

1.A. Desarrollo del software

Sitio: [VIRGEN DE LA PAZ](#)

Curso: Entornos de desarrollo

Libro: 1.A. Desarrollo del software

Imprimido por: DIEGO ARMANDO SIMBAÑA QUISILEMA

Día: sábado, 29 de octubre de 2022, 16:36

Tabla de contenidos

1. Software y programa. Tipos de software.

2. Relación hardware-software.

3. Desarrollo del software.

3.1. Análisis.

3.2. Diseño.

3.3. Codificación.

3.4. Pruebas.

3.5. Verificación en cliente / Explotación.

3.6. Mantenimiento.

3.7. Documentación.

4. Ciclos de vida del software.

5. Herramientas de apoyo al desarrollo del software.

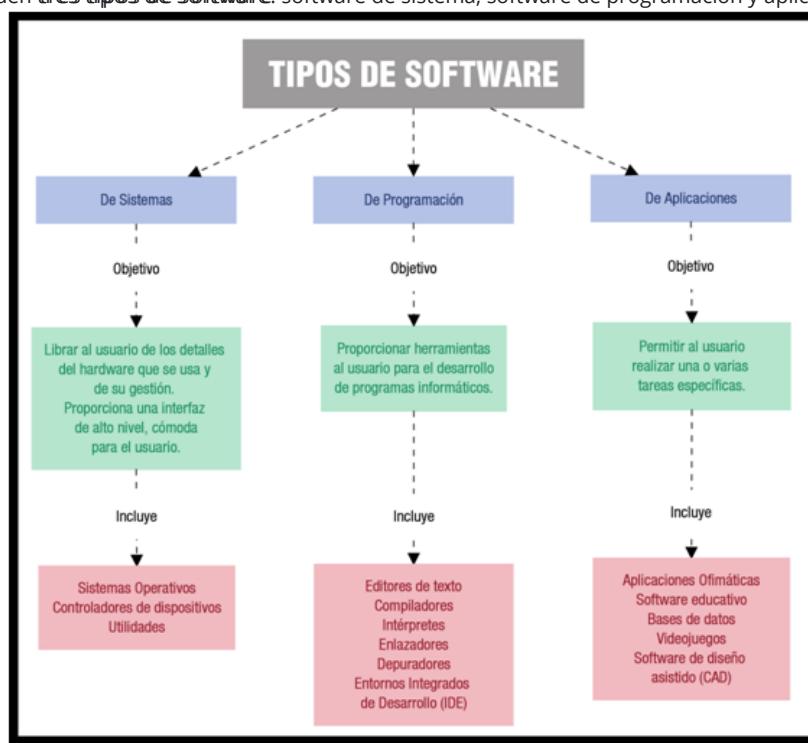
6. Frameworks.

1. Software y programa. Tipos de software.

Un ordenador se compone de dos partes bien diferenciadas: hardware y software.

El **software** es el conjunto de programas informáticos que actúan sobre el hardware para realizar una tarea específica.

Según su función se distinguen **tres tipos de software**: software de sistema, software de programación y aplicaciones.



	<p>El software de sistema es el software base que ha de estar instalado y configurado en nuestro ordenador para que las aplicaciones puedan ejecutarse y funcionar. El principal tipo de software de sistema es el sistema operativo. Algunos ejemplos de sistemas operativos son: Windows, Linux, Mac OS X ...</p>
	<p>El software de programación es el conjunto de herramientas que nos permiten desarrollar programas informáticos.</p> <p>Algunos ejemplos son los editores de texto/código, compiladores, intérpretes, entornos de desarrollo integrados (IDE).</p>
	<p>Las aplicaciones informáticas son un conjunto de programas que tienen una finalidad más o menos concreta.</p> <p>Son ejemplos de aplicaciones los procesadores de textos, las hojas de cálculo, el software para reproducir música, los videojuegos, etc.</p>

Un **programa** es un conjunto de instrucciones escritas en un lenguaje de programación, que indican a la máquina que operaciones realizar sobre unos determinados datos.

A lo largo de esta primera unidad se tratarán los conceptos fundamentales de software y las fases del llamado ciclo de vida de una aplicación informática.

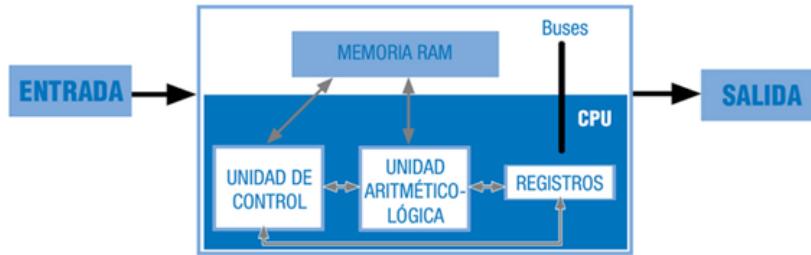
También aprenderás a distinguir los diferentes lenguajes de programación y los procesos que ocurren hasta que el programa funciona y realiza la acción deseada.

2. Relación hardware-software.

Al hablar de un ordenador, la relación hardware-software es inseparable. El software se ejecuta sobre los dispositivos físicos y éstos precisan del software para proporcionar sus funciones.



La primera arquitectura hardware se estableció en 1946 por John Von Neumann, véase en la siguiente figura los principales bloques que la conforman.



En la actualidad, los equipos todavía se basan en esos mismos conceptos.

Esta relación software-hardware la podemos poner de manifiesto desde dos puntos de vista:

- **Desde el punto de vista del sistema operativo**

El sistema operativo es el encargado de coordinar al hardware durante el funcionamiento del ordenador, actuando como intermediario entre éste y las aplicaciones que están corriendo en un momento dado.

Todas las aplicaciones necesitan recursos hardware durante su ejecución (tiempo de CPU, espacio en memoria RAM, tratamiento de interrupciones, gestión de los dispositivos de entrada/salida, etc.). Será siempre el sistema operativo el encargado de controlar todos estos aspectos de manera "oculta" para las aplicaciones (y para el usuario).

- **Desde el punto de vista de las aplicaciones**

Como ya sabemos, una aplicación no es otra cosa que un conjunto de programas y que éstos están escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar.

Hay multitud de lenguajes de programación diferentes (como ya veremos en su momento). Sin embargo, todos tienen algo en común: estar escritos con sentencias de un idioma que el ser humano puede aprender y usar fácilmente. Por otra parte, el hardware de un ordenador sólo es capaz de interpretar magnitudes físicas (ausencias o presencias de tensión, luz ...) que, en informática, se traducen en secuencias de 0 y 1 (código binario).

Esto nos hace plantearnos una cuestión: ¿Cómo será capaz el ordenador de "entender" algo escrito en un lenguaje que no es el suyo?.

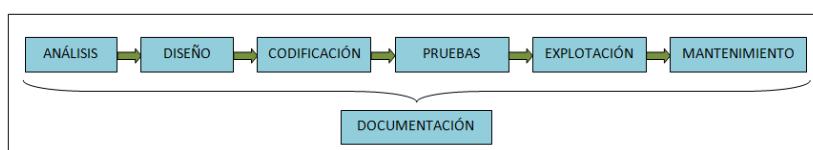
Como veremos a lo largo de esta unidad, tendrá que pasar algo (un proceso de traducción de código) para que el ordenador ejecute las instrucciones escritas en un lenguaje de programación.

3. Desarrollo del software.

Entendemos por Desarrollo de Software todo el proceso que ocurre desde que se concibe una idea hasta que un programa está implementado en el ordenador y funcionando.

El proceso de desarrollo, que en un principio puede parecer una tarea simple, consta de una serie de pasos de obligado cumplimiento, pues sólo así podremos garantizar que los programas creados son eficientes, fiables, seguros y responden a las necesidades de los usuarios finales (aquellos que van a utilizar el programa).

Como veremos con más detenimiento a lo largo de la unidad, el desarrollo de software es un proceso que conlleva una serie de pasos. Genéricamente, al conjunto de pasos se denomina ciclo de vida del proyecto. Estos pasos son los siguientes:



Es muy importante dedicar los recursos necesarios en las primeras etapas del desarrollo del software. Avanzar a las etapas finales sin un análisis y diseño libres de errores, implicará que se propaguen durante toda la vida del proyecto y como consecuencia el producto obtenido sea de mala calidad.

Estas etapas son:

Fase	Tareas
Análisis 	Analizar las necesidades de la aplicación a generar. Consensuar todo lo que se requiere del sistema, siendo el punto de partida para las siguientes etapas. Estos requisitos "deberían" ser cerrados para el resto del desarrollo.
Diseño 	Realizar los algoritmos necesarios para el cumplimiento de los requisitos planteados en el proyecto. Determinar las herramientas a utilizar en la codificación.
Codificación 	Implementar el código fuente y obtener los ficheros en código máquina que es capaz de entender el ordenador.
Pruebas 	Los elementos, ya programados, se enlazan para componer el sistema y se comprueba que funciona correctamente y que cumple con los requisitos, antes de ser entregado al usuario final.

Verificación en cliente / Explotación 	Es la fase donde el usuario final utiliza el sistema en un entorno de preproducción o pruebas. Si todo va correctamente, el producto estará listo para ser pasado a producción.
Mantenimiento 	Se trata de modificaciones al producto, generando nuevas versiones del mismo.
Documentación 	Las tareas de documentación están presentes a lo largo de todo el ciclo de vida del proyecto, por lo que muchos autores no la consideran una etapa en sí misma.

La construcción de software es un proceso que puede llegar a ser muy complejo y que exige gran coordinación y disciplina del grupo de trabajo que lo desarrolle.

3.1. Análisis.

Es la primera etapa del proyecto, la más complicada y la que más depende de la capacidad del analista. Está comprobado que los errores en esta fase son los que mayor impacto tienen en el proyecto en su conjunto.

En líneas generales, de esta fase obtendremos dos salidas:

- **Documento especificación de requisitos software**, que considerará tanto requisitos funcionales como no funcionales del sistema.
 - **Funcionales**: qué funciones tendrá que realizar la aplicación. Qué respuesta dará la aplicación ante todas las entradas. Cómo se comportará la aplicación en situaciones inesperadas.
 - **No funcionales**: tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.

Considerando un programa para la gestión de ventas de una cadena de tiendas, podríamos considerar como ejemplo de requisitos:

Funcionales	No funcionales
Si se desea que la lectura de los productos se realice mediante códigos de barras.	Los PCs suministrados deberán ser de color azul por tratarse del color corporativo.
Si se van a detallar las facturas de compra y sus formatos.	La venta online tendrá que garantizar un servicio ininterrumpido a lo largo de año. La disponibilidad deberá ser 24x7.
Si los trabajadores de las tiendas trabajan a comisión, tener información de las ventas de cada uno.	Los materiales entregables deberán cumplir la normativa requerida por la comunidad europea en el sector del comercio.
Si se desea un control del stock en almacén.	La empresa debe hacer sus desarrollos de acuerdo a algún tipo de certificación.
La interfaz tiene que ser fácil de usar para usuarios con pocos conocimientos de informática.	
.....	

- **Documento de diseño de arquitectura**, que contiene la descripción de la estructura relacional global del sistema y la especificación de lo que debe hacer cada una de sus partes, así como la manera en que se combinan unas con otras. En ocasiones este documento se genera como una de las primeras tareas de la fase de diseño.

Es imprescindible una buena comunicación entre el analista y el cliente para que la aplicación que se va a desarrollar cumpla con sus expectativas. Y habrá que asegurar que se definen aspectos como los siguientes:

- La planificación de las reuniones que van a tener lugar.
- Relación de los objetivos del usuario cliente y del sistema.
- Relación de objetivos prioritarios y temporización.
- Mecanismos de actuación ante contingencias.
-

3.2. Diseño.

En este punto, ya sabemos lo que hay que hacer, el análisis ya ha definido los requisitos y el documento de análisis arquitectónico identifica cómo dividir el programa para afrontar su desarrollo, pero ¿cómo hacerlo?.

Para cada bloque habrá que realizar un diseño en detalle, donde habrá que tomar decisiones importantes, tales como:

- Entidades y relaciones de las bases de datos.
- Selección del lenguaje de programación que se va a utilizar.
- Selección del sistema gestor de base de datos.
- Determinar los algoritmos necesarios.
- Herramientas a utilizar.

En líneas generales, de esta fase obtendremos dos salidas:

- **Documento de Diseño del Software**, que recoge información de los aspectos anteriormente mencionados.
- **Plan de pruebas** a utilizar en la fase de pruebas, sin entrar en detalles de las mismas.

Nota: en ocasiones, el diseño de arquitectura, que aquí ha sido tratado como una labor a realizar en la fase de análisis, es considerado como una primera tarea de la fase de diseño.

3.3. Codificación.

Es la fase en donde se implementa el código fuente y se obtienen los ficheros en código máquina que es capaz de entender el ordenador.

En el siguiente [enlace](#) podrás aprender mucho más sobre la codificación y sus lenguajes de programación.

3.4. Pruebas.

Una vez obtenido el software, la siguiente fase del ciclo de vida es la realización de pruebas.

Normalmente, éstas se llevan a cabo sobre un conjunto de datos de prueba, que consisten en una colección predefinida de datos límite a los que la aplicación es sometida.

La realización de pruebas es imprescindible para asegurar la validación y verificación del software construido.

Entre todas las pruebas que se efectúan sobre el software podemos distinguir básicamente:

- **Pruebas unitarias**

Consisten en probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente). Como resultado de las pruebas unitarias se genera un **documento de procedimiento de pruebas**, que tendrá como partida el plan de pruebas de la fase de diseño. Éste incluye los resultados obtenidos y deben ser comparados con los resultados esperados que se habrán determinado de antemano.

JUnit es el entorno de pruebas unitarias para Java.

- **Pruebas de integración**

Consiste en la puesta en común de todos los programas desarrollados una vez pasadas las pruebas unitarias de cada uno de ellos. Para las pruebas de integración se genera un **documento de procedimiento de pruebas de integración**, que podrá partir de un plan de pruebas de integración si durante la fase de análisis fue generado. Al igual que en las pruebas unitarias los resultados esperados se compararán con los obtenidos.

En el siguiente [enlace](#) encontrarás información interesante sobre los tipos de prueba que debemos hacer a nuestro software.

3.5. Verificación en cliente / Explotación.

Nuestro trabajo ya recoge todos los requisitos a los que tiene que dar respuesta y ha sido sometido a todo tipo de pruebas.

La verificación en cliente es la fase en donde el usuario final utiliza el sistema en un entorno de preproducción o pruebas, ya en sus propias instalaciones. Si todo va correctamente, el producto estará listo para ser pasado a producción (entorno de explotación).

El resultado de esta fase será el reporte de errores detectados o ejecutable final.

3.6. Mantenimiento.

El mantenimiento se define como el proceso de control, mejora y optimización del software tras la implantación.

Por su naturaleza, el software es cambiante y deberá actualizarse y evolucionar con el tiempo. Deberá ir adaptándose de forma paralela a las mejoras del hardware en el mercado y afrontar situaciones nuevas que no existían cuando el software se construyó.

Además, siempre surgen errores que habrá que ir corrigiendo y nuevas versiones del producto mejores que las anteriores.

Por todo ello, se pacta con el cliente un servicio de mantenimiento de la aplicación (que también tendrá un coste temporal y económico).

Su duración es la mayor en todo el ciclo de vida del software, ya que también comprende las actualizaciones y evoluciones futuras del mismo.

Los tipos de cambios que hacen necesario el mantenimiento del software son los siguientes:

- **Correctivo.** Resuelve un error detectado.
- **Evolutivo.** El cliente propone mejoras para el producto. Implica nuevos requisitos.
- **Adaptativos.** El programa precisa adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, nuevas condiciones especificadas por organismos reguladores, etc.

3.7. Documentación.

En realidad, la documentación no debe ser considerada como una etapa más en el desarrollo del proyecto, la elaboración de documentos es constante durante todo su ciclo de vida.

Documentar un proyecto se hace necesario para dar toda la información a los usuarios de nuestro software y para poder acometer futuras revisiones.

Una correcta documentación permitirá pasar de una etapa a otra de forma clara y definida. También se hace imprescindible para la reutilización de parte de los programas en otras aplicaciones, siempre y cuando se desarrollen con diseño modular.

De acuerdo al destinatario final de los documentos, podemos distinguir tres tipos:

Personal técnico en informática (analistas y programadores). Guías técnicas.	
Aspectos que quedan reflejados:	El diseño de la aplicación. La codificación de los programas. Las pruebas realizadas.
¿Cuál es su objetivo?	Facilitar un correcto desarrollo, realizar correcciones en los programas y permitir un mantenimiento futuro.

Usuarios (clientes). Guías de uso.	
Aspectos que quedan reflejados:	Descripción de la funcionalidad de la aplicación. Forma de comenzar a ejecutar la aplicación. Ejemplos de uso del programa. Requerimientos software de la aplicación. Solución de los posibles problemas que se pueden presentar.
¿Cuál es su objetivo?	Dar a los usuarios finales toda la información necesaria para utilizar la aplicación.

Personal informático responsable de la instalación. Guías de instalación.	
Aspectos que quedan reflejados:	Puesta en marcha. Explotación. Seguridad del sistema.
¿Cuál es su objetivo?	Dar toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa.

4. Ciclos de vida del software.

Se puede definir el **ciclo de vida de un proyecto** como: conjunto de fases o etapas, procesos y actividades requeridas para ofrecer, desarrollar, probar, integrar, explotar y mantener un producto software.

Al principio, el desarrollo de una aplicación era un proceso individualizado, carente de planificación donde únicamente se hacía una codificación y prueba/despuración del código a desarrollar. Pero pronto se detectaron muchos inconvenientes:

- Dependencia total de la persona que programó.
- Se desconoce el progreso y la calidad del proyecto.
- Falta de flexibilidad ante cambios.
- Posible incremento del coste o incluso imposibilidad de completarlo.
- Puede no reflejar las necesidades del cliente.

De ahí surgió la necesidad de hacer desarrollos más estructurados, aportando valor añadido y calidad al producto final, apareciendo el concepto de **ciclo de vida del software**.

Algunas ventajas que se consiguen son:

- En las primeras fases, aunque no haya líneas de código, invertir en pensar el diseño es avanzar en la construcción del sistema, pues facilitará la codificación.
- Asegura un desarrollo progresivo, con controles sistemáticos, que permite detectar defectos con mucha antelación.
- El seguimiento del proceso permite controlar los plazos de entrega retrasados y los costes excesivos.
- La documentación se realiza de manera formal y estandarizada simultáneamente al desarrollo, lo que facilita la comunicación interna entre el equipo de desarrollo y la de éste con los usuarios.
- Aumenta la visibilidad y la posibilidad de control para la gestión del proyecto.
- Supone una guía para el personal de desarrollo, marcando las tareas a realizar en cada momento.
- Minimiza la necesidad de rehacer trabajo y los problemas de puesta a punto.

Diversos autores han planteado distintos modelos de ciclos de vida, pero los más conocidos y utilizados son los citados a continuación:

- **Modelo en cascada**.

Es el modelo de vida clásico del software.

Se pasa de unas etapas a otras sin retorno posible, cualquier error en las fases iniciales ya no será subsanable aunque sea detectado más adelante.

Este escaso margen de error lo hace prácticamente imposible de utilizar. Sólo es aplicable en pequeños desarrollos.

Todos los requisitos son planteados para hacer un único recorrido del proyecto.

- **Modelo en cascada con realimentación**.

Es uno de los modelos más utilizados. Proviene del modelo anterior, pero se introduce una realimentación entre etapas, de forma que podamos volver atrás en cualquier momento para corregir, modificar o depurar algún aspecto. No obstante, si se prevén muchos cambios durante el desarrollo no es el modelo más idóneo.

Es el modelo perfecto si el proyecto es rígido (pocos cambios, poco evolutivo) y los requisitos están claros.

- **Modelos evolutivos**.

Son más modernos que los anteriores. Tienen en cuenta la naturaleza cambiante y evolutiva del software.

Distinguimos las siguientes variantes:

- **Modelo iterativo incremental**.

Está basado en el modelo en cascada con realimentación, donde las fases se repiten y refinan, y van propagando su mejora a las fases siguientes.

Cada iteración cubre una parte de los requisitos requeridos, generando versiones parciales y crecientes para el producto software en desarrollo. Cada versión obtenida será el punto de partida para la siguiente iteración.

Los incrementos a considerar en cada vuelta ya vienen establecidos desde las etapas iniciales.

- [Modelo en espiral.](#)

Se trata de un modelo iterativo como el anterior, pero en este caso, tras cada vuelta se lleva a cabo un análisis de riesgos y se determinan los objetivos a conseguir en las siguientes iteraciones.

Es un modelo muy abierto a cambios de requisitos, incluso una vez puesto en marcha el proyecto. El modelo en espiral corre el riesgo de alargarse excesivamente en el tiempo y aumentar el coste de su desarrollo.

- Modelos ágiles.

Se trata de modelos que están ganando gran presencia en los desarrollos software. Muy centrados en la satisfacción del cliente, muestran gran flexibilidad a la aparición de nuevos requisitos, incluso durante el desarrollo de la aplicación.

Al tratarse de metodologías evolutivas, el desarrollo es incremental, pero estos incrementos son cortos y están abiertos al solapado de unas fases con otras.

La comunicación entre los integrantes del equipo de trabajo y de éstos con el cliente son constantes.

Un ejemplo de metodología ágil es [Scrum](#).

5. Herramientas de apoyo al desarrollo del software.

En la práctica, para llevar a cabo varias de las etapas vistas en el punto anterior contamos con herramientas informáticas, cuya finalidad principal es automatizar las tareas y ganar fiabilidad y tiempo.

Esto nos va a permitir centrarnos en los requerimientos del sistema y el análisis del mismo, que son las causas principales de los fallos del software.

Las herramientas **CASE (Computer Aided Software Engineering)** son un conjunto de aplicaciones que se utilizan en el desarrollo de software con el objetivo de automatizar las fases del desarrollo y reducir tanto costes como tiempo del proceso. Como consecuencia, se consigue mejorar la productividad, la calidad del proceso y el resultado final.

¿En qué fases del proceso nos pueden ayudar?. En el diseño del proyecto, en la codificación, en el diseño de su apariencia visual, detección de errores...

En concreto, estas herramientas permiten:

- Mejorar la planificación del proyecto.
- Dar agilidad al proceso.
- Poder reutilizar partes del software en proyectos futuros.
- Hacer que las aplicaciones respondan a estándares.
- Mejorar la tarea del mantenimiento de los programas.
- Mejorar el proceso de desarrollo, al permitir visualizar las fases de forma gráfica.

CLASIFICACIÓN

Normalmente, las herramientas CASE se clasifican en función de las fases del ciclo de vida del software en la que ofrecen ayuda:

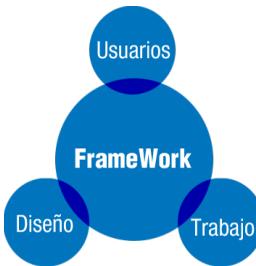
- **U-CASE (Upper-Case)**: ofrece ayuda en las fases de planificación y análisis de requisitos.
- **M-CASE (Medium-Case)**: útil en análisis y diseño.
- **L-CASE (Lower-Case)**: ayuda en la programación del software, detección de errores del código, depuración de programas y pruebas y en la generación de la documentación del proyecto.

Pulsa este [enlace](#) para ver una ampliación de los tipos y ayudas concretas de la herramientas CASE.

6. Frameworks.

Un framework es una estructura de ayuda al programador, en base a la cual podemos desarrollar proyectos sin partir desde cero.

Se trata de una plataforma software donde están definidos programas soporte, bibliotecas, lenguaje interpretado, etc., que ayuda a desarrollar y unir los diferentes módulos o partes de un proyecto.



Con el uso de framework podemos pasar más tiempo analizando los requerimientos del sistema y las especificaciones técnicas de nuestra aplicación, ya que la tarea laboriosa de los detalles de programación queda resuelta.

Ventajas de utilizar un framework:

- **Desarrollo rápido** de software.
- **Reutilización** de partes de código para otras aplicaciones.
- **Diseño** uniforme del software.

Inconvenientes de utilizar un framework:

- **Dependencia del framework.** Posible dependencia del código respecto al framework utilizado (si cambiamos de framework, habrá que reescribir parte de la aplicación).
- **Consumo de recursos.** La instalación y uso del framework en nuestro equipo consume bastantes recursos del sistema.

Ejemplos de Frameworks:

- **.NET** es un framework para desarrollar aplicaciones sobre Windows. Ofrece el "Visual Studio .net" que nos da facilidades para construir aplicaciones y su motor es el ".Net framework" que permite ejecutar dichas aplicaciones.
- **Spring de Java**. Es un conjunto de bibliotecas (API's) para el desarrollo y ejecución de aplicaciones Java.
- **Qt**. Framework multiplataforma para el lenguaje C++. Admite adaptaciones para ser utilizado en otros lenguajes.
- **Angular**. Framework de Javascript para aplicaciones web.