

Clase 1

Principios de ingeniería del software

Introducción

Un sistema informático se puede definir como un conjunto de partes o componentes cuya finalidad es brindar soporte al procesamiento, almacenamiento, y entrada-salida de información. Por información nos referimos a la agrupación de datos que, propiamente ordenados, y estructurados, resultan útiles para alcanzar determinado resultado u objetivo.

Un sistema informático está fundamentado en la utilización de la computación, es decir, en el procesamiento automático de la información mediante dispositivos electrónicos.

Las partes fundamentales que componen un sistema informático son el componente físico, o hardware; el componente lógico, o software; y el componente humano.

En esta primera parte de la materia nos enfocaremos sobre el componente lógico, en particular sobre el proceso de desarrollo del software.

El software

El software en sentido estricto es el soporte lógico e inmaterial que permite que la computadora pueda desempeñar tareas, almacenar información o servir como medio para administrar el hardware.

Por lo general se denomina software a los programas de computación. Dichos programas se basan en un conjunto de instrucciones que las computadoras utilizan para manipular datos, en muchos casos de manera automática y en otros con intervención del usuario, haciendo uso de los componentes físicos del sistema.

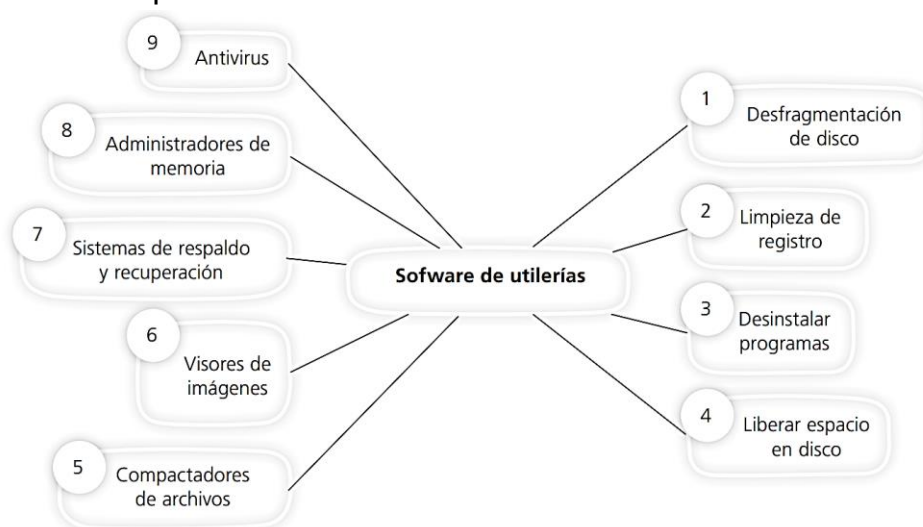
Sin la presencia de software, la computadora sería un conjunto de componentes físicos sin posibilidad de ser utilizados.

A grandes rasgos, podemos clasificar el software informático en **software de sistema, software de aplicación y software de programación**.

- **Software de sistema:** Se llama software de sistema o software de base al conjunto de programas que sirve para interactuar con el sistema informático, delegando el control sobre el hardware y dándole el soporte a

otros programas. El software de sistema se divide a su vez en tres tipos: sistema operativo, controladores de dispositivos (drivers) y utilitarios.

- **Sistema Operativo:** Es el conjunto de programas que gestionan los recursos de la computadora y controlan sus actividades. Este cúmulo de sistemas tiene como finalidad cinco funciones básicas: suministro de interfaz al usuario, administración de recursos, administración de archivos, administración de tareas y servicio de soporte.
- **Controladores de dispositivos o drivers:** Los controladores de dispositivos son programas que permiten que otros de mayor nivel como un sistema operativo interactúe con un componente de hardware.
- **Utilitarios:** Son programas o partes de un programa que tiene un fin determinado, esto es, programas que nos asisten para realizar un trabajo. Sin las utilerías no serían posibles todos los programas que realizan un sinfín de cosas. Cada programa tiene sus propias utilerías; es decir, aplicaciones extras que no son el programa mismo. A continuación, se muestra una imagen con algunos utilitarios del sistema operativo.



- **Software de aplicación:** Son los programas diseñados para o por los usuarios para facilitar la ejecución de tareas específicas en la computadora, como pueden ser las aplicaciones ofimáticas (procesador de texto, hoja de cálculo, programa de presentación, sistema de gestión de base de datos), y todo otro tipo de software especializado como software médico, software educativo, editores de música, programas de contabilidad, entre otros.
- **Software de programación:** Estos programas son la base donde se escribe el código para desarrollar nuevos programas dentro de un sistema operativo. Este tipo de software incluye principalmente compiladores, intérpretes, ensambladores, enlazadores, depuradores, editores de texto y


un entorno de desarrollo integrado que contiene las herramientas anteriores, y normalmente cuenta una avanzada interfaz gráfica de usuario (G.U.I.). A través del desarrollo de algoritmos mediante lenguajes de programación de alto nivel, es posible diseñar software que ejerza diversas funciones, tal como es el caso del software de aplicación.

En particular el software de programación permite la implementación de funciones, métodos y procedimientos para finalmente concluir en el desarrollo de programas. Para ello se requiere del conocimiento de ciertos estándares respecto a las aplicaciones disponibles en el mercado y de lenguajes de programación, entre otros aspectos.

Es así que desarrollo de software es un aspecto fundamental dentro de un sistema informático, y como tal responderá a la necesidad de establecer condiciones de eficiencia y calidad, entre otros aspectos. Esto es lo que ampliaremos a continuación a través de la ingeniería de software.

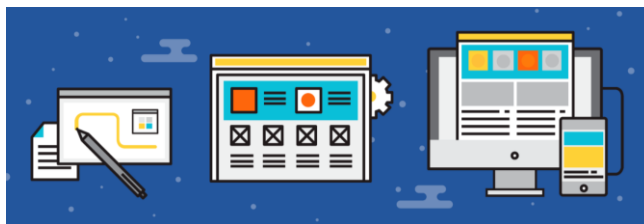
¿Qué es la ingeniería de software?

La ingeniería de software se ha definido por varios autores:

- Según Ian Sommerville, considerado uno de los padres de la ingeniería de software, la ingeniería de software *"es una disciplina de la ingeniería que comprende todos los aspectos de la producción del software"* [Somerville, 2004].
- La IEEE (The Institute of Electrical and Electronics Engineers: el Instituto de Ingenieros Electricistas y en Electrónica), es la asociación internacional más grande del mundo formada por profesionales de las nuevas tecnologías, como ingenieros electricistas, electrónicos, en sistemas y en telecomunicaciones. La IEEE define a la ingeniería de software como *"la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software"*.
- Finalmente, una de las definiciones más interesantes es la de Bohem (1976): *"ingeniería de software es la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como desarrollo de software o producción de software"*.

Cualquiera de estas definiciones es válida y no contradice a la otra. Sin embargo, ninguna de ellas incluye el específicamente concepto de calidad del software.

Podríamos agregar también que la ingeniería de software es una disciplina en la que se aplican técnicas y principios de forma sistemática en el desarrollo de sistemas de software para garantizar su calidad.



La importancia de la ingeniería de software

Normalmente, los clientes que mandan construir un sistema de software lo quieren lo más pronto posible (o antes). Es por esto que para “acabar más rápido” existe la tentación de comenzar a codificar y probar sin hacer antes un buen análisis de cuáles son los requerimientos del cliente ni un diseño.

Es decir, si bien es cierto que cualquier persona con tiempo y que le guste programar podría construir software, lo más seguro es que el producto final sería mucho menos eficiente y mucho menos seguro que un software desarrollado con procedimientos adecuados y bien estudiados.

Si no se tienen claros los requerimientos del cliente, entonces lo más probable es que el sistema que se desarrolle no cumpla con sus necesidades. Cuando no se cuenta con un diseño, se cometen todo tipo de errores y es muy difícil encontrarlos y resolverlos de forma adecuada. Lo más probable es que el tiempo invertido para que el sistema funcione haya sido mayor al que se hubiera invertido aplicando ingeniería de software. Como consecuencia, los desarrolladores quedan frustrados y exhaustos, y el cliente difícilmente quedará satisfecho. Además, al final se entrega un producto de mala calidad al que difícilmente se le podrá dar un buen mantenimiento.



La experiencia dice que, si en un proyecto grande no se aplica ingeniería de software, éste fracasará. La probabilidad de que un proyecto sea exitoso es mucho mayor cuando se aplica la ingeniería de software.

De allí la importancia de la ingeniería del software, la cual ofrece herramientas y técnicas que trascienden más allá de la codificación del software y que es sumamente importante para construir o mantener un software de calidad.

Guía del cuerpo de conocimiento de la ingeniería de software

El SWEBOK¹ (*Guide to the software engineering body of knowledge*: Guía del cuerpo de conocimiento de la ingeniería de software) es un documento editado por la IEEE y describe el conocimiento que existe en la disciplina de ingeniería de software.

Divide a la ingeniería de software en doce áreas:

1. Requerimientos
2. Diseño
3. Construcción
4. Pruebas
5. Mantenimiento
6. Gestión de la configuración
7. Gestión de la ingeniería de software (Administración de Proyectos)
8. Procesos software
9. Métodos y herramientas
10. Calidad
11. Medición
12. Seguridad

En la próxima clase veremos una introducción a algunas de estas áreas, incluidas en el proceso de desarrollo de software.

Principios de la ingeniería de software

Aunque la ingeniería de software no tiene principios fundamentales universalmente reconocidos, se han hecho varios intentos por establecerlos. El libro "Fundamentals of software Engineering " (Ghezzi, 2002) es un reconocido texto de ingeniería de software que propone siete principios que constituyen las propiedades deseables durante el desarrollo de un sistema de software.

A continuación, explicaremos estos principios:

✓ *Rigor y formalidad*

En cualquier proceso creativo existe la tendencia a seguir la inspiración del momento de forma no estructurada, sin ser precisos; el desarrollo de software es de por sí una actividad creativa. Sin embargo, si la documentación del sistema es ambigua o inconsistente será difícil encontrar los defectos y saber dónde hacer cambios. Mientras más rigor y formalidad

¹ <https://www.computer.org/education/bodies-of-knowledge/software-engineering>

en la documentación y el código, el sistema será más confiable, verificable y mantenible.

El rigor y la formalidad va desde la inclusión de la explicación y fundamento matemático y lógico de las principales funcionalidades del programa, hasta la documentación que acompaña al programa final y los comentarios dentro del código fuente.

En las siguientes imágenes podemos observar el mismo programa, escrito en pseudocódigo.

Con comentarios:

```
/*
Conversión decimal a binario. Entrada: un número decimal entero.
Salida: una cadena binaria que represente al número antes ingresado.
*/
function dec2bin(n:integer) // Solo convierte números enteros, una mejora a incorporar sería que permita números reales
b=abs(n);
bin="";
repeat while b>=2
    if mod(b,2)==1 then
        bin="1"+bin;
    elseif mod(b,2)==0 then
        bin="0"+bin;
    endif;
    b=floor(b/2);
end;
//A continuación se añade el último resultado de las sucesivas divisiones, siendo el primer valor del número binario
if b==1 then bin="1"+bin;
else bin="0"+bin;
endif;
//Se añade el signo del número entero al número en binario
if sign(n)==-1 then bin="-"+bin; endif;
return bin;
endfunction
```

Y a continuación sin comentarios:

```
function dec2bin(n:integer)
b=abs(n);
bin="";
repeat while b>=2
    if mod(b,2)==1 then
        bin="1"+bin;
    elseif mod(b,2)==0 then
        bin="0"+bin;
    endif;
    b=floor(b/2);
end;
if b==1 then bin="1"+bin;
else bin="0"+bin;
endif;
if sign(n)==-1 then bin="-"+bin; endif;
return bin;
endfunction
```

Como se puede observar, aun siendo un programa pequeño con una única función, se entiende mucho mejor si se incluyen ciertos comentarios entre las líneas del código fuente.

En programas que normalmente tienen miles de líneas de código sería muy útil incluir comentarios acerca de las funciones o procedimientos más importantes. Los comentarios a incluir deben ser útiles y deben tener el mayor rigor matemático y lógico posible.

✓ Modularidad

Para resolver un problema complejo de desarrollo de software, conviene separarlo en partes más pequeñas que se puedan diseñar, desarrollar, probar y modificar de manera sencilla y lo más independientemente posible del resto de la aplicación. A cada una de estas partes se les llama módulos.

En la programación estructurada los módulos son procedimientos y funciones que, en conjunto, proporcionan la funcionalidad del sistema, mientras que en la programación orientada a objetos los módulos están formados por las clases que son parte del problema.

Este aspecto es uno de los más importantes a la hora de diseñar el código de un programa y una de las primeras cosas que se enseña al comenzar a programar.

Como se observa en las siguientes figuras² los módulos en programación estructurada están orientados a procesar una parte de la solución del problema:



Mientras que en programación orientada a objetos están enfocados a determinar cuáles son las propiedades de las entidades que intervienen en el problema:

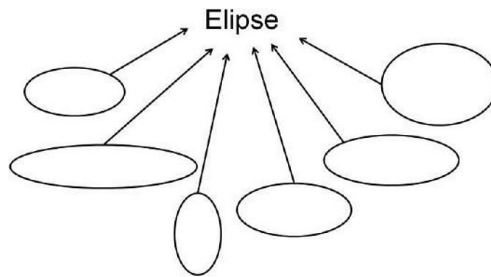


✓ Abstracción

Abstraer significa obtener la esencia al identificar o percibir el problema. Con la abstracción se extraen características comunes a partir de ejemplos específicos. En el ejemplo de la siguiente figura se observa que, de las figuras geométricas con diferentes tamaños y excentricidades, se hace la abstracción de que todas ellas son, en esencia, elipses porque todas tienen en común las características de una elipse. Esto nos permite simplificar y unificar las funciones o procedimientos a codificar ya que, en lugar de

² Gomez, M. C., Ojeda, J. C., & González, P. P. (2019). [Imagen]. En Fundamentos de Ingeniería de Software (p. 17).

programar cada figura por separado, crearíamos una función que nos permita obtener una elipse según ciertos datos necesarios.



✓ *Anticipación al cambio*

El software sufre cambios constantemente. Las principales causas de los cambios son: la necesidad de eliminar defectos que no fueron detectados antes de liberar la aplicación, el surgimiento de nuevos requerimientos del usuario o cambios en los requerimientos existentes. Se requiere un esfuerzo especial en las fases iniciales del desarrollo de un proyecto de software para anticipar cómo y dónde es probable que se den los cambios. La gestión de la configuración facilita que se administran las diferentes versiones del software de forma controlada.

✓ *Generalidad*

Generalizar es buscar un problema que sea lo más general posible en lugar de tener varias soluciones especializadas. Para resolver un problema se debe buscar un problema más general que posiblemente esté oculto tras el problema original. El problema original puede reutilizarse. Así, un mismo módulo puede ser invocado desde más de un punto en la aplicación en lugar de tener varias soluciones en módulos especializados.

Por ejemplo, para un software de un colegio podríamos tener una clase que defina al **Estudiante** y otra que defina al **Profesor**.

Generalizar, en este caso, podría ser definir otra clase **Persona**, que incluya las características generales, y hacer que Estudiante y Profesor sean subclases de Persona.

✓ *Incrementalidad*

La experiencia ha demostrado que los requerimientos del usuario van cambiando o se van definiendo mejor mientras se desarrolla el producto. Bajo estas circunstancias, es común que se hagan varias entregas parciales del sistema cada vez más completas. Cuando se construye una aplicación de esta forma, los pasos intermedios pueden ser prototipos del producto final que permiten ir teniendo retroalimentación del usuario y descubrir y acordar así cuáles son sus verdaderos requerimientos.

✓ Separación de intereses

Bajo este principio se separan diferentes aspectos de un problema para concentrarse en un aspecto y después atender los otros. Por ejemplo, si se requiere que un programa sea correcto (que hace lo que debe hacer) y eficiente (el desempeño del software se considera que es el esperado), nos concentramos primero en una solución que resuelva correctamente el problema y posteriormente lo modificamos para lograr mayor eficiencia.

Áreas de aplicación del software

Como ya hemos mencionado al inicio de esta clase, el software es el elemento lógico de un sistema informático, a diferencia del hardware, que es el elemento físico. El software se crea a través de una **metodología de desarrollo de software** con fases y actividades claramente definidas por la ingeniería de software, y sobre las cuales ampliaremos en la próxima clase; y no se fabrica en un sentido clásico como la mayoría de los productos.

Los componentes del software son los programas ejecutables y los datos existentes en una computadora. Dichos datos incluyen distintos tipos de archivos, los cuales se pueden definir como un conjunto ordenado de información expresada en bytes (veremos esta unidad de medida informática dentro de algunas clases), que interpretados de una determinada manera representan información útil, ya sea para el usuario, para el programa o para ambos.

Los archivos informáticos son usados por los programas informáticos. Estos programas crean, modifican, interpretan y borran archivos para su propio uso bajo demanda del usuario o bien de manera automática. Los programadores que crean los programas deciden qué archivos necesitan, cómo se van a usar, y sus extensiones.

Según la función que tenga el programa será el tipo de archivo que utilice (por ejemplo, de imagen, de sonido, de diseño 3d, de base de datos, de código de programación, de texto con formato, etc.).

Las áreas de aplicación del software son diversas. Por ejemplo, en la imagen de la derecha se muestran algunas de dichas áreas.

