software vaya avanzando en su desarrollo, pasando por todas las fases de su ciclo de vida. En algunas de las fases, estas herramientas pueden aportar incluso la automatización de ciertos procesos, lo que permite a los analistas y programadores aliviar considerablemente el trabajo a realizar.

1. ¿Qué conseguirás?

Al terminar de estudiar este tema habrás conseguido reforzar dos aspectos de tus conocimientos como programador:

1. **Tu capacidad para detectar y resolver problemas de forma satisfactoria habrá aumentado gracias a la ayuda del depurador** y sus opciones. Efectivamente, notarás que la dificultad que tenías para resolver problemas en tus programas habrá disminuido considerablemente, no solo por conocer una buena herramienta como es el depurador y sus opciones disponibles, sino también porque con la ejecución pausada que te ofrece el mismo comprenderás mucho mejor el funcionamiento de las instrucciones del lenguaje y por tanto del flujo de ejecución de tu código fuente.
2. Tendrás una visión más cercana y real del trabajo que suponen las fases del ciclo de vida del software, y del **ahorro de tiempo y esfuerzo que aportan las Herramientas CASE** para tales menesteres.

Imagen en Flickr de Ruiwen Chua con Algunos derechos reservados

Para estudiar la depuración aprovecharás una herramienta que ya te es conocida y que has usado en temas anteriores, nos referimos a **PSeInt**, que entre otras opciones dispone de un depurador con las características típicas de otros muchos y que te será de gran ayuda para comprender su utilidad.

Para comprender mejor las características que poseen las herramientas CASE y su utilidad, practicarás con **Umbrello UML Modeller**, que al igual que PSeInt, también es software libre y fácil de usar.

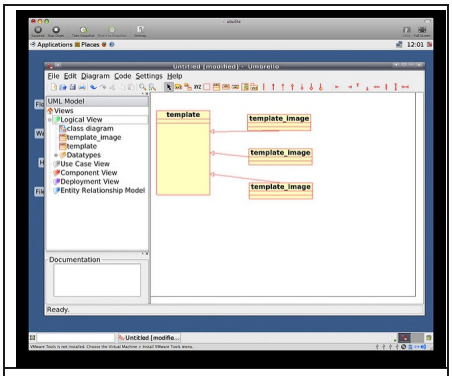


Imagen en Flickr de Clive Darra con Algunos derechos reservados

2. Depuración

Depurar un programa consiste en realizar el proceso de **detección e identificación de errores y problemas** que puede contener el mismo para su posterior corrección y eliminación. Se dice que un programa está depurado cuando está libre de errores. Para referirse a la depuración también se suele utilizar el término inglés "**debugging**", que procede de la traducción literal de eliminación de bichos (bugs), nombre con el que se conoce coloquialmente a los errores en programación.

Imagen en Flickr de Charly W. Karl con Algunos derechos reservados

En el proceso de depuración **se realiza un seguimiento del funcionamiento del programa**, en el que **se observan los distintos valores que van tomando las variables** y los resultados que se obtienen en las operaciones. Es lo que se denomina "hacer una **traza del programa**".

Depurar un programa es una tarea ardua, engorrosa y agotadora, por ello, para simplificarla, es conveniente utilizar herramientas destinadas a tal fin. El software que permite realizar este proceso de revisión y detección de errores se llama depurador o debugger.

El debugger permite, entre otras cosas:

- Establecer puntos de interrupción en la ejecución del programa.
- Continuar la ejecución después de una interrupción.
- Ejecutar paso a paso el programa.
- Examinar los valores que van tomando las variables utilizadas en el programa.

Curiosidad

No todos los entornos de programación ofrecen depuradores a los programadores. **Cuando no se dispone de un depurador** los programadores suelen recurrir a un proceso de **depuración manual**, consistente en colocar en puntos estratégicos del código fuente **instrucciones que muestren por pantalla el valor de las variables**. De esa manera, se suple la posibilidad que brindan los depuradores de poder visualizar los valores contenidos en las variables en determinados puntos durante la ejecución del programa. Una vez que se han podido ver dichos valores durante la ejecución del programa, las instrucciones que permiten ver dichos valores son eliminadas del código fuente, o son comentadas (se las convierte en comentarios) por si en algún otro momento vuelven a ser necesarias.

2.1 Puntos de interrupción (breakpoints)

Un punto de interrupción o punto de ruptura **es una marca que el programador fija**, mediante el uso del software depurador, **en alguna de las instrucciones del código fuente** de un programa **para que la ejecución del mismo se detenga ahí momentáneamente**. Esta detención en la ejecución del programa **permitirá al programador examinar los valores que tienen almacenados las variables en ese preciso instante**. De esta forma, el programador posee un mecanismo muy potente para verificar el funcionamiento del programa en cualquier punto. Después, el depurador, mediante otra de sus opciones, permitirá al programador continuar la ejecución del programa hasta el final, o hasta el siguiente punto de interrupción fijado. Efectivamente, **los depuradores suelen permitir establecer más de un punto de interrupción**, lo que posibilita hacer controles en diversos puntos durante la ejecución del programa.

En la mayoría de depuradores, los puntos de interrupción **suelen simbolizarse mediante un punto rojo grueso o una flecha de color situados a la izquierda de la línea del código fuente** donde el programador lo fija, como puedes apreciar en las siguientes imágenes:

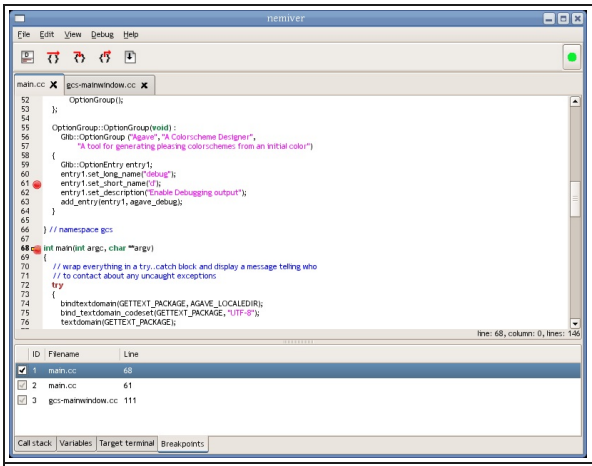


Imagen en Flickr de J Jongsma con Algunos derechos reservados

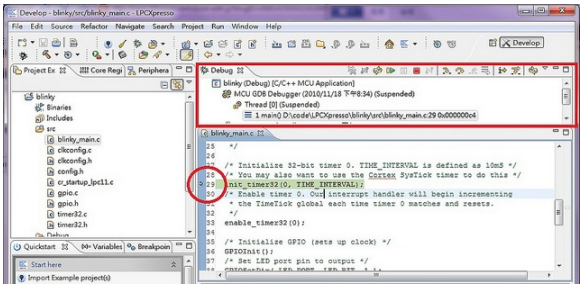


Imagen en Flickr de george079 con Algunos derechos reservados

2.2 Ejecución paso a paso

Imagen en Flickr de Esparta Palma con Algunos derechos reservados

Otra de las opciones que poseen los depuradores es la de permitir la **ejecución de un programa instrucción a instrucción**, deteniéndose en cada una para que el programador pueda ir examinando los valores que van almacenando las variables después de cada instrucción ejecutada. Es decir, que hace el mismo efecto que si se colocara un punto de interrupción en cada instrucción del programa. Por supuesto, **en cualquier momento el propio programador podrá abandonar la ejecución paso a paso** haciendo que el programa se ejecute de forma normal, si detenerse, hasta llegar al final o al siguiente punto de interrupción fijado.

Los depuradores suelen tener asociada la ejecución paso a paso a una opción del menú, así como a una tecla concreta (suele ser alguna de las teclas de función del teclado), de esta forma, el programador irá haciendo clic en dicha opción o pulsando la tecla para cada instrucción a ejecutar.

Puede que te encuentres depuradores con **diferentes opciones de ejecución paso a paso** (paso a paso por instrucciones, paso a paso por procedimientos, paso a paso para salir, etc), pero en definitiva son opciones cuyo objetivo es el de ir ejecutando las instrucciones poco a poco y por tanto las diferencias entre ellas serán mínimas.

Estas opciones, combinadas con otras también interesantes, **permitirán al programador realizar un seguimiento totalmente controlado de la ejecución de sus programas**. Por ejemplo, se podría ejecutar el programa sin parar hasta llegar a un punto de interrupción, o hasta la posición del cursor, y luego ir ejecutando paso a paso las instrucciones siguientes, o al contrario, comenzar ejecutando el programa paso a paso hasta llegar a una instrucción concreta y de ahí en adelante hacer que el depurador ejecute el resto del programa sin pausas hasta el final (o hasta el siguiente punto de interrupción).

El poder que ofrece el depurador al programador hace de este una herramienta esencial y en multitud de ocasiones indispensable.

2.3 Ventanas de depuración: Traza de un programa

Ya hemos hablado de las posibilidades que ofrecen los depuradores a los programadores. En todas ellas, la finalidad es la misma: poder examinar los distintos valores que va tomando cada variable utilizada en el programa, para así, verificar el correcto funcionamiento del mismo, o por el contrario, tener mayor precisión para detectar la causa de los errores que se presenten, además de ganar en rapidez en la resolución de estos últimos. Para examinar los valores almacenados en las variables en cada interrupción, **el depurador ofrece al programador una serie de ventanas en las que se podrán introducir los nombres de las variables a examinar**. También existe la posibilidad, no solo de introducir en esas ventanas el nombre de las variables, sino **también expresiones formadas con las mismas**. A estas ventanas, en general, se les suele dar el nombre de **ventanas de inspección**, y puede haber varias: una ventana para variables locales (las que existen en la función o procedimiento que está en ejecución en ese preciso instante), otra ventana para variables globales (aquellas que existen a lo largo de toda la ejecución del programa), otra para la pila de llamadas a subprogramas, etc.

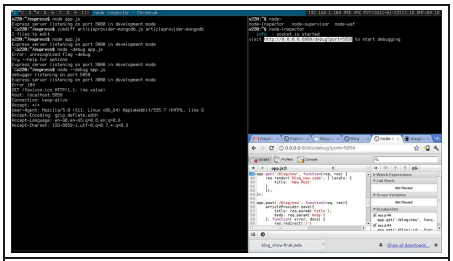


Imagen en Flickr de Kai Hendry con Algunos derechos reservados

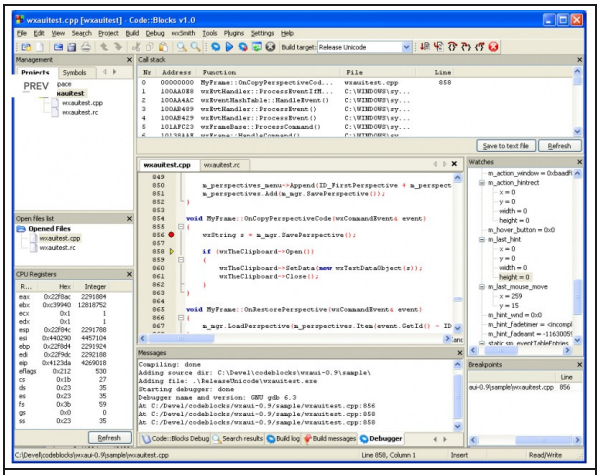


Imagen en Flickr de Linux Screenshots con Algunos derechos reservados

2.4 Depuración en PSeInt

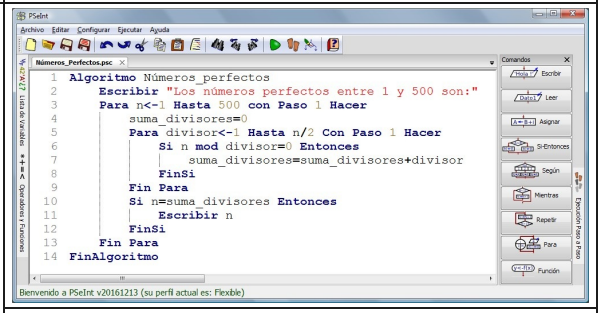
A continuación vamos a utilizar un ejemplo de un **programa implementado en PseInt para ver en él cómo funciona la opción de depuración "paso a paso"** y comprobaremos cómo el depurador nos indica por dónde va el flujo del programa en cada momento, circulando a través de sus bucles de repetición según las condiciones impuestas por el programador. **En una ventana se muestra el programa**, la secuencia de órdenes y el flujo, dirección y orden de ejecución, **en otra se va ejecutando** propiamente el programa, mostrando los valores y resultados buscados, y a la **derecha tenemos el panel "paso a paso"** donde podemos elegir varias opciones que te ofrece el depurador. Así, **si marcamos la opción que nos muestra el detalle**, abajo se nos abre **otra ventana donde se explica qué está haciendo el programa en cada momento**, donde guarda valores, en qué variables, cuales son los valores que guarda, etc. Vamos a verlo de forma más detallada.

El programa mostrará los números perfectos entre 1 y 500. Un **número perfecto** es un **número** natural que es igual a la suma de sus divisores propios positivos (los divisores propios de un número son todos los divisores del mismo, incluyendo el 1, a excepción del propio número).

El código fuente del programa sería el siguiente:

```
Algoritmo Números_perfectos
    Escribir "Los números perfectos entre 1 y 500 son:"
    Para n<-1 Hasta 500 con Paso 1 Hacer
        suma_divisores=0
        Para divisor<-1 Hasta n/2 Con Paso 1 Hacer
            Si n mod divisor=0 Entonces
                suma_divisores=suma_divisores+divisor
            FinSi
        Fin Para
        Si n=suma_divisores Entonces
            Escribir n
        FinSi
    Fin Para
FinAlgoritmo
```

PSeInt lo mostraría así:

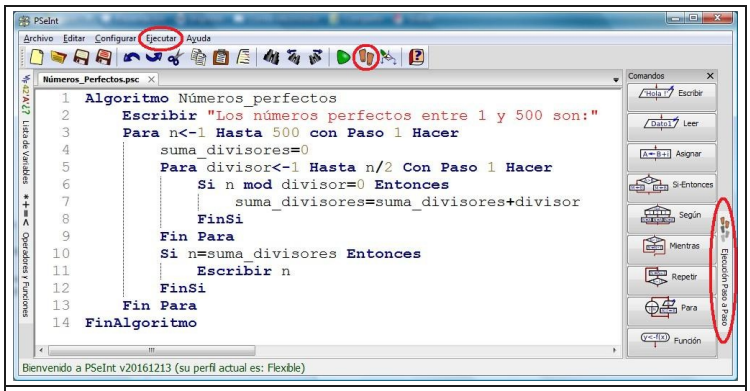


Fuente propia

A continuación, para comenzar la ejecución paso a paso del programa tendríamos varias opciones:

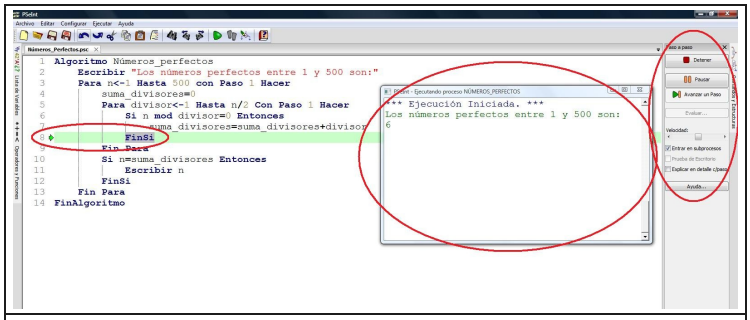
- En el menú superior elegir la opción "Ejecutar" y dentro de ella seleccionar "Ejecutar paso a paso"
- O bien pulsar la tecla F5
- O bien hacer clic en el icono de ejecución paso a paso que aparece en la barra de herramientas, arriba a la derecha, representado por dos pies
- También podemos hacer clic en la pestaña que aparece en la parte lateral derecha de la ventana de PSeInt, junto a los botones de los comandos.

La imagen siguiente muestra algunas de estas posibilidades:



Fuente propia

Inmediatamente después, el algoritmo comenzará a ejecutarse paso a paso, mostrando la ventana de ejecución del programa y marcando sobre el código fuente, en la ventana principal de PSeInt, qué instrucción se está ejecutando en cada momento.



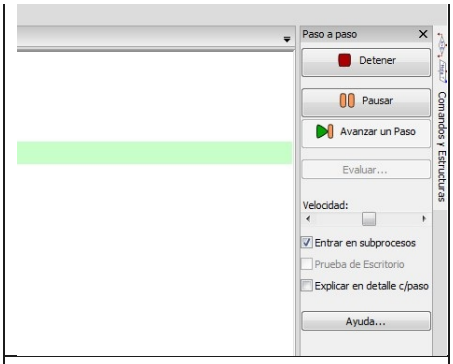
Fuente propia

También verás en la parte derecha de la ventana de PSeInt un panel lateral con distintas opciones que podrás utilizar para el proceso de ejecución paso a paso. Entre esas opciones encontrarás:

- Un primer **botón para detener** la ejecución paso a paso por completo.
- Un **botón** que será utilizado **para hacer pausas** durante la ejecución paso a paso.
- Un **botón para poder avanzar instrucción a instrucción**.
- Otro **botón que permitirá evaluar expresiones** cuando se haga una pausa en la ejecución paso a paso. Este botón estará habilitado solo cuando la ejecución esté pausada.
- Una **barra horizontal que marcará la velocidad** a la que se irá ejecutando cada paso. Desplazando esta barra hacia la derecha configurarás PSeInt para que se haga un tiempo de retardo menor entre cada instrucción ejecutada. Por el contrario, desplazándola hacia la izquierda PSeInt tardará más tiempo en pasar de una instrucción a otra.
- Una **opción que configurará PSeInt para que la ejecución paso a paso sea también a nivel de subprocesos** (funciones) o solo del programa principal. Es decir, que si está marcada esta opción, los subprocesos también se ejecutarán paso a paso. En caso contrario, los subprocesos se ejecutarán sin hacer paradas o pausas.
- **Otra opción** llamada **Prueba de Escritorio que mostrará una nueva ventana** en la

parte inferior de la pantalla **en la que se pueden agregar variables o expresiones para seguir los valores que van tomando** durante la ejecución.

- También aparecerá una **opción que permitirá mostrar de forma detallada cada paso** durante la ejecución.
- Por último, aparece un **botón para acceder a la ayuda** de PSeInt.



Fuente propia

Ejercicio resuelto

Para que puedas practicar con PSeInt y su depurador, te proponemos un segundo ejemplo. Intenta hacer el programa y luego ejecutarlo paso a paso.

En este caso se trata de un programa que calcule con una renta dada de 1.000 euros, 5 combinaciones posibles de consumo de 2 bienes (A y B) que tienen respectivamente unos costes de 20 euros y 7 euros, haciendo que el usuario introduzca cada vez una cantidad a consumir del bien A. También deberá mostrar los costes invertidos respectivamente en los bienes y el sobrante monetario.

¿Te atreves a desarrollarlo tú mismo? ¿No? No te preocupes, te facilitamos la solución para que solo tengas que copiarlo, pegarlo en el editor de PSeInt y comenzar la ejecución paso a paso.

```
Algoritmo Consumo_de_dos_Bienes
    renta=1000
    veces=1
    Mientras veces<6 Hacer
        Escribir "¿Cuántas unidades del bien A a 20 euros quieres consumir?"
        Leer uda
        costea=20*uda
        rtadpble=1000-costea
        resto=rtadpble MOD 7
        costeb=rtadpble-resto
        udb=costeb/7
        rtafinal=1000-costea-costeb
        Escribir "COMBINACIÓN ",veces,":"
        Escribir "CONSUMO DE ",uda," UNIDADES DE A, CON UN COSTE DE ",costea " EUROS Y ",udb," UNIDADES DE B, CON UN COSTE DE ", costeb," EUROS. TE SOBRAN ",rtafinal,"EUROS"
        veces=veces+1
    FinMientras
    Escribir "Estas son las 5 combinaciones posibles con los consumos predeterminados del bien A"
FinAlgoritmo
```

3. Herramientas CASE

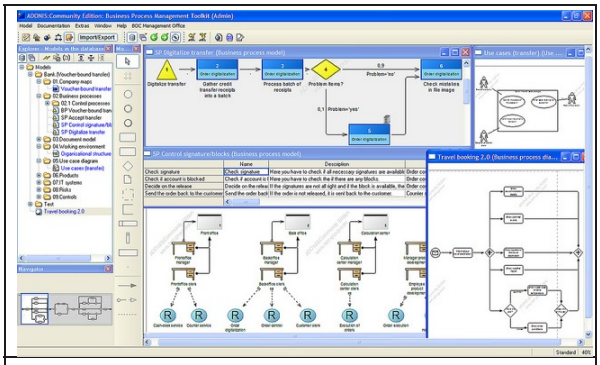


Imagen en Flickr de ADONIS Community con Algunos derechos reservados

Las **herramientas CASE** (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora) son un **conjunto de aplicaciones informáticas que tienen como finalidad ayudar y facilitar el desarrollo de software**. Permiten incrementar la productividad y disminuir costes en el desarrollo del software, ya que ahorran tiempo y por tanto también dinero. Estas herramientas **pueden aplicarse en todas las fases del ciclo de vida del software**, sobre todo en tareas como el proceso de realizar un diseño del proyecto, la estimación y cálculo de costes, la implementación de parte del código de forma automática, la compilación automática del mismo, la generación de documentación o incluso la detección de errores, entre otras.

Los entornos CASE suelen utilizar para todo ello un **conjunto de herramientas** de programación con una **interfaz común** tanto para diseñar, como para desarrollar y depurar software. Por tanto, un entorno CASE consta de una serie de componentes que aportan un prototipo o modelo visual de una aplicación, componentes que pueden crear y generar código a través de interfaces visuales y que finalmente, mediante un depurador, posibilitan la fase de prueba del código final.

Concretando más, podemos ampliar los objetivos esenciales de las herramientas CASE en los siguientes:

- **Mejorar la productividad** en el desarrollo y el mantenimiento de software.
- **Mejorar la planificación** del proyecto e **incrementar la calidad** del software.
- **Reducir el tiempo y el coste** de desarrollo y mantenimiento del software y de los sistemas informáticos.
- **Automatizar** el desarrollo de software, la documentación, la generación de código, las pruebas y la gestión del proyecto.
- **Ayudar en la reutilización** de software, la portabilidad **y la estandarización de la documentación**.
- Permitir una **gestión global y con una misma herramienta** de todas las fases de desarrollo del software.
- **Facilitar el uso de las distintas metodologías** propias de la ingeniería del software.

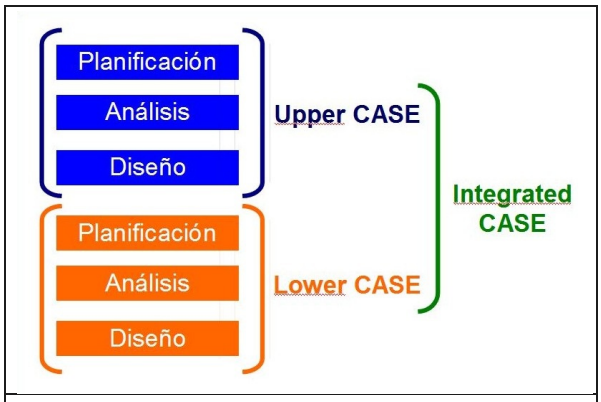
3.1 Tipos de Herramientas CASE

Existen **numerosas clasificaciones** de las herramientas CASE, aunque **la más extendida** es la que divide las herramientas en tres categorías de la siguiente forma:

- **Herramientas de gestión**: Son las que se encargan de la estimación, de la planificación y del seguimiento del proyecto.
- **Herramientas técnicas**: divididas a su vez en dos:

  a. **Upper CASE** (o CASE frontales), dedicadas a las primeras fases de planificación, análisis de requisitos y diseño del software.
  b. **Lower CASE** (o CASE dorsales), que suelen aplicarse en el diseño detallado y la generación de código.

- **Herramientas de soporte**: Son las que abordan sistemas de diccionario/repositorio, seguridad, control y configuración, etc.



Fuente propia

En cualquier caso, las clasificaciones de las herramientas CASE son siempre difusas, es decir, que suele haber herramientas difíciles de clasificar, ya que están a caballo entre dos o más categorías. Por ejemplo, las **I-CASE (Integrated CASE)** engloban aspectos tanto de CASE frontales como dorsales, contemplando por tanto todo el ciclo de desarrollo.

Sin embargo, **existe un denominador común** en la gran mayoría de las herramientas CASE, y es **el uso del Lenguaje Unificado de Modelado (UML)**, que es un estándar utilizado para crear esquemas, diagramas y documentación relativa a los desarrollos de software. El término "lenguaje" aquí no indica que UML sea un lenguaje de programación, ya que no es un lenguaje propiamente dicho, sino un **conjunto de normas y estándares gráficos para representar los esquemas relativos al software**.

En apartados posteriores verás los principios básicos de UML.

Curiosidad

Si quieres ver una muestra de la diversidad y variedad de herramientas CASE que te puedes encontrar, visita por ejemplo esta web.

Fuente propia

Las herramientas CASE para análisis y diseño ayudan en la definición de los requisitos del sistema y sus propiedades, permitiendo crear y modificar diagramas Entidad/Relación (E/R), diagramas de flujo de datos, diagramas de estructura, diagrama de clases, etc. Incluso a veces engloban herramientas de prototipado como diseñadores de pantallas, generadores de menús, generadores de informes, y lenguajes de especificación ejecutables.

Una característica importante de las herramientas CASE es la **posibilidad de generación de código y documentación**. Efectivamente, a partir de las especificaciones del diseño se puede generar código. Utilizan para ello diversos lenguajes de cuarta generación (lenguajes 4GL). Aunque no se puede generar en todos los casos el 100% del código de los programas, sí permiten generar una buena parte del código, de tal manera que al programador solo le quede completar la implementación de los programas y refinarlos.

Otra propiedad importante, como ya se ha mencionado anteriormente, es la integración en los entornos CASE de herramientas para creación y **generación automatizada de un repertorio muy variado de documentación**, que va desde la descripción textual en una especie de pseudocódigo hasta diagramas complejos.

No suele faltar un **entorno integrado para pruebas y depuración**, con múltiples funcionalidades para la gestión, definición, diseño y monitorización de las mismas, así como opciones para construir entornos apropiados de ejecución, en los que se permite verificar e implementar código. Finalmente también suelen aportar **herramientas de evaluación de las pruebas**, aportando informes sobre los fallos en forma de estadísticas para posteriormente valorar la calidad tanto de las pruebas como del software.

En entornos de desarrollo complejos se hace imprescindible la incorporación de una herramienta capaz de gestionar la configuración de los sistemas, destacando sobre todo la gestión del **control de versiones**, es decir, la capacidad de proporcionar almacenamiento y acceso controlado a los datos, así como de registrar los cambios sobre los mismos y poder recuperar versiones anteriores. Para almacenar las distintas versiones de cada elemento, las herramientas de gestión de configuración suelen emplear un método de almacenamiento incremental, consistente en guardar sólo la primera (o la última) versión del elemento de forma completa, mientras que, para el resto de las versiones, se almacena exclusivamente la diferencia entre éstas y la versión inmediatamente anterior (o posterior).

3.3 Lenguaje Unificado de Modelado (UML)

Como ya se ha comentado con anterioridad, el Lenguaje Unificado de Modelado (UML) no es un lenguaje de programación, sino un método o notación gráfica basado en diagramas para especificar, visualizar y documentar modelos de sistemas de software orientado a objetos, teniendo un uso limitado para otros modelos de programación. UML no es un método de desarrollo, lo que significa que no le indica qué debe hacer primero y qué debe hacer después, ni cómo debe diseñar su sistema, sino que le ayuda a visualizar su diseño y a comunicarse con otras personas.

UML está formado por muchos elementos que representan las distintas partes de un sistema de software. Los elementos de UML se usan para crear diagramas, que representan partes del sistema. Para ello cuenta con una serie de diagramas de distintos tipos, como por ejemplo los siguientes:

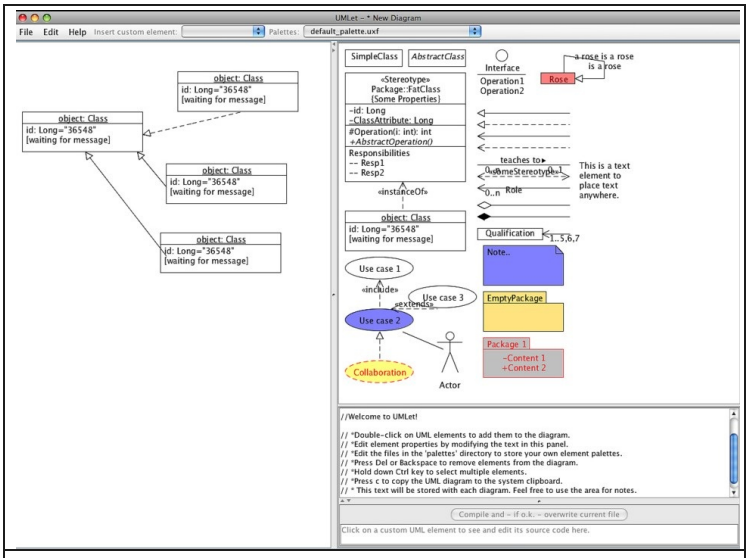


Imagen en Flickr de Clive Darra con Algunos derechos reservados

- **Diagramas de casos de uso**: Aquellos que muestran actores (personas u otros usuarios del sistema), casos de uso (escenarios donde se usa el sistema) y sus relaciones. Mediante estos diagramas representamos la forma en que el usuario (Actor) opera con el sistema. Estas operaciones (o casos de usos) se realizan tras la orden de un actor o la llamada desde otra operación (otro caso de uso). Las relaciones entre los actores y los casos de uso se expresan en el diagrama mediante flechas de distinto tipo para expresar relaciones de asociación, dependencia o generalización por ejemplo.
- **Diagramas de clases**: Como su propio nombre indica, muestran clases y las relaciones entre ellas. Veremos con más detalle este tipo de diagramas y a través de un ejemplo lo entenderemos mejor, analizando todos sus elementos, tales como objetos (instancias de una clase), sus atributos, métodos, relaciones, herencia, asociación, etc.
- **Diagramas de secuencia**: Son los que muestran objetos y una secuencia de las llamadas a métodos que hacen a otros objetos.La base de estos diagramas es la característica temporal, es decir, que la idea principal es mostrar la interacción entre los objetos participantes a lo largo del tiempo. Es un concepto intrínseco al de secuencia. Cada participante individual (línea de vida) tiene un rectángulo con el nombre del objeto y las interacciones en el tiempo se representan mediante mensajes dibujados como flechas. Estos mensajes pueden ser completos, perdidos o encontrados, síncronos o asíncronos, llamadas o señales.
- **Diagramas de colaboración**: Representan objetos y sus relaciones, poniendo énfasis en aquellos que participan en el intercambio de mensajes. Se muestran los mensajes enviados entre si. Un ejemplo de mensaje podría ser la solicitud del resultado de un cálculo. En este caso los dos objetos, origen y destino, emiten mensajes, el primero solicitando el resultado y el segundo enviándolo. A su vez este resultado puede ser enviado a otro objeto que lo necesita para volver a operar o conjugar con otros.
- **Diagramas de estados**: Permiten plasmar los estados del sistema, los eventos de los objetos que hacen cambiar de estado y por supuesto, los cambios de estado. Es decir tenemos tres elementos básicos, estados (inicial, intermedios y final), eventos (razones o causas que hacen variar el estado inicial a otro estado diferente) y el cambio de estado. Mostramos los la secuencia de estados por las que puede pasar un objeto, su flujo de estados y la forma en que se producen esos cambios. Un buen ejemplo podría consistir en considerar el interés que puede tener una persona en un trabajo, pudiendo estar muy interesada, interesada o desinteresada. Puede cambiar de pensamiento (estado) por diferentes motivos. También podría ser el estado de ánimo de una persona, pudiendo pasar de contenta a estado normal o enfadada por diferentes razones.
- **Diagramas de actividades**: Muestran actividades y los cambios de una actividad a otra con los eventos que ocurren en alguna parte del sistema. Representan el comportamiento dinámico (activo) de un sistema, exponiendo la transición entre las actividades desde un estado inicial a uno final. Por ejemplo el pago automatizado de un aparcamiento donde tendríamos cuatro actividades: introducir tiket en la máquina, esperar la información sobre el coste, introducir el pago en dinero y retirar el tiket.
- **Diagramas de componentes**: Representan los componentes de programación de alto nivel. Es decir, la estructura física del código de un programa. O sea, se muestran los elementos de diseño de software tales como tablas, archivos, ejecutables, documentos, librerías, archivos de configuración, etc., y la forma de acceder a ellos, o sea la interfaz de usuario.
- **Diagramas de despliegue**: Plasman las instancias de los componentes y sus relaciones. Muestran la distribución o estructura, la arquitectura, que forman los elementos de software mencionados en el apartado anterior, que se denominan aquí artefactos, y describe el despliegue físico de información generada por el programa en los componentes de hardware correspondientes llamados nodos.
- **Diagrama Entidad-Relación**: Representan datos y las relaciones y restricciones entre ellos. Está basado en la percepción del mundo real donde los objetos son entidades y se reflejan las relaciones entre las mismas. Un ejemplo puede ser el diagrama de una empresa donde coexisten clientes, proveedores y personal empleado. Las relaciones se manifiestan a través de los productos comercializados teniéndose que realizar todas las gestiones necesarias y típicas, desde transporte y almacenado, hasta manipulado, transformado, envasado, y venta.

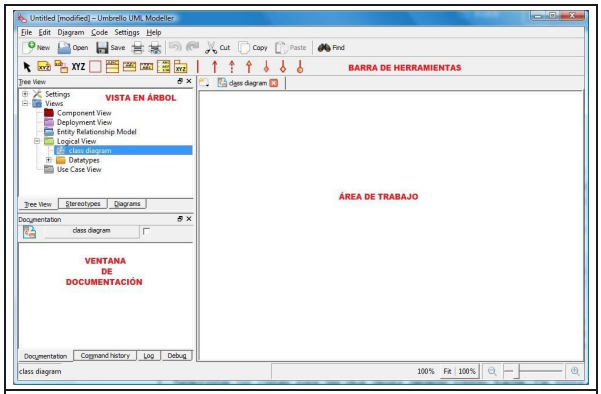
Son múltiples los usos de UML en las herramientas CASE, en el siguiente apartado podrás estudiar los principios básicos de una de ellas: Umbrello UML Modeller.

3.4 Umbrello: Primeros pasos

Umbrello UML Modeller es un software libre que permite construir y editar diagramas UML para facilitar la creación y desarrollo de software de calidad. También permite, entre otras muchas cosas, generar código fuente para diversos lenguajes de programación a partir de los diagramas UML creados. Desde su página oficial puedes descargarlo para el sistema operativo que tengas instalado en tu ordenador.

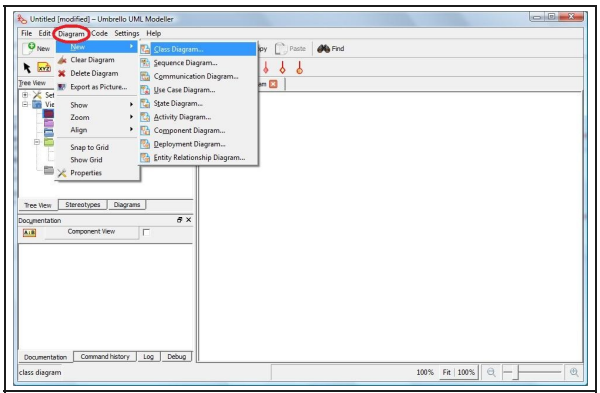
Es capaz de gestionar una gran variedad de diagramas distintos mediante una interfaz gráfica que tiene 3 partes:

- **Área de trabajo**: El área de trabajo es la ventana principal de Umbrello UML Modeller, y es la zona donde se editarán y verán los distintos diagramas del esquema que se va a representar.
- **Vista en árbol**: Es la zona donde se muestran todos los elementos de los que estará compuesto el esquema a representar (diagramas, clases, actores, casos de uso,...). Proporciona una forma rápida de pasar de un diagrama a otro de los que componen el esquema representado, ya que al hacer clic en el árbol sobre uno de los diagramas, los elementos del mismo aparecerán en el área de trabajo.
- **Ventana de documentación**: Permite previsualizar rápidamente la documentación para el objeto que previamente tengamos seleccionado.



Fuente propia

También dispone de una **barra de herramientas** en la parte superior, en la que se muestran los botones que permiten insertar los elementos necesarios para elaborar cada diagrama. Tan solo hay que hacer clic en el botón del elemento a insertar y luego hacer de nuevo clic en el punto del área de trabajo donde se desee insertar el elemento.



Fuente propia

Umbrello ofrece, como ya se ha comentado con anterioridad, una gran variedad de diagramas posibles. Para ello, solo hay que elegir en el menú superior la opción de "Diagrama" y dentro de ella seleccionar el diagrama deseado. También se puede hacer clic con el botón derecho del ratón sobre una de las carpetas que se visualizan en la vista en árbol y seleccionar el diagrama deseado.

A modo de ejemplo, en el siguiente apartado verás cómo trabajar con Umbrello UML Modeller para crear uno de los tipos de diagrama que ofrece. Nos centraremos, más concretamente, en la construcción de un diagrama de clases.

3.4.1 Diagrama de Clases

Un diagrama de clases es una **herramienta destinada a representar y describir la estructura de un sistema**. En él se plasma de forma visual la arquitectura, los requisitos de entidades, los patrones y los diseños de un proyecto de software. Los diagramas de clases **se utilizan cuando se va a programar mediante orientación a objetos**, ya que **son una representación de todas las clases que tiene el sistema a desarrollar, y sus relaciones**, incluyendo además los **atributos y métodos** de cada clase.

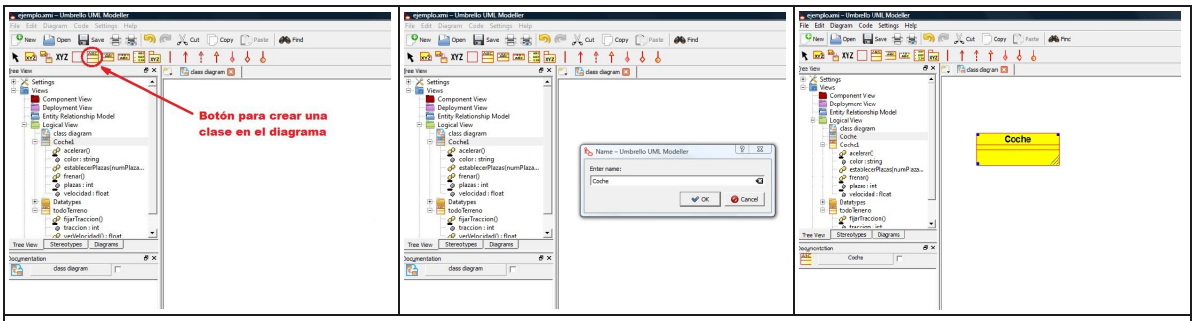
Como ya sabemos, en orientación a objetos una clase es la unidad básica que aglutina y encapsula toda la información de un objeto, y un objeto sería una instancia de una clase. Por tanto, mediante las clases podemos modelar el sistema que queramos representar (casas, personas, coches, cuentas bancarias, etc.). Hay que recordar que las clases tienen atibutos (las propiedades o características de los objetos de la clase) y métodos (las acciones que pueden realizar los objetos de la clase). En UML, **una clase se representa mediante un rectángulo dividido horizontalmente en tres zonas**, de tal manera que la zona superior se utiliza para escribir el nombre de la clase, la zona media para anotar en ella los atributos y la zona inferior para exponer los métodos que posee. Quedaría de esta forma:

Nombre de la clase
Lista de atributos de la clase
Lista de métodos de la clase

Por ejemplo, para representar la clase Coche, cuyos atributos son el color, la velocidad y las plazas, y sus métodos establecerPlazas, frenar y acelerar, usaremos el siguiente rectángulo:

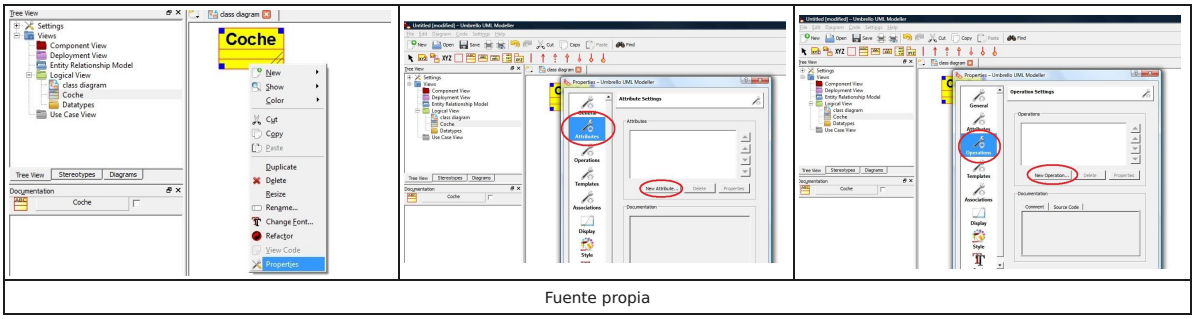
Coche
color velocidad plazas
establecerPlazas frenar acelerar

Para crear una clase en Umbrello contamos con un botón de la barra de herramientas, tan solo hay que hacer clic en él y luego hacer clic en cualquier parte del área de trabajo. Umbrello mostrará una pequeña ventana solicitando el nombre de la clase. Después de introducir el nombre, la clase aparecerá en el área de trabajo en forma de un rectángulo con fondo amarillo. En las siguientes imágenes se puede apreciar con mayor claridad:



Fuente propia

Una vez creada la clase, para introducir tanto sus atributos como sus métodos, basta con hacer clic con el botón derecho del ratón sobre el rectángulo amarillo que la representa y seleccionar la opción de "Propiedades" del menú contextual que aparecerá en pantalla. Las propiedades de la clase aparecen organizadas y estructuradas en bloques que son accesibles mediante grandes botones situados en la parte izquierda de la ventana. Entre esos bloques podremos encontrar los atributos y los métodos, con sendos botones para agregar ambos. Las siguientes imágenes ofrecen con nitidez estos aspectos:



Fuente propia

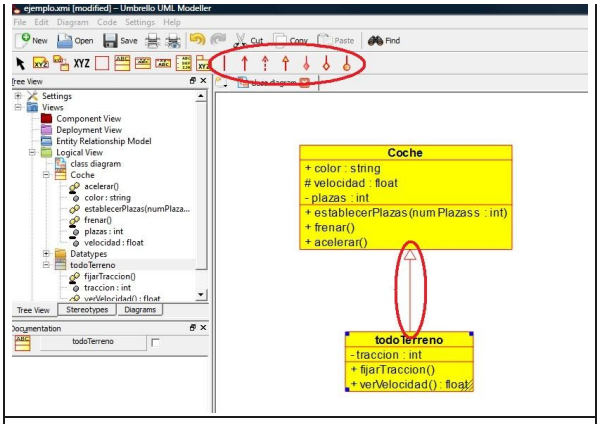
Una vez creadas las clases, el siguiente paso sería establecer las relaciones entre las mismas. En UML existen varios tipos de relaciones que se pueden dar entre las clases:

- **Asociaciones**: Representan relaciones de objetos que colaboran entre sí. Son relaciones débiles, es decir, en ellas la existencia de un objeto no depende de la del otro.
- **Dependencias**: Representan un tipo de relación muy particular, en la que una clase es instanciada desde otra. Se denota por una flecha punteada. Son las relaciones más débiles que existen.
- **Herencia (Especialización/Generalización)**: Se utiliza cuando una clase (clase hija o subclase) hereda métodos y atributos de otra clase (clase padre o superclase), por tanto la subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la superclase (public y protected).
- **Composiciones**: Se dan cuando un atributo de una clase hace referencia a un objeto de otra clase. Son tipos de relaciones estáticas, en las que el tiempo de

vida del objeto incluido esta condicionado por el tiempo de vida del que lo incluye, es decir, que la existencia de los objetos de una clase dependerán de la existencia del los objetos de la otra. Por ejemplo, si tuviéramos una clase Coche y otra clase Motor, en la clase Coche habría un atributo que referencia a un objeto de la clase Motor (un coche sin motor no tiene razón de ser, ya que lo necesita para su funcionamiento).

- **Agregaciones**: También se dan cuando un atributo de una clase hace referencia a un objeto de otra clase, pero a diferencia de las anteriores, son relaciones dinámicas, en las que el tiempo de vida del objeto incluido es independiente del que lo incluye. En efecto, en este caso, la existencia del objeto referenciado no depende de la clase que lo referencia. Por ejemplo, si tuviésemos una clase Ciudad relacionada con otra clase Aeropuerto, en la clase Ciudad habría un atributo para hacer referencia a un objeto de la clase Aeropuerto, sin embargo, podrían existir ciudades sin aeropuertos.

Tras la creación de las clases con sus atributos y métodos correspondientes, el siguiente paso sería realizar las relaciones entre ellas. Para establecer relaciones en un diagrama de clases, Umbrello ofrece una serie de botones disponibles en la barra de herramientas.



Fuente propia

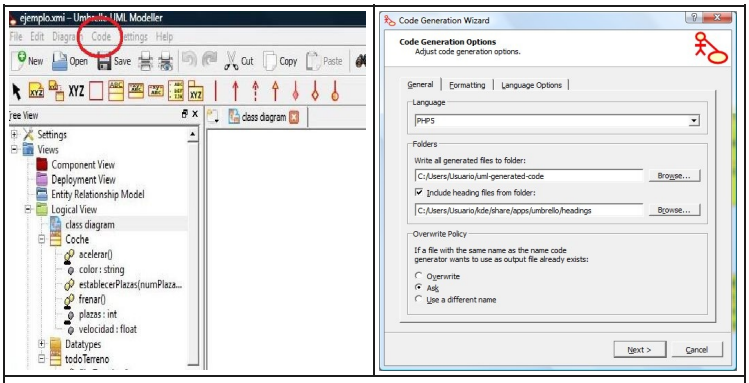
3.4.2 Generando código fuente

Una de las características o utilidades que aportan las herramientas CASE es la **posibilidad de generar código fuente a partir de los diagramas UML** realizados de forma gráfica en su entorno. Normalmente suelen disponer de un **asistente que irá guiando al diseñador en el proceso de generación de código**, lo cual hace que las herramientas aporten una **gran agilidad y rapidez a la hora de comenzar la implementación de los programas**. Entre los pasos del asistente es frecuente encontrar una opción para determinar el lenguaje de programación en el que se quiere generar el código, lo que consigue que las Herramientas CASE y los diagramas que en ellas se generen alcancen cierta independencia respecto a los lenguajes de programación que se utilizarán para crear el software o producto final.

Concretamente, en Umbrello, dichas opciones las puedes encontrar en el menú superior de la aplicación, seleccionando en dicho menú la opción de "Código" (o "Code" si aparece en inglés), y dentro de ella podrás acceder al "Asistente para generación de código" (o "Code Generation Wizard" si el programa viene configurado en inglés).

El asistente permitirá configurar una serie de aspectos:

- Elegir el lenguaje destino (entre ellos se encuentran C++, Java, Javascript, MySQL, PHP, Python, etc.).
- Establecer opciones sobre los detalles y comentarios a incluir.
- Seleccionar las carpetas donde se generarán los archivos, tanto del código fuente como de los archivos de cabezera (archivos pueden contener información sobre derechos de autor o sobre licencias, así como variables que se evalúan durante la generación del código).
- O incluso condifurar la política de sobreescritura, para que Umbrello actúe de una determinada forma en caso de que ya existan archivos de código fuente generados en pasos anteriores.



Fuente propia

En definitiva, **la generación de código fuente con herramientas CASE puede convertirse en uno de los puntos fuertes para los programadores** a la hora de iniciar la implementación de sus programas.

4. Inténtalo tú mismo

Ha llegado la hora de que pongas en práctica todo lo que has visto en el tema. Para ello te proponemos las siguientes tareas:

- Realiza en PSeInt un programa escrito en pseudocódigo que simule el lanzamiento de una moneda el número de veces que el usuario introduzca por teclado y cuente cuántas caras y cuántas cruces han salido. Para ello puedes usar la función "Aleatorio" de PSeInt, que puedes consultar aquí. Posteriormente, realiza una ejecución paso a paso del programa y comprueba que el programa funciona de forma correcta.
- Introduce el siguiente algoritmo en PSeInt y, ayudándote de la ejecución paso a paso, averigua qué hace exactamente el programa:

```
Algoritmo Desconocido1
   Escribir "Escribe un texto" sin saltar
   Leer cad
   Para i<-1 Hasta longitud(cad) Con Paso 1 Hacer
      c<-subcadena(cad,i,i)
      Si c<>" " Entonces
         Escribir c sin saltar
      Sino
         Escribir ""
      Fin Si
   Fin Para
   Escribir ""
FinAlgoritmo
```

- Introduce el siguiente algoritmo en PSeInt y, ayudándote de la ejecución paso a paso, averigua qué valores va tomando la variable "p". ¿Sabrías decir lo que hace el algoritmo?

```
Algoritmo Desconocido2
   p<-0
   Escribir "Teclea una frase" sin saltar
   Leer frase
   i<-1
   Mientras i<=longitud(frase) Hacer
      c=subcadena(frase,i,i)
      Mientras i<=longitud(frase) Y c=" " Hacer
         i<-i+1
         Si i<=longitud(frase) Entonces
            c=subcadena(frase,i,i)
         FinSi
      Fin Mientras
      s<-0
      Mientras i<=longitud(frase) Y c<>" " Hacer
         i<-i+1
         s<-1
         Si i<=longitud(frase) Entonces
            c=subcadena(frase,i,i)
         FinSi
      Fin Mientras
      p<-p+s
      i<-i+1
   Fin Mientras
   //Escribir p
FinAlgoritmo
```

Observa que la penúltima línea de código es un comentario, si no consigues averiguar el valor de "p" al final de la ejecución paso a paso, puedes quitarle las dos barras y descomentar la línea. Eso te ayudará a resolver el enigma.

- El siguiente algoritmo contiene un error cuando se ejecuta. Intenta descubrir el problema y solvéntalo para ver qué hará el programa.


```
Algoritmo Desconocido3
   Dimension n[20],v[5]
   Para i<-1 Hasta 6 Con Paso 1 Hacer
      v[i]<-0
   FinPara
   Para i<-1 Hasta 20 Con Paso 1 Hacer
      n[i]<-Aleatorio(1,6)
      v[n[i]]<-v[n[i]]+1
   Fin Para
   Para i<-1 Hasta 6 Con Paso paso Hacer
      Escribir i," -> ",v[i]
   Fin Para
FinAlgoritmo
```

- El siguiente programa intenta debería calcular la suma y el producto de los 10 primeros números naturales, pero el resultado que da no es ese. ¿Puedes descubrir por qué falla y corregirlo para que funcione correctamente?

```
Algoritmo Desconocido4
   suma<-0
   producto<-0
   Escribir "Cálculo de la suma y el producto de los 10 primeros números naturales"
   Para i<-1 Hasta 10 Con Paso 1 Hacer
      suma<-suma+1
      producto<-producto*i
   FinPara
   Escribir "La suma de los 10 primeros números naturales es ",suma
   Escribir "El producto de los 10 primeros números naturales es ",producto
FinAlgoritmo
```

- Si no lo has hecho aún, crea un diagrama de clases en Umbrello en el que plasmes las clases que aparecen en el ejemplo del apartado 3.4.1, incluyendo los atributos y métodos de las clases y por supuesto la relación de herencia que aparece.
- Amplía el ejercicio anterior para incluir en él una clase llamada Vehículo que tenga como atributo público el número de ruedas y un método

para inflar la rueda.

- Relaciona la clase Coche del ejercicio anterior para que sea una clase hija de la clase Vehículo.
- Siguiendo con el ejercicio anterior, crea una nueva clase llamada Ciclomotor, con un atributo llamado cilindrada y un método llamado Comprobar_aceite. Establece una relación de herencia entre la clase Ciclomotor y la clase Vehículo en la que esta última sea la clase padre.
- Utilizando en Umbrello el Asistente para Generación de Código, intenta generar los fuentes de las clases del ejercicio anterior, tanto en lenguaje C++ como en PHP.

Comprueba lo aprendido

Los errores en programación pueden ser:

- ◯ Estructurales, también llamados de sintaxis.
- ◯ Estructurales, también llamados de funcionamiento.
- ◯ De sintaxis o de funcionamiento.

Incorrecto

Incorrecto

Opción correcta

Solución

1. Incorrecto (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Opción correcta (Retroalimentación)

Los errores de funcionamiento:

- ◯ Pueden interrumpir la ejecución de un programa.
- ◯ Si ocurren, el programa se interrumpe obligatoriamente.
- ◯ No existe tal tipo de errores.

Opción correcta

Incorrecto

Incorrecto

Solución

1. Opción correcta (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Incorrecto (Retroalimentación)

Depurar un programa consiste en:

- ◯ Retocar su diseño.
- ◯ Detectar, identificar y corregir errores.
- ◯ Dar por finalizado su código.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

El "Debugger":

- ◯ Es software básico y por tanto presente en todos los entornos de programación.
- ◯ Examina los valores que van tomando las variables del programa.
- ◯ Impide la ejecución paso a paso del programa.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)

2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

"Hacer una traza del programa" es realizar:

- ◯ Un borrador o boceto del mismo incluyendo sólo las ideas principales.
- ◯ Un seguimiento del programa.
- ◯ Una copia del programa convenientemente "a prueba de errores".

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

Si un entorno de programación no dispone de "Debugger":

- ◯ Se puede hacer una depuración manual.
- ◯ No es posible depurar el programa.
- ◯ Todos los entornos disponen de "Debugger".

Opción correcta

Incorrecto

Incorrecto

Solución

1. Opción correcta (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Incorrecto (Retroalimentación)

Los puntos de interrupción:

- ◯ Suponen tener que reiniciar el programa una vez analizadas las variables.
- ◯ Pueden ser múltiples y en cualquier parte del programa.
- ◯ Son fijados automáticamente por el "Debugger".

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

La ejecución paso a paso:

- ◯ Nos permitirá examinar los valores de las variables después de cada instrucción.
- ◯ Es lo mismo que establecer un número elevado de puntos de interrupción.
- ◯ Si se ejecuta es obligatoria hasta el final del programa, no puede abandonarse antes.

Opción correcta

Incorrecto

Incorrecto

Solución

1. Opción correcta (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Incorrecto (Retroalimentación)

Las ventanas de inspección:

- ◯ Sólo pueden existir para variables locales.
- ◯ Son ventanas de depuración y puede haber varias.
- ◯ Son ventanas de actualización de sistemas operativos tales como Windows.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

Las herramientas CASE:

- ◯ Dan formato a las instrucciones.
- ◯ Ahorran tiempo y facilitan el desarrollo del programa.
- ◯ Sólo se usan en la detección de errores.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

Son objetivos de las herramientas CASE:

- ◯ Incrementar la cantidad de software producido.
- ◯ Incrementar la calidad del software producido y mejorar la productividad en el desarrollo de programas.
- ◯ Exclusivamente agilizar la depuración de errores.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

Permitir una gestión global y con una misma herramienta es:

- ◯ Imposible a día de hoy.
- ◯ Posible y fácil con herramientas CASE.
- ◯ Deseable pero poco probable.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

Las "Upper CASE":

- ◯ Se encargan de la estimación inicial y el seguimiento del proyecto.
- ◯ Sus funciones principales son Seguridad y Control.
- ◯ Se usan para analizar requisitos y diseño del software.

Incorrecto

Incorrecto

Opción correcta

Solución

1. Incorrecto (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Opción correcta (Retroalimentación)

El "Lenguaje Unificado de Modelado" (UML):

- ◯ Es un lenguaje de programación como su propio nombre indica.
- ◯ No es un lenguaje de programación.
- ◯ Es raramente usado por las herramientas CASE.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

Las partes de la interfaz gráfica de UMBRELLO son:

- ◯ Área de trabajo, Vista en árbol y Ventana de documentación.
- ◯ Área de trabajo y Vista en árbol.
- ◯ Área de trabajo y Ventana de documentación.

Opción correcta

Incorrecto

Incorrecto

Solución

1. Opción correcta (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Incorrecto (Retroalimentación)

Los Diagramas de Clases:

- ◯ Se usan en programación orientada a objetos.
- ◯ Constan de dos partes: Nombre de la clase y Lista de atributos de la clase.
- ◯ Constan de dos partes: Nombre de la clase y Lista de métodos de la clase.

Opción correcta

Incorrecto

Incorrecto

Solución

1. Opción correcta (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Incorrecto (Retroalimentación)

Comprueba lo aprendido

Errar en la forma de escribir una instrucción de programa es un error de funcionamiento.

◯ Verdadero ◯ Falso

Falso

Las herramientas CASE nos ayudan en el diseño, desarrollo y depuración de programas.

◯ Verdadero ◯ Falso

Verdadero

Observar los valores de las variables y los resultados obtenidos es lo que se denomina "hacer una traza del programa".

◯ Verdadero ◯ Falso

Verdadero

El "Debugger" impide la continuación de la ejecución después de un punto de interrupción.

◯ Verdadero ◯ Falso

Falso

Un punto de ruptura es una marca que el programador fija en alguna de las instrucciones del código fuente de un programa.

◯ Verdadero ◯ Falso

Verdadero

La ejecución saltando de un punto de interrupción al siguiente es lo que se denomina ejecución paso a paso.

◯ Verdadero ◯ Falso

Falso

No es objetivo de las herramientas CASE mejorar la planificación de un proyecto.

◯ Verdadero ◯ Falso

Falso

Las herramientas CASE ayudan a la estandarización de documentos.

◯ Verdadero ◯ Falso

Verdadero

Las Lower CASE suelen emplearse en el diseño detallado y la generación de código.

◯ Verdadero ◯ Falso

Verdadero

Los Diagramas de Despliegue plasman las instancias de los componentes y sus relaciones.

◯ Verdadero ◯ Falso

Verdadero

Los Diagramas Entidad-Relación (E/R) muestran objetos y la secuencia de llamadas a métodos que hacen otros objetos.

◯ Verdadero ◯ Falso

Falso

UMBRELLO es un software libre para construir y editar diagramas UML.

◯ Verdadero ◯ Falso

Verdadero

Curiosidad

Imagen en Flickr de dlmanrg con Algunos derechos reservados

En ocasiones, en lugar de partir de las primeras fases del ciclo de vida del software e ir avanzando por ellas hasta conseguir el resultado final, **se necesita** hacer lo contrario, es decir, **partir de los resultados finales e ir retrocediendo** para obtener productos de software en fases anteriores. Es lo que se llama **ingeniería inversa**. Es decir, que normalmente, el ciclo de vida del software va encaminado a obtener un código fuente final, y la ingeniería inversa parte de ese código fuente final para abstraerse e ir obteniendo las fases anteriores al código fuente. Las herramientas que permiten tal operación son las conocidas como Herramientas de ingeniería inversa.

Además de todos los tipos de herramientas que poseen los entornos CASE, y que ya se han mencionado, también suelen incluir alguna herramienta de ingeniería inversa. Dentro de éstas destacan por ejemplo las que llevan a cabo las siguientes operaciones:

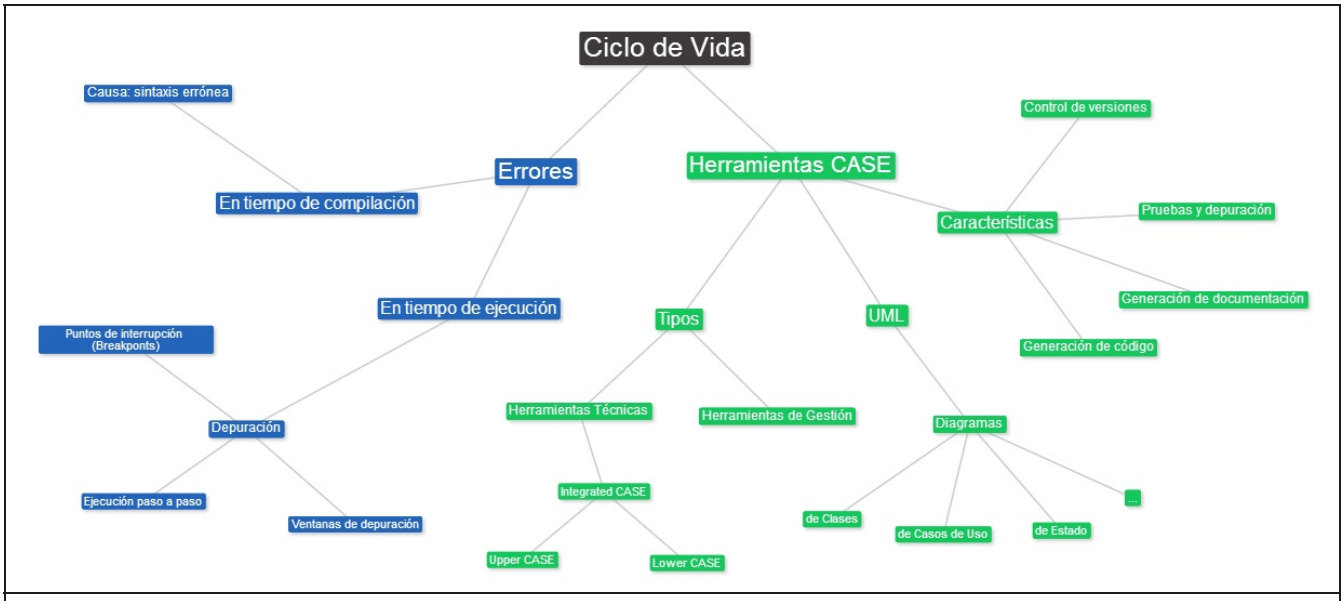
- Ingeniería inversa de datos, que son capaces de extraer la información del código fuente que describe la estructura de los elementos de datos, construyendo así diagramas Entidad/Relación partiendo de esquemas relacionales, jerárquicos o ficheros.
- Ingeniería inversa de procesos, que incluso permiten aislar la descripción lógica de las entidades y las reglas del negocio a partir del código de los programas.
- Reestructuración de código fuente, que modifican su formato o implantan un formato estándar.
- Redocumentación, que permiten generar diagramas a fin de que se comprenda mejor el código.
- Análisis de código, cuyas funcionalidades van desde la tabulación automática del código fuente hasta la posibilidad de ir visualizando dinámicamente las llamadas existentes en el mismo.

Puedes indagar más sobre las Herramientas de Ingeniería Inversa aquí.

Para saber más

Haciendo una búsqueda en internet encontrarás multitud de sitios que te pueden servir para conocer más **detalles de los distintos diagramas que UML integra**. De entre todos ellos, hemos elegido un artículo muy interesante que recoge, de forma bastante resumida, los fundamentos básicos de cada uno. Puedes leerlo visitando esta web.

Incluso puedes encontrar **herramientas online, para generar diagramas** sin necesidad de instalar ningún programa en tu ordenador, como por ejemplo esta.



Fuente propia

Aviso legal

El presente texto (en adelante, el "**Aviso Legal**") regula el acceso y el uso de los contenidos desde los que se enlaza. La utilización de estos contenidos atribuye la condición de usuario del mismo (en adelante, el "**Usuario**") e implica la aceptación plena y sin reservas de todas y cada una de las disposiciones incluidas en este Aviso Legal publicado en el momento de acceso al sitio web. Tal y como se explica más adelante, la autoría de estos materiales corresponde a un trabajo de la **Comunidad Autónoma Andaluza, Consejería de Educación (en adelante Consejería de Educación**).

Con el fin de mejorar las prestaciones de los contenidos ofrecidos, la Consejería de Educación se reservan el derecho, en cualquier momento, de forma unilateral y sin previa notificación al usuario, a modificar, ampliar o suspender temporalmente la presentación, configuración, especificaciones técnicas y servicios del sitio web que da soporte a los contenidos educativos objeto del presente Aviso Legal. En consecuencia, se recomienda al Usuario que lea atentamente el presente Aviso Legal en el momento que acceda al referido sitio web, ya que dicho Aviso puede ser modificado en cualquier momento, de conformidad con lo expuesto anteriormente.

1. Régimen de Propiedad Intelectual e Industrial sobre los contenidos del sitio web

1.1. Imagen corporativa

Todas las marcas, logotipos o signos distintivos de cualquier clase, relacionados con la imagen corporativa de la Consejería de Educación que ofrece el contenido, son propiedad de la misma y se distribuyen de forma particular según las especificaciones propias establecidas por la normativa existente al efecto.

1.2. Contenidos de producción propia

En esta obra colectiva (adecuada a lo establecido en el artículo 8 de la Ley de Propiedad Intelectual) los contenidos, tanto textuales como multimedia, la estructura y diseño de los mismos son de autoría propia de la Consejería de Educación que promueve la producción de los mismos.

Descargar PDF