Elementos de desarrollo del Software

ÍNDICE

1. Introducción

En esta unidad aprenderemos a

- Reconocer la relación de los programas con los componentes del sistema informático.
- Identificar las fases de desarrollo de una aplicación informática.
- Diferenciar los distintos modelos de desarrollo.
- Clasificar los lenguajes de programación.
- Diferenciar código fuente, objeto y ejecutable.
- Reconocer las características de generar código intermedio para ejecutar en MV.
- Evaluar la funcionalidad ofrecida por herramientas usadas para la programación.

2. Conceptos básicos

2.1. ¿Qué es un sistema informático?

Un **sistema informático** es el conjunto de recursos disponibles para la resolución de problemas mediante el uso de las ciencias de la computación. Esto incluye:

- **Los equipos informáticos**: ordenadores, periféricos...
- **El software (programas) de dichos equipos**: sistemas operativos, aplicaciones...
- **Los usuarios y administradores**: las personas que utilizan dicho sistema y las que se encargan de que funcione.
- **Las relaciones entre todos estos elementos**: esto incluye las políticas de uso, de cuota...

2.2. ¿Qué es el software?

Según la definición del IEEE:

"**Software** es la suma total de los **programas de ordenador, procedimientos, reglas, la documentación asociada y los datos** que pertenecen a un sistema de cómputo" y "un producto de software es un producto diseñado para un usuario"

Un **programa** no es más que una serie de órdenes que se llevan a cabo secuencialmente, aplicadas sobre un conjunto de datos.

La **tarea de un programador** es escoger qué órdenes constituirán un programa de ordenador, en qué orden se deben llevar a cabo y sobre qué datos hay que aplicarlas para que el programa lleve a cabo la tarea que debe resolver.

RESUMEN:

- **SW**: es el conjunto de programas informáticos que actúan sobre el hardware para realizar una tarea específica.
- **Programa**: es un conjunto de instrucciones escritas en un lenguaje de programación, que indican a la máquina que operaciones realizar sobre unos determinados datos.

2.3. Tipos de Software

Se divide en tres clasificaciones

- Según la tarea que realiza
- Según el método de distribución
- Según licencias

A. Según la tarea

1. **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 o los drivers. Algunos ejemplos de sistemas operativos son: Windows, Linux, Mac OS X ...

2. **De programación:** es el conjunto de herramientas que nos permiten desarrollar programas informáticos. Algunos ejemplos son los editores de texto/código, compiladores, intérpretes, entornos de desarrollo integrados (IDE).

3. **De aplicaciones:** son un conjunto de programas que tienen una finalidad más o menos concreta. Son ejemplos de aplicaciones los procesadores de textos, las hojas de cálculo, el software para reproducir música, los videojuegos, etc.

- **B. Según el método de distribución**

1. **Shareware:** Es un tipo de software que se distribuye con limitaciones, bien como versión de demostración, de prueba o evaluación. permite que usuario pueda evaluar el software de forma gratuita por un tiempo especificado. Para adquirir la licencia completa se requiere de un pago. Ej. WinRar, Paint Shop Pro, ...

2. **Freeware:** Se considera freeware aquel software que se distribuye de manera gratuita. Suele incluir una licencia de uso que permite su distribución, pero con algunas restricciones como no modificar la aplicación. Los programas de software libre no son necesariamente freeware. No tiene que confundirse con el software libre o de código abierto, puesto que existe la posibilidad de freeware con código propietario. No confundir con FREE SOFTWARE.

3. **Adware:** Son programas gratuitos pero que incluyen publicidad de algún modo programas shareware que de forma automática descargan publicidad en nuestro ordenador (a veces puede evitarse su descarga). Al comprar la licencia del programa se elimina la publicidad.

4. **Software multimedia:** programas utilizados para presentar de forma integrada textos, gráficos, sonidos y animaciones. Usado principalmente en el ámbito educativo. Ej. las enciclopedias multimedia.

5. **Software de uso específico:** se desarrolla especialmente para resolver un problema de alguna organización o persona. Requiere de un experto en informática para su creación o adaptación. Ej. programa para la gestión de un videoclub.

- **B. Según la licencia**

1. **Software libre (Free Software):** el autor cede usa serie de libertades básicas al usuario: libertad de uso, acceso y adaptación del código fuente, distribución de copias con o sin modificaciones y libertad de mejorar el programa. La licencia más utilizada en los productos y desarrollos de software libre es la licencia GPL (General Public License - Licencia Pública General) que da derecho al usuario a usar y modificar el programa, con la obligación de hacer públicas las versiones modificadas de este. No confundir con FREEWARE.

2. **Código abierto (Open Source):** A diferencia del software libre, el software de código abierto puede distribuirse, copiarse y modificarse, pero en todo momento se necesita notificar de los cambios a los usuarios de la comunidad que soportan y apoyan este software como colaboradores.

3. **Software con Dominio Público:** carece de licencia o no hay forma de determinarla pues se desconoce el autor.

4. **Software Propietario:** Este es una forma de software muy popular y que es la que conoce la mayoría de las personas, se refiere en si a todo aquel software que no es libre.

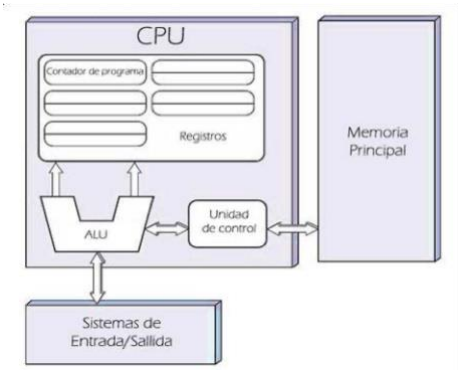
5. **Software Comercial:** El software de tipo comercial es simplemente aquel que genera ganancias monetarias a la empresa o persona que lo ha desarrollado. El software propietario casi siempre es comercial, pero en muchos casos el software comercial puede llegar a ser software libre pues pagando por él luego puedes acceder a su código fuente para modificarlo a gusto.

3. Relación Hardware-Software

- El hardware lo forman los componentes físicos.
- El software forma la parte lógica del ordenador que no se puede tocar.

Al hablar de un ordenador, la relación hardware-software es inseparable. El software se ejecuta sobre los dispositivos físicos (hardware) 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.



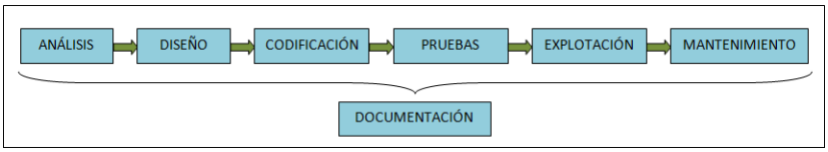
- **Memoria Principal (RAM)**: almacena de forma temporal el código binario de los archivos ejecutables y los archivos de datos necesarios.

- **CPU**: lee y ejecuta instrucciones almacenadas en memoria RAM, así como los datos necesarios. .
- **E/S**: recoge nuevos datos desde la entrada, se muestran los resultados, se leen/guardan a disco, … ¿Es el HD o memorias externas un dispositivo de E/S?

4. Ciclo de vida del software

Entendemos por **Ciclo de vida del 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). Genéricamente, al conjunto de pasos se denomina **ciclo de vida** del proyecto. Estos pasos son los siguientes:



4.1. 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 la fase de análisis se utilizarán distintas herramientas como los Diagrama de flujo de Datos (DFD), Diagramas de caso de uso, Diagrama de Transiciones, Diagramas E-R, etc. que veremos más adelante.

En líneas generales, de esta fase obtendremos dos salidas:

1. **Documento especificación de requisitos software** que considerará tanto requisitos funcionales como no funcionales del sistema.

   - **Funcionales**: Describen con detalle 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**: Tratan sobre las características del sistema, como puede ser la fiabilidad, mantenibilidad, sistema operativo, plataforma hardware, restricciones, limitaciones, tiempos de respuesta del programa, legislación aplicable, etc.

2. **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.

Actividad. Ejemplo análisis del software

Considerando un programa para la gestión de ventas de una cadena de tiendas. Analiza e indica si los siguientes requisitos indicados por el cliente al analista son funcionales o no funcionales:

a) Se desea que la lectura de los productos se realice mediante códigos de barras. F

b) Los PCs suministrados deberán ser de color azul por tratarse del color corporativo. No F

c) La venta online tendrá que garantizar un servicio ininterrumpido a lo largo de año. La disponibilidad deberá ser 24x7. No F

d) Se van a detallar las facturas de compra y sus formatos. F

e) Los trabajadores de las tiendas trabajan a comisión, por tanto, se debe tener información de las ventas de cada uno. F

f) Se desea un control del stock en almacén. F

g) La empresa debe hacer sus desarrollos de acuerdo a algún tipo de certificación o criterios de calidad. No F

h) La interfaz tiene que ser fácil de usar para usuarios con pocos conocimientos de informática. F

4.2. 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?

Se descompone y organiza el sistema en componentes o bloques que pueden ser desarrollados por separado. Para cada bloque habrá que realizar un diseño en detalle, donde habrá que tomar decisiones importantes, tales como:

- Diseño de datos: 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.
- Diseño de la interfaz de usuario.

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.

4.3. Codificación

Se escribe el código fuente de cada componente o programa que vamos a desarrollar.

Pueden utilizarse distintos lenguajes informáticos para desarrollar el código fuente:

- Lenguajes de programación: C, C++, Java, Javascript, ...
- Lenguajes de otro tipo: HTML, XML, JSON, ...

Hemos de saber que para que el programa sea entendible por el hardware el código fuente tendrá que traducirse como veremos más adelante.

4.4. Pruebas

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

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.

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

- **Pruebas unitarias:** Pruebas de integración: 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.

Ampliación pruebas

4.5. Explotación

El software que hemos generado ya recoge todos los requisitos a los que tiene que dar respuesta y ha sido sometido a todo tipo de pruebas.

La explotación o 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 por parte del cliente y ejecutable final por parte de los desarrolladores.

4.6. 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. Tipos de mantenimiento:

- **Correctivo**: se corrigen defectos.
- **Perfectivo**: se mejora la funcionalidad.
- **Evolutivo**: se añade funcionalidades nuevas.
- **Adaptativo**: se adapta a nuevos entornos.

4.7. 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 e Informática (analistas y programadores) → Guía Técnica	
Aspectos 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. ▸ Permitir un mantenimiento futuro.

Usuarios (clientes) → Guía de uso	
Aspectos 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 responsable de Instalación → Guía de instalación	
Aspectos reflejados:	▸ Puesta en marcha.
¿Cuál es su objetivo?	▸ Implantación de la aplicación se realice de forma segura, confiable y precisa.

5. Modelos de desarrollo del software

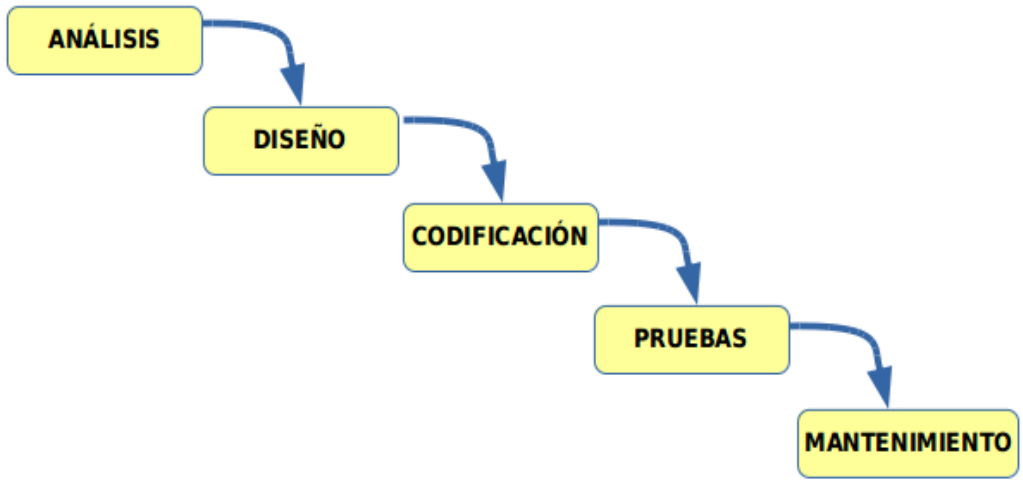
Se pueden dividir en tres grupos:

- Modelos clásicos (predictivos)
  - Modelo en cascada
  - Modelo en cascada con retroalimentación
  - Modelo en V
- Modelo de construcción de prototipos
- Modelos evolutivos o incrementales
  - Modelo en espiral (iterativos)

- Metodologías ágiles (adaptativos)

5.1. Modelos clásicos

- **Modelo en cascada:**



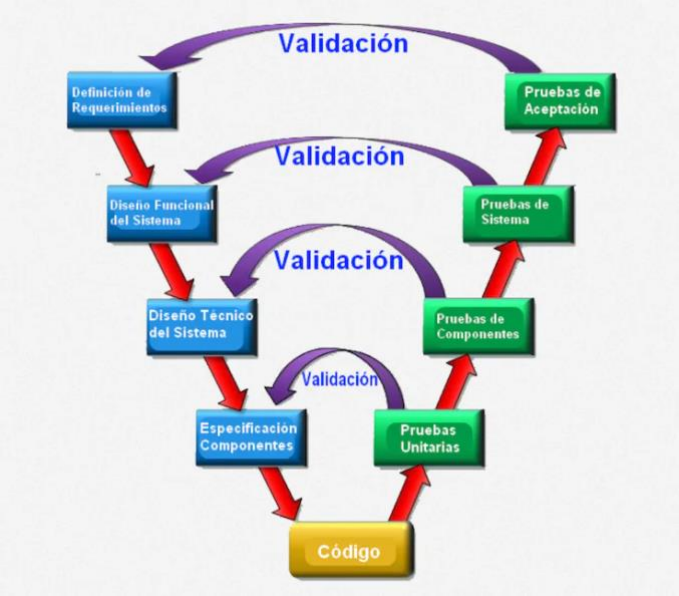
  - Modelo clásico y de mayor antigüedad.
  - Las fases han de realizarse en el orden indicado sin vuelta atrás posible.
  - Cualquier error en las fases iniciales ya no será subsanable, aunque sea detectado más adelante.
  - El resultado de una fase es la entrada de la siguiente fase.
  - Es un modelo bastante rígido que se adapta mal al cambio continuo de especificaciones.
  - Este escaso margen de error lo hace prácticamente imposible de utilizar. Sólo es aplicable en pequeños desarrollos.

Existen variantes de este modelo como podemos ver en los siguientes modelos

- **Modelo en cascada con retroalimentación:**

  - 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.

- **Modelo en V**



  - Modelo muy parecido al modelo en cascada.
  - Visión jerarquizada con distintos niveles.
  - El resultado de una fase es la entrada de la siguiente fase.

5.2. Modelos evolutivos o incrementales

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

- **Modelo en espiral (iterativos)**
  - 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.
  - Corre el riesgo de alargarse excesivamente en el tiempo y aumentar el coste de su desarrollo.

Metodologías ágiles

En 2001 se resumieron los principios sobre los que se basan los métodos alternativos en cuatro postulados, lo que ha quedado denominado como Manifiesto Ágil:

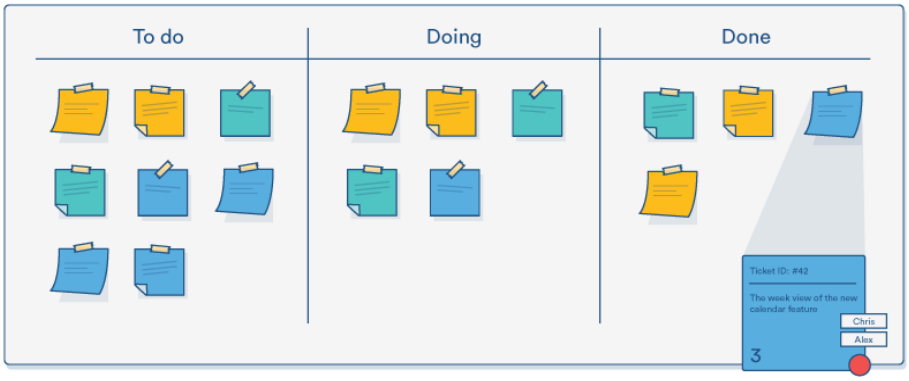
> Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:
>
> - **Individuos e interacciones sobre procesos y herramientas**
> - **Software funcionando sobre documentación extensiva**
> - **Colaboración con el cliente sobre negociación contractual**
> - **Respuesta ante el cambio sobre seguir un plan**
>
> Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.

- Son métodos de ingeniería del software basados en el desarrollo iterativo e incremental.
- Los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto.
- El trabajo es realizado mediante la colaboración de equipos auto-organizados y multidisciplinarios, inmersos en un proceso compartido de toma de decisiones a corto plazo.
- Las metodologías más conocidas son:
  - Kanban
  - Scrum

- XP (eXtreme Programming)

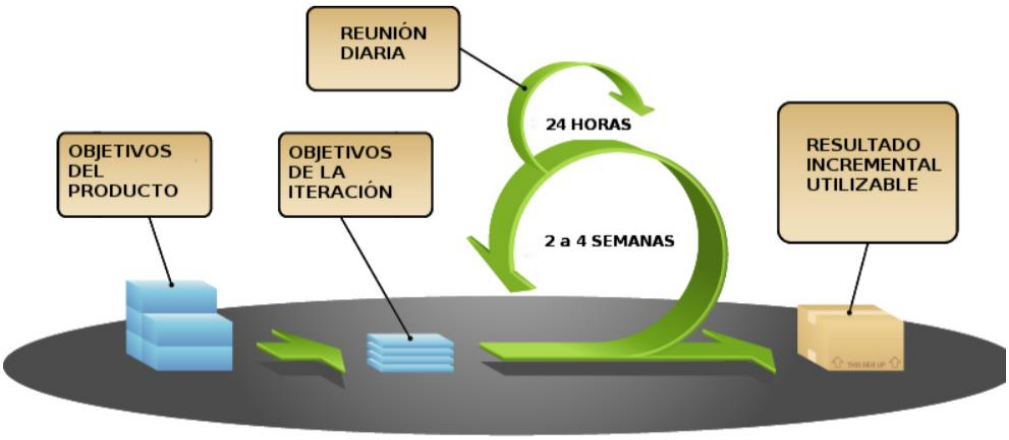
A. Kanban

- También se denomina "sistema de tarjetas".
- Desarrollado inicialmente por Toyota para la industria de fabricación de productos.
- Controla por demanda la fabricación de los productos necesarios en la cantidad y tiempo necesarios.
- Enfocado a entregar el máximo valor para los clientes, utilizando los recursos justos.
- Pizarra Kanban



B. Scrum

- Modelo de desarrollo incremental.
- Iteraciones (sprint) regulares cada 2 a 4 semanas.
- Al principio de cada iteración se establecen sus objetivos priorizados (sprint backlog).
- Al finalizar cada iteración se obtiene una entrega parcial utilizable por el cliente.
- Existen reuniones diarias para tratar la marcha del sprint.

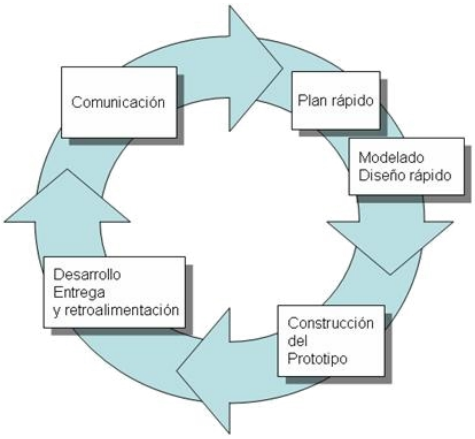


C. XP (Programación Extrema)

- Potencia las relaciones interpersonales del equipo de desarrollo como clave del éxito mediante el trabajo en equipo, el aprendizaje continuo y el buen clima de trabajo.
- Esta metodología pone el énfasis en la retroalimentación continua entre cliente y el equipo de desarrollo.
- Es idónea para proyectos con requisitos imprecisos y muy cambiantes.
- Diseño sencillo
- Pequeñas mejoras continuas
- Integración continua
- Se realiza programación por parejas
- El cliente se integra en el equipo de desarrollo
- Propiedad del código es compartida

5.3. Modelo de construcción de prototipos

Modelo adecuado cuando los requisitos no están especificados claramente: por no existir experiencia previa o por omisión o falta de concreción del usuario/cliente.



- Proceso de desarrollo:
  - Se crea un prototipo durante la fase de análisis y es probado por el usuario/cliente para refinar los requisitos del software a desarrollar.
  - Se repite el paso anterior las veces necesarias.

6. 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.

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.

Ejemplos de herramientas CASE: Microsoft Proyect, Rational Rose, JDeveloper, etc.

7. Framework

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.

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.
- **Angular**: Framework de Javascript para aplicaciones web.

Actividad. Investigación de conceptos

Tras la exposición de tu profesor no te ha quedado nada claro ciertos conceptos expuestos, y por eso, has decidido realizar una presentación para aclarar dudas a tus compañeros, que tampoco han quedado conformes con la exposición dada por el profesor de entornos.

Has decidido aclarar estos conceptos en una presentación:

1. Presentación donde aclaras los distintos tipos de licencias que ha nombrado el profesor
2. Presentación donde aclaras las diferentes arquitecturas de diseño de software usados en aplicaciones Android: MVP, MVC y MVVM
3. Presentación sobre metodologías Kanban, Scrum y XP (eXtreme Programming)
4. Presentación donde aclaras las diferentes Herramientas CASE.