Unidad Didáctica III

Introducción al software y a la ingeniería del software

*CASO. **SINUNEURO** (**S**ervicios **IN**mobiliarios de la **UN**ión **EURO**pea) es una empresa dedicada a la venta y alquiler de locales, solares y viviendas, a la que no le ha ido nada mal en los últimos años y que tras un estudio sobre su forma de trabajar, ha decido actualizar el sistema informático. Esta necesidad ha surgido al detectar errores de gestión, así como la ausencia de muchas de las funciones que han comprobado que sí se realizan en empresas similares.*

***Julia** es la persona seleccionada para recopilar las necesidades de la que será nueva aplicación de gestión de la empresa, así como de buscar y seleccionar las soluciones más adecuadas de entre las que se presenten por empresas dedicadas al desarrollo del software.*

*De entre todas las soluciones propuestas por diferentes empresas, **Julia** presentó tres al Consejo de Administración de **SINUNEURO** y éste se decidió por la solución presentada por la empresa **SI Andalucía**, empresa que se dedica a los servicios informáticos y entre ellos al desarrollo del software.*

*Dos de los especialistas de **SI Andalucía**, **José** y **María**, se reúnen con **Julia** con el fin de establecer la planificación del trabajo, los pasos a seguir y la información que necesitan para ajustar la solución por la que ha optado el Consejo de Administración.*

***Julia** tiene muy claro lo que necesita su empresa, pero no conoce nada sobre el trabajo para desarrollar una aplicación informática. **María** tras tranquilizarla, le explica que las actividades a realizar y los procesos a seguir están definidos por la Ingeniería del Software, lo que les garantiza no olvidar ninguno de los aspectos a tratar y seguir una metodología contrastada, propuesta por grandes expertos y que permite obtener software de calidad.*

El **ordenador** es sin duda una de las herramientas más útiles, que actualmente se utiliza en la mayoría de las tareas de la actividad humana, pero hay que entender al ordenador no sólo como el electrodoméstico físico (hardware) que colocamos sobre la mesa, sino que debe ir acompañado de programas y aplicaciones informáticas que le proporcionan la capacidad de ser útiles en tareas profesionales o de ocio. Es como decir; "¿para qué quieres vías si no tienes tren?". Ambos conceptos son inseparables.

Cuando se habla de actividades concretas que se pueden realizar con un ordenador, se está hablando de software. El ordenador sirve para muchas cosas, pero para hacer una factura necesitamos un software específico, igual que para tratar imágenes o sonidos, incluso para jugar. Y en cada una de estas actividades se necesita el **software** adecuado, que finalmente será el que nos permita realizar esas diferentes tareas nos facilite las cosas y nos muestre los resultados de forma clara y sencilla.

Un software de baja calidad es probable que dificulte el trabajo e incluso que proporcione resultados incomprensibles o poco claros.

Por ello a veces hay quien prefiere realizar ese trabajo de forma manual. **Con un software de calidad esto no puede ocurrir porque se adapta a las necesidades del** usuario y es creado para cubrir unas

necesidades concretas, si las cumple será útil.

Conseguir un software de calidad, con un alto grado de eficacia lleva tiempo y dinero. Esto unido a la falta de profesionales cualificados, lo convierte cada vez más en algo raro de encontrar.

Podemos **resumir** diciendo que **con un ordenador se puede hacer cualquier cosa siempre que dispongamos del software adecuado.** Realmente esto es así. Hasta hace pocos años resultaba ingenuo pensar que se pudieran automatizar tareas como la cocina, jugar al ajedrez, reparar vehículos, explorar lugares, etc.

¿Y hasta donde podemos llegar?

Lo que podemos asegurar es que la imaginación del ser humano no tiene límites y desde luego el software no se los pone, así que podemos esperar casi cualquier cosa que podamos imaginar.

PARA SABER MÁS

El software se utiliza constantemente y de forma masiva a cada momento, y la tendencia es que siga creciendo su uso en cada vez más actividades. En este enlace puedes ver cómo se ubica el software en un sistema informático y cuál es su utilidad.

¿Para qué sirve el software?

Ciclo de vida del software

Unidad Didáctica III

Software

El **software** debemos entenderlo como algo vivo, que se adapta a las necesidades del usuario y que "mejora" con el uso. Necesita datos para generar resultados útiles, con o sin la intervención de personas. Por tanto podemos decir que el software

- está compuesto por el código fuente con el que están desarrollados los diferentes programas,
- los datos con los que trabajan y
- la documentación que debe acompañar a cualquier aplicación informática. En la documentación deben establecerse claramente los objetivos (requisitos) que se persiguen y las especificaciones que ayudan a alcanzarlos.

Aunque lo parezca, no todo el software es igual.

- Hay aplicaciones concretas que llevan a cabo tareas **específicas** para una determinada empresa o un sistema de trabajo concreto y
- otras que son creadas de forma sistemática y que se emplean de forma **general** por muchos usuarios o empresas sin necesidad de ser modificados para adaptarlos a cualquiera de ellos.

Por ello podemos hablar de dos **tipos de software atendiendo a la forma en que ha sido creado**:

- **Software a medida**. Software que se adapta a las necesidades y forma de trabajar del cliente.
- **Software de propósito general**. Está desarrollado y contrastado su funcionamiento suficientemente Si alguien quiere usarlo, debe adaptarse a él.

Pero esta no es la única clasificación posible, también podemos decir que existen muchas **clases de software atendiendo a su ámbito de aplicación**:

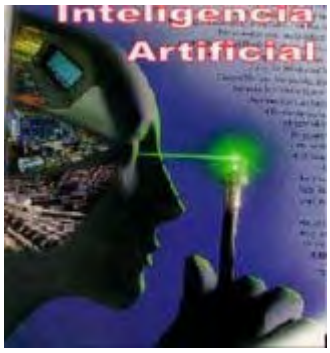
- **De sistemas**. Se trata de los programas específicos que gestionan dispositivos, tales como maquinaria industrial, electrodomésticos avanzados o cajeros automáticos. Suele ir "empotrado" dentro de los productos y sistemas de los mercados industriales que controla. Reside en memorias ROM instaladas en lavadoras, microondas y todo tipo de electrodomésticos y componentes de automóvil.

- **De tiempo real**. Se incluye en esta categoría principalmente al software que controla instrumentos

simulación de sistemas, control de vuelos, etc., en los que el tiempo de respuesta de la aplicación suele ser un factor crítico. Por **ejemplo**, el controlador aéreo no puede esperar a que la información que captura el radar tarde varios minutos en mostrarse en la pantalla tras ser procesada por la aplicación, porque en ese tiempo dos aviones pueden haber chocado o haberse estrellado. La respuesta debe ser inmediata, y la información debe procesarse de forma casi instantánea.

- **De gestión**. Básicamente incluimos en esta categoría aquellas aplicaciones que facilitan al usuario la gestión de una empresa, un proyecto o una forma de trabajar. Por **ejemplo** el software de ofimática.
- **Científico**. Las aplicaciones científicas, en especial las relacionadas con lainvestigación y el desarrollo, tienen características específicas que las hacen diferentes. Normalmente son creadas por científicos especializados, ya que requieren un gran conocimiento específico sobre una materia que va más allá de lo que puede controlar un informático sin conocimientos de la misma.

- **De Inteligencia Artificial**. La IA (Inteligencia Artificial) pretende que el software "aprenda" con la experiencia y pueda obtener soluciones, por sí mismo, a los problemas que se le plantean. Ese aprendizaje, naturalmente es simulado y se basa en una recopilación de datos que le permite ajustar más la respuesta del programa, pero no hay una verdadera adquisición de conocimiento, aunque puede llegar a parecerlo.



- **De ordenador personal**. En esta categoría incluimos todo el software que puede utilizar un usuario en casa con su ordenador personal.

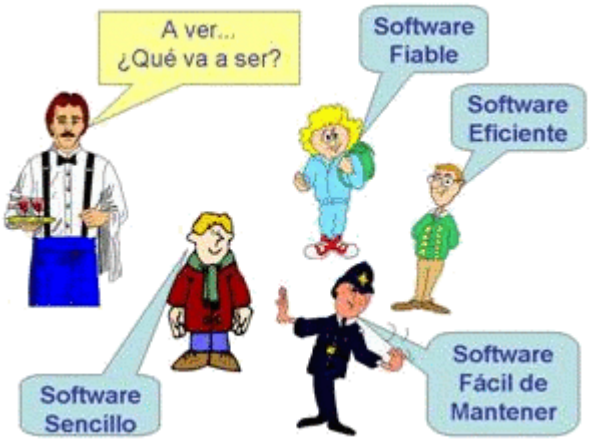
El **software** es un producto muy diferente al resto de los que consume actualmente nuestra sociedad.

¿Qué tiene de especial el software?

Pues una serie de características propias que lo hacen singular:

1. Es **desarrollado**, no fabricado.
2. Es un elemento **lógico**, no físico.
3. **Se deteriora** y no hay piezas de repuesto.
4. Se puede construir **a medida**.

Y este producto debe cumplir una serie de **requisitos** que nos van a permitir distinguir entre un buer producto y un producto mediocre que en ocasiones puede dar más problemas de los que resuelve.



Por tanto podemos decir que **el software debe ser**:

1. **Fácil de mantener**. Construido y documentado para permitir cambios sin demasiado coste n esfuerzo.
2. **Fiable**. Debe hacer aquello para lo que fue construido, sin errores y con rapidez.
3. **Eficiente**. Debe aprovechar al máximo los recursos sin utilizarlos de forma innecesaria.
4. **Fácil de usar**. La comunicación entre el software y el usuario o usuarios que lo utilicen, debe se clara, sencilla y amigable.

En definitiva **lo que debe ocurrir es que el software facilite el trabajo a los usuarios**, de modo que sea el usuario quien dirija el ordenador y no al contrario.

Y podemos preguntar... ¿Existen métodos y procedimientos para construir "buen software"?

Evidentemente la respuesta afirmativa a esta pregunta es la razón de ser de la **Ingeniería del Software**.

PARA SABER MÁS

Una definición muy completa de software, la podemos encontrar en el siguiente enlace de Wikipedia. Si buscamos en diferentes libros sobre la definición del software podemos encontrar algunas que parece que definen cosas diferentes, pero básicamente podemos quedarnos con la realizada por el IEEE. En este enlace además puedes encontrar un completo artículo sobre todo lo relacionado con el Software.

Definición de software **[Versión en caché]**

Unidad Didáctica III

Evolución del software

¿Conoces a alguien que sea "aficionado" a programar y que haya aprendido de forma "autodidacta" a hacer pequeñas aplicaciones?

Al principio el **software** era desarrollado más o menos así, o incluso peor, por virtuosos que basándose en la intuición y en la experiencia acumulada al desarrollar programas, se aventuraban a crear software que naturalmente nadie entendía ni podía modificar, a veces ni ellos mismos. El progreso en el desarrollo era incierto y muy azaroso. Y en la mayoría de los casos no se utilizaban las herramientas más adecuadas sino aquellas que se tenían más mano.

Enseguida se observó que esto del software podía ser útil y que había procedimientos y actividades más adecuados que otros para conseguir productos mejores, sobretodo basándose en la experiencia de algunos de los virtuosos anteriores.

El planteamiento era sencillo: **¿Porqué no intentar aplicar las técnicas de ingeniería empleadas para la producción de otro tipo de productos, y que tan buenos resultados daban, también al desarrollo del software?**

Con todo ello se consiguió dotar al software de una importancia que no tenía hasta ese momento y fue entonces cuando se empezó a pensar que era necesario establecer los criterios que debía cumplir el software y cómo conseguirlos.

Por estos motivos apareció **la Ingeniería del software**:

- con **profesionales bien formados**,
- que desarrollan su labor con **fundamentos teóricos**,
- siguiendo metodologías **específicas** de **reutilización de código**,
- basándose en **análisis detallados** de los problemas y sus soluciones.

La **ingeniería del software** tiene además un importante mercado detrás, que lo está convirtiendo en uno de los productos más demandado y útiles del mundo empresarial.

Esta evolución podemos observarla en el siguiente cuadro que resume la historia reciente del software como uno de los principales protagonistas de la Informática, prácticamente desde que ésta apareció. E progreso en las diferentes técnicas inventadas por científicos y expertos, ha ido abriendo nuevos caminos y líneas de investigación que han sido aplicadas a toda clase de programas.

Artesanía	Producción	Ciencia	Comercialización	Ingeniería Informática Profesional
1950		**1956**		
Se comienzan a utilizar Programas pequeños e intuitivos		IBM inventa el Fortran		
		1965		
		Algoritmos y Estructuras de Datos		
1970	**1970**	**1970**	**1970**	
Los programas grandes con éxito eran una excepción	Aceptación creciente de los métodos de programación estructurada	Aparece el primer compilador de Pascal	Aparecen las primeras empresas de servicios informáticos	
		1972		
		Aparece el concepto de lenguajes orientados a objetos		
		1978		
		Nace el lenguaje de programación C		
	1980		**1980**	
	Aparecen lenguajes de 4ª generación		Se usa Control de producción en el desarrollo del software.	
			1985	
			Se empieza a utilizar el marketing para comercializar software	
1990 ...	**1990 ...**			**1990 ...**
Se continúan haciendo programas sin metodología	Aparece la Reusabilidad en el software			- Profesionales cada vez mejor formados. - Se utilizan metodologías

de desarrollo.

- Se emplean equipos de desarrollo.

- Y se automatizan muchas de las tareas.

Ciclo de vida del software

Unidad Didáctica III

La crisis del software

¿Piensas que es fácil producir software de calidad? ¿Cuáles crees que fueron los principales **problemas** que llevaron a los desarrolladores de software a cambiar de estrategia, adoptando metodologías y desarrollando la nueva Ingeniería del Software?

Es más frecuente de lo deseable encontrar deficiencias en el software, que es necesario detectar a tiempo. Esto nos lleva a establecer métodos para diagnosticar algunos problemas. **Los problemas que suelen aparecer son**:

- Cuando aumenta la demanda del producto, **los desarrolladores no alcanzan una Productividad suficiente** y las prisas no suelen ayudar a la hora de mejorar la calidad.
- A veces **los clientes no se sienten satisfechos** con el resultado final, porque no es lo que esperaban.
- **Las aplicaciones fallan** y se rompen con cierta frecuencia, lo que puede provocar la **pérdida de datos**.
- La **baja calidad** durante el desarrollo, principalmente porque los **equipos de profesionales no están suficientemente preparados o formados**.
- Puede ocurrir que **los costes sean superiores a lo presupuestado**, lo cual suele implicar **retrasos y desconfianza**.
- **La actualización del software, suele ser muy costosa** y generalmente es preferible un producto nuevo a modificar otro existente.
- **No se cumplen los plazos de entrega**. Lo cual se achaca normalmente a problemas y fallos de última hora que nos llevan a desconfiar.
- **No suele existir portabilidad en el software** y debe ser retocado e incluso reconstruido en algunas fases para diferentes plataformas.
- Suele ser frecuente la **ausencia del aprovechamiento óptimo de los recursos** disponibles.

Cuando se dan los problemas y deficiencias anteriores, podemos decir que el software está en crisis, y los motivos más frecuentes por los que suele ocurrir esto, podemos resumirlos en los puntos siguientes:

- El **Hardware es cada vez más potente**, y esto **puede llevar a que algunas aplicaciones no funcionen correctamente**. Suele solucionarse con reajustes en la configuración de los dispositivos.
- **Aumento de la demanda por encima de la productividad** de los desarrolladores, lo que significa retrasos, prisas y disminución en la calidad por falta de rigor en la fase de pruebas.
- **Ausencia de metodologías y técnicas de análisis**. Lo que seguramente desembocará en olvidos y situaciones imprevistas.
- **Uso inadecuado de los recursos**, lo que puede incluso suponer mayor trabajo del que resuelve e software. Por ejemplo no aprovechar la existencia de una red local para usar una única impresora en una oficina y decidirse por utilizar una para cada puesto de trabajo, lo que supone multiplicar el gasto de consumibles, el mantenimiento y la ocupación del espacio físico (que repercute en la comodidad) sin que realmente se produzca ningún beneficio.
- Cuando **se incrementa la complejidad del** Sistema, es más recomendable el seguimiento de una metodología de desarrollo adecuada y más tiempo de dedicación.
- **Ausencia de cauces de comunicación adecuados** entre el equipo de desarrollo y los clientes incluso entre los miembros del equipo de desarrollo.

Todo esto lleva a entender esta crisis del software como causa de que la producción del software no sea la adecuada y de que el software que se produce no aporte el suficiente grado de satisfacción entre los usuarios, lo que resumidamente significa:

- **Baja Productividad.** No existe una producción masiva como en otros productos, en la que además no se siguen técnicas o métodos de ingeniería industrial.
- **Baja Calidad**. No cubre las expectativas de los usuarios suficientemente, es difícil de mantener y modificar, no es intuitivo o sencillo...

Unidad Didáctica III

Ingeniería del software

¿Qué hacer ante la situación descrita en el apartado anterior, donde la crisis del software claramente hacía que nadie estuviera satisfecho con los programas?

La solución para salvar esta la crisis del software, sería aplicar la Ingeniería del Software en la construcción de sistemas informáticos.

La necesidad de un enfoque de ingeniería en el desarrollo del software fue propuesta en una conferencia de la OTAN en 1968. En esta conferencia se establecen las bases de lo que se pretende conseguir con la Ingeniería del software, que podemos reducir en los siguientes puntos:

- **Reducir costes.**
- **Mejorar la calidad**.
- **Explotar y aprovechar el potencial que proporciona el hardware**.

Pero no es sencillo hacer software de **calidad; eficiente, fiable, fácil de mantener y usar** e incluso adecuado a las necesidades del cliente. El desarrollo de una aplicación es una labor de **equipo**, por lo que todo el equipo tiene que ser competente y hacer aportaciones en cada una de las diferentes fases del desarrollo del software. Esas aportaciones serán utilizadas por parte de otros miembros del proyecto en fases siguientes. Incluso es posible que ante la detección de fallos o errores sea necesario volver a fases anteriores para llevar a cabo cambios o reajustes de requerimientos.

Llegar a esta situación de producción del software ha supuesto una compleja **evolución hacia la ingeniería** a través de múltiples disciplinas. Cada una de estas disciplinas ha ido aportando elementos, métodos y técnicas para la obtención de un producto diferente a cualquiera de los conocidos, que se aplica a todo tipo de actividades empresariales o domésticas, y que va asociado al hardware, con una inevitable dependencia entre ambos.

PARA SABER MÁS

La Ingeniería del Software nos va a garantizar el desarrollo de software de calidad, siempre que tengamos en cuenta una serie de criterio a seguir. Este enlace ubica la Ingeniería del Software en este proceso.

Ingeniería del Software **[Versión en caché]**

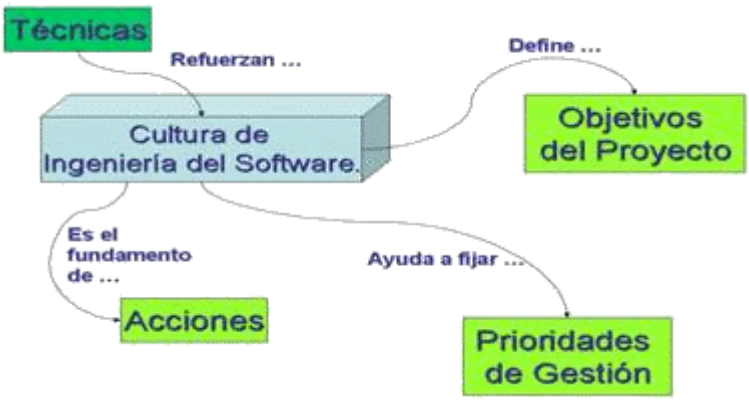
El siguiente enlace es la Página del Instituto de Ingeniería del Software que recoge todos los avances y disciplinas relacionadas para asegurar el desarrollo y operaciones destinadas a predecir y mejorar costes, planificación y calidad del Software

El principal desafío de la Ingeniería del Software es desarrollar y mantener software garantizando:

- **Calidad**. Será fácil de mantener y actualizar, aportando un alto grado de satisfacción a los usuarios.
- **Fiabilidad**. Hará aquello para lo que ha sido diseñado y proporcionará resultados correctos.
- **Facilidad de uso**. Realmente va a suponer mejoras en la actividad laboral de los usuarios, a los que les resultará sencillo su manejo, con una documentación adecuada.
- **Minimizar el mal uso**. Será muy difícil hacer un uso inadecuado del software de modo que sólo va a permitir obtener resultados correctos, habiendo previsto situaciones erróneas o imposibles.

Después de todo esto podemos concluir con algunas definiciones formales que el **IEEE** (el comité de estandarización internacional) ha establecido sobre los conceptos de Software e Ingeniería del Software:

- **Ingeniería del Software**: "La aproximación sistemática al desarrollo, operación y mantenimiento de software".
- **Software**: "programas de ordenador, procedimientos, reglas, documentación y datos asociados a un sistema de ordenador".



PARA SABER MÁS

Página oficial del Instituto internacional de estandarización. Podrás encontrar todo tipo de actividades llevadas a cabo por el Instituto, publicaciones, voluntariado, etc. Además se pueden conocer los informes que actualmente se están realizando.

Las actividades del IEEE están organizadas por regiones a nivel mundial. En el siguiente enlace puedes comprobar esta organización:

Mapa de distribución de las regiones del IEEE **[Versión en caché]**

Europa se encuadra en la región 8, junto a Oriente Medio y África. Éste el enlace a su Web.

IEEE.Región 8

A su vez cada región está estructurada en secciones. Éste es el enlace a la sección reservada para España, pero no esperes encontrarla en castellano.

IEEE. Sección de España

Este Instituto tiene un comité que trata sobre computación. Éste es el enlace para acceder a sus actividades y noticias de interés.

IEEE. Computer Society

En Wikipedia podemos encontrar algunos comentarios sobre las actividades que realiza el IEEE que pueden resultar muy esclarecedores sobre su funcionamiento y utilidad.

IEEE en Wikipedia **[Versión en caché]**

Unidad Didáctica III

Objetivos de la ingeniería del software

Quizás no te has dado cuenta, pero en los apartados anteriores **hemos asistido a la aparición de ur producto, el Software**, que cada vez es más demandado y al que cada vez se le exige más, debido a que su uso se ha extendido a una velocidad vertiginosa en la mayoría de los trabajos que se desarrollan en la sociedad actual.

Pero el **software** en general debe ajustarse a unos requisitos y pautas que han establecido los organismos internacionales, con el fin de unificar criterios y delimitar las tareas de los desarrolladores.

Hasta ahora hemos conseguido determinar los **problemas** que se presentan a la hora de desarrolla aplicaciones y desde luego, si no se siguen las técnicas y procedimientos adecuados, el producto obtenido suele tener una baja calidad y presentar problemas.

Hemos conseguido diagnosticar y detectar estos problemas del Software y hemos visto que su solución es la Ingeniería del Software.

Por todo ello **es conveniente establecer claramente los objetivos de la Ingeniería del Software**:

- Como cualquier ingeniería,... **construir instrumentos que ayuden o faciliten al ser humano la realización de alguna tarea.**
- **Conseguir un producto software fiable, de alta calidad y bajo coste.**
- **Conducir el proceso de desarrollo y mantenimiento software de manera eficiente y con éxito.**
- **Mejorar la calida de los productos software.**
- **Aumentar la productividad de los desarrolladores.**
- **Facilitar el control y seguimiento del proceso de desarrollo.**
- **Suministrar a los desarrolladores las bases para construir software de alta calidad de forma eficiente.**
- **Definir una disciplina que garantice la producción sistemática y el mantenimiento de los productos software desarrollados en el plazo fijado dentro del coste estimado.**

Ciclo de vida del software

Unidad Didáctica III

Fundamentos de la Ingeniería del Software

¿En qué se basa la Ingeniería del Software para conseguir esos objetivos, teniendo en cuenta las peculiaridades del producto que se desarrolla?

La Ingeniería del Software no es como el resto de ingenierías, principalmente porque:

- Se basa en un producto **intangible**,
- Es de muy **reciente** aparición y
- Utiliza muchos **menos recursos** comparada con cualquiera de las otras ramas de la Ingeniería.

En general las ingenierías construyen instrumentos que imitan, aumentan, ayudan, facilitan o sustituyen capacidades físicas del ser humano. **La Ingeniería del Software además puede simular capacidades mentales del ser humano, permitiendo la resolución de problemas o ayudando en esa tarea.**

Pero la ingeniería del software es mucho más que **Ingeniería**,
necesita de las aportaciones de otras disciplinas humanas con el fin de conseguir un producto de calidad
Ya hemos visto su relación con la rama de ingeniería, pero es importante también las aportaciones de:

- La **Ingeniería,** evidentemente.
- La **Gestión** de recursos o personal en un proyecto,
- Las técnicas de **Economía** de tiempos (ajuste de plazos) o de costes (ajuste de presupuestos),
- La participación de la **Informática** es indiscutible por el uso de técnicas, metodologías y herramientas.
- Es imprescindible finalmente una gran dosis de **Creatividad**, especialmente en proyectos innovadores y de gran envergadura, aplicada a cada una de las cuatro disciplinas anteriores.

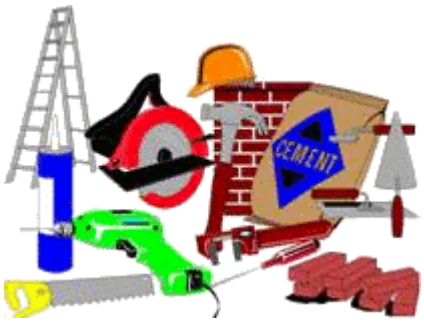
Visualiza una demo sobre las disciplinas implicadas en la ingeniería del software

La Ingeniería del Software es una tecnología estratificada en capas bien definidas sobre un enfoque de calidad, que permiten cubrir las necesidades del equipo de desarrollo. Así, podemos decir que disponemos de técnicas que implican **Métodos, Procedimientos y Herramientas**.

Un determinado conjunto de estas técnicas es lo que se denomina **Paradigma de la Ingeniería del Software** o **Ciclo de Vida del Software**. La elección de un ciclo de vida del software concreto va a depender de la naturaleza del proyecto y de la aplicación a desarrollar.

- Los **Métodos** definen cómo construir el software desde el punto de vista técnico.
  - **Planificación y estimación de proyectos**. Fase inicial que permite establecer plazos a cumplir y recursos a utilizar durante el proyecto.
  - **Análisis de requisitos**. Que va a concretar las necesidades del usuario y cuáles se pueden llevar a cabo y de qué modo.
  - **Diseño**. Va a permitir especificar cómo solucionar las necesidades del cliente y cómo llevar a cabo dichas soluciones.
  - **Codificación**. Consiste en la elaboración del programa de ordenador que sintetiza dichas soluciones mediante la programación del código.
  - **Pruebas**. Fase durante la que se realizan las pruebas que permitan asegurar que el software funciona adecuadamente.
  - **Mantenimiento**. Una vez que la aplicación informática está funcionando en un sistema real es necesario hacer un seguimiento periódico para concretar ajustes y solucionar cualquier problema que pudiera surgir.

- Las **Herramientas**, proporcionan un soporte automático o semi-automático para los métodos Ayudan a automatizar muchas de las tareas anteriores.



  - **Herramientas CASE**. ***C****omputer* ***A****ided* ***S****oftware* ***E****ngineering*. (Ingeniería del Software Asistida por Computador) Las herramientas CASE permiten realizan dentro del ordenador las tareas de análisis y diseño que hasta entonces venían haciéndose con lápiz y papel a lo sumo con la ayuda de editores de texto y de gráficos no pensados para desarrollar y organizar los diferentes elementos de un proyecto informático.
  - **Herramientas CAD.** ***C****omputer* ***A****ided* ***D****esign* (Diseño Asistido por Computador). Básicamente se centran en tareas de diseño.

En capítulos posteriores veremos en detalle estos tipos de herramientas.

PARA SABER MÁS

Las Herramientas CASE han adquirido una gran importancia en los últimos años, ya que han conseguido facilitar considerablemente las tareas de planificación y desarrollo, minimizando costes y tiempo, permitiendo la reutilización y aumentando la calidad del producto final.

Wikipedia. Herramientas Case **[Versión en caché]**

Clasificación de las herramientas CASE por categorías

Las Herramientas CAD pretenden ayudar al equipo de desarrollo en el diseño de aplicaciones. Pero esto no es exclusivo de la ingeniería del software. En el siguiente enlace puedes comprobar cómo proliferan este tipo de herramientas en cualquiera de las ingenierías.

Comentarios sobre herramientas CAD

- Finalmente los **Procedimientos**, son el punto de unión entre métodos y herramientas y definen:
  - La **secuencia** en la que se aplican los métodos,
  - Cómo usar las **herramientas**,
  - Las **entregas** que se requieren,
  - **Controles** de seguimiento y calidad,
  - **Guías** para facilitar la labor de gestores y desarrolladores,
  - etc.

En la Ingeniería del Software además se suelen emplear algunas **técnicas de apoyo** que facilitan:

- la planificación de las tareas,
- la representación de los datos y estados del proyecto y
- la evaluación a la hora de tomar decisiones

Esas técnicas de apoyo son:

1. **Abstracciones**. Con estas técnicas es posible crear modelos virtuales sobre cómo será el producto final y cómo será el proceso de desarrollo de ese producto basándose en uno de los Modelos de Ciclo de Vida del Software.
2. **Representaciones**. Técnicas utilizadas en las diferentes fases de desarrollo para la planificación de tiempos y recursos, los procesos a seguir, la estimación de costes, tareas del personal, etc. Podemos citar; las diferentes notaciones y símbolos, Diagramas de Gantt, Lenguajes de programación...
3. **Evaluaciones**. Que van a permitir determinar el grado de finalización del producto concluido e incluso de cada una de las fases del producto. Normalmente incluye diferentes técnicas de medición específicas para cada una de las fases, que permitan ponderar una situación frente a otra.

Unidad Didáctica III

Actividades del equipo de trabajo de Ingeniería del Software

¿Piensas que es frecuente trabajar individualmente desarrollando software o es una tarea que requiere de la cooperación de todo un **equipo**?

La tarea de desarrollar software hace ya mucho

tiempo que dejó de ser una labor individual del virtuoso de turno, aunque actualmente hay determinadas aplicaciones, normalmente de poca dificultad y reducido tamaño, que aún son programadas por desarrolladores de forma individual. Actualmente **una aplicación moderna es desarrollada por un equipo de personas especializadas en las diferentes fases del desarrollo de un proyecto**. En este aspecto podemos concretar que el personal dedicado a la ingeniería del software debe...

- **Trabajar en Equipo.**
- **Analizar y estudiar los problemas adelantándose a los mismos.**
- **Trabajar bajo restricciones de tiempo, costes y recursos.**
- **Interactuar con clientes y usuarios del futuro sistema software.**
- **Tomar decisiones constantemente**.

El tipo de **actividades** que va a llevar a cabo el personal de dicho equipo de Ingeniería del Software serán...

- De **Desarrollo**. Se centran básicamente en la construcción del producto, desde las necesidades de cliente hasta las pruebas finales que se realizan para comprobar su funcionamiento.
- De **Control**. Van a permitir asegurarnos un software de calidad evaluando determinados aspectos de proyecto en sus diferentes fases, incluso algún tiempo después de su implantación.
- De **Gestión**. Tareas que van a garantizar el adecuado desarrollo del proyecto.
- De **Operación**. Principalmente se centran en el trato con el usuario, desde las entrevistas iniciales para especificar los requerimientos, hasta su formación en el uso del producto finalizado, e incluso de los reajustes durante su mantenimiento.

Cada uno de los siguientes conceptos será tratado en detalle en unidades posteriores, por el momento sólo te damos un breve comentario sobre cada uno de ellos, en la siguiente simulación.

Básicamente el **trabajo de Desarrollo** de software (obtención del producto) se compone de las siguientes fases.

ANÁLISIS

Decidir qué hacer.

- Estudio de viabilidad.
- Deducción de requisitos.
- Análisis de requisitos.
- Modelado del sistema.
- Prototipado
- ...

DISEÑO

Decidir cómo hacerlo.

- Arquit. del sistema.
- Detallado.
- Interfaz de usuario.
- Datos.
- ...

Durante todas estas fases ...

Aceptación del producto.

VALIDACIÓN Y VERIFICACIÓN

Inspecciones y revisiones.

CONSTRUCCIÓN

- **CODIFICACIÓN. Hacerlo.**
- **PRUEBAS. Probar el producto.**
- **INSTALACIÓN. Usar el producto obtenido.**

1. Documentación.
2. Codificación.
3. Debug (Depuración).
4. ...

Planificación de prueba.

Pruebas de unidad.

Pruebas de integración.

Pruebas de regresión.

Pruebas del sistema.

Pruebas de aceptación.

...

MANTENIMIENTO

Seguimiento del producto durante su funcionamiento real.

Las actividades de **Control**, se ocupan de evaluar y asegurar la calidad del software.

Métricas.
Garantía de calidad.
Gestión de configuraciones.

Las actividades de **Gestión**. Cada proyecto depende de una gestión apropiada, adaptada a sus características, para alcanzar sus propios objetivos.

Planificación y estimación
Seguimiento de los proyectos
Administración de los proyectos
Dirección de los proyectos

Actividades de **Operación**. Definen las diferentes etapas por las que debe pasar el producto final hasta que es utilizado definitivamente por el cliente.

Entrega (e instalación)
Puesta en marcha
Formación de los usuarios

Unidad Didáctica III

Ciclo de vida del software

*CASO. **María** explica a **Julia** que todo el proceso de desarrollo del software desde que se decide la necesidad de su existencia hasta que se retira de uso y es sustituido por otro, está claramente definido por la Ingeniería del Software. Lo que deben decidir en este momento es qué modelo seguir y para ello es imprescindible conocer perfectamente el problema a tratar, el equipo de trabajo de que se dispone y las herramientas y recursos disponibles para llevarlo a cabo.*

***Julia** insiste en que no entiende muy bien qué es lo que tienen que hacer. Ha llegado a comprender la necesidad de utilizar la Ingeniería del Software en el desarrollo de la aplicación, pero no entiende porqué deben decidirse por un ciclo de vida, ¿es que no hay un método ideal, que se aplique a todos los proyectos?*

***María** le aclara que hasta hace unos años se utilizaba casi exclusivamente el ciclo de vida clásico, pero que se ha demostrado que en determinados proyectos hay paradigmas que permiten obtener mejores resultados con menor coste. Por eso es conveniente dedicar un poco de tiempo a decidir cuál es el procedimiento más adecuado a seguir.*

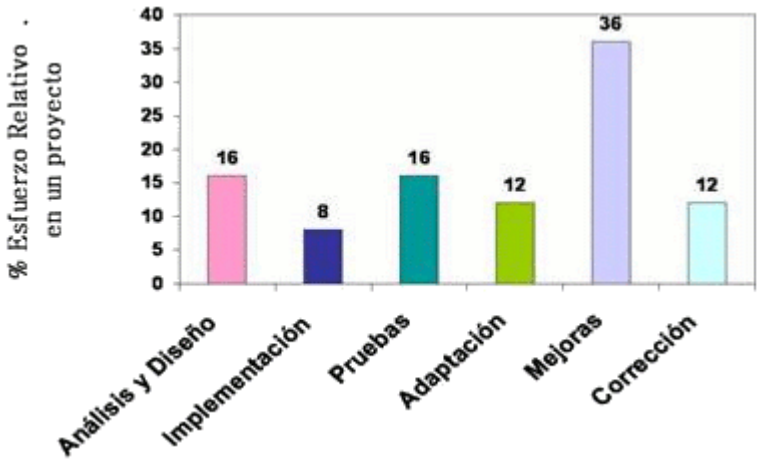
Podemos definir el **Ciclo de Vida del Software** como el conjunto de fases por las que pasa el sistema que se está desarrollando desde que nace la idea inicial hasta que el software es retirado o reemplazado por otro más adecuado.

Se denomina a veces **paradigma del software** y puede presentarse bajo dos puntos de vista:

- **Transformación del producto**. Se refiere al producto en sí, es decir, al Software que vamos a conseguir, que será de utilidad al cliente que lo ha pedido y del cual ha surgido la necesidad de su construcción.
- **Proceso mediante el que se transforma el producto**. Se centra en el proyecto mediante el que va a ser creado el producto final, cómo enfocarlo y cómo llevarlo a cabo.

Durante el ciclo de vida del software se realiza un **reparto del esfuerzo** de desarrollo del mismo, en cada una de las fases que lo componen. La tabla siguiente muestra cuales son esas fases, y el gráfico que le sigue muestra el porcentaje de esfuerzo y por tanto de coste que supone cada fase sobre el total de un proyecto.

Análisis y diseño.	**Estudio del problema y planteamiento de soluciones.**
Implementación.	**Confección de la solución elegida**
Pruebas.	**Proceso para comprobar la calidad del producto**
Adaptación.	**Instalación al cliente para que pueda usarlo.**
Mejoras.	**Retoques que permiten hacer más atractivo el uso.**
Correcciones.	**Solución de errores o ajustes para evitar problemas.**



Las características que debe poseer un Ciclo de Vida del Software podemos concretarlas en la siguiente imagen:

No existe un único modelo de Ciclo de Vida a seguir a la hora de desarrollar el software. Existen varios tipos que permiten adaptar la construcción de un producto

- según el problema a tratar,
- el equipo de trabajo disponible y
- las herramientas y recursos con los que podemos contar para llevarlo a cabo.

Por eso tenemos diferentes alternativas para llevar a cabo un proyecto, de entre las cuales es necesario elegir una de ellas según un conjunto de criterios que debemos ordenar para cada proyecto.

PARA SABER MÁS

En el siguiente enlace puedes encontrar un resumen muy completo sobre los ciclos de vida del software en PDF. A destacar la parte que trata sobre las cualidades del Software.

Ciclo de Vida del Software **[Versión en caché]**

También es muy recomendable el artículo que Wikipedia recoge sobre las fases de desarrollo del

Software, aunque da un enfoque diferente y siempre enriquecedor al tema.

Fases del Software **[Versión en caché]**

Esta página pretende ser un lugar de encuentro de gente interesada en la Ingeniería del Software Basada en Componentes, CBSE (Component-Based Software Engineering), especialmente en España.

Ingeniería del software basada en componentes (ISBC)

La seguridad en la Ingeniería de Software es un tema amplio. Este artículo está dirigido hacia la definición y la discusión de la seguridad y confiabilidad del software, así como en la responsabilidad del desarrollador y del usuario.

Seguridad en la Ingeniería del Software **[Versión en caché]**

Ciclo de vida del software

Unidad Didáctica III

Tipos de ciclos de desarrollo

CASO. Ante la necesidad de seguir un modelo de ciclo de vida para desarrollar el software que necesita la empresa ***Sinuneuro****, las reuniones que mantienen diariamente* ***María*** *y* ***Julia*** *tratan de hacer un enfoque adecuado para conseguir un software de calidad en el menor tiempo posible.* ***María*** *explica que en este punto de decisión de un método de trabajo, es importante conocer las herramientas y recursos de que pueden disponer para este trabajo, y actuar en consecuencia.*

Julia *le pide que sea más concreta, porque no entiende muy bien a qué se refiere. ¿Qué significa eso de los recursos? ¿A qué herramientas se refiere?*

María *le explica que hay muchas formas de abordar el desarrollo del software y es preciso establecer de antemano cuál será el ciclo de vida del proyecto y de este modo hacer una planificación consistente del trabajo a desarrollar. En primer lugar le habla de los recursos humanos para ajustar las fases del proyecto y otros recursos que pueden influir en el planteamiento.*

Estos recursos serían:

- *los plazos de entrega,*
- *el presupuesto disponible,*
- *los diferentes estados por los que pasará el proyecto,*
- *etc.*

Además, en función de esos recursos, es preciso establecer qué herramientas son necesarias, de cuales se dispone, cuales se necesitan y de cuales es necesario prescindir porque no hay posibilidad de utilizarlas. Con todo ello, una vez establecidas las fases de desarrollo, hay que consensuar las entradas (información necesaria para ponerla en marcha) y salidas (información generada u obtenida durante su desarrollo) para cada una de las fases, los criterios de transición para pasar a la fase siguiente, las actividades a realizar y definir cada uno de los esquemas de trabajo, asignando tareas a cada uno de los miembros del equipo de desarrollo.

Al comienzo, el modelo que se utilizaba era el de Codificar y Corregir. Normalmente lo hacía todo la misma persona que conocía sólo algunas de las técnicas y no le iba mal con ellas. En ocasiones existían mejores opciones que no podía utilizar al no conocerlas. Este modelo básico contiene dos pasos:

- Escribir código.
- Corregir problemas en el código.

Se trata de primero implementar algo de código y luego pensar acerca de requisitos, diseño, validación, y mantenimiento. Algo así como solucionar los problemas cuando aparezcan, sin realizar ninguna previsión Esto a veces funciona.

Este modelo tiene **tres problemas** principalmente:

- Después de un número de correcciones, el código puede tener una muy mala estructura, hace que los arreglos sean cada vez más costosos.
- Con cierta frecuencia, aún el software bien diseñado, no se ajusta a las necesidades del usuario, por lo que es rechazado o su reconstrucción es muy cara.
- El código es difícil de reparar por su pobre preparación para probar y modificar.

Por todo esto **no es el modelo más recomendable** y por lo que fueron apareciendo modelos que han ido proporcionando buenos resultados que aportan al desarrollo del software una base sólida sobre la que planificar cualquier trabajo. Todos ellos han surgido de la experiencia que ha sentado las bases de los ciclos de vida del Software.

Hay un gran número de modelos de ciclo de vida del software, entre los que vamos a tratar...

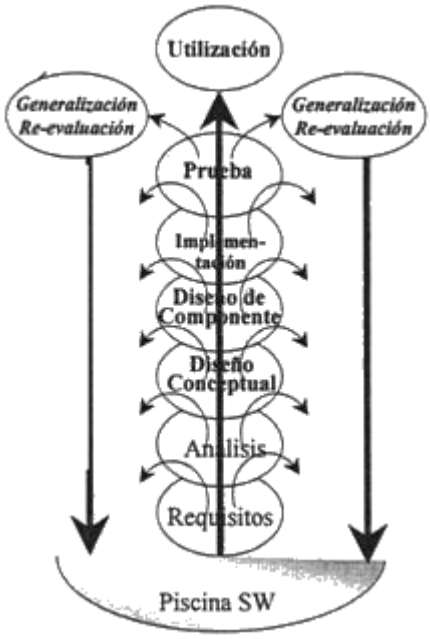
- Modelo en Cascada.
- Modelos Evolutivos:
  - Desarrollo Exploratorio.
  - Enfoque utilizando prototipos.
- Modelo en espiral. Evolutivo.
- Modelo Incremental.
- Modelo Basado en reutilización.

Unidad Didáctica III

Modelo en Cascada

El primer modelo de desarrollo de software que se publicó se derivó de otros procesos de ingeniería. Toma las actividades fundamentales de especificación, desarrollo, validación y evolución y las representa como fases separadas del proceso.

Consta de las siguientes fases:



1. **Definición de los requisitos**: Los servicios, restricciones y objetivos son establecidos con los usuarios del sistema. Se busca hacer esta definición en detalle de todo el sistema.
2. **Diseño de software**: El sistema es dividido en subsistemas de software y hardware. Se establece la arquitectura total del sistema. Se identifican y describen las abstracciones y relaciones de todos los componentes del sistema.
3. **Implementación y pruebas unitarias**: Construcción de los módulos y unidades de software. Se realizan conjuntos de pruebas para cada unidad.
4. **Integración y pruebas del sistema**: Se integran todas las unidades, se prueban en conjunto y se entrega el conjunto probado al cliente.
5. **Operación y mantenimiento**: Generalmente es la fase más larga. El sistema es puesto en marcha y se realiza la corrección de los errores descubiertos. Se realizan mejoras de implementación y es posible identificar nuevos requisitos.

La **interacción entre fases** puede observarse en la imagen siguiente. Cada fase tiene como resultado **documentos** que deben ser aprobados por el usuario. Una fase no comienza hasta que termine la fase anterior y generalmente incluye la corrección de los problemas encontrados en fases previas.

Algunos **problemas** que se observan en el modelo de cascada son:

- Las **iteraciones** son costosas e implican rehacer trabajo debido a la necesidad de producción y aprobación de documentos.
- Los problemas se dejan para su posterior resolución, lo que lleva, en ocasiones a que sean ignorados o corregidos de una forma poco satisfactoria.
- Existe una alta probabilidad de que el software no cumpla con los **requisitos del usuario** por el largo tiempo de entrega del producto.
- Es **inflexible** a la hora de evolucionar para incorporar nuevos requisitos. Es difícil responder a cambios en los requisitos.

Este modelo sólo debe usarse si se conocen y entienden completamente los requisitos. También se utiliza como parte de grandes proyectos.

Ejemplo del modelo.

Se trata del caso más habitual y sencillo. Cuando el cliente nos llama sabe lo que necesita y lo que quiere conseguir. Vamos a trabajar de forma conjunta y abordar el trabajo por etapas, cada vez que concluimos una etapa, alcanzamos el punto de partida de la siguiente. Un ejemplo claro es el de un proyecto de desarrollo de una aplicación para gestión de almacén. El cliente tiene claro su funcionamiento y lo que necesita, además es probable que haya visto alguna aplicación similar ya funcionando, le haya gustado y haya apreciado sus ventajas.

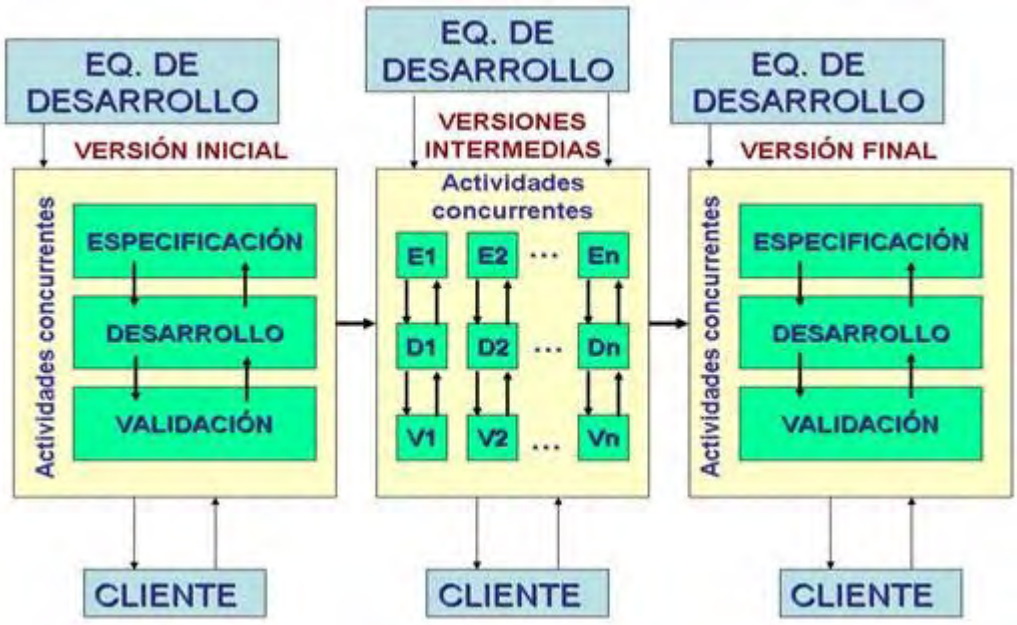
La planificación según este modelo es sencilla porque requiere la consecución de diferentes fases con la participación directa del cliente:

a. ***Análisis del problema para definir los requisitos que se necesitan. En este caso, gestión de productos (altas, bajas y modificaciones), pedidos a proveedores, control de existencias, etc.***
b. ***Una vez que el cliente valida esos requisitos se puede continuar con la siguiente fase de diseño, en la que, basándose en estos requisitos, es elaborada una aplicación que permita utilizarlos según las especificaciones del cliente.***
c. ***Durante la fase de diseño, el problema es tratado por partes, de modo que cada una de ellas se centra en uno de los requisitos definidos durante el análisis. Al concluir cada una de las partes, esta debe ser revisada por el cliente para comprobar que su funcionamiento es el esperado.***
d. ***La fase de diseño finaliza cuando cada una de las partes del proyecto supera las pruebas de validación. Entonces es el momento de integrarlas todas juntas y llevar a cabo pruebas de todo el sistema completo.***
e. ***Finalmente hemos obtenido una solución que permite automatizar mediante el uso de los ordenadores la gestión de almacén, es entonces el momento para que el cliente la utilice a modo de prueba durante un tiempo, durante el cual debe indicar qué partes es preciso revisar y reconsiderar. Los cambios que es preciso aplicar, nos pueden llevar a cualquiera de las fases anteriores, en la que se procedería de idéntico modo a como se ha realizado antes.***

Unidad Didáctica III

Modelos Evolutivos

La idea de este modelo parte del desarrollo de una implantación del sistema inicial, exponerla a los comentarios del usuario y refinarla en sucesivas versiones hasta que se desarrolle el sistema adecuado. En la imagen se observa cómo las **actividades concurrentes** de especificación, desarrollo y validación, son realizadas durante el desarrollo de cada una de las distintas versiones hasta llegar al producto final.



Una ventaja de este modelo es que se **obtiene una rápida realimentación del usuario**, ya que las actividades de especificación, desarrollo y pruebas se ejecutan en cada versión.

Existen dos tipos de desarrollo en este modelo:

- **Desarrollo Exploratorio:** El objetivo de este enfoque es explorar con el usuario los requisitos hasta llegar a un sistema final. El desarrollo comienza con las partes que se tiene más claras. El sistema va evolucionando a la vez que se añaden nuevas características propuestas por el usuario.

- **Enfoque utilizando prototipos:** El objetivo es entender los requisitos del usuario y trabajar para mejorar la calidad de los requisitos. A diferencia del desarrollo exploratorio, se comienza por definir los requisitos que no están claros para el usuario y se utiliza un prototipo para experimentar con ellos. El prototipo ayuda a terminar de definir estos requisitos.

Uno de los principales representantes de este tipo es el **modelo en Espiral**, que veremos detallado en el apartado siguiente.

Entre **los puntos favorables de este modelo** están:

- La especificación **puede desarrollarse de forma creciente**.
- Los usuarios y desarrolladores logran un **mejor entendimiento del sistema**. Esto se refleja en una mejora de la calidad del software.
- Es más efectivo que el modelo de cascada, ya que **cumple con las necesidades inmediatas del usuario**.

Desde una perspectiva de ingeniería y administración

se identifican los siguientes **problemas del modelo**:

- **Proceso no Visible**: Los administradores necesitan entregas para medir el progreso. Si el sistema se necesita desarrollar rápido, no es efectivo producir documentos que reflejen cada versión de sistema.
- **Sistemas pobremente estructurados**: Los cambios continuos pueden ser perjudiciales para la estructura del software haciendo costoso el mantenimiento.
- Se **requieren técnicas y herramientas**: Para el rápido desarrollo se necesitan herramientas que pueden ser incompatibles con otras o que poca gente sabe utilizar.

Este modelo es efectivo en proyectos pequeños (menos de 100.000 líneas de código) o medianos (hasta 500.000 líneas de código) con poco tiempo para su desarrollo y sin generar documentación para cada versión.

Ejemplo del modelo.

Otra forma de hacer un planteamiento del trabajo para la aplicación de gestión de almacén es según los modelos evolutivos. Las situaciones en las que se aplica este modelo suelen ser muy variadas, entre ellas podemos destacar cuando el cliente no sabe muy bien lo que busca, tiene claro que necesita una aplicación informática para facilitar el trabajo, pero no sabe cómo quiere que funcione. Evidentemente no ha visto nada que funcione como él quiere y no está muy seguro de que con el software se produzcan mejoras en la organización de la empresa.

El modelo evolutivo, parte de cada una de las especificaciones planteadas por el cliente, por ejemplo "Generación de Pedidos a Proveedores", es diseñada, desarrollada y presentada al cliente para que opine al respecto. Normalmente hay que mostrarle facilidad de uso y eficacia en el trabajo. El cliente durante esa validación irá haciendo peticiones sobre cómo le gusta que funcione cada uno de los apartados. Y una vez concluida, para la siguiente versión se incorpora una nueva especificación. De este modo vamos obteniendo versiones sucesivas del producto final, cada vez con mayores prestaciones.

Para proyectos largos es mejor combinar lo mejor del modelo de cascada y evolutivo:

- se puede hacer un prototipo global del sistema y
- posteriormente volver a implementarlo con un acercamiento más estructurado.

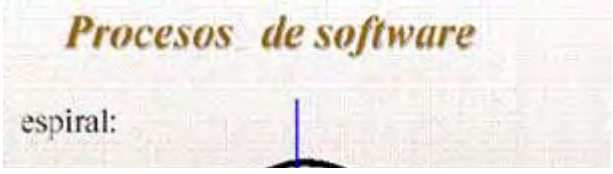
Los subsistemas con requisitos bien definidos y estables se pueden programar utilizando cascada y la interfaz de usuario se puede especificar utilizando un enfoque exploratorio.

Unidad Didáctica III

Modelo en Espiral. Evolutivo

El modelo de **desarrollo en espiral** es una variante de los modelos evolutivos y actualmente uno de los más conocidos. El ciclo de desarrollo se representa como una espiral, en lugar de una serie de actividades sucesivas con retrospectiva de una actividad a otra, que realmente es lo que ocurre. Una característica importante del modelo en espiral es que cada ciclo se completa con una revisión en la que participan los principales agentes (personas u organizaciones) que tienen relación con el producto.

Cada ciclo de desarrollo se divide en cuatro fases:



1. **Definición de objetivos**:
   - Se definen los objetivos, estableciendo

las restricciones del proceso o de la parte del producto que está siendo elaborada.
    - Se realiza un diseño detallado del plan de desarrollo, con alternativas para esa parte de producto, que pueden ser no software (organización empresarial, aumento de plantilla inversión en recursos, etc.).
    - Se identifican los riesgos y se elaboran estrategias alternativas dependiendo de éstos. Los riesgos que nos podemos encontrar son los siguientes; volver atrás para rehacer o retocar una fase, seleccionar herramientas de desarrollo inadecuadas, no validar una parte del producto a la que se le ha dedicado cierto tiempo, hacer una estimación de tiempos y costes inadecuada el bajo grado de satisfacción del cliente, el mal funcionamiento del producto tras las pruebas pertinentes (puede ocurrir que lo que sea inadecuado sean las propias pruebas), etc.
2. **Evaluación y reducción de riesgos**:
    - Se realiza un análisis detallado de cada riesgo identificado durante la definición de los objetivos realizada anteriormente.
    - Pueden desarrollarse prototipos para disminuir el riesgo que se produce cuando el cliente no tiene muy claro qué necesita o qué busca (requisitos dudosos).
    - Se llevan a cabo los pasos para reducir los riesgos identificados. Por ejemplo es posible considerar la posibilidad de usar prototipos para contrarrestar el riesgo de requisitos dudosos o también podemos aplicar rigurosas pruebas de funcionamiento del producto antes de pasarlo a la validación del cliente.
3. **Desarrollo y validación**:
    - Se escoge el modelo de desarrollo después de la evaluación del riesgo. Es posible optar por un modelo diferente para cada etapa, es decir, el modelo que se utilizará (cascada, sistemas formales, evolutivo, etc.) puede depender del riesgo identificado para esa fase.
4. **Planificación**:
    - Se determina si continuar con otro ciclo, en el caso que se considere inadecuado el actual porque se hayan detectado alto nivel de riesgos.
    - Se planea la fase siguiente del proyecto y se vuelve a aplicar la espiral.

Este modelo a diferencia de los otros **toma en consideración explícitamente el riesgo**, que puede llegar a ser determinante en administración del proyecto.

Ejemplo del modelo.

El modelo en espiral comienza con la identificación de los objetivos, restricciones y soluciones alternativas del proyecto, cada una de esas alternativas presenta una serie de inconvenientes o riesgos que pueden dificultar el resultado final del producto.

Si utilizamos el ejemplo de la aplicación para la gestión de almacén, supongamos que el cliente es la primera vez que realiza este tipo de trabajos (no ha hecho la gestión de almacén de forma manual) por lo que no sabe muy bien lo que busca, además es posible que haya visto algunas aplicaciones similares y tenga diferentes preferencias de cada una de ellas, incluso cuando intentamos consensuar las especificaciones que buscamos puede llegar a contradecirse, precisamente porque no conoce el tema y no puede imaginar el modelo que necesita.

En esta situación lo ideal es definir claramente los objetivos (o especificaciones) que se buscan, y proponer al cliente las diferentes alternativas para conseguirlos (si es que las hay), entonces identificar los riesgos de cada alternativa (difícilmente una alternativa carece de riesgos) y evaluarlos junto al cliente que debe tomar la decisión (con el debido asesoramiento) sobre la alternativa a seguir. Normalmente las diferentes alternativas se representan mediante modelos de simulación de la aplicación o prototipos, en los que podemos identificar con cierta facilidad los riesgos.

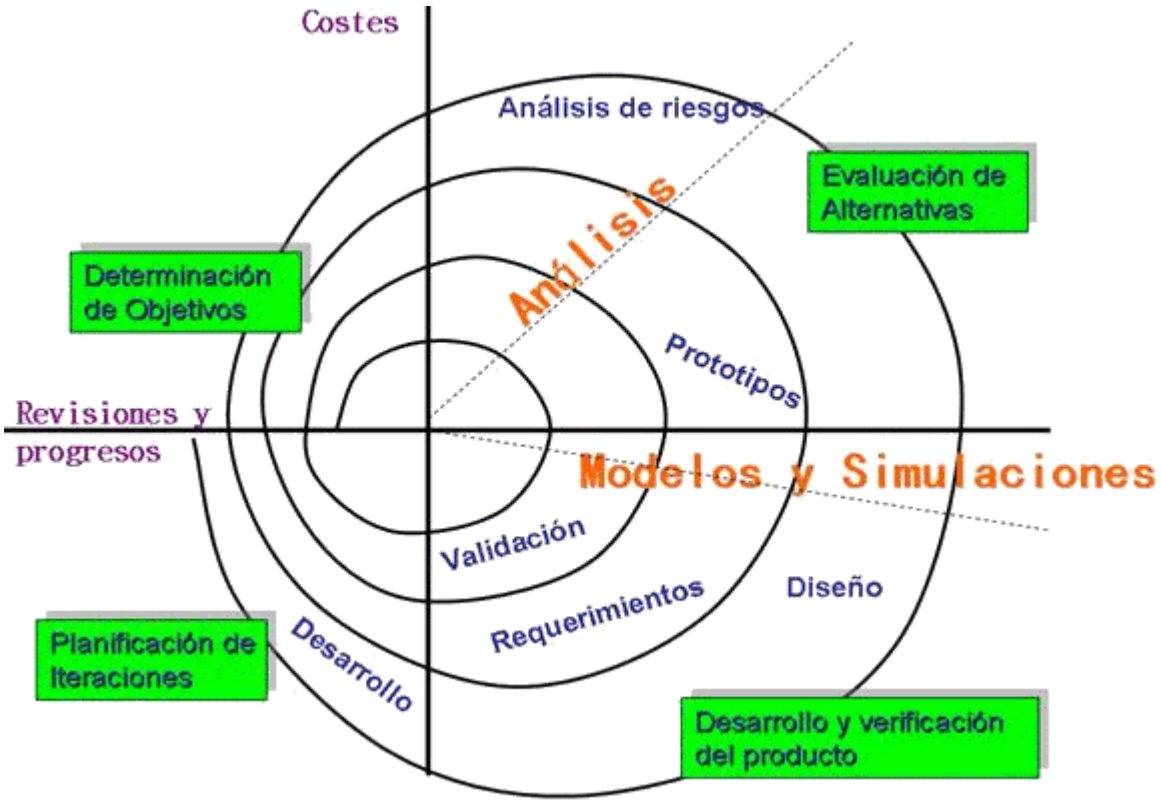
Por ejemplo si se opta por la alternativa de comenzar el proyecto desde que se realiza el pedido de un producto a proveedores, ésta puede hacerse sobre un producto nuevo o sobre un producto que ya existe, además cuando almacenamos el producto debe existir una correspondencia exacta entre lo que aparece físicamente en el almacén y lo que está registrado en el software. Los riesgos que presenta esta alternativa pueden ser los siguientes:

- ***Si el cliente no tiene muy claro cómo funciona un almacén de forma manual, probablemente le va a costar más entender el funcionamiento del software.***
- ***Es posible que pida cosas que el programa ya proporciona. Por ejemplo puede pretender conocer en cada momento el número de existencias de un producto, cuando puede conocer el de todos los productos almacenados.***

Una vez que se ha desarrollado parte del sistema, junto al cliente se planifica la siguiente fase en la que se deben concretar nuevas alternativas para la consecución de los objetivos buscados.

El Proceso a seguir:

- El ciclo de vida se inicia con la definición de los objetivos.
- De acuerdo a las restricciones se determinan distintas alternativas.
- Se identifican los riesgos al sopesar los objetivos contra las alternativas.
- Se evalúan los riesgos con actividades como análisis detallado, simulación, prototipos, etc.
- Se desarrolla un poco el sistema.
- Se planifica la siguiente fase.



Unidad Didáctica III

Modelo Incremental

El enfoque incremental de desarrollo surge como una forma de **reducir la repetición del trabajo en e proceso de desarrollo y dar oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema**. Podemos interpretarlo como una combinación del Modelo de Cascada y Modelo Evolutivo.

Reduce el proceso de rehacer trabajo durante el desarrollo y permite retrasar las decisiones hasta conocer mejor el sistema.

Durante el desarrollo de cada incremento se puede utilizar el modelo de cascada o evolutivo, dependiendo del conocimiento que se tenga sobre los requisitos a implementar. Si se tiene un buen conocimiento, se puede optar por cascada, si es dudoso, evolutivo.

Ejemplo del modelo.

Otra forma de trabajar en el desarrollo del software es la de elaborar una pequeña aplicación que cumpla los objetivos básicos del proyecto, para después ir añadiendo especificaciones (incrementando prestaciones) que van a permitir cubrir por etapas las necesidades del clientes.

En el ejemplo de gestión de almacén, se elaboraría una pequeña aplicación con los requisitos básicos de un programa de este tipo, sobre la que se irán haciendo añadidos (incrementos) que el cliente va decidiendo cuando la utiliza. Una especie de aplicación básica, que se irá ampliando según el cliente vaya detectando nuevas necesidades. El desarrollo de esos incrementos va a depender del grado de claridad respecto a las especificaciones del cliente, si el cliente conoce lo que busca y tiene claras sus necesidades el desarrollo será muy diferente al de la situación

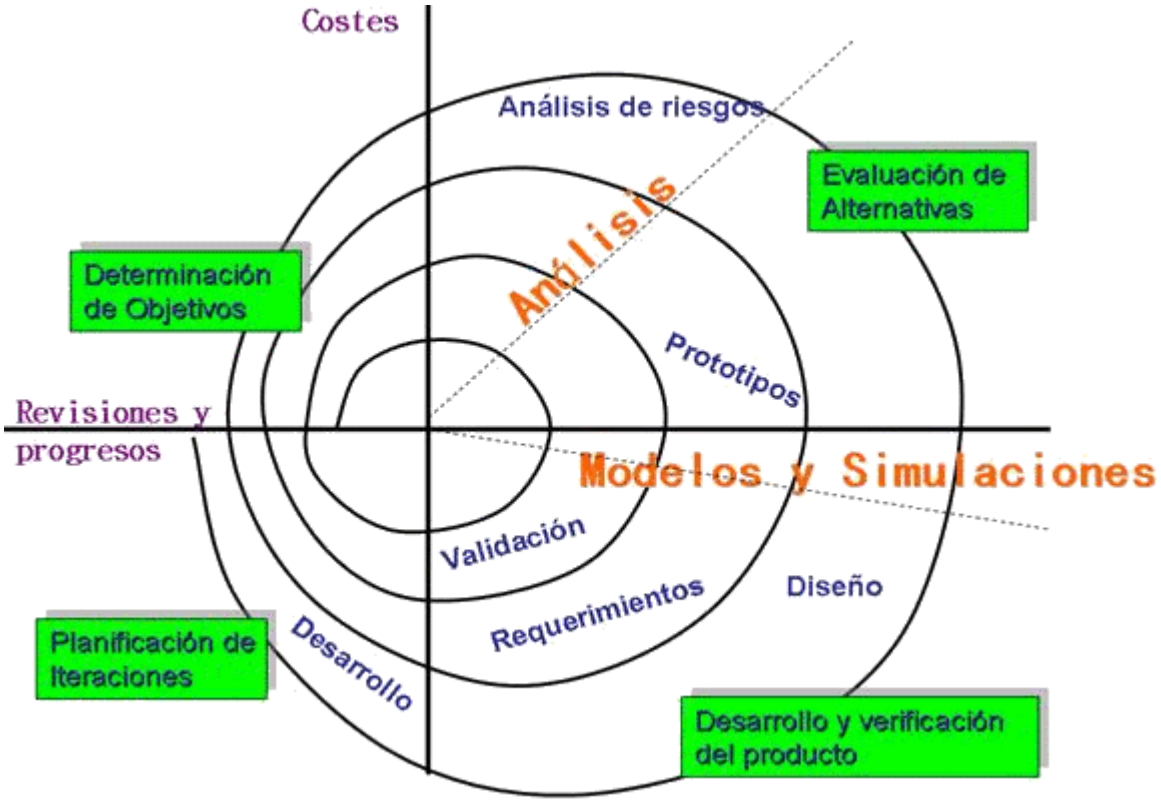
contraria, pero en cualquier caso debe ir validando incrementos del proyecto que irán siendo integrados a la aplicación hasta conseguir un sistema final aceptado o que deben volver a ser desarrollados con las revisiones precisas.

Entre las ventajas del modelo incremental se encuentran:

- **Los clientes no esperan** hasta el fin del desarrollo para utilizar el sistema. Pueden empezar a usarlo desde el primer incremento.
- Los clientes **pueden aclarar los requisitos** que no tengan claros conforme ven las entregas de sistema.
- Se **disminuye el riesgo de fracaso** de todo el proyecto, ya que se puede distribuir en cada incremento.
- **Las partes más importantes del sistema son entregadas primero**, por lo cual se realizan más pruebas en estos módulos y se disminuye el riesgo de fallos.

Algunas de las desventajas identificadas para este modelo son:

- **Cada incremento debe ser pequeño** para limitar el riesgo (menos de 20.000 líneas).
- Cada incremento **debe aumentar la funcionalidad**.
- Es difícil establecer las correspondencias de los requisitos y los incrementos.
- Es difícil detectar las unidades o servicios genéricos para todo el sistema.



Ciclo de vida del software

Unidad Didáctica III

Modelo orientado a la reutilización

Como su nombre lo indica, es un modelo fuertemente orientado a la reutilización. Este modelo consta de 4

fases. A continuación se describe cada fase:

- **Análisis de componentes**: Se determina qué componentes pueden ser utilizados para el sistema en cuestión. Casi siempre hay que hacer ajustes en cada uno de ellos para adecuarlos.
- **Modificación de requisitos**: Se adaptan (en lo posible) los requisitos para concordar con los componentes de la etapa anterior. Si no se pueden realizar modificaciones en los requisitos, hay que seguir buscando componentes más adecuados (es preciso volver a la fase 1).
- **Diseño del sistema con reutilización**: Se diseña o reutiliza el marco de trabajo para el sistema. Se deben tener en cuenta los componentes localizados en la fase anterior para diseñar o determinar este marco.
- **Desarrollo e integración**: El software que no puede comprarse, se desarrolla. Se integran los componentes y subsistemas. La integración es parte del desarrollo en lugar de una actividad separada.

Ejemplo del modelo.

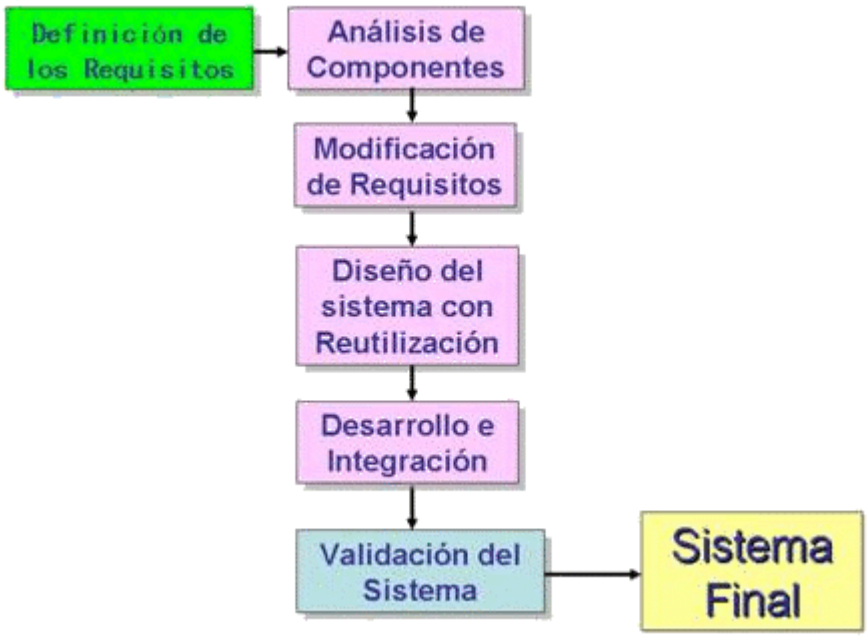
Actualmente un equipo de desarrollo no puede trabajar pensando en un único proyecto, sino que debe intentar aprovechar al máximo el tiempo y los recursos de la empresa, ya sean humanos, de software o materiales. Por ello es habitual que este tipo de empresas dispongan de librerías o módulos de código que pueden ser utilizados fácilmente en diferentes proyectos, con el consiguiente ahorro costes (en tiempo y económicos). Pero eso significa que a la hora de desarrollar nuevos módulos es necesario hacerlo pensando en que posteriormente podría ser utilizado en otros proyectos, por lo que a la hora de desarrollar uno de estos módulos deben ajustarse a ciertos patrones que facilitarán la compatibilidad con el resto de los módulos desarrollados.

Si nos situamos en el proyecto de gestión de almacén, un módulo que puede utilizarse de otros anteriores podría ser la entrada de datos de tipo numérico, texto o fecha que se suele utilizar en todo tipo de aplicaciones a la hora de introducir datos a través de teclado, de modo que no exista la posibilidad al introducir valores numéricos de cometer errores añadiendo caracteres.

Además si en nuestra aplicación se realiza un módulo para la venta de productos con la consiguiente disminución del número de existencias del producto en almacén, es posible plantearlo de tal modo que pueda ser reutilizado en las posteriores aplicaciones que se desarrollen e incluyan ventas de productos.

Las ventajas de este modelo son:

- **Disminuye los costes y el esfuerzo** de desarrollo.
- **Reduce el tiempo** de entrega.
- **Disminuye los riesgos** durante el desarrollo.



Desventajas de este modelo:

- **Es posible que el software no cumpla las expectativas del cliente** debido a que se encuentra predefinido, porque fue diseñado para otro proyecto.
- **Las actualizaciones de los componentes adquiridos no están en manos de los desarrolladores** del sistema.

Unidad Didáctica III

¿Cuál es el modelo más adecuado?

Cada proyecto de software requiere de una forma particular de abordar el problema. Las propuestas comerciales y académicas actuales promueven procesos iterativos, donde en cada iteración puede utilizarse uno u otro modelo de proceso, teniendo en cuenta un conjunto de criterios, como por ejemplo; grado de definición de requisitos, tamaño del proyecto o riesgos identificados, entre otros.

Modelo de proceso	Se adapta con requisitos y arquitectura no predefinidos	Produce software fiable	Gestiona riesgos	Admite correcciones sobre la marcha	Visión del progreso por el Cliente y el Jefe del proyecto
Codificar y corregir	Poco	Poco	Poco	Mucho	Suficiente
Cascada	Poco	Mucho	Poco	Poco	Poco
Evolutivo exploratorio	Bastante	Bastante	Suficiente	Bastante	Bastante
Evolutivo prototipado	Mucho	Suficiente	Suficiente	Mucho	Mucho
Espiral	Mucho	Mucho	Mucho	Suficiente	Suficiente
Incremental	Poco	Mucho	Suficiente	Poco	Poco

Desarrollo orientado a reutilización	Suficiente	[1]Depende...	Suficiente	Mucho	Mucho

Poco:	Significa que sucede en raras ocasiones.
Suficiente:	Que se produce algunas veces.
Bastante:	Que se da con cierta frecuencia.
Mucho:	Que ocurre la mayoría de las veces.

[1] Cuando se decide por la reutilización, se supone esta decisión ha sido tomada con garantías de calidad y mejora en el tiempo de desarrollo, aunque si se hace de forma precipitada y sin los fundamentos precisos, los resultados pueden llegar a ser peores y muy difíciles de reparar. De modo que según sea utilizado o según sea la cualificación de quien lo utiliza, es posible conseguir desde un producto poco fiable, hasta otro muy fiable.