UNIDAD 2: ENTORNOS DE DESARROLLO.

Módulo Profesional: Entornos de Desarrollo

Índice

RESUMEN INTRODUCTORIO

En la presente unidad se estudiarán conceptos relacionados con los entornos de desarrollo y las herramientas CASE. En este sentido, se pretenden analizar las ventajas del uso de este tipo de herramientas, así como los diferentes tipos y clasificaciones. Seguidamente, se pasará a conocer la importancia que en la actualidad desempeñan los entornos de desarrollo y sus diferentes tipos. A continuación, se analiza el entorno de desarrollo NetBeans a través de una guía de instalación y, por último, se estudian metodologías de desarrollo presentes en las empresas como son las de tipo ágil.

INTRODUCCIÓN

Como se estudió en la unidad anterior, el proceso de desarrollo de las aplicaciones informáticas viene determinado, generalmente, por una serie de fases. Estas fases están muy relacionadas entre sí y, en muchas ocasiones, es fundamental poder automatizar lo máximo posible todas las tareas involucradas. En esta unidad se pretende conocer qué tipo de herramientas existen para ayudar a la automatización del proceso de desarrollo de las aplicaciones. Este tipo de herramientas van a facilitar mucho la labor a todos los miembros de los equipos implicados.

Además, se introducen nociones básicas de metodologías ágiles, muy utilizadas en la actualidad. Este tipo de metodologías son aplicables tanto al proceso de desarrollo software como en empresas u organizaciones de otro sector. Entre ellas se destaca la metodología SCRUM, cuyo fundamento se basa en llevar a cabo reuniones diarias entre los miembros del equipo y realizar entregables al cliente de forma periódica. Esta metodología de trabajo permite un mayor control sobre el producto a entregar y una relación más cercana entre cliente y equipo de trabajo. Las reuniones son muy útiles para solucionar errores que estén ocurriendo y, de esta forma, no alargar estos problemas en el tiempo.

CASO INTRODUCTORIO

En la unidad anterior, Juan y su equipo de analistas llegaron a establecer una reunión con el cliente. Una vez recopilados todos los requisitos y analizadas todas las características de la aplicación software, se disponen a ejecutar la fase de diseño. En ella, los diseñadores del equipo de Juan empiezan a hacer uso de las herramientas CASE instaladas en su equipo: ArgoUML para el diseño de diagramas UML, Subversion para el control de versiones de los ficheros de código y los correspondientes a la documentación, y MySQL Workbench, para diseñar la base de datos. Además, el equipo de programadores pone a punto sus entornos de desarrollo, mediante los cuales implementarán todo el código de la aplicación. Al ser un software creado en lenguaje Java, se acuerda configurar Eclipse con la última versión de JDK disponible. Con todas estas herramientas, y con la buena gestión de Juan para el reparto de tareas entre los miembros de su equipo, se procede al desarrollo de las fases del software.

Al finalizar la unidad el alumnado:

- ✓ Conocerá las principales herramientas CASE utilizadas para la automatización de las fases del ciclo de vida del software.
- ✓ Conocerá los principales entornos de desarrollo del mercado.
- ✓ Afianzará sus conocimientos con las diferentes fases presentes en el desarrollo de las aplicaciones.
- ✓ Tomará consciencia de la importancia de la automatización en los procesos de desarrollo de software.

1. ENTORNOS DE DESARROLLO

Los **entornos de desarrollo** están formados por un conjunto de software que facilitan o automatizan las actividades de desarrollo. Lo ideal sería poder automatizar todo el proceso de desarrollo de una aplicación desde el principio (fase de análisis) hasta el final (explotación y mantenimiento) pero normalmente sólo se automatizan las fases de implementación y las pruebas del software. Las llamadas actividades verticales del ciclo de vida son las específicas de una fase de la vida del software, por ejemplo, la etapa de análisis o diseño. Por su parte, las actividades horizontales son actividades que tienen lugar en cualquiera de las fases del ciclo de vida del software, por ejemplo, la documentación. Los entornos de desarrollo deben dar soporte tanto a las actividades verticales como horizontales.

1.1 Herramientas CASE

Las siglas CASE provienen del inglés "Computer Aided Software Engineering", que traducido al inglés podría entenderse como Ingeniería del Software Asistida por Ordenador. El uso de estas herramientas empezó a popularizarse a partir de los buenos resultados obtenidos a través de herramientas informáticas de ayuda al diseño (CAD) y a la fabricación (CAM).

Como se estudiará más adelante, las herramientas CASE pueden cubrir una o varias de las actividades de ingeniería del software. Por ejemplo, se puede acudir a una herramienta CASE para representar el modelo Entidad Relación de una base de datos, y extraer automáticamente de dicho modelo la documentación correspondiente al **diccionario de datos**.

Igual ocurre con otras áreas, como son la estimación de recursos y costes, la planificación, el diseño de datos y control, etc. De hecho, las últimas tendencias en herramientas CASE apuntan hacia la integración de diferentes sub-herramientas (correspondientes a distintas actividades) que compartan y aprovechen los resultados que cada una va obteniendo. Estas herramientas se las conocen como 'herramientas CASE integradas".

PARA SABER MÁS

Para ampliar información sobre las herramientas CASE, su historia, clasificación y ejemplos, le recomendamos visitar la web:

https://www.ecured.cu/Herramienta_CASE

1.1.1 Objetivo

Para mejorar la calidad y la productividad de los sistemas de información a la hora de construir software se plantean los siguientes objetivos:

- Permitir la aplicación práctica de metodologías estructuradas, las cuales al ser realizadas con una herramienta se consigue agilizar el trabajo.
- Facilitar la realización de prototipos y el desarrollo conjunto de aplicaciones.
- Simplificar el mantenimiento de los programas.
- Mejorar y estandarizar la documentación.
- Aumentar la portabilidad de las aplicaciones.
- Facilitar la reutilización de componentes software.
- Permitir un desarrollo y refinamiento visual de las aplicaciones, mediante la utilización de gráficos.

Imagen: Objetivos a la hora de construir un software

1.1.2 Estructura

De una forma esquemática se puede afirmar que una herramienta CASE se compone de los siguientes elementos:

- **Repositorio** (diccionario) donde se almacenan los elementos definidos o creados por la herramienta, y cuya gestión se realiza mediante el apoyo de un Sistema de Gestión de Base de Datos (SGBD) o de un sistema de gestión de ficheros.

- **Metamodelo** (no siempre visible), que constituye el marco para la definición de las técnicas y metodologías soportadas por la herramienta.

- **Carga o descarga de datos**, son facilidades que permiten cargar el repertorio de la herramienta CASE con datos provenientes de otros sistemas, o bien generar a partir de la propia herramienta esquemas de base de datos, programas, etc. que pueden, a su vez, alimentar otros sistemas. Este elemento proporciona así un medio de comunicación con otras herramientas.

- **Comprobación de errores**, facilidades que permiten llevar a cabo un análisis de la exactitud, integridad y consistencia de los esquemas generados por la herramienta.

- **Interfaz de usuario**, que constará de editores de texto y herramientas de diseño gráfico que permitan, mediante la utilización de un sistema de ventanas, iconos y menús y con la ayuda del ratón, definir los diagramas, matrices, etc. que incluyen las distintas metodologías. Los módulos de diagramación y modelización se encargan de proporcionar las utilidades para diseñar las interfaces de usuario.

Imagen: Elementos de una herramienta CASE

Algunos de los diagramas y modelos utilizados con mayor frecuencia son:

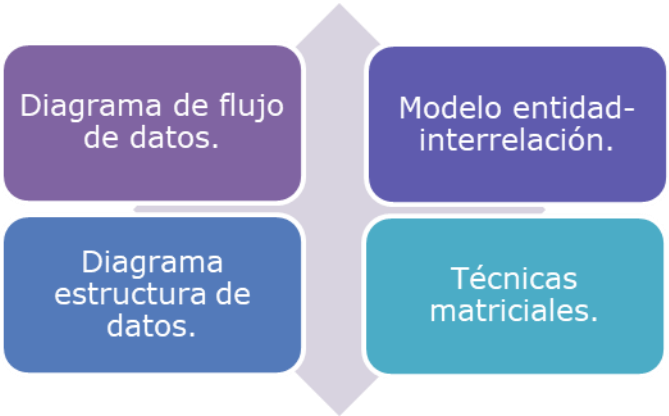


Imagen: Diagramas y modelos realizados con herramientas CASE

PARA SABER MÁS

Para ampliar información sobre el diagrama de flujo de datos y ver más ejemplos, le recomendamos visitar la web:

http://mundoinformatico321.blogspot.com/2013/02/diagrama-de-flujo-de-datos.html

1.1.3 Repositorio

Se conoce como **repositorio** a la base de datos central de una herramienta CASE. El repositorio amplía el concepto de diccionario de datos para incluir toda la información que se va generando a lo largo del ciclo de vida del sistema, como por ejemplo: componentes de análisis y diseño (diagramas de flujo de datos, diagramas entidad-relación, esquemas de bases de datos, diseños de pantallas), estructuras de programas, algoritmos, etc. En algunas referencias se le denomina Diccionario de Recursos de Información. Apoyándose en la existencia del repositorio, se efectúan comprobaciones de integridad y consistencia:

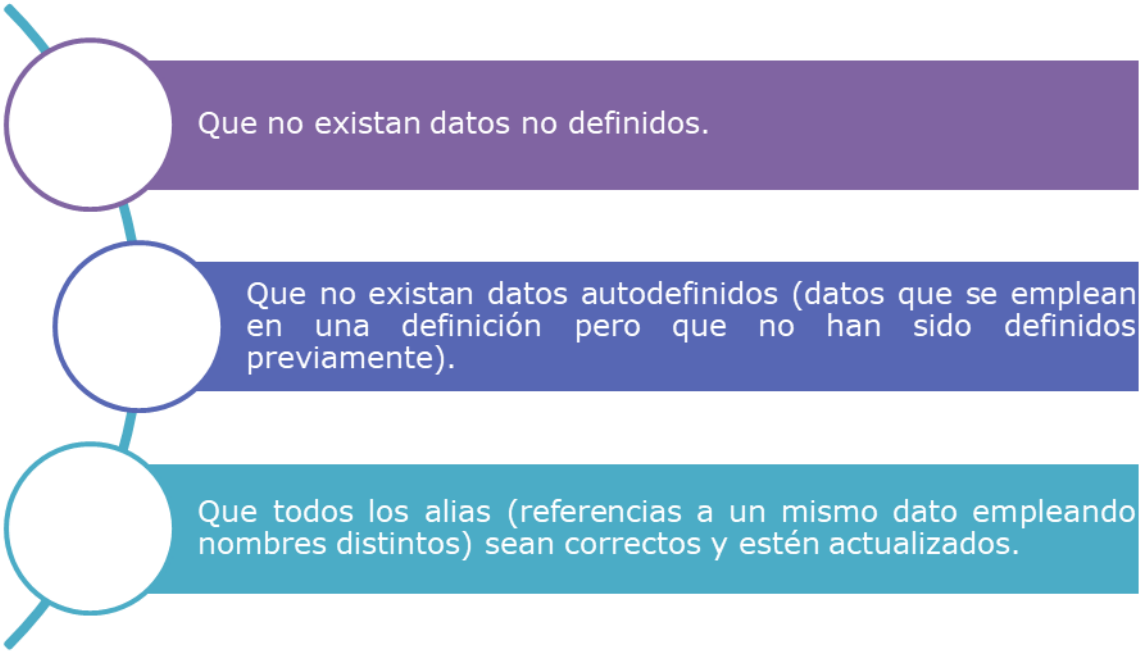


Imagen: Comprobaciones de integridad y consistencia

Las características más importantes de un repositorio son:

- **Tipo de información que contiene**: metodología concreta, datos, gráficos, procesos, informes, modelos o reglas, etc.

- **Tipo de controles**: Si incorpora algún módulo de gestión de cambios, de mantenimiento de versiones, de acceso por clave, de redundancia de la información. La gestión de cambios y el mantenimiento de versiones, ayudarán en el caso de que convivan diferentes versiones de la misma aplicación o se tengan que realizar simultáneamente cambios en la versión de producción y de desarrollo.

- **Tipo de actualización**: Si los cambios en los elementos de análisis o diseño se ven reflejados en el repositorio en tiempo real o mediante un proceso por lotes (batch). Esto será importante en función a la necesidad de que los cambios sean visibles en el momento por todos los usuarios.

- **Reutilización de módulos para otros diseños**: El repositorio es la clave para identificar, localizar y extraer distintos elementos para su reutilización.

- **Posibilidad de exportación e importación** para extraer información del repositorio y tratarla con otra herramienta (formateo de documentos, mejora de presentación) o incorporar al repositorio, información generada por otros medios.

- **Interfaces automáticas** con otros repositorios o bases de datos externas.

1.1.4 Clasificación

Generalmente se han clasificado las herramientas CASE atendiendo a la actividad o fase de la ingeniería del software a la que asisten o automatizan. En ocasiones es difícil asignar una herramienta comercial concreta a una categoría, ya que puede cubrir varias actividades distintas.

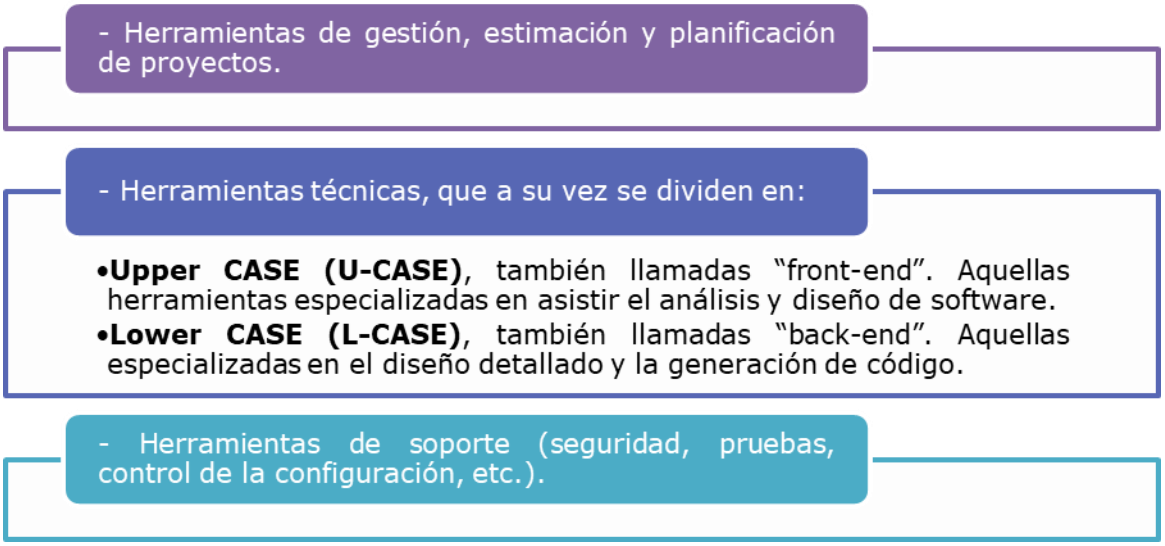


Imagen: Clasificación de las herramientas CASE

A continuación, se analizan con detalle los tipos más comunes de herramientas CASE.

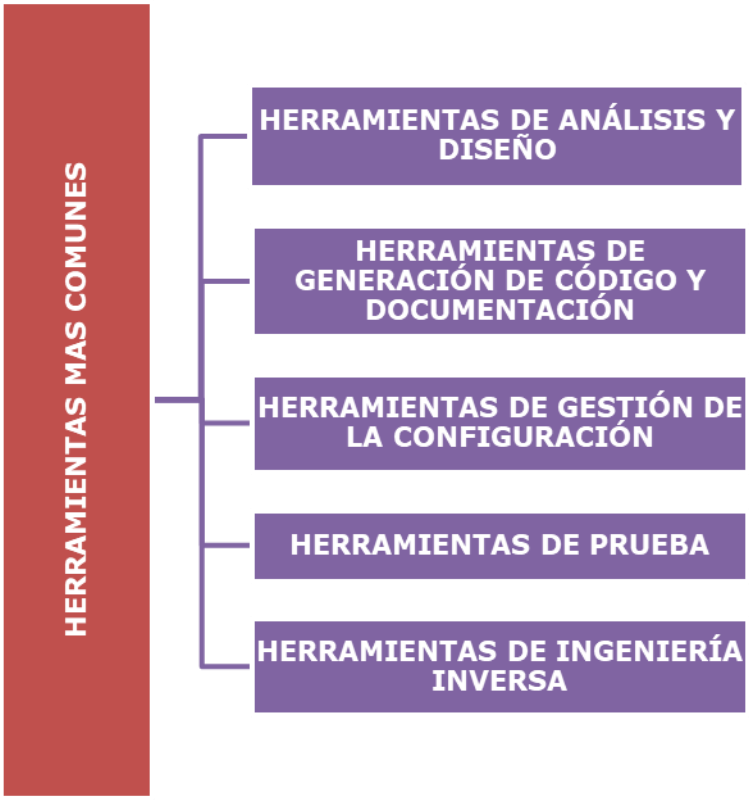


Imagen: Tipos más comunes de herramientas CASE

Herramientas de Análisis y Diseño: Las herramientas de análisis y diseño son, de entre todas las categorías, las de mayor difusión en la actualidad. Su principal objetivo es ayudar a la definición de requisitos del sistema y sus propiedades. Dentro de esta categoría destacan las herramientas que asisten las fases de análisis y diseño facilitando la aplicación de una determinada metodología. Así, estas herramientas permiten editar diagramas E/R, diagramas de flujo de datos, diagramas de clases, etc. También son muy importantes las herramientas de prototipado, como los diseñadores de pantallas, generadores de menús, generadores de informes, etc.

Herramientas de Generación de código y documentación: A partir de las especificaciones de diseño se puede generar código, tanto programas (por ejemplo, en lenguaje C o Java) como esquemas de base de datos (sentencias SQL de definición). Actualmente, las herramientas CASE ofrecen interfaces con diversos lenguajes de cuarta generación para la construcción de sistemas de forma rápida. Las herramientas CASE también soportan la creación automatizada de un conjunto muy variado de documentación, que va desde la descripción textual de un pseudocódigo hasta diagramas más o menos complejos.

Herramientas de Gestión de la Configuración: En entornos de desarrollo complejos, especialmente si se integran diversas herramientas de ingeniería de software, se hace imprescindible la incorporación de una herramienta capaz de

gestionar la configuración de los sistemas. Este tipo de herramientas ofrecen distintas capacidades:

- **Control de versiones**: Es decir, capacidad de proporcionar almacenamiento y acceso controlado a los datos por parte de uno más usuarios, así como registrar los cambios sobre los mismos, y poder recuperar versiones anteriores.

- **Trazabilidad de requisitos**: Permiten que un requisito pueda ser rastreado hasta su implementación.

- **Análisis de impacto**: Permite conocer los elementos del sistema que se ven afectados ante un cambio.

Herramientas de Prueba: Las herramientas de prueba, o CAST (Computer Aided Software Testing), son desarrollos especialmente recientes dentro de la tecnología CASE. Algunas funcionalidades que suelen ofrecer este tipo de herramientas son las siguientes:

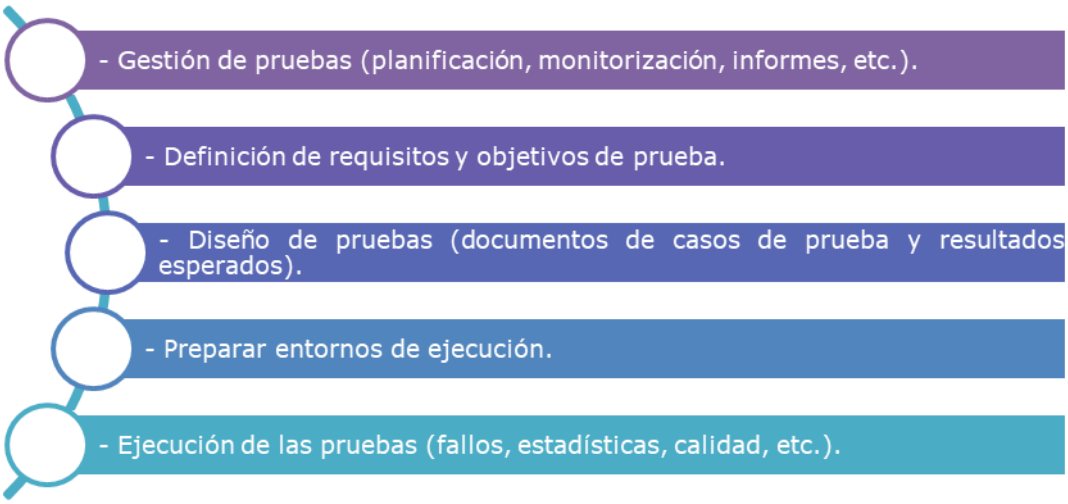


Imagen: Funcionalidades de las herramientas CAST

Herramientas de Ingeniería Inversa: El objetivo de la ingeniería inversa es obtener información a partir de un producto, con el fin de determinar cómo fue fabricado. El método se denomina así porque avanza en dirección opuesta a las tareas habituales de ingeniería. Dentro de este apartado destacan diversas herramientas, que llevan a cabo las siguientes tareas:

- **Ingeniería inversa de datos**, capaces de extraer información del código fuente que describe la estructura de los datos, por ejemplo, construyendo diagramas E/R partiendo de un modelo de datos ya implementado.

- **Ingeniería inversa de procesos**, a partir del código permiten aislar la lógica de las entidades y las reglas de negocio.

- **Reestructuración de código fuente**, modifican su formato o implantan un formato estándar.

- **Análisis de código**, cuyas funcionalidades van desde la tabulación automática del código fuente, hasta la posibilidad de ir visualizando dinámicamente las llamadas del mismo.

1.2 Entornos de Programación

Son las herramientas CASE que se encargan de dar soporte al desarrollador para la creación del código de los programas que se van a implementar. Juegan un papel importante en las fases de diseño y codificación del ciclo de vida del software, siendo útil en las fases de pruebas y explotación y mantenimiento. En los entornos de programación se tendrá soporte para la codificación, edición del código fuente, ejecución del software, compilación, creación de ejecutables, etc.

También se incluyen herramientas muy útiles como la búsqueda, el reemplazo, control de versiones, generación automática de pruebas, ejecución en modo depuración, etc.

Un entorno de desarrollo estará como mínimo formado por un editor de texto orientado al lenguaje de programación en el que se desee crear el programa, además de contener un depurador. Un entorno de programación u otro puede estar organizado de maneras muy diferentes. Los entornos de programación se pueden clasificar según el siguiente esquema:

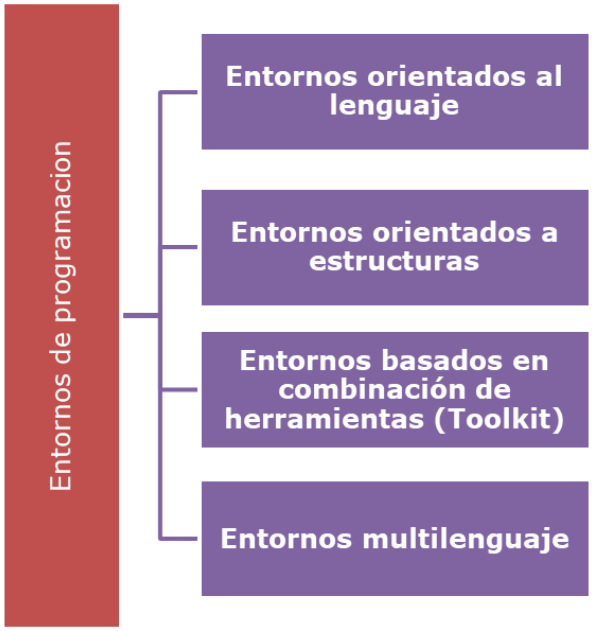


Imagen: Clasificación de los entornos de programación

1.2.1 Entornos orientados al lenguaje

Son entornos específicos de un lenguaje en particular, fuertemente integrados como una única herramienta. Su editor de código fuente está muy orientado al lenguaje de modo que cualquier fallo en alguna de las palabras reservadas del lenguaje se mostrará como un error.

Debido a esto son muy poco flexibles en la integración o interoperación con otros productos, la principal ventaja es que son muy fáciles de usar. Ejemplos: DEV-C++, VISUAL C++, DELPHI.

1.2.2 Entornos orientados a estructuras

También son orientados al lenguaje, por lo que tienen todas las ventajas e inconvenientes de estos. Además del editor de código fuente posee un editor de estructura o editor sintáctico, basado en representar internamente el código fuente de la estructura que se está implementando o editando, para crear, editar código en lenguaje XML, HTML, por ejemplo.

1.2.3 Entornos basados en combinación de herramientas (Toolkit)

Presentan una integración débil, fáciles de ampliar y adaptar a nuevas herramientas. Presentan al desarrollador un conjunto de herramientas de modo que son capaces de interoperar entre ellas. Ejemplos de este tipo de entornos: EMAX, GUIM o VIM, entre otros.

1.2.4 Entornos multilenguaje

Están a disposición de desarrolladores que necesitan crear aplicaciones que combinan código fuente en distintos lenguajes de programación. Algunas posibilidades de combinación son:

Entornos genéricos:

- No combina lenguajes en un mismo programa, sino que hay varios programas cada uno con su lenguaje de programación, siendo tarea del desarrollador combinar las correspondientes herramientas.

Entornos específicos:

- Son como los entornos orientados al lenguaje pero admiten más de un lenguaje de programación, la integración está fuertemente controlada. Ejemplos: Con GPS, se puede combinar ADA y C++.

Lenguajes ejecutables sobre máquina virtual:

- En este lenguaje, es la máquina la que establece el formato del código binario; pueden combinarse módulos escritos en diferentes lenguajes para los que exista el compilador apropiado. Pueden combinarse los diferentes entornos de programación o tener uno específico para todos ellos. Ejemplo: Eclipse.

Imagen: Entornos multilenguaje

1.3 Ejemplos de Herramientas CASE

Como se ha estudiado, existe una gran variedad de herramientas CASE con características muy específicas. A continuación, se describen algunas de ellas:

1.3.1 Microsoft Project

Microsoft Project es un software de administración de proyectos para asistir a administradores de proyectos en el desarrollo de planificaciones, asignación de recursos a tareas, dar seguimiento al progreso, administrar presupuesto y analizar cargas de trabajo.

La aplicación puede crear un calendario de rutas críticas, y reconocer diferentes clases de usuarios, los cuales pueden contar con distintos niveles de acceso a proyectos, vistas y otros datos. Los objetos personalizables como calendarios, vistas, tablas, filtros y campos, son almacenados en un servidor que comparte la información a todos los usuarios.

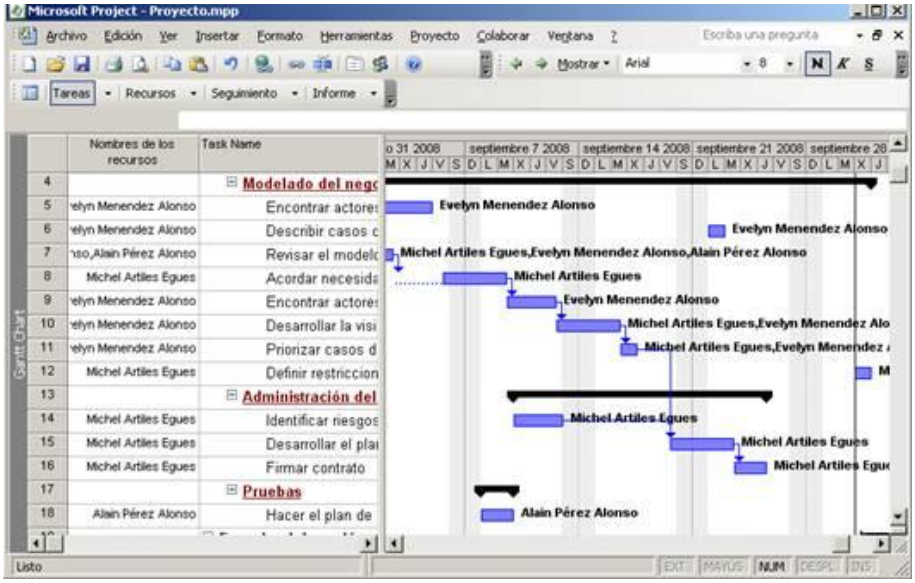


Imagen: Interfaz de Microsoft Project

1.3.2 Oracle JDeveloper

Es un entorno integrado que cubre todo el ciclo de vida de desarrollo de software e integra distintas tecnologías utilizadas para la construcción de aplicaciones empresariales estándar. Sus principales características son:

Entorno de desarrollo integrado: Integra en una única herramienta el desarrollo en Java, SOA, Web 2.0, bases de datos, XML y servicios Web, incluyendo la posibilidad de compartir elementos de la aplicación.

Soporte al ciclo de vida completo: JDeveloper cubre el ciclo de vida completo de una aplicación. Los desarrolladores pueden diseñar, generar y visualizar su código con diagramas UML, Java y de base de datos. El entorno de codificación y los editores declarativos y visuales, facilitan el desarrollo. Las pruebas integradas y las auditorías de código aseguran la calidad de la aplicación.

Framework de desarrollo: Oracle JDeveloper es el entorno de desarrollo para Oracle ADF (Oracle Application Development Framework), un entorno de trabajo completo que proporciona la infraestructura necesaria para las distintas capas de una aplicación (acceso a datos, desarrollo de servicios de negocio, control de la aplicación, interfaz gráfica, seguridad, etc.).

ENLACE DE INTERÉS

Se recomienda visitar la web oficial de Oracle Developer para ampliar información:

http://www.oracle.com/technetwork/developer-tools/jdev/overview/index.html

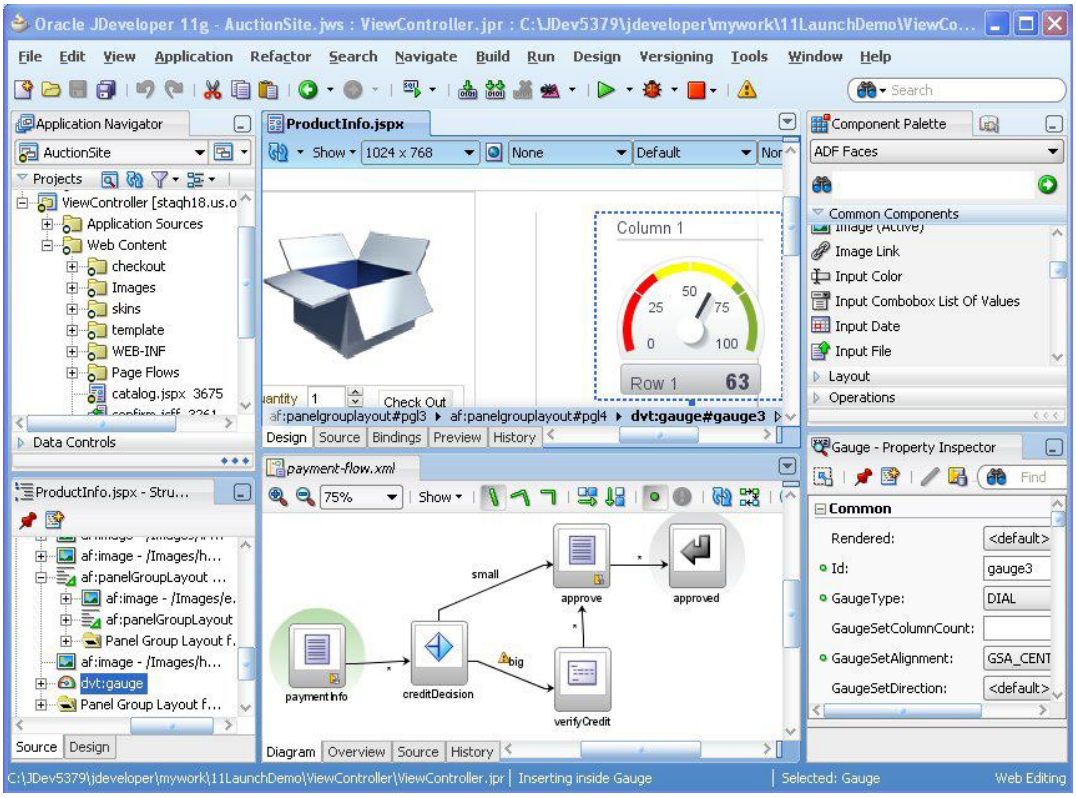
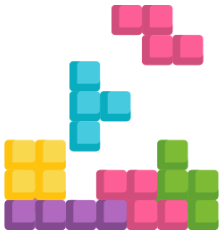


Imagen: Interfaz de Oracle JDeveloper



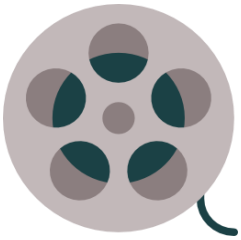
EJEMPLO PRÁCTICO

Un jefe de proyecto debe llevar un control total sobre el trabajo de su personal. Como tal, es su labor realizar una gestión del proyecto atendiendo a varios criterios: número de miembros del equipo, tiempos, retrasos, recursos materiales, etc. ¿Qué tipo de herramienta CASE será necesario utilizar por un jefe de proyecto para controlar todos estos elementos?

Solución:
Una herramienta CASE como Microsoft Project o GanttProject, que permite crear una situación actual en base a todos los elementos que intervienen en el proyecto y estimar tiempo y esfuerzo.

COMPRUEBA LO QUE SABES

En el anterior ejemplo práctico se mencionan 2 herramientas para la gestión de proyectos: Microsoft Project o GanttProject. ¿Cuál de ellos es software libre?



VIDEO DE INTERÉS

En el siguiente vídeo se muestra un ejemplo de utilización de Microsoft Project, atendiendo a: Inicio del proyecto, Vinculación de tareas de forma simple, diagrama de Gantt y cálculo de ruta crítica.

https://www.youtube.com/watch?v=5UWZFcSHdTw

NOTA: El diagrama de Gantt permite visualizar un diseño del proyecto atendiendo a las diferentes tareas y a los tiempos involucrados en ellas.

2. INSTALACIÓN DE NETBEANS

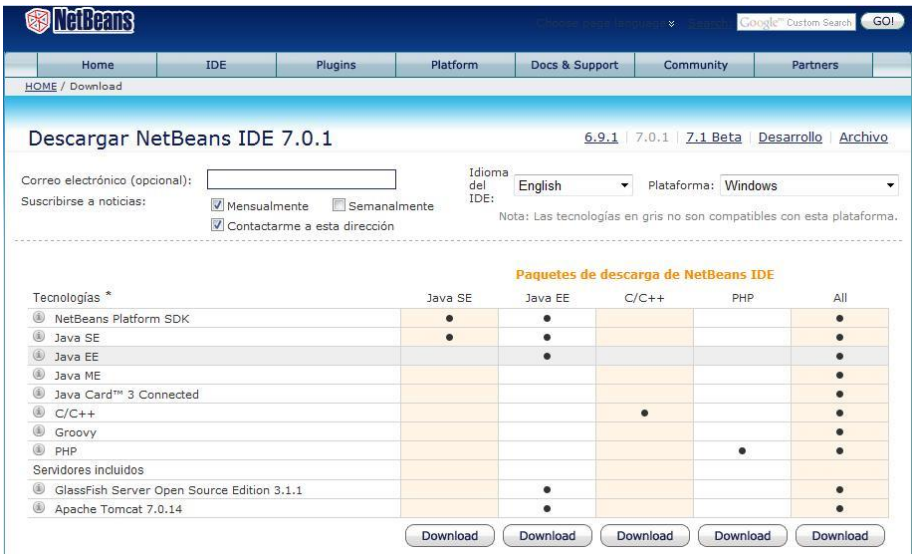


Imagen: Descarga de NetBeans

Para trabajar con varios lenguajes de programación lo ideal es descargar el programa completo, aunque se puede optar por el básico e ir añadiendo plugins:

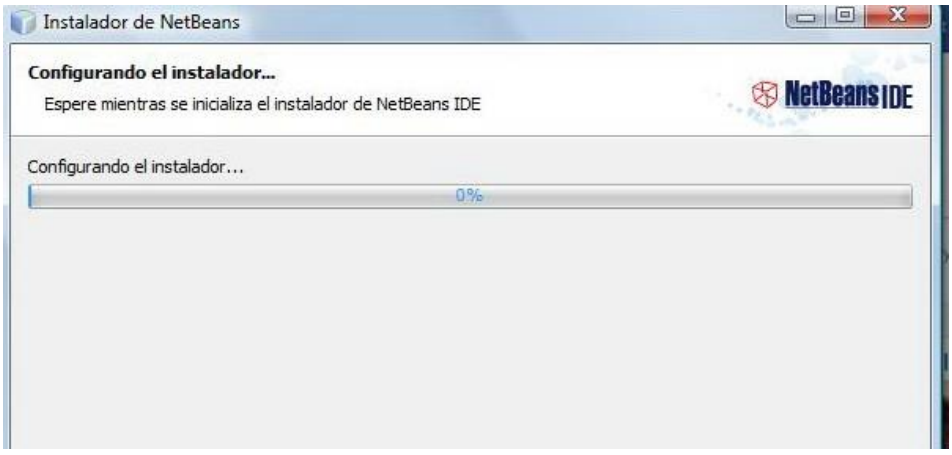


Imagen: Configurando el instalador

Una vez que se configura el instalador, se avanza a la siguiente pantalla:

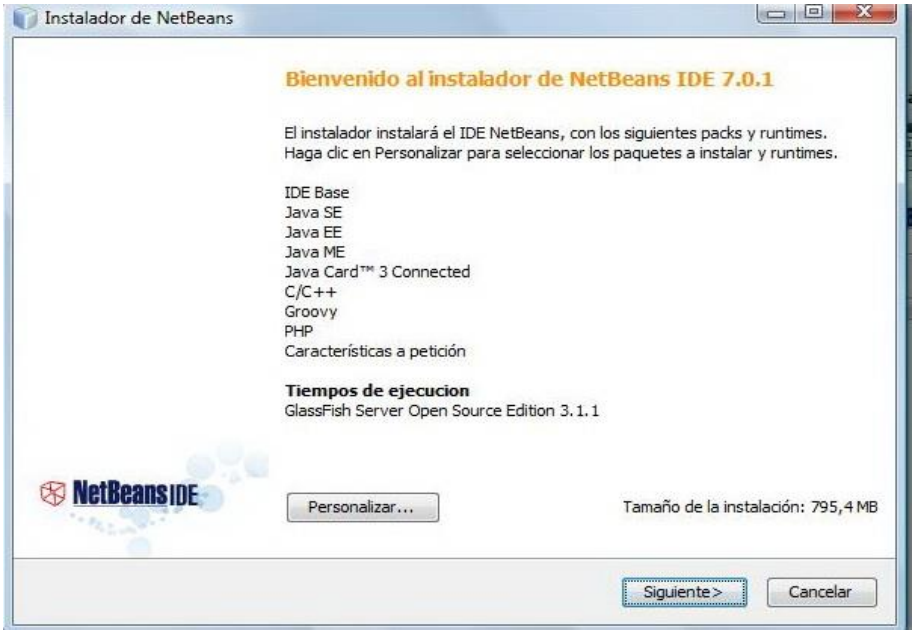


Imagen: Asistente para la instalación de NetBeans

En esta pantalla, se puede personalizar los paquetes a instalar pulsando sobre el botón Personalizar:

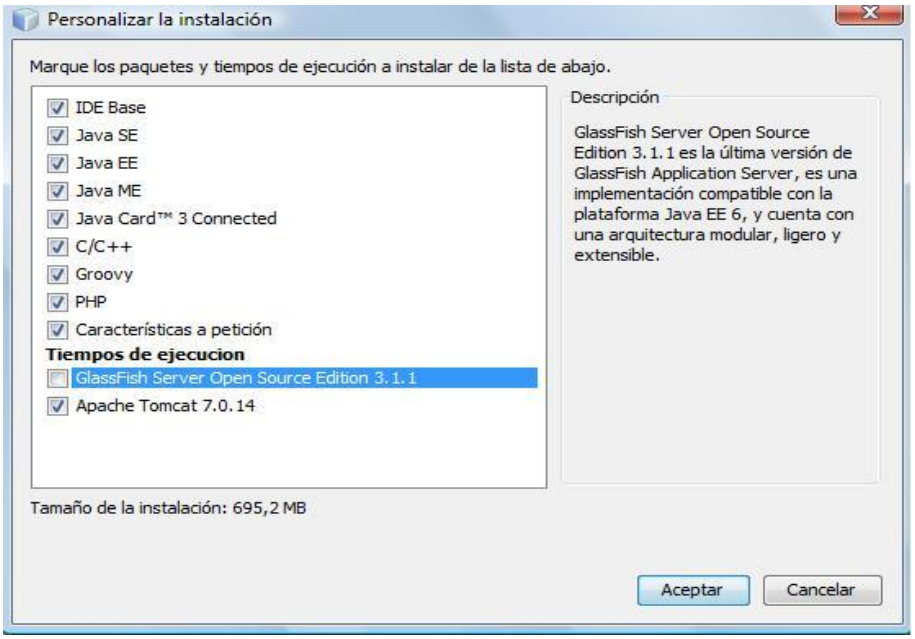


Imagen: Personalización de la instalación

Acuerdo de licencia, leer detenidamente, aceptar para continuar:

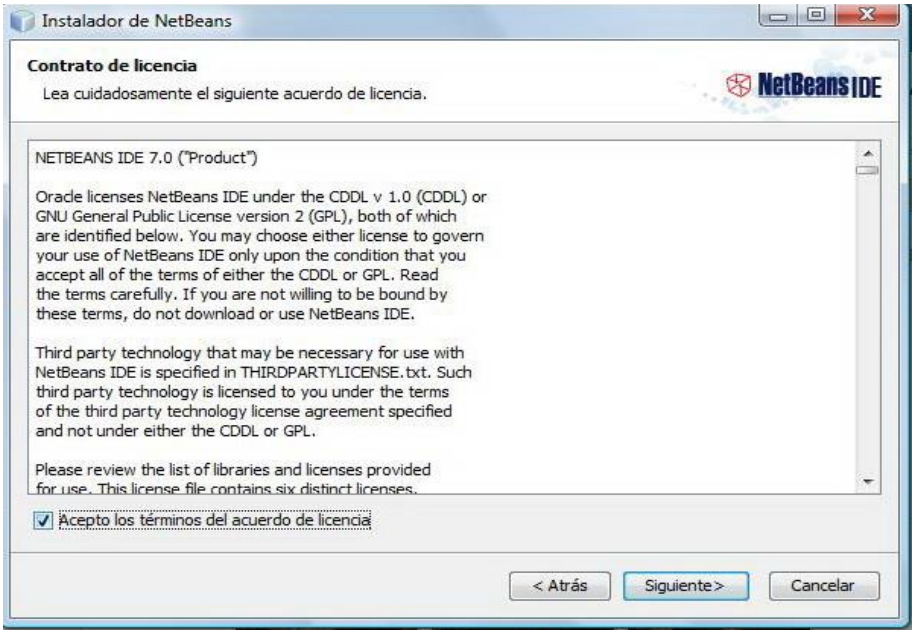


Imagen: Acuerdo de licencia

En el paquete de instalación descargado, está contenido el framework JUnit, que se usará en el tema dedicado a las pruebas. En esta pantalla, se instala dicho framework:

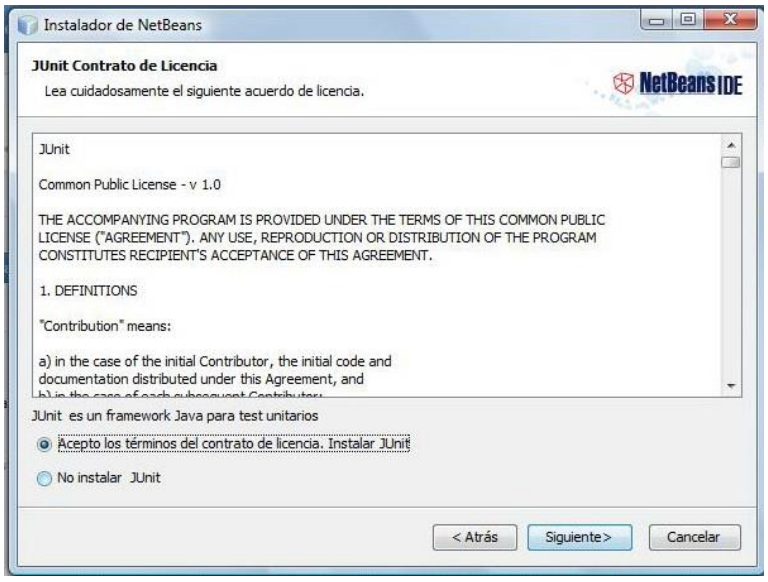


Imagen: Acuerdo de licencia de JUnit

En el siguiente paso, se especifica el directorio de instalación de NetBeans, y el JDK (Plataforma de Desarrollo) para NetBeans:

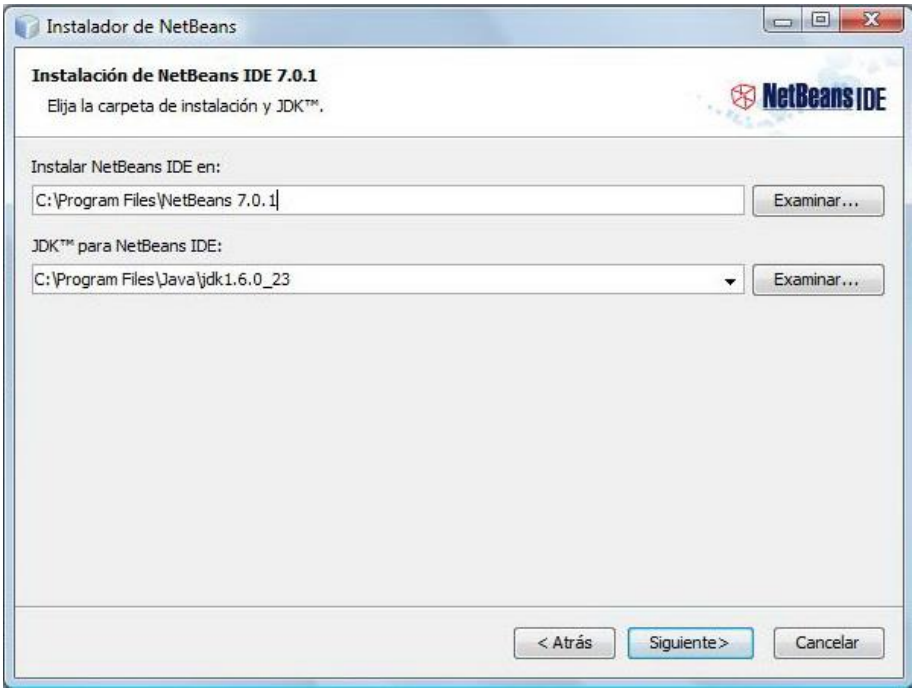


Imagen: Directorio de instalación de NetBeans

Si se ha optado por instalar el servidor GlassFish, se pedirá también el directorio donde se desea instalarlo:

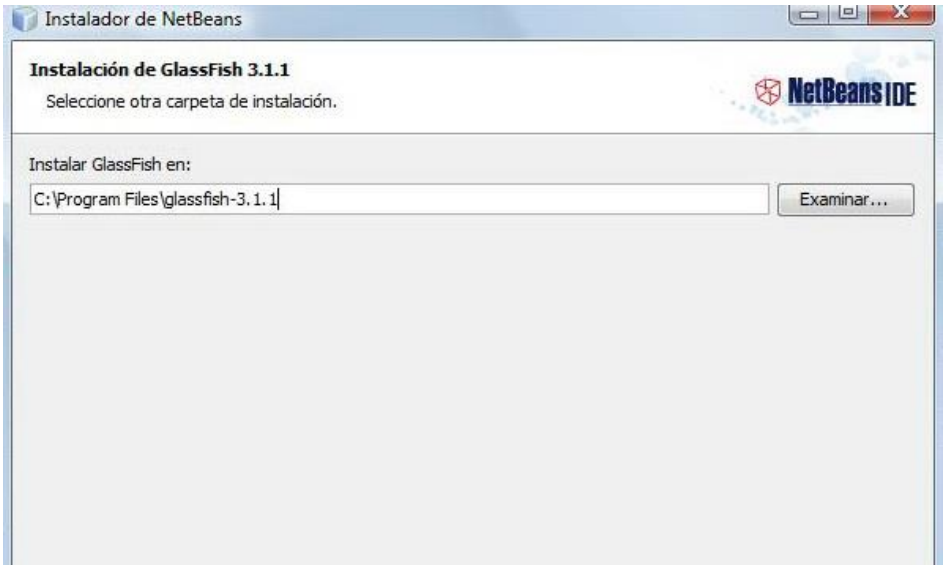


Imagen: Directorio de instalación del servidor GlassFish

A partir de aquí, comienza la instalación:

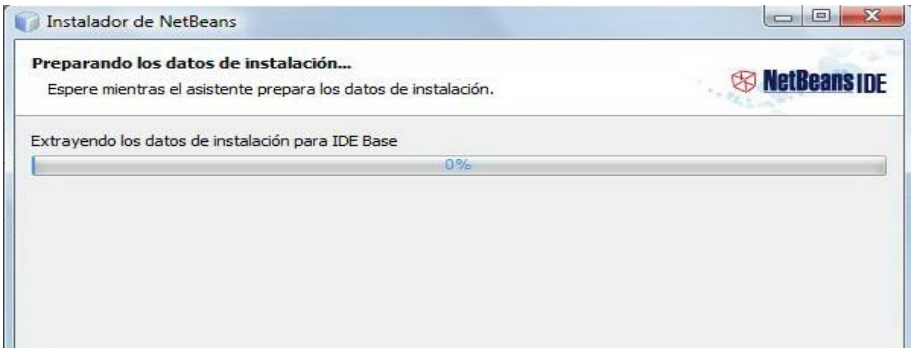


Imagen: Preparando la instalación de NetBeans

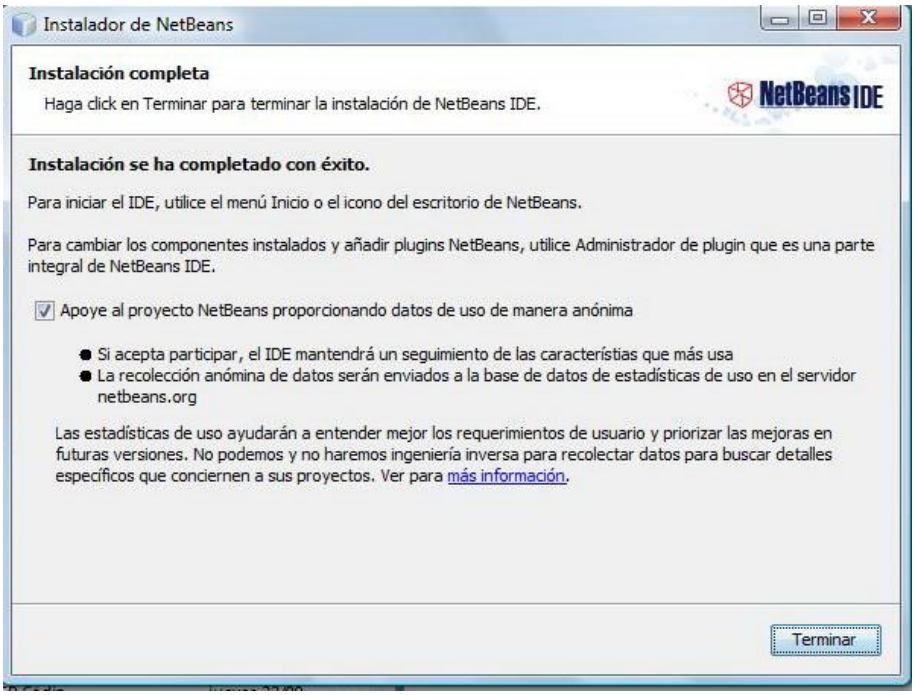


Imagen: Fin de la instalación de NetBeans

COMPRUEBA LO QUE SABES

¿Qué otro tipo de entorno de desarrollo parecido a NetBeans permite su descarga e instalación gratuita, es compatible con el lenguaje Java y presenta una interfaz bastante similar?

ENLACE DE INTERÉS

Puede descargar la última versión de NetBeans desde la web oficial:

http://netbeans.org/downloads/index.html

Actualización del entorno de desarrollo NetBeans: Se pueden añadir plugins, es decir, actualizar el entorno de desarrollo NetBeans de dos formas, manualmente y automática.

- **Agregar plugins manualmente**: Para agregar un plugin de manera manual, se deben seguir los siguientes pasos:
  Ir a Herramientas → Complementos:

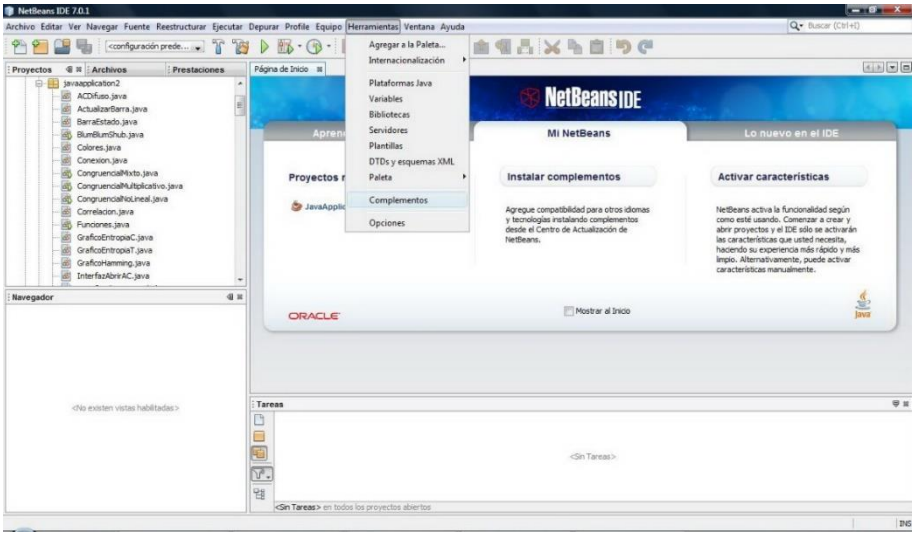


Imagen: Herramientas → Complementos

Seleccionar Descargado (archivos .nbm):

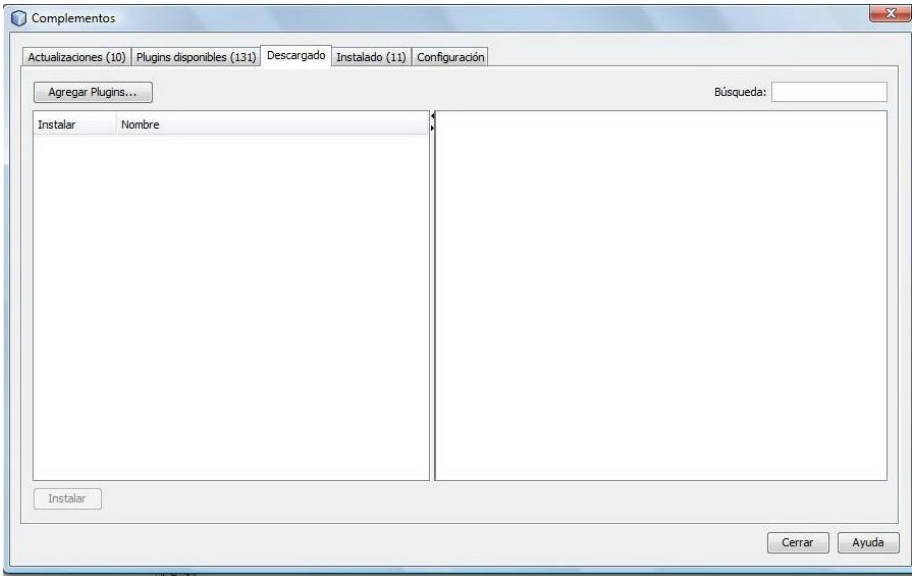


Imagen: Pestaña descargado

Clic sobre el botón Agregar Plugins → Seleccionar el archivo .nbm → Se realiza la carga de los módulos, clic el botón Instalar:

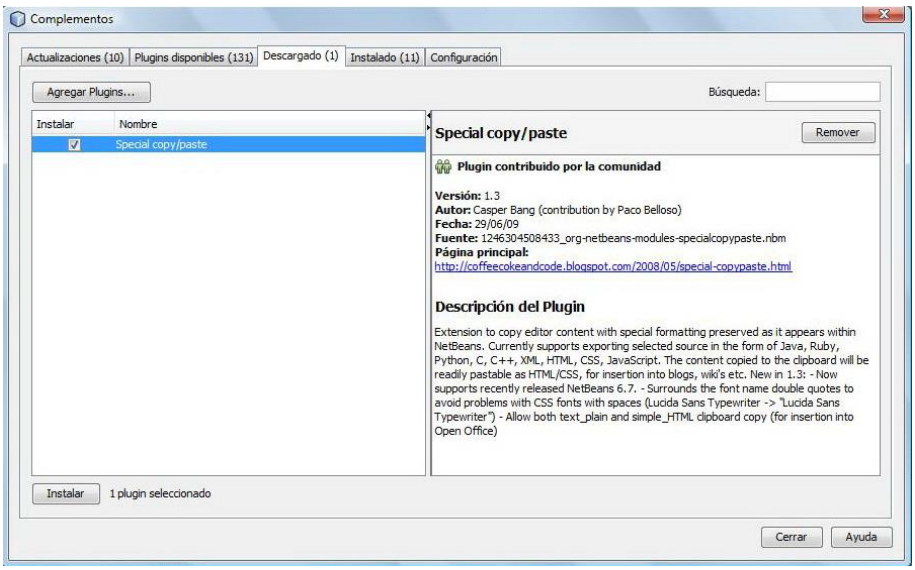


Imagen: Carga de módulos

Seleccionar la casilla Siguiente:

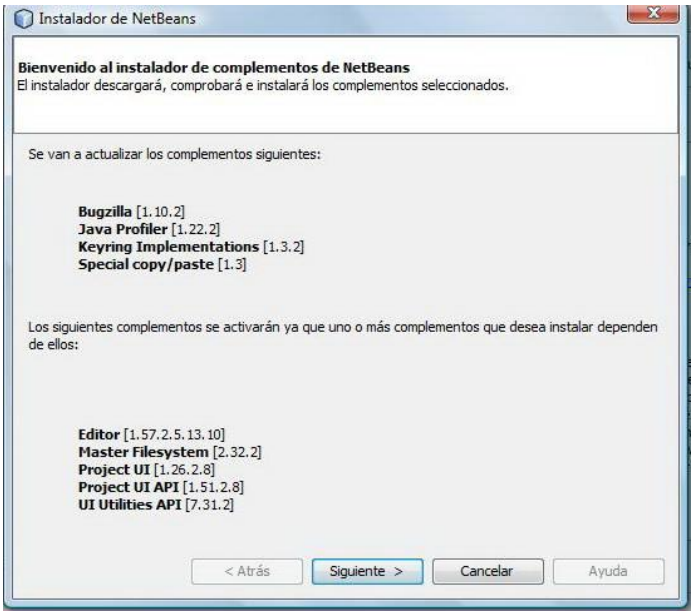


Imagen: Instalar complementos

Si el módulo no está firmado, pregunta si se está de acuerdo en firmarlo:

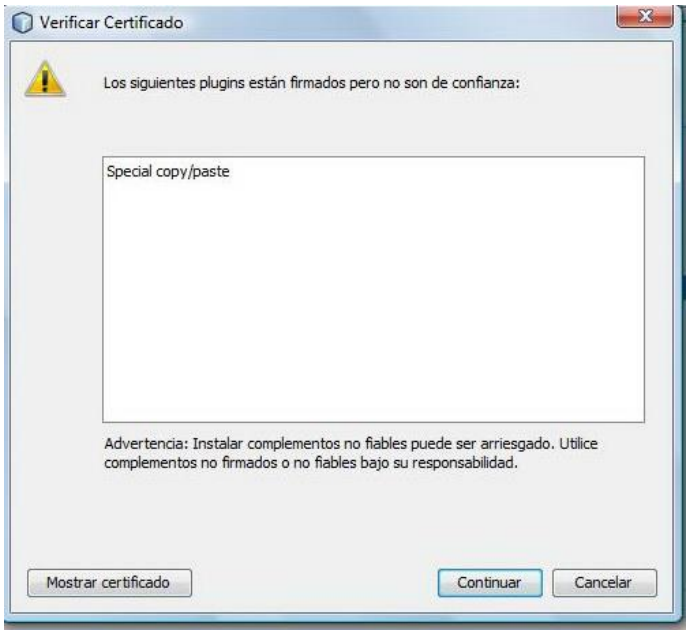


Imagen: Verificar certificado

Al final pregunta si desea Reiniciar o no:

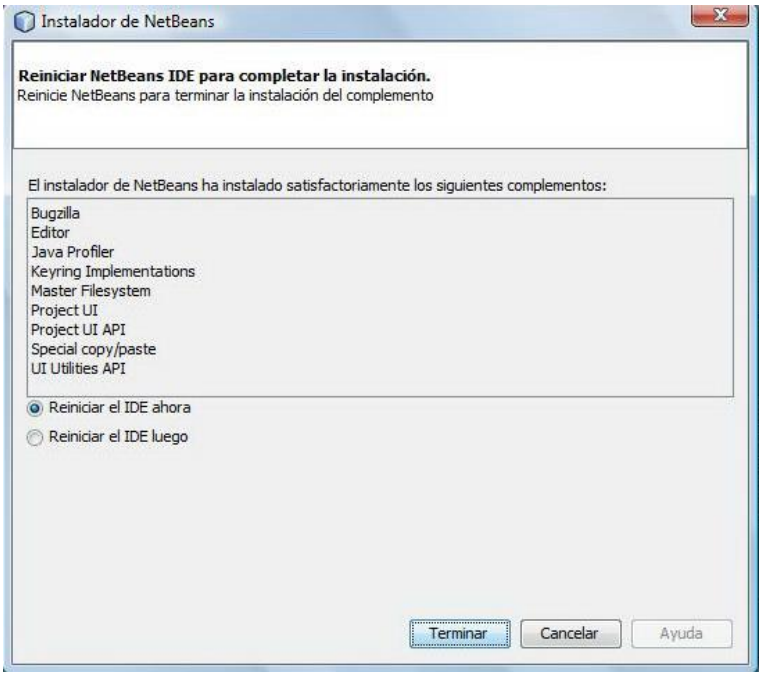


Imagen: Reiniciar NetBeans

- **Agregar plugins de manera automática**: Para agregar un plugin de manera automática, se deben seguir los siguientes pasos:
  Ir a Herramientas → Complementos → Actualización. Ahí es posible ver actualizaciones de plugins.

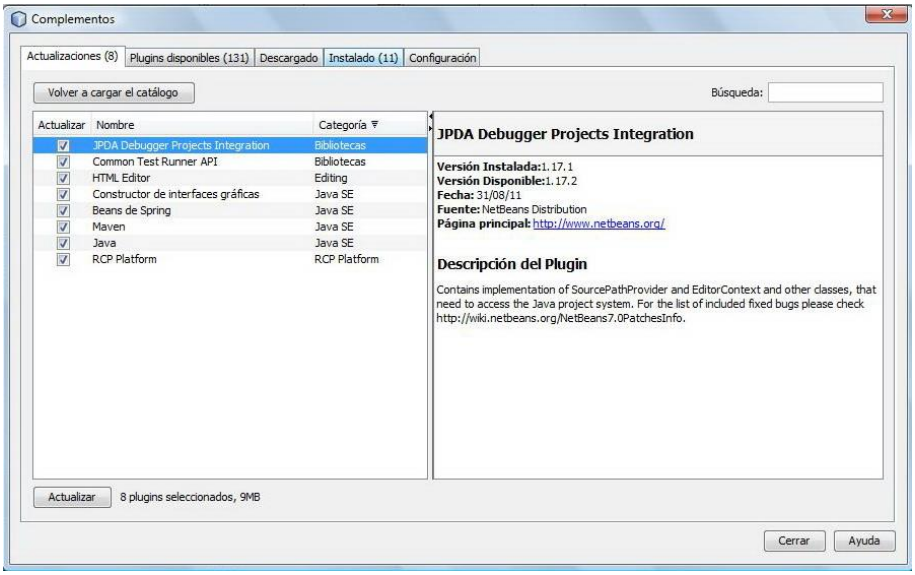


Imagen: Actualización automática

Se pueden ver plugins disponibles para instalar, ir a la pestaña de Plugins disponibles.

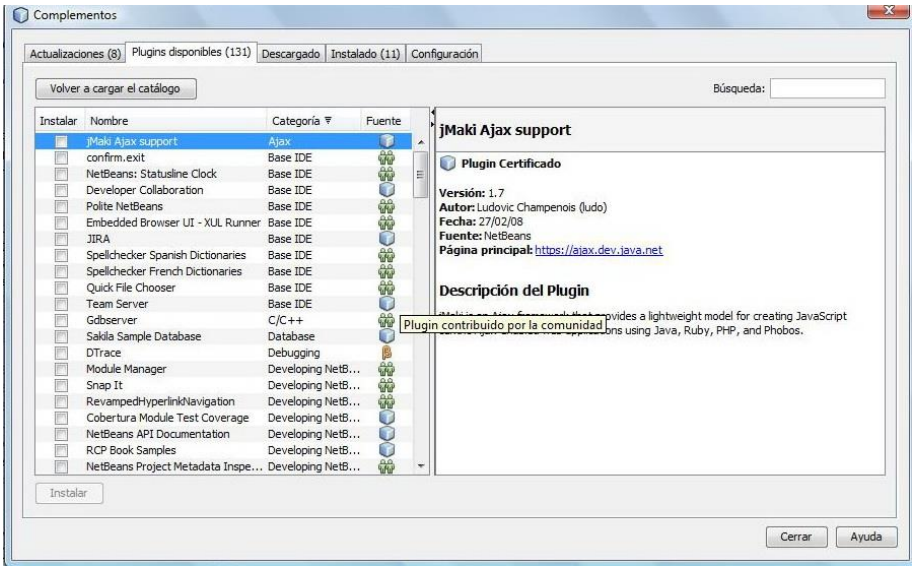


Imagen: Plugins disponibles

Por último, es posible configurar el centro de actualización para, por ejemplo, ver el periodo de tiempo en el cual se comprueba si existen actualizaciones disponibles

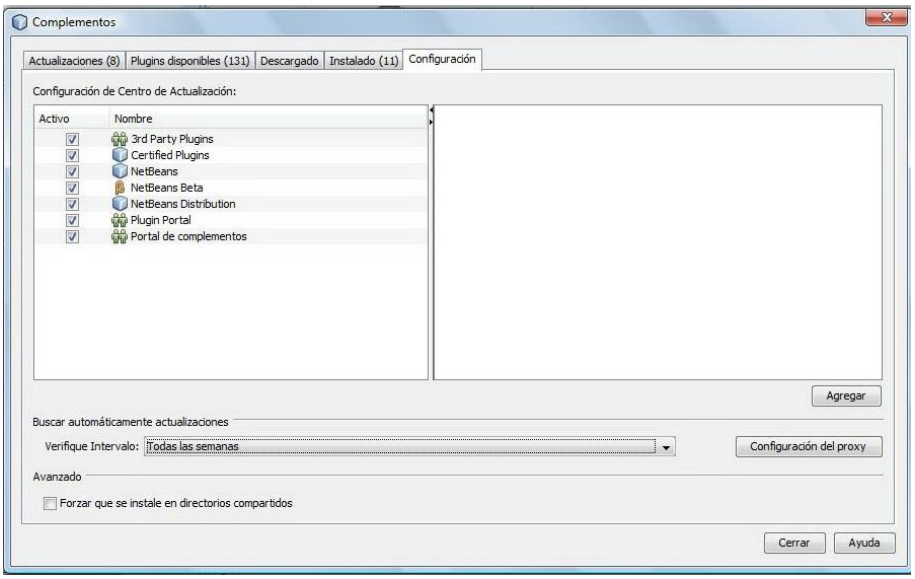


Imagen: Configurar el centro de actualización

3. REPASO DE METODOLOGÍAS

En esta sección se pretende realizar un repaso de las metodologías de desarrollo de software más habituales en las factorías de software. Hasta hace pocos años, las metodologías que se aplicaban eran las tradicionales, cuya forma de trabajar se organiza en fases las cuales han de ser implementadas siguiendo un orden establecido. En los últimos tiempos, han surgido una evolución en el desarrollo de software, con la participación activa del cliente en todo el proceso y el reparto de tareas atendiendo a la criticidad y a la prioridad. Este hecho ha originado el nacimiento de metodologías ágiles como Scrum o Kanban, entre otras.

3.1 Metodologías tradicionales de desarrollo

En presente y anterior unidad, se han detallado las diferentes fases del desarrollo software en base a la metodología tradicional en cascada. Esta metodología no es única, ya que existen otras como la espiral, incremental, etc. En el siguiente enlace se puede profundizar en todas y cada una de ellas.

ENLACE DE INTERÉS

Más información sobre las diferentes metodologías de desarrollo software tradicionales:

https://okhosting.com/blog/metodologias-del-desarrollo-de-software/

3.2 Metodologías ágiles

En los últimos años, han aparecido una serie de metodologías, conocidas como ágiles, que pretenden ayudar aún más a las empresas de cualquier sector a mejorar sus procesos de desarrollo. El nacimiento de estas metodologías radica en la necesidad de paliar algunas carencias existentes en las tradicionales, con el objetivo de desarrollar software de forma más rápida y responder mejor a los posibles cambios que puedan surgir. El llamado manifiesto ágil está formado por 4 valores y 12 principios, los cuales se listan a continuación:

3.2.1 Valores del manifiesto ágil

- Valorar a las personas y las interacciones entre ellas sobre los procesos y las herramientas.
- Valorar el software funcionando sobre la documentación detallada.
- Valorar la colaboración con el cliente sobre la negociación de contratos.
- Valorar la respuesta a los cambios sobre el seguimiento estricto de los planes.

3.2.2 Principios del manifiesto ágil

- 1. La mayor prioridad es satisfacer al cliente a través de entregas tempranas y frecuentes de software con valor.
- 2. Aceptar el cambio incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan los cambios para darle al cliente ventajas competitivas.
- 3. Entregar software funcionando en forma frecuente, desde un par de semanas a un par de meses, prefiriendo el periodo de tiempo más corto.
- 4. Expertos del negocio y desarrolladores deben trabajar juntos diariamente durante la ejecución del proyecto.
- 5. Construir proyectos en torno a personas motivadas, generándoles el ambiente necesario, atendiendo sus necesidades y confiando en que ellos van a poder hacer el trabajo.
- 6. La manera más eficiente y efectiva de compartir la información dentro de un equipo de desarrollo es la conversación cara a cara.
- 7. El software funcionando es la principal métrica de progreso.
- 8. Los procesos ágiles promueven el desarrollo sostenible. Los sponsors, desarrolladores y usuarios deben poder mantener un ritmo constante indefinidamente.
- 9. La atención continua a la excelencia técnica y buenos diseños incrementan la agilidad.
- 10. La simplicidad "el arte de maximizar la cantidad de trabajo no hecho" es esencial.
- 11. Las mejores arquitecturas, requerimientos y diseños emergen de equipos auto-organizados.
- 12. A intervalos regulares, el equipo reflexiona acerca de cómo convertirse en más efectivos, luego mejora y ajusta su comportamiento adecuadamente.

3.2.3 Metodologías ágiles existentes

Actualmente, existen una serie de metodologías ágiles implantadas en equipos de desarrollo. Entre ellas se encuentran: Agile Unified Process (AUP), Essential Unified Process (EssUP), Extreme Programming (XP), Open Unified Process (OpenUP), Dynamic Systems Development Method (DSDM), Projects in Controlled Environments (PRINCE2), Lean Software Development, Kanban y Scrum.

Una de las más extendidas y que ha revolucionado la forma de trabajar de los equipos de desarrollo ha sido **Scrum**. En las siguientes líneas se resume en qué consiste su práctica.

3.2.4 La metodología Scrum

Se trata de un marco de trabajo para proyectos, principalmente, complejos. La idea que busca es ejecutar un proyecto en iteraciones de entre 2 y 4 semanas, llamadas **sprints**, con una duración fija. Por tanto, las fechas que se estiman no pueden ser modificadas. El producto se va desarrollando de una forma evolutiva e incremental. A la hora de trabajar con Scrum se establecen una serie de **roles**:

- **Equipo de desarrollo**: Son los encargados de crear el producto. Estará formado por aquellas personas necesarias para tal fin. Este equipo se auto organiza en su trabajo. Es importante que cada miembro del equipo sea capaz de realizar cualquier función dentro del mismo y no existan especialistas de una determinada área. El equipo debe ser responsable de lo siguiente: estimar los esfuerzos de las tareas a realizar, compromiso en la construcción del producto para la estimación realizada y entrega al finalizar cada sprint.

- **Product Owner**: Es el responsable del éxito del producto. Como tal, debe atender a las siguientes necesidades: revisar el producto, realizar la toma de requisitos, generar, actualizar y mantener el plan (fechas de entrega y contenidos), determinar/cambiar las prioridades de las características de los proyectos y aceptar/rechazar el producto al final de cada sprint.

- **ScrumMaster**: Es el líder del equipo y el que acompaña en el trabajo día a día, garantizando tanto al equipo de desarrollo como al Product owner a cumplir la metodología Scrum satisfactoriamente. Se encarga de detectar y solventar cualquier tipo de problema o conflicto interpersonal dentro del equipo, así como del progreso del desarrollo de las actividades, del avance y de las comunicaciones y cooperaciones dentro del equipo.

Una vez que se disponen de estos roles, ¿cómo funciona la metodología Scrum en la práctica? El elemento principal de trabajo es el **Product Backlog**. También es conocido como Pila del Producto. No es más que una lista de ítems (Product Backlog Items, PBIs), es decir, un listado de acciones a realizar con prioridades, para establecer un orden. Estas prioridades las asigna el Product Owner.

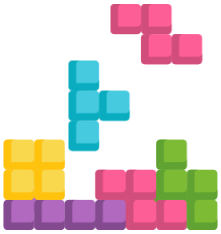
Llegado a este punto, se procede a iniciar el desarrollo de Scrum. Este inicio comienza con una reunión para planificar el primer sprint, y llegar a un acuerdo entre el Product Owner y el equipo de desarrollo sobre el alcance del mismo. En esta reunión se suelen establecer 2 pautas: una estratégica (¿qué es lo que se va a realizar?) y otra táctica (¿cómo se va a realizar?). En esta reunión, el Product Owner presenta los posibles PBIs que pueden formar parte del sprint. Será el equipo de desarrollo junto con el ScrumMaster los encargados de diseñar la forma en la que se ejecutarán estos ítems. Tras esta reunión, el equipo conforma un **Sprint Backlog** o **Commited Backlog** que representa el sprint analizado y se fija en el famoso **taskboard** (pizarra de actividades) del equipo, dando comienzo al desarrollo del producto para este sprint. En la metodología Scrum es fundamental llevar a cabo una serie de reuniones, las cuales se resumen a continuación:

- **Reunión diaria**: Facilita la comunicación entre los miembros del equipo y el ScrumMaster, se da visibilidad a los compromisos y a los problemas que hayan podido ocurrir. La duración no debería superar los 15 minutos.

- **Reunión de revisión del producto**: Al finalizar cada sprint, para que el Product Owner evalúe el producto. Al finalizar la revisión se recomienda definir la fecha de la próxima reunión de revisión del siguiente sprint.

- **Reunión de retrospectiva**: Se origina después de la reunión de revisión del producto, entre el equipo de desarrollo y el ScrumMaster reflexionar sobre el trabajo realizado y posibles mejoras y buenas prácticas a aplicar.

- **Reunión de refinamiento del Backlog**: se realiza durante el Sprint y en función de necesidades, para refinar PBIs y posibles riesgos involucrados.



EJEMPLO PRÁCTICO

En la empresa Atos acaba de entrar Javier, un informático con cierta experiencia en programación, pero nunca ha tenido la oportunidad de realizar un proyecto con Scrum. ¿Qué primera lección, puesto que acaba de llegar a la empresa, debería aprender sobre este tipo de metodología?

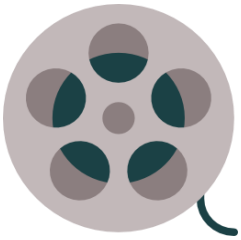
Solución:
Trabajo en equipo. Es importante inculcar en Javier un desempeño de trabajo en equipo, donde las reuniones van a tener un peso muy importante y con una frecuencia diaria. Además, Javier ha de ser consciente que este tipo de metodología es, como su propio nombre indica, ágil, es decir, flexible a cambios y propuestas que puedan surgir tanto por parte del cliente como del ScrumMaster.

COMPRUEBA LO QUE SABES

¿Podría aplicarse este tipo de metodologías ágiles en una empresa totalmente distinta al ámbito informático y de creación de software? Piense un caso para una organización como, por ejemplo, una ONG.

¿SABÍAS QUE...?

Las metodologías ágiles de gestión de proyectos son el futuro, y cada vez más, la única forma posible de adaptarse a los cambios que envuelven al ámbito de las empresas a día de hoy.



VIDEO DE INTERÉS

En el siguiente vídeo se muestra de forma esquemática como se puede implementar una metodología ágil como Scrum.

https://www.youtube.com/watch?v=PlLHc60egiQ

RESUMEN FINAL

En esta unidad se han estudiado las características y componentes de los entornos de desarrollo. Además, se ha analizado el concepto de herramienta CASE y los principales existentes en el mercado. Por último, se ha aprendido a instalar uno de los principales IDEs existentes en el mercado como NetBeans y se han analizado diferentes metodologías de desarrollo tradicionales y ágiles.