Software

Es el conjunto de los programas de cómputo, procedimientos, reglas, **documentación** y datos asociados que forman parte de las operaciones de un sistema de computación.

El software no son sólo programas, sino todos los **documentos asociados** y la configuración de datos necesarios para hacer que estos programas operen de manera correcta. Un sistema de software consiste en diversos programas independientes, archivos de configuración que se utilizan para ejecutar estos programas, **un sistema de documentación que describe la estructura del sistema, la documentación para el usuario que explica cómo utilizar el sistema** y sitios web que permitan a los usuarios descargar la información de productos recientes. [Sommerville, 2005]

El software de computadora es el producto que los ingenieros de software construyen y después mantienen en el largo plazo. El software se forma con:

(1) Las instrucciones (programas de computadora) que al ejecutarse proporcionan las características, funciones y el grado de desempeño deseados;

(2) Las estructuras de datos que permiten que los programas manipulen información de manera adecuada; y

(3) Los documentos que describen la operación y uso de los programas. [Pressman, 2005]

Programación Orientada a Objetos

La **Programación Orientada a Objetos (POO)** como paradigma, "es una forma de pensar, una filosofía, de la cual surge una cultura nueva que incorpora técnicas y metodologías diferentes. La POO como paradigma es una postura ontológica: el universo computacional está poblado por objetos, cada uno responsabilizándose por sí mismo, y comunicándose con los demás por medio de mensajes" [Greiff 1994].

La Programación Orientada a Objetos desde el punto de vista computacional "es un método de implementación en el cuál los programas se organizan como grupos cooperativos (relacionados) de objetos, cada uno de los cuales representa una instancia de alguna clase, y todas estas clases son miembros de una jerarquía de clases unidas vía relaciones de herencia" [Greiff 1994].

El paradigma OO se basa en el concepto de objeto. Un **objeto** es aquello que tiene estado (propiedades más valores), comportamiento (acciones y reacciones a mensajes) e identidad (propiedad que lo distingue de los demás objetos). La estructura y comportamiento de objetos similares están definidos en su clase común. Una **clase** es la definición de un conjunto de objetos que comparten una estructura y comportamiento común.

La diferencia entre un objeto y una clase es que un objeto es una entidad concreta que existe en tiempo y espacio, mientras que una clase representa una abstracción, la "esencia" de un objeto, tal como es.

Ingeniería del software

Este término fue introducido a finales de los 60 a raíz de la **crisis del software**.

Esta crisis fue el resultado de la introducción de la tercera generación del hardware.

A partir de ese momento, el hardware dejó de ser un impedimento para el desarrollo de la informática al aumentar sus prestaciones y reducirse sus costos, impulsando la necesidad de mejorar la calidad y eficiencia en el software producido.

La crisis se caracterizó por los siguientes problemas:

- Imprecisión en la planificación del proyecto y la estimación de los costos.
- Baja calidad del software.
- Dificultad de mantenimiento de programas con un diseño poco estructurado, etc.

Como consecuencia, se hace indispensable que el software sea eficaz y barato tanto en el desarrollo como en la compra. También se requiere una serie de características como fiabilidad, facilidad de mantenimiento y de uso, eficiencia, etc.

En aquellos momentos se puso de manifiesto la problemática inherente al desarrollo de software de calidad:

- Las técnicas de desarrollo que funcionan de forma individual o para equipos pequeños no escalan adecuadamente para el desarrollo de sistemas complejos.
- La velocidad de cambio tecnológico es muy elevada. Los clientes tienen nuevas expectativas y todo ello aumenta la presión limitando la habilidad de crear software de calidad en tiempos aceptables

La **ingeniería de software** busca dar soluciones a estos problemas.

Objetivos de la ingeniería de software

En la construcción y desarrollo de proyectos deben aplicarse métodos y técnicas para resolver los problemas inherentes al desarrollo de software. La informática aporta herramientas y procedimientos sobre los que se apoya la Ingeniería de Software con el fin de:

- Mejorar la calidad de los productos de software
- Aumentar la productividad y trabajo de los ingenieros del software.
- Facilitar el control del proceso de desarrollo de software.
- Suministrar a los desarrolladores las bases para construir software de alta calidad en una forma eficiente.
- Definir una disciplina que garantice la producción y el mantenimiento de los productos software desarrollados en el plazo fijado y dentro del costo estimado.

Un programador NO ES EQUIVALENTE a un ingeniero de software

- “Todo el mundo” puede sentarse a programar pero esto no garantiza que pueda crear una solución compleja en tiempo, costo y calidad adecuados
- Desarrollar software sin conocer técnicas de ingeniería representa un enfoque “artesanal” de desarrollo que hoy en día sigue siendo habitual. Los esfuerzos de profesionalización buscan reducir este enfoque “artesanal” en el desarrollo de

software.

El desarrollo de software necesita tanto los fundamentos desarrollados dentro de las ciencias de la computación, como el rigor que las disciplinas de ingeniería aportan a la fiabilidad de los artefactos que se producen.

Definiciones de Ingeniería del Software

- Ingeniería del Software es el estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas de software. [Zelkovitz, 1978]
- Ingeniería del Software es la Aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada requerida para desarrollar, operar (funcionar) y mantenerlos. Así como también desarrollo de software o producción de software. [Bohem, 1976]
- Ingeniería del Software es el establecimiento y uso de principios sólidos de la ingeniería para obtener económicamente un software confiable y que funcione de modo eficiente en maquinas reales. [Bauer, 1972]
- Ingenieria de Software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento del software: es decir, la aplicación de ingeniería al software. [IEEE, 1993]
- La Ingenieria de Software es una disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema hasta el mantenimiento de éste después de que se utiliza. [Sommerville, 2004]
- La Ingenieria de Software es una disciplina que integra el proceso, los métodos, y las herramientas para el desarrollo de software de computadora. [Pressman, 2006]
- La IS es aplicar el **sentido común** al desarrollo de sistemas software. [Navarro, (UCM)].

  ¿Qué es sentido común?

  1. Planificar antes de desarrollar.
  2. Diseñar antes de programar.
  3. Reutilizar diseños que funcionan y son mantenibles.
  4. ...utilizar las herramientas apropiadas, [Gómez Sanz, Jorge Jesús (UCM)]
  5. ...y recursos humanos de capacidad media. [Pavón Mestras, Juan (UCM)]

Aplicación de la Ingeniería del Software al desarrollo de aplicaciones

La Ingeniería del Software se utiliza, sobre todo, para desarrollar aplicaciones de gran envergadura (de miles o millones de instrucciones) en las que suelen participar distintos equipos de personas y, a veces, de distintas empresas de software. Suelen ser proyectos que pueden durar varios meses o incluso años. No obstante, por pequeño que sea un proyecto software, siempre es conveniente aplicar los principios de la Ingeniería del Software, ya que esto ayudará a desarrollar un software de mayor calidad.

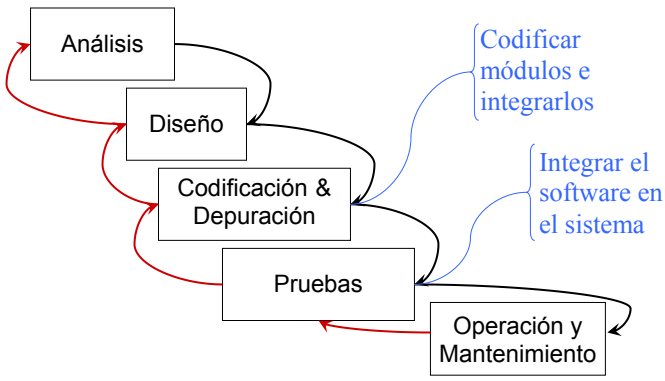
La **calidad de un programa** se puede medir en base a tres aspectos principales:

1. Sus características operativas. Se debe valorar si el software hace lo que se espera de él (*corrección*) y si, para ello, se utilizan óptimamente los recursos de la computadora (*eficiencia*), tales como: la memoria, el tiempo de CPU, etc. También se debe evaluar si la aplicación ofrece una interfaz adecuada al usuario (*facilidad de uso*) y si es seguro con respecto a los datos (*integridad*).

2. Su capacidad para sufrir cambios. En este sentido, es importante estimar en qué medida el programa es susceptible de ser corregido (*facilidad de mantenimiento*) o modificado (*flexibilidad*). También hay que ver si resulta fácil hacer pruebas de su funcionamiento (*facilidad de prueba*).

3. Su adaptabilidad a entornos distintos. Hay que preguntarse hasta qué punto se podría volver a usar parte de dicho software en otro proyecto (*reusabilidad*). Asimismo, se debe valorar si el software puede interactuar con otros sistemas informáticos (*facilidad de interoperación*) y si se puede usar en otra máquina que utilice un procesador distinto (*portabilidad*), aunque sea realizando pequeños cambios en el software.

El proceso de construcción de una aplicación informática de calidad lleva consigo realizar una serie de tareas repartidas en cinco etapas, llamadas: *análisis*, *diseño*, *codificación*, *pruebas* y *mantenimiento*. Estas cinco etapas constituyen el denominado *ciclo de vida de un producto software*. Dicho de otra forma, el **ciclo de vida de un programa** son las distintas etapas por las que éste tiene que pasar durante su existencia.



Etapas del ciclo de vida de un programa

Las distintas fases del ciclo de vida de un programa se deben realizar secuencialmente, tal y como se muestra en la figura. En cada una de las etapas se generará una documentación que servirá para iniciar la siguiente. A este proceso se le conoce como ciclo de vida clásico o en cascada, y es la base fundamental sobre la que se apoya la Ingeniería del Software. Existen otros tipos o modelos de ciclos de vida, tales como: clásico con prototipo, automático, en espiral, etc. Pero, todos ellos están basados de alguna manera en el modelo clásico.

¿En qué consiste y cuál es la finalidad cada una de estas fases?:

❖ **Fase de Análisis**

Un software siempre se crea para dar solución a un problema. Ahora bien, no todos los problemas planteados por los humanos se pueden informatizar. Para determinar si algo es posible, lo primero que hay que hacer es analizar el problema en cuestión. Esto implica determinar cuáles son las exigencias del problema y estudiar si se puede resolver poniendo en práctica las técnicas y conocimientos científicos disponibles. En el caso de que se considere viable, se debe realizar un análisis muy exhaustivo del problema, fruto del cual se obtendrá una documentación, en la que se especificará, claramente, cuáles son los *requisitos* que deberá contemplar el programa, entendidos éstos como características que exhibir el programa. A dicho documento se le llama **Especificación de Requisitos Software (ERS)** y en él quedará escrito *qué* tiene que hacer el programa que se va a desarrollar, tanto en lo que se refiere al comportamiento interno (gestión de los datos) como al externo (interacción con el usuario y con otras aplicaciones).

❖ **Fase de Diseño**

Una vez que los *requisitos de un programa* han sido establecidos, ya se puede iniciar la fase de diseño. En esta etapa se tiene que encontrar una solución informática al problema planteado. Dicha solución determinará *cómo* se va a resolver el problema. Por norma general, no suele ser fácil encontrar una solución óptima y, menos aún, cuando el problema tiene cierta envergadura. Para encontrar dicha solución, el desarrollador puede hacer uso del **diseño modular**.

❖ **Fase de Codificación**

Una vez diseñada la aplicación, se puede iniciar la fase de **codificación**. En esta etapa se tiene que traducir el diseño a un lenguaje de programación específico; es decir, las acciones representadas en el diseño deben convertirse en instrucciones software. Para codificar el diseño hay que conocer la sintaxis del lenguaje al que se va a traducir. Sin embargo, independientemente del lenguaje de programación en que esté escrito un programa, será su diseño el que determine su lógica. La **lógica de un programa** establece cuáles son sus acciones y en qué orden se deben ejecutar. Por tanto, es conveniente que todo programador aprenda a diseñar aplicaciones antes de pasar a la fase de codificación.

❖ **Fase de Pruebas**

Una vez obtenido el código ejecutable de un programa, tras depurarlo lo máximo posible, hay que comprobar exhaustivamente su funcionalidad. Para ello, se tiene que ejecutar tantas veces como se considere necesario, proporcionando, en cada ejecución, datos de entrada distintos y comprobando si los datos de salida son siempre los esperados.

No es posible que el código ejecutable de un programa tenga errores de sintaxis, ya que éstos habrán sido detectados por el compilador y corregidos por el programador en la fase de codificación. Por tanto, las **pruebas** a realizar se deben centrar en la búsqueda de errores de ejecución o de lógica.

Para estar totalmente seguros del buen funcionamiento de un programa se debería probar con todas las posibles combinaciones de entrada, cosa que en la mayoría de ocasiones suele ser imposible, ya que éstas pueden ser infinitas. Así pues, las pruebas tienen que ser muy bien elegidas, intentando abarcar el mayor número posible de casos y poniendo a prueba el programa en aspectos críticos. Además, en todos los programas pueden darse situaciones inesperadas, es decir, situaciones no previstas por el programador. En esos casos, el programa reaccionará de manera imprevisible.

Cuando se desarrolla una aplicación grande, las pruebas pueden tardar semanas o

incluso meses. Las pruebas no sólo se deben centrar en la comprobación del tratamiento de los datos, sino también en aspectos como: la adaptación de la aplicación al resto del sistema informático o la interacción del software con otras aplicaciones ya existentes. Una aplicación informática de envergadura puede estar formada por cientos o miles de programas, y todos ellos deben ser probados individual y conjuntamente. Antes de implantar un software de estas características, lo normal es hacer simulaciones con datos reales para verificar su buen funcionamiento.
Para corregir los errores de ejecución o de lógica encontrados en la fase de pruebas, casi siempre, por no decir siempre, hay que retocar el diseño y, en algunos casos, incluso hay que volver a analizar el problema, volviendo a pasar por todas las fases de desarrollo; de lo cual se deduce que, cuanto mejor se hagan el análisis y el diseño de una aplicación, la probabilidad de encontrar errores en la fase de pruebas será menor.

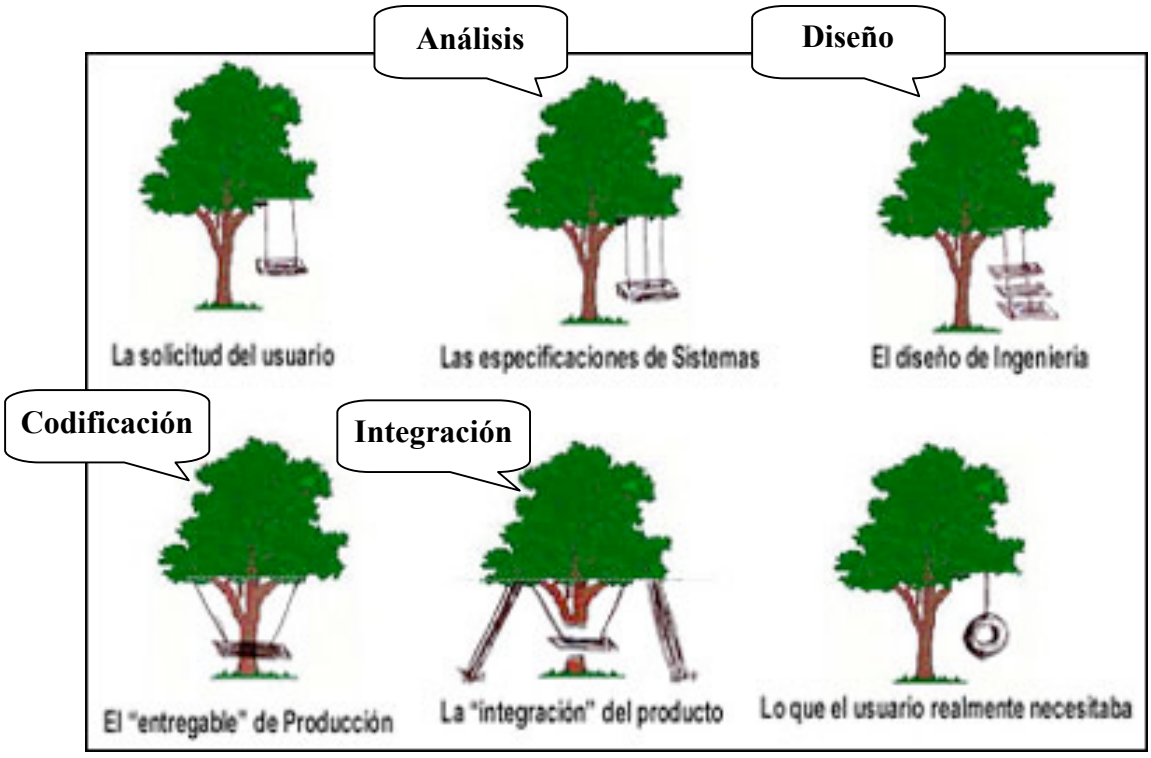
❖ **Fase de Mantenimiento**

El **mantenimiento de un software** se puede realizar, básicamente, por dos motivos: *reparación* o *modificación*. Una vez implantada la aplicación, todavía pueden producirse errores no detectados en fases anteriores, lo que implicará efectuar **reparaciones**. Por otra parte, puede ocurrir que, en cualquier momento, se desee ampliar o cambiar alguna funcionalidad de la aplicación, lo cual llevará consigo realizar **modificaciones**.
En un programa pequeño, las modificaciones o reparaciones pueden ser fáciles de realizar. Sin embargo, en aplicaciones grandes, una pequeña modificación o reparación puede resultar muy costosa. Además, hay que tener presente que aunque el software no se puede estropear (cosa que sí le puede ocurrir al hardware), el software sí se puede deteriorar debido a los cambios. Tanto es así que, en algunos casos, puede ser preferible empezar de cero toda la aplicación en vez de intentar actualizarla. Con esto, el **software viejo** puede dejar de utilizarse al quedarse obsoleto, lo que significa el final de su existencia, es decir, el final de su ciclo de vida. Esto ha pasado siempre, por ejemplo, con algunos sistemas operativos antiguos, con aplicaciones de gestión que se han dejado de usar por la aparición de otras mucho más funcionales, con juegos que ya no son tan atractivos porque no usan gráficos, etc.

Todos los factores que influyen en la calidad de un proyecto software deben medirse a lo largo de todo su proceso de desarrollo, es decir, en el transcurso de todas las etapas del ciclo de vida, y no sólo al final. De esta forma, la calidad del producto software resultante, se puede ir mejorando sobre la marcha.

Un proyecto desarrollado de forma indisciplinada tiene muchas posibilidades de fracasar...

Ejemplo: Diseñar un columpio



Desarrollar software es como construir un edificio: hay mucho que hacer antes de poder entregar el producto "final" definitivo...

- ✓ Planificar minuciosamente
- ✓ Elegir materiales
- ✓ Establecer y respetar una temporización
- ✓ Inspeccionar frecuentemente la obra
- ✓ Los errores son muy costosos de reparar
- ✓ La dificultad depende del tamaño del edificio

Los problemas de organización y gestión son tan complicados como los problemas técnicos

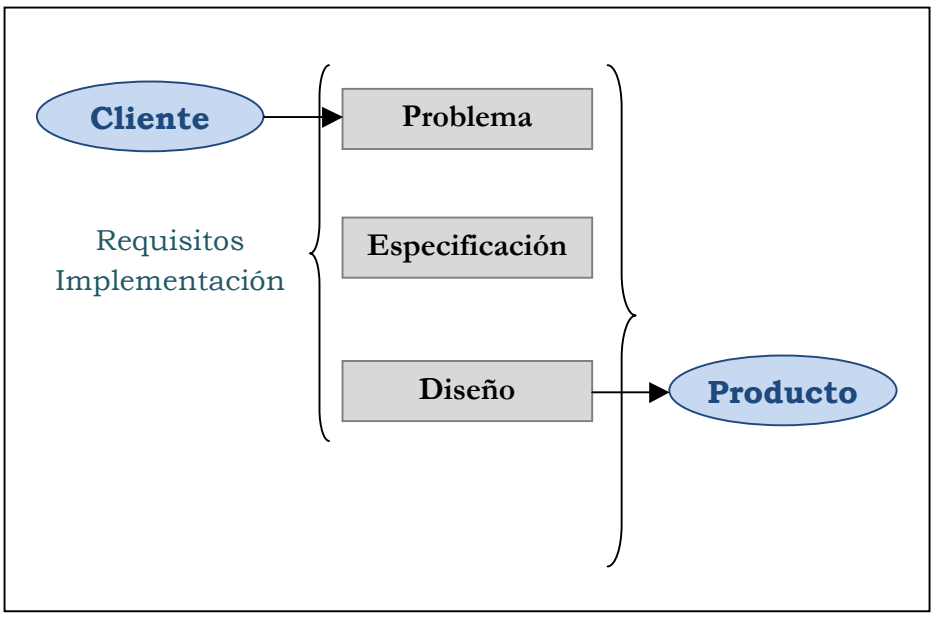
Cada fase puede introducir errores …

Problema mal planteado
↓
Especificación incorrecta
↓
Diseño inadecuado
↓
Implementación errónea

Los errores se propagan …

No empieces a codificar hasta que SEPAS BIEN lo que estás haciendo

Las metodologías planifican el desarrollo



Fase 1: definición del problema original...

¿Qué problema hay que resolver?
Definir bien el problema antes de empezar...
Asegurarse de conocer bien cuál es el problema...

- ✓ En lenguaje natural
- ✓ Desde el punto de vista del usuario
- ✓ Sin referirse a una posible solución

Fase 2: especificación de la solución...

- ✓ Describe en detalle qué hace el sistema
- ✓ No describe cómo se hace
- ✓ Realiza una descripción correcta y completa (contempla todos los casos)
- ✓ Emplea diagramas y notaciones formales
- ✓ Permite realizar cambios de forma cómoda (se producirán)

"Las especificaciones son como el agua, es más fácil construir sobre ellas cuando se han congelado"

Fase 3: diseño de la solución...

- ✓ Describe cómo funciona el sistema
- ✓ Define la estructura del sistema:
  - qué componentes existen
  - qué papel juega cada componente
  - cómo se relacionan los componentes
- ✓ Justifica las decisiones de diseño
- ✓ Emplea diagramas y notaciones formales
- ✓ Permite cambios de forma cómoda (se producirán)
- ✓ Es independiente del lenguaje, el S.O. y la máquina
- ✓ Guía la implementación

Las metodologías mejoran la calidad del Software: **internamente** (de cara a los desarrolladores) y **externamente** (de cara a los usuarios)

Bibliografía

[Braude, 2003] Eric J. Braude. *Ingeniería de Software: Una perspectiva orientada a objetos*. Editorial Ra-Ma. 2003

[Greiff, 1994] Greiff W. R. *Paradigma vs Metodología; El Caso de la POO*. *Soluciones Avanzadas*. Ene-Feb 1994. pp. 31-39.

[Piattini et al., 2003] Mario G. Piattini y otros. *Analisis y Diseño de Aplicaciones Informáticas de Gestión: Una perspectiva de Ingenieria del Software.* Editorial Ra-Ma. 2003

[Pressman, 2006] Roger Pressman. *Ingenieria del Software**: Un Enfoque Practico***. McGraw-Hill. 2006

[Sommerville, 2004] Ian Sommerville. *Ingenieria de Software*. Pearson. 2005

[Weitzenfeld, 2005] Alfredo Weitzenfeld. *Ingenieria de Software Orientada a Objetos: Teoría y Práctica con UML y Java*. Thomson Paraninfo. 2005