UNIDAD I: FUNDAMENTOS DE INGENIERIA DE SOFTWARE

CONTENIDOS

Introducción
1.1. Concepto de ingeniería de software
- 1.1.1. Definiendo la ingeniería de software
- 1.1.2. Modelado de sistemas
  - a. Modelos de contexto
  - b. Modelos de comportamiento
  - c. Modelos de datos
  - d. Modelos de objetos
- 1.1.3. El proceso de la ingeniería de sistemas
  - a. Definición de requerimientos del sistema
  - b. Diseño del sistema
  - c. Desarrollo de los subsistemas
  - d. Integración del sistema
  - e. Instalación del sistema
  - f. Operación del sistema
  - g. Evolución del sistema
  - h. Desmantelamiento del sistema
- 1.1.4. Modelos del ciclo de vida del desarrollo de software

1.2. Marco histórico (proceso evolutivo del software)
- 1.2.1. Evolución de la ingeniería de sistemas

1.3. Áreas de aplicación de la ingeniería de software
1.4. Importancia del software
1.5. Problemas del software

OBJETIVOS DE LA UNIDAD

- Aplicar los conceptos de Ingeniería de Software al proceso de creación de software.
- Definir el alcance del proceso evolutivo de la ingeniería de software.
- Identificar las áreas de aplicación de la Ingeniería de Software.
- Describir la importancia del software en los sistemas de producción actuales.
- Identificar los problemas del software y las mejores prácticas para minimizarlos.

Introducción

La industria del software tiene un efecto transversal en toda la economía y la sociedad por el alto grado de involucramiento de la investigación, el desarrollo, la comercialización y distribución de software; no obstante, la industria del software está evolucionando hacía un modelo más racional para los usuarios, con menos costos de licencia y una mayor prestación de servicios. El desarrollo de software estará regido por la estandarización, el auge de la ingeniería de la web y los sistemas online.

Este contexto ha convertido al "software de computadora en la tecnología individual más importante en el ámbito mundial. También es uno de los ejemplos principales de la ley de las consecuencias imprevistas[1]".

Nadie podría haber previsto que el software estaría relacionado con sistemas de todo tipo: de transporte, médicos, de telecomunicaciones, militares, industriales, de entretenimiento, máquinas de oficina, entre otras. Y si se toma en cuenta la ley de las consecuencias imprevistas, hay muchos efectos que todavía es imposible predecir en el trabajo diario.

Por último, nadie podría haber predicho que millones de programas de computadora tendrían que corregirse, adaptarse y mejorarse conforme pasara el tiempo y que la labor de desarrollar estas

[1] Son los efectos profundos e inesperados en otras tecnologías con las que en apariencia no tiene ninguna relación, como en empresas comerciales, en personas y aún en la cultura en su conjunto.

actividades de "mantenimiento" absorbería más gente y recursos que todo el trabajo aplicado para la creación del software nuevo.

1.1. Concepto de ingeniería de software

1.1.1. Definiendo la ingeniería de software

Para definir el término ingeniería de software comenzaremos por definir ¿qué es software? ¿qué características lo hacen diferente? y ¿cuáles son los atributos de un buen software?

El software se forma con las instrucciones (programas de computadora) que al ejecutar se proporcionan las características, funciones y el grado de desempeño deseado; las estructura de datos que permiten que los programas manipulen información de manera adecuada y los documentos que describen la operación y el uso de los programas (PRESSMAN, 2006).

Para entender el software (y la ingeniería de software), es importante examinar las características que lo hacen diferente de otras cosas que construye el ser humano:

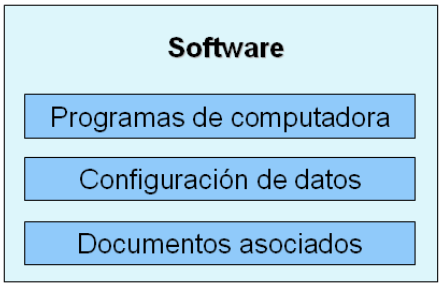

Figura 1. Definición de software.

- El software se desarrolla o construye; no se manufactura en el sentido clásico.
- El software no se "desgasta".
- A pesar que la industria tiene una tendencia hacia la construcción por componentes, la mayoría del software aún se construye a la medida.

El software es el conjunto completo de programas, procedimientos y documentación relacionada que se asocia con un sistema, y especialmente con un sistema de computadora. En un sentido específico, software son los programas de computadora [SANCHEZ, 2012].

El software debe tener la funcionalidad y el rendimiento requeridos por el usuario, además de ser mantenible, confiable y utilizable.

Tabla 1. Atributos esenciales de un buen software.

Características del producto	Descripción
Mantenibilidad	El software debe escribirse de tal forma que pueda evolucionar para cumplir las necesidades de cambio de los clientes. Este es un atributo crítico debido a que el cambio en el software es una consecuencia inevitable de un cambio en el entorno de negocios.
Confiabilidad	La confiabilidad del software tiene un gran número de características, incluyendo la fiabilidad, seguridad y protección. El software confiable no debe causar daños físicos o económicos en el caso de una falla del sistema.
Eficiencia	El software no debe hacer que se malgasten los recursos del sistema, como la memoria y los ciclos de procesamiento. Por lo tanto, la eficiencia incluye tiempos de respuesta y de procesamiento, utilización de la memoria, entre otros.

Características del producto	Descripción
Usabilidad	El software debe ser fácil de utilizar, sin esfuerzo adicional, por el usuario para quien está diseñado. Esto significa que debe tener una interfaz de usuario apropiada y una documentación adecuada.

Ingeniería de software

La ingeniería de software ha sido definida de diferentes maneras y por diferentes autores. Algunas definiciones a utilizar en este curso son:

Es una disciplina que comprende todos los aspectos de la producción del software desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de éste después que se utiliza (SOMMERVILLE, 2002).

La ingeniería de software es el establecimiento y uso de principios sólidos de ingeniería para obtener económicamente un software confiable y que funcione de modo eficiente en máquinas reales (BAUER, NAU69).

La ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software; es decir, la aplicación de la ingeniería de software y el estudio de enfoques sistemático (IEEE93).

La ingeniería de software centra su interés en el desarrollo de componentes eficientes, es decir, capaces de utilizar en la mejor forma posible los recursos de computación y comunicación disponibles.

La ingeniería de software es una tecnología estratificada. La base que soporta la ingeniería de software es el enfoque en la calidad. Para alcanzar niveles de calidad aceptable requiere de adoptar enfoques de gestión de la calidad y el fomento a una cultura de mejora continua.

"Un proceso define quién está haciendo qué, cuándo y cómo lograr cierta meta" (Ivar Jacobson, Grady Booch y James Rumbaugh).

Figura 2. Estratos de la ingeniería de software.

Los métodos de la ingeniería de software se basan en un conjunto de principios básicos que gobiernan cada área de la tecnología e incluye actividades de modelado y otras técnicas descriptivas.

Las herramientas de ingeniería de software proporcionan el soporte automatizado o semiautomatizado para el proceso y los métodos.

Es importante diferenciar los conocimientos científicos que se aplican en la Ingeniería de Software, la ciencia de la Ingeniería de Software en sí misma, y la práctica de la ingeniería [SANCHEZ, 2012]:

- Las **ciencias que se aplican en la Ingeniería de Software** son la ciencia de la computación y otras ciencias que son de utilidad para aspectos determinados, como las relativas a la organización, la economía, la psicología, y por supuesto las matemáticas en general.
- La **Ingeniería de Software** como ciencia es la aplicación del método científico a la teorización y creación de conocimiento sobre la propia Ingeniería de Software. Está dedicada al estudio de sus actividades, y centrada en generar teorías, modelos explicativos o enunciados descriptivos sobre la práctica de la ingeniería.

- La **práctica de la ingeniería**, que está orientada a prescribir cómo deben realizarse las actividades propias de la disciplina. Es un aspecto complementario con la ciencia de la ingeniería, pues la ciencia necesita de la observación de la práctica, y la práctica a su vez se perfecciona de acuerdo con el conocimiento generado por la ciencia.

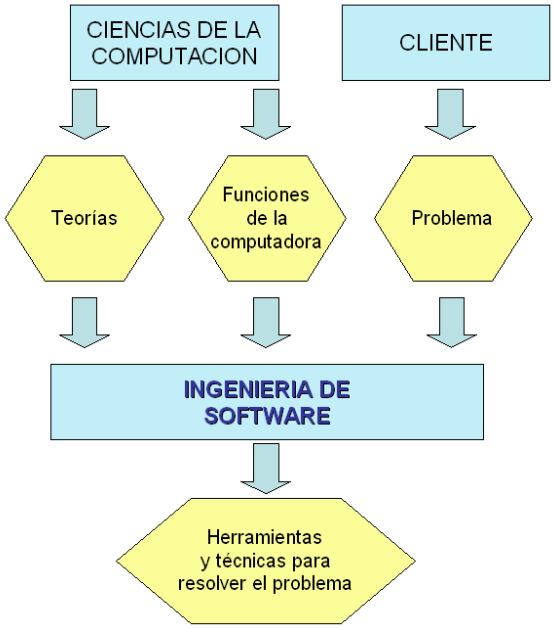

Figura 3. Relación entre ciencias de la computación e ingeniería de software.

1.1.2. Modelado de sistemas

Un modelo de sistemas es un conjunto de componentes relacionados recíprocamente, denominados subsistemas. Estos componentes vistos desde la perspectiva de subsistemas proporcionan una función única.

El modelado de sistemas es un elemento importante del proceso de ingeniería de sistemas. Sin importar que el enfoque esté en la visión global o en la visión detallada, el ingeniero crea modelos que:

- Definen los procesos que satisfacen las necesidades de la visión que se considera.
- Representen el comportamiento de los procesos y los supuestos en los que se basa el comportamiento.
- Define de modo explícito las entradas exógenas y endógenas de información al modelo.
- Representan todas las uniones (incluidas las salidas) que permiten al ingeniero entender mejor la visión.

Los modelos se utilizan en el proceso de análisis para desarrollar una comprensión del sistema existente a reemplazar o a mejorar, o para especificar el sistema requerido. Se pueden utilizar para representar el sistema desde diferentes perspectivas:

- Una perspectiva externa en la que se modela el contexto o entorno del sistema.
- Una perspectiva de comportamiento en la que se modela el comportamiento del sistema.
- Una perspectiva estructural en la que se modela la arquitectura del sistema o la estructura de los datos procesados por este.

a. Modelos de contexto

En una de las primeras etapas de la obtención de requerimientos y del proceso de análisis se deben definir los límites del sistema. Esto comprende trabajar conjuntamente con los stakeholders del sistema para distinguir lo que es el sistema y su entorno.

El límite entre el sistema y su entorno se debe trazar durante el proceso de ingeniería de requerimientos. Una vez que se han tomado las decisiones sobre los límites del sistema, una parte de la actividad de análisis es la definición de ese contexto y las dependencias que un sistema tiene con su entorno. Por lo regular, el primer paso en esta actividad es la producción de un modelo arquitectónico sencillo.

Por lo general los modelos arquitectónicos de alto nivel se expresan como diagramas de bloque en los que cada subsistema es representado por un rectángulo que incluye una frase y líneas que indican que existe algún tipo de asociaciones entre los subsistemas.

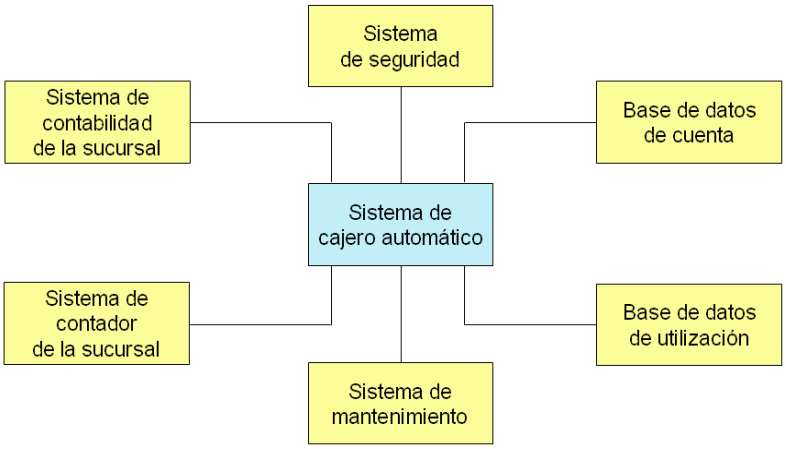

Figura 6. Modelo arquitectónico que ilustra la estructura del sistema de información incluido en una red de cajeros automáticos.

Los modelos arquitectónicos describen el entorno de uno. Sin embargo, no muestran las relaciones entre otros sistemas del entorno y el sistema que se está especificando. Todas esas relaciones podrían afectar los requerimientos del sistema que se está definiendo, por lo que deben tomarse en cuenta.

Habitualmente los modelos arquitectónicos sencillos se complementan con otros modelos, como los de procesos, los cuales muestran las actividades de los procesos comprendidos en el sistema, y los de flujo de datos, que muestran como s transfieren los datos entre el sistema y otros sistemas de su entorno.

b. Modelos de comportamiento

Estos modelos se utilizan para describir el comportamiento completo del sistema. Se discuten dos tipos de modelos de comportamiento:

- Los modelos de flujo de datos.
- Los modelos de máquina de estado.

Modelos de flujo de datos

Son una forma intuitiva de mostrar la manera en que un sistema procesa los datos. En el nivel de análisis, se utilizan para modelar la forma en la que estos se procesan en el sistema existente. La notación utilizada en estos modelos representa el procesamiento funcional, los almacenes y los movimientos de datos entre las funciones.

Los modelos de flujo de datos se utilizan para mostrar cómo fluyen los datos a través de una secuencia de pasos de procesamiento. Los datos se transforman en cada paso antes de moverse a la siguiente etapa. Si los diagramas de flujo de datos se utilizan para documentar un diseño de software, estos pasos de procesamiento o transformaciones se representan por funciones en los programas. Sin embargo, en un modelo de análisis, el procesamiento se puede llevar a cabo por las personas o por las computadoras.

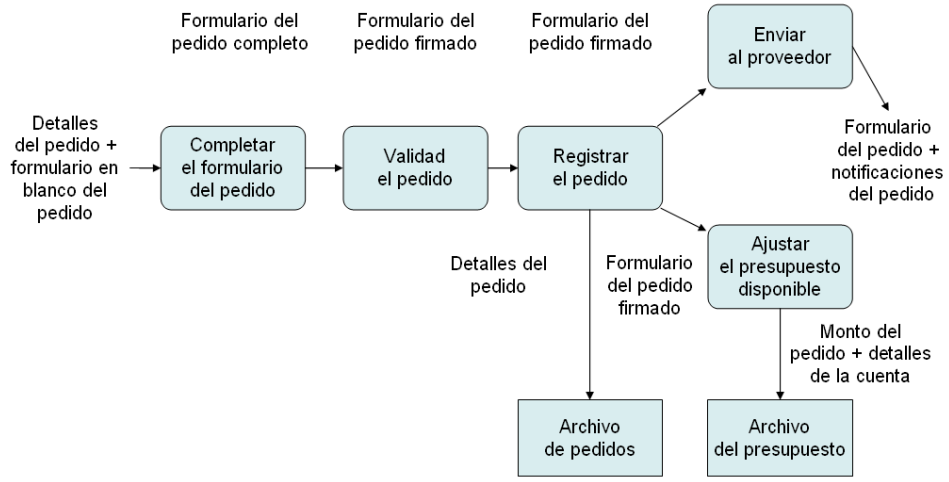

Figura 7. Diagrama de flujo de datos del procedimiento de pedidos.

Modelos de máquina de estado

Los modelos de máquina de estado se utilizan para modelar el comportamiento de un sistema en respuesta a eventos internos o externos. Dichos modelos muestran los estados del sistema y los eventos que provocan las transiciones de un estado a otro. No muestra el flujo de datos del sistema. Este tipo de modelo es de utilidad para modelar los sistemas de tiempo real debido a que estos sistemas a menudo están dirigidos por estímulos provenientes del entorno del sistema.

Un modelo de máquina de estado de un sistema supone que, en cualquier momento, el sistema está en uno de varios estados posibles. Cuando se recibe un estímulo, este dispara una transición a un estado diferente.

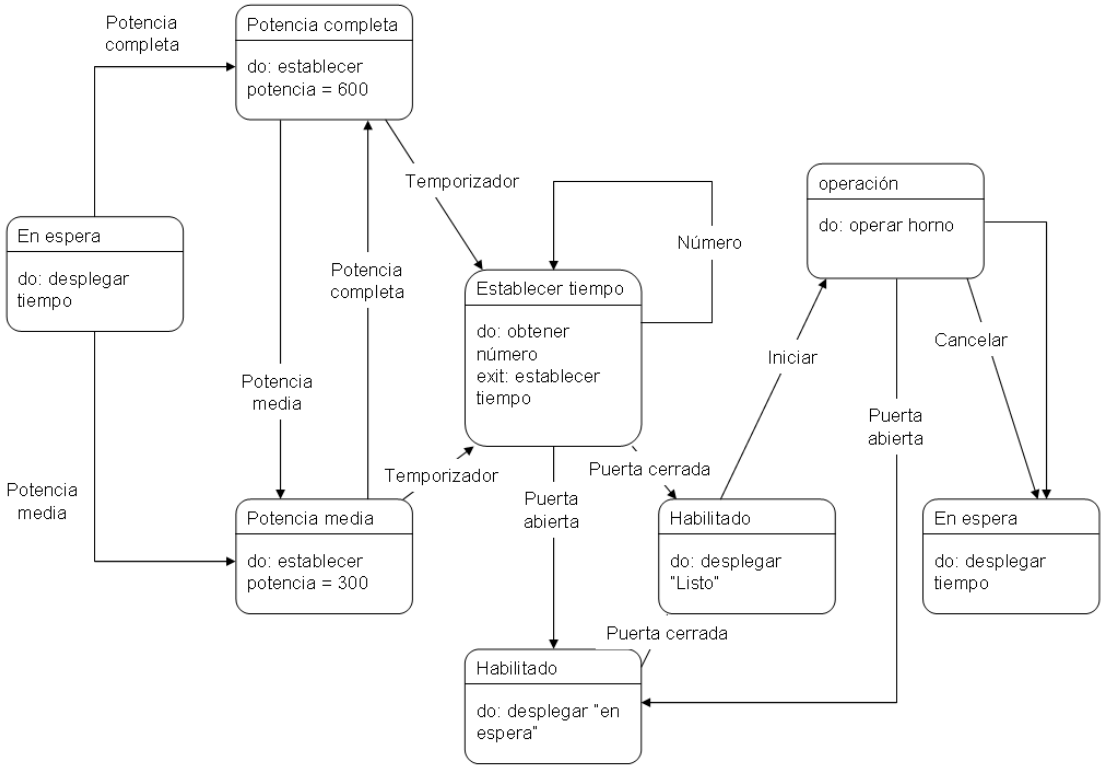

Figura 8. Modelo de máquinas de estado de un horno de microondas sencillo.

c. Modelos de datos

Muchos de los sistemas de software grande utilizan bases de datos de información de gran tamaño. En algunos casos, esta base de datos existe de forma independiente del sistema de software; en otros, se crea para el sistema que se está desarrollando. Una parte importante del modelado de sistemas es definir la forma lógica de los datos procesados por el sistema.

La técnica de modelado de datos más utilizada es la de entidad-relación-atributo (modelado ERA) que muestra las entidades de datos, sus atributos asociados y las relaciones entre estas entidades.

UML (Unified Modeling Language - Lenguaje Unificado de Modelado) no incluye una notación específica para este tipo de modelado de datos ya que supone un proceso de desarrollo orientado a objeto y modela los datos utilizando objetos y sus relaciones.

A menudo, los modelos de datos se utilizan en conjunto con los de flujo de datos para describir la estructura de la información que se está procesando.

Debido a la limitante que tienen los modelos ERA se sugiere recolectar las descripciones más detalladas de las entidades, las relaciones y los atributos que se incluyen en el modelo en un repositorio o diccionario de datos. Los diccionarios de datos se utilizan para administrar toda la información proveniente de todos los tipos de dichos modelos.

El diccionario contiene las características lógicas de los sitios donde se almacenan los datos del sistema, incluyendo nombre, descripción, alias, contenidos y organización; también identifica los procesos donde se emplean los datos y los sitios donde se necesita el acceso inmediato a la información. Sirve como punto de partida para identificar los requerimientos de las bases de datos durante el diseño del sistema

Los analistas utilizan los diccionarios de datos por cinco razones importantes:
1. Para manejar los detalles en sistemas grandes.

2. Para comunicar un significado común para todos los elementos del sistema.
3. Para documentar las características del sistema.
4. Para facilitar el análisis de los detalles con la finalidad de evaluar las características y determinar dónde efectuar cambios en el sistema.
5. Localizar errores y omisiones en el sistema.

El diccionario de datos debe ser visto como una actividad paralela al análisis y diseño de sistemas, ya que este no es un fin en si mismo, ni debe serlo.

Es importante considerar que el diccionario de datos siempre debe estar actualizado, en caso contrario se vuelve simplemente una colección histórica.

Las estructuras de datos para un diccionario de datos se describen usando una notación algebraica, con los siguientes símbolos:
1. Un signo de igual (=) significa "está compuesto de".
2. Un signo de suma (+) significa "y".
3. Las llaves {} indican elementos repetitivos, también llamados grupos de repetición o límites superiores e inferiores para el número de repeticiones.
4. Los corchetes [] representan una situación de uno u otro. Se podría representar un elemento u otro, pero no ambos. Los elementos listados entre los corchetes son mutuamente excluyentes.
5. Los paréntesis () representan un elemento opcional. Los elementos opcionales se podrían dejar en blanco en la entrada de las pantallas y podrían contener espacios o ceros para campos numéricos en las estructuras de archivos.

Tabla 2. Ejemplo de un diccionario de datos.

Cliente =	Nombre + Nombre + Dirección + Saldo actual + {Información del pedido} + Pago
Nombre =	Apellido + Nombre + (Inicia el segundo nombre)
Dirección =	Calle + (Departamento) + Ciudad + Estado + Código postal + País
Información del pedido =	Número del pedido + Fecha del pedido + Fecha de envío + Total
Pago =	[Cheque | Tarjeta de crédito] + Fecha de pago + Monto del pago
Cheque =	Número de cheque
Tarjeta de crédito =	Número de tarjeta de crédito + Fecha de expiración

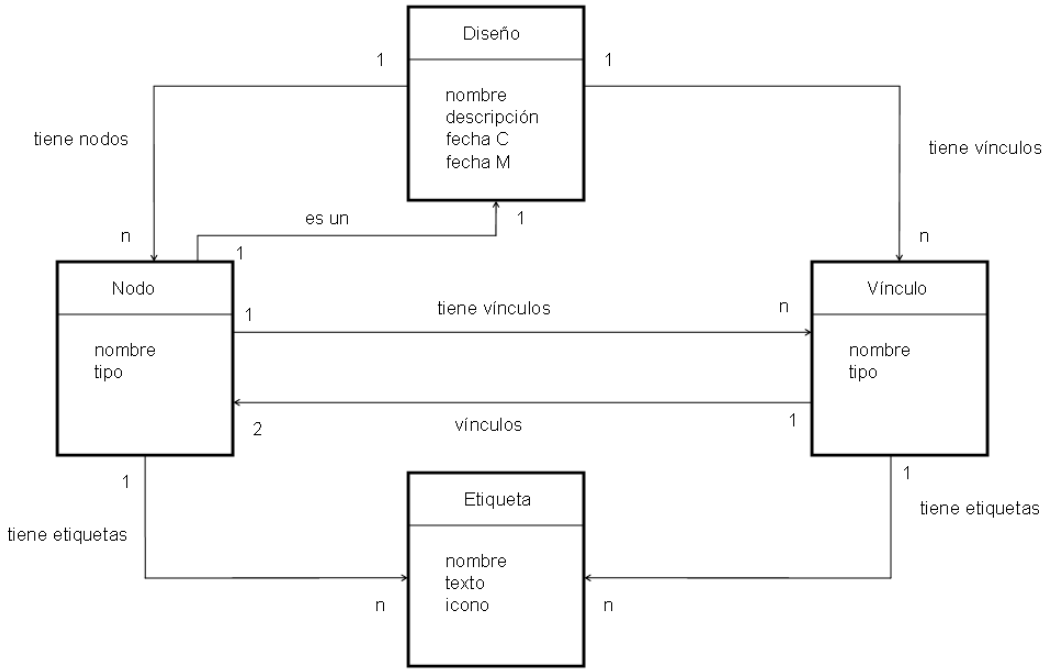

Figura 9. Modelo de datos semántico de un diseño de software.

d. Modelos de objetos

Utilizar un modelo de objetos implica:

- Los requerimientos del sistema utilizan un modelo de objetos.
- El diseño del sistema utiliza un modelo de objetos
- Uso de un enfoque orientado a objetos para el desarrollo del software (se utiliza un lenguaje de programación orientado a objetos como Java o C++).

Los modelos de objetos desarrollados durante el análisis de requerimientos se utilizan para representar los datos del sistema y su procesamiento, combinando algunas de las formas de utilización de los modelos de flujos de datos y semántico de datos. También son útiles para mostrar la manera en que las entidades en el sistema se clasifican y se componen de otras entidades.

Los modelos de objetos desarrollados durante el análisis de requerimientos simplifican la transición al diseño y programación orientada a objetos. Sin embargo, a menudo se observa que los usuarios finales de un sistema encuentran dichos modelos pocos naturales y difíciles de comprender; por consiguiente la recomendación es complementar dichos modelos con los de flujo de datos que muestran el procesamiento de datos en el sistema.

Actualmente el estándar efectivo para el modelado de objetos es UML. La “orientación a objetos” es un enfoque de desarrollo de software que organiza tanto el problema como la solución como una colección de objetos discretos; tanto la estructura de datos como el comportamiento están incluidos en la representación.

Una representación orientada a objetos puede reconocerse por sus siete características:

- Identidad
- Abstracción
- Clasificación
- Encapsulamiento
- Herencia
- Polimorfismo
- Persistencia

- **Identidad.** Se refiere al hecho que los datos son organizados en entidades discretas, distinguibles, denominadas objetos.

- **Abstracción.** Es esencial para construir cualquier sistema, sea orientado a objetos o no. La abstracción en un sistema orientado a objetos (OO) ayuda a representar los diferentes puntos de vista incorporados en el sistema que se desarrolla. En conjunto, las abstracciones forman una jerarquía que muestra cómo las diferentes perspectivas de un sistema se relacionan unas con otras. La abstracción es una técnica que ayuda a identificar qué información específica debe ser visible y cuál debe ser ocultada.

- **Clasificación.** Se utiliza para agrupar objetos que tienen atributos y comportamientos en común.

- **Encapsulamiento.** Es la manera en la cual los métodos forman un límite protector alrededor del objeto, aislándolo de cosas que ocurren a otros objetos. Una clase encapsula los atributos y comportamientos de un objeto, ocultando los detalles de implementación; sin embargo, encapsulamiento no es lo mismo que ocultamiento de información. El encapsulamiento es un técnica para empaquetare la información de tal forma que se oculte lo que debe ser ocultado y se haga visible lo que está pensado que sea visible.

- **Herencia.** Son las propiedades comunes entre clases y subclases organizadas jerárquicamente.

- **Polimorfismo.** Es el comportamiento manifiesto de manera diferente en diferentes clases o subclases.

- **Persistencia.** Es la capacidad del nombre, estados y comportamientos de un objeto para trascender el espacio y el tiempo; en otras palabras, el nombre, estado y comportamiento de un objeto se conservan cuando el objeto es transformado.

Las ventajas más importantes de la programación orientada a objetos son las siguientes:

- **Mantenibilidad (facilidad de mantenimiento).** Los programas que se diseñan utilizando el concepto de orientación a objetos son más fáciles de leer y comprender y el control de la complejidad del programa se consigue gracias a la ocultación de la información que permite dejar visibles sólo los detalles más relevantes.
- **Modificabilidad (facilidad para modificar los programas).** Se pueden realizar añadidos o supresiones a programas simplemente añadiendo, suprimiendo o modificando objetos.
- **Reusabilidad.** Los objetos, si han sido correctamente diseñados, se pueden usar numerosas veces y en distintos proyectos.
- **Fiabilidad.** Los programas orientados a objetos suelen ser más fiables ya que se basan en el uso de objetos ya definidos que están ampliamente testados.

Conceptos básicos de la orientación a objeto

Objeto. Un objeto es una cosa, generalmente extraída del vocabulario del espacio del problema o del espacio de la solución. Todo objeto tiene un nombre (se le puede identificar), un estado (generalmente hay algunos datos asociados a él) y un comportamiento (se le pueden hacer cosas a objeto y él puede hacer cosas a otros objetos). Un objeto es una instancia de una clase.

Clase. Es una descripción de un conjunto de objetos similares. Por ejemplo la clase *Coches.* Una clase contiene los atributos y las operaciones sobre esos atributos que hacen que una clase tenga la entidad que se desea.

Una clase describe un conjunto de objetos que comparten una estructura común y tienen comportamientos comunes. En una clase todos los objetos tienen el mismo conjunto de atributos y el mismo número de operaciones; difieren sólo en los valores de sus atributos respectivos.

Atributo. Es una característica concreta de una clase. Por ejemplo atributos de la clase Coches pueden ser el Color, el Numero de Puertas...

Método. Es una operación concreta de una determinada clase. Por ejemplo de la clase Coches podríamos tener un método arrancar() que lo que hace es poner en marcha el coche.

Instancia. Es una manifestación concreta de una clase (un objeto con valores concretos). También se le suele llamar ocurrencia. Por ejemplo una instancia de la clase Coches puede ser: un Ford Mustang, de color Gris con 3 puertas.

Para comprender mejor en qué consiste la orientación a objetos analicemos la siguiente analogía: el presidente Francisco Flores gobernó El Salvador desde 1999 a 2004; asimismo el presidente Enrique Bolaños gobernó Nicaragua desde el 2002 a 2007.

Nombre:	presidente Francisco Flores
País:	El Salvador
inicioDeGobierno:	1999
finDeGobierno:	2004

Tabla 3. Presidente Francisco Flores de El Salvador

Nombre:	Presidente Enrique Bolaños
País:	Nicaragua
inicioDeGobierno:	2002
finDeGobierno:	2007

Tabla 4. Presidente Enrique Bolaños de Nicaragua

Al comparar las tablas 3 y 4 queda claro que todos los presidentes tienen algo en común: los presidentes tienen un nombre, gobiernan un país específico y su gobierno comienza en un año específico y termina en otro año específico.

Entonces tenemos que Presidente es una clase y nombre, país, inicioDeGobierno y finDeGobierno son atributos de la Clase Presidente. Por otra parte, un presidente gobierna y si suspende su gobierno, abdica. Estas funciones realizadas por los presidentes se les denominan operaciones.

De acuerdo al ejemplo, los atributos del presidente Francisco Flores son {presidente Francisco Flores, El Salvador, 1999, 2004} y los atributos del presidente Enrique Bolaños son {presidente Enrique Bolaños, Nicaragua, 2002, 2007}.

Para el mismo ejemplo el presidente Francisco Flores es una instancia de la Clase Presidente y también lo es el presidente Enrique Bolaños; es decir, una clase es un conjunto de objetos relacionados y, a la inversa, un objeto es una instancia de una clase.

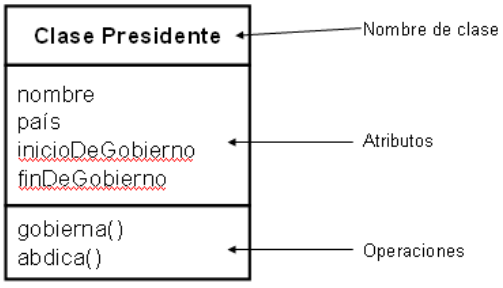

Figura 10. Representación en UML de la Clase Presidente.

Una clase se representa en UML como se muestra en la figura 10: aparecen tres recuadros, el recuadro superior contiene el nombre de la clase, en negritas con la letra inicial de los nombres en mayúsculas; el recuadro al centro contiene los nombres de los atributos de una instancia de esa clase, es decir, de un objeto de ese tipo y el recuadro inferior contiene los nombres de las operaciones que pueden realizarse por o con una instancia de esa clase.

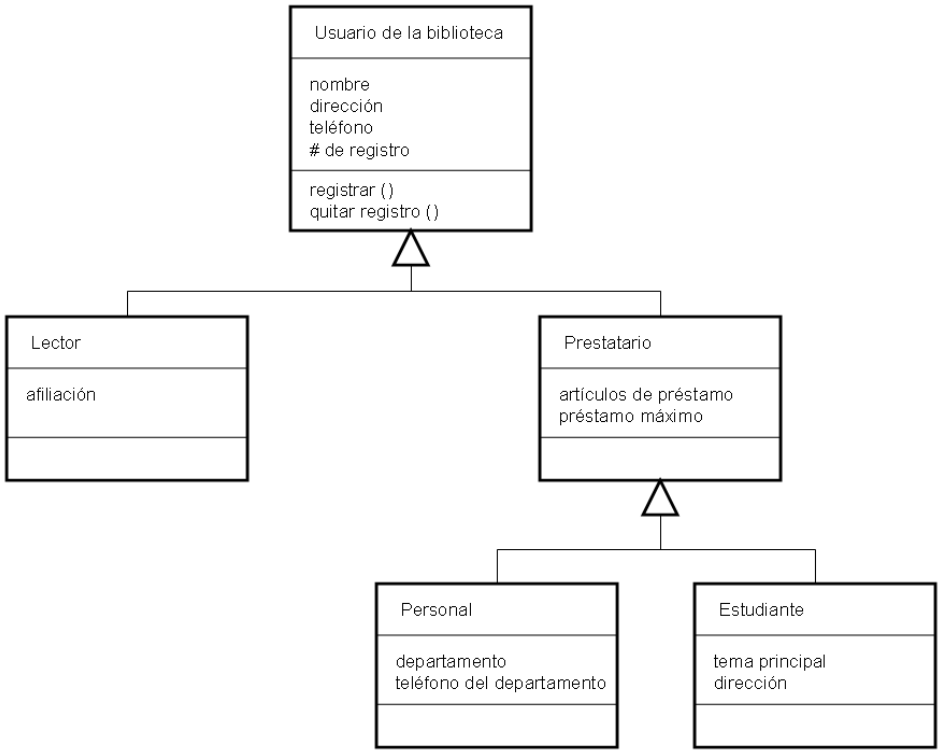

Figura 11. Clase jerárquica del usuario de un sistema de biblioteca.

1.1.3. El proceso de ingeniería de sistemas

Existen diferencias importantes entre el proceso de la ingeniería de sistemas y el desarrollo de software:

- **Implicación interdisciplinaria:** varias disciplinas de la ingeniería se conjuntan en la ingeniería de sistemas.

- **Alcance reducido para rehacer el trabajo durante el desarrollo de sistemas:** una vez que se han tomado decisiones en la ingeniería de sistemas cuesta mucho cambiarla. Raramente es posible rehacer el trabajo en el diseño de sistemas para resolver estos problemas. Una razón por la que el software ha llegado a ser tan importante en los sistemas es que permite flexibilidad en los cambios que se hacen durante el desarrollo de sistemas como respuesta a los nuevos requerimientos.

Para comprender mejor el alcance del proceso de ingeniería de sistema se iniciará definiendo qué es un proceso de software: es un conjunto coherente de políticas, estructuras organizativas, tecnologías, procedimientos y artefactos que se necesitan para concebir, desarrollar, implantar y mantener un producto software.

Esta definición abarca todo el ciclo del software, desde la concepción hasta el mantenimiento (ver la Figura 12).

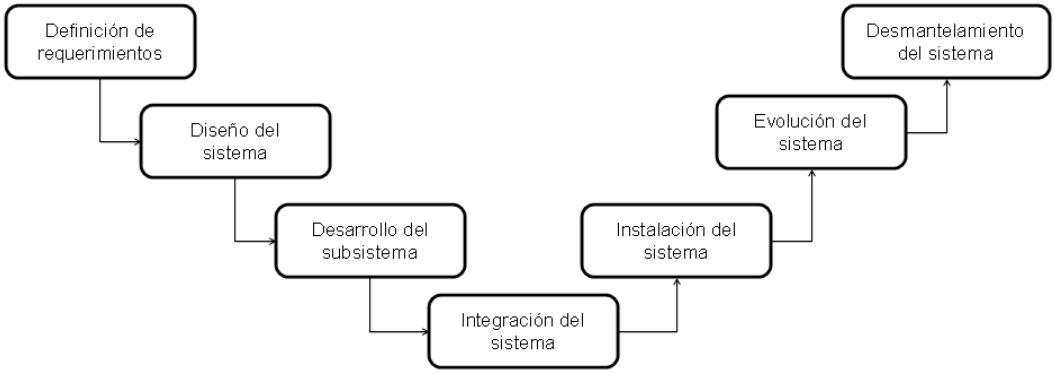

Figura 12. El proceso de la ingeniería de sistemas.

La IEEE propone, a través de la Guía SWEBOK[2], 5 etapas o fases en el proceso de ingeniería de software:

- Requerimientos de software
- Diseño de software
- Construcción de Software
- Pruebas de software (testing)
- Mantenimiento de software

a. Definición de requerimientos del sistema

La actividad de definición de requerimientos del sistema pretende descubrir los requerimientos completos de éste. El proceso requiere consultar con los clientes del sistema y con los usuarios finales. Esta fase de definición de requerimientos usualmente se concentra en la desviación de tres tipos de requerimientos:

Requerimientos funcionales abstractos: las funciones básicas que el sistema debe proporcionar se definen en un nivel abstracto. La especificación detallada de requerimientos funcionales tiene lugar en el nivel de subsistemas.

Propiedades del sistema: estas son propiedades emergentes no funcionales del sistema. Incluyen propiedades como la disponibilidad, el rendimiento, la protección, entre otras. Estas propiedades no funcionales del sistema afectan los requerimientos para todos los subsistemas.

Características que no debe mostrar el sistema: algunas veces tiene igual importancia especificar lo que el sistema debe y no debe hacer.

Una parte importante de la fase de definición de requerimientos es establecer un conjunto completo de objetivos que el sistema debe cumplir. Este no necesariamente debe expresarse en términos de la funcionabilidad del sistema, pero debe definir el porqué se construye el sistema para un entorno particular.

b. Diseño del sistema

El diseño de sistema se centra en proporcionar la proporcionalidad del sistema a través de sus diferentes componentes. Las actividades que se realizan en este momento son:

[2] Software Engineering Body of Knowledge, la cual es un compendio de conocimientos compilados para la profesión de Ingeniería de Software.

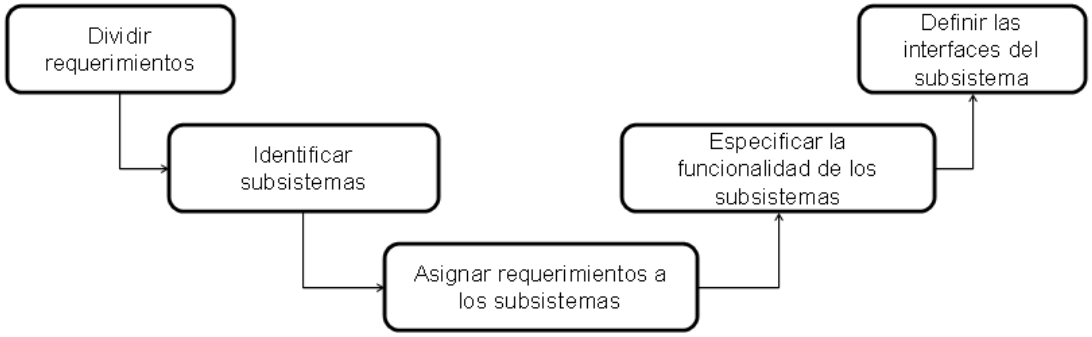

Figura 13. El proceso diseño de sistemas.

Dividir requerimientos. Los requerimientos se analizan y se recolectan en grupos relacionados. Normalmente existen varias opciones posibles de división, la mayoría de las cuales puede producirse en esta etapa del proceso.

Identificar subsistemas. Se identifican los diferentes subsistemas que pueden, individual o colectivamente, cumplir con los requerimientos. Los grupos de requerimientos están normalmente relacionados con los subsistemas, de tal forma que esta actividad y la de partición de requerimientos se ven disminuidas. Sin embargo, la identificación de subsistemas se puede ver influenciada por otros factores organizacionales y del entorno.

Asignar requerimientos a los subsistemas. Los requerimientos son asignados a los subsistemas. En principio, esto debe ser directo si la partición de requerimientos se utiliza para la identificación de subsistemas. En la práctica, no existe igualdad entre las particiones de requerimientos y la identificación de subsistemas.

Especificar la funcionalidad de los subsistemas. Se deben enumerar las funciones específicas asignadas a cada subsistema. Esto puede verse como parte de la fase de diseño del sistema o, si el subsistema es un sistema de software, como parte de la actividad de especificación de requerimientos para ese sistema. En esta etapa, también se deben especificar las relaciones entre los subsistemas.

Definir las interfaces del subsistema. Esto comprende definir las interfaces necesarias y requeridas por cada subsistema. Una vez que estas interfaces se han acordado, es posible el desarrollo paralelo de los subsistemas.

En este proceso de diseño existe un compromiso de realimentación e iteración de una etapa a la otra. A menudo es necesario rehacer el trabajo cuando surgen problemas y preguntas.

c. Desarrollo de los subsistemas

Durante el desarrollo de los subsistemas se siguen las siguientes etapas:

- Iniciar un proceso del software (esto comprende requerimientos, diseño, implementación, entre otros).
- Construcción de los subsistemas (en algunos casos se compran los subsistemas para integrarse en el sistema). Es común que diferentes subsistemas se desarrollen en paralelo.

Al respecto se deben hacer las siguientes consideraciones:

- A menudo se deben realizar 'revisiones de trabajo' con el fin de detectar los problemas. Estas 'revisiones de trabajo' comúnmente implican cambios en el software debido a la flexibilidad inherente a él.
- Es importante diseñar software para el cambio.

d. Integración del sistema

Consiste en tomar subsistemas desarrollados de forma independiente y conjuntar para crear el sistema completo. La integración se puede llevar a cabo utilizando el enfoque del 'big bang' que consiste en integrar todos los sistemas al mismo tiempo. Sin embargo, por razones tanto técnicas como de administración, el mejor enfoque es un proceso de integración creciente donde los sistemas se integran uno a uno.

Este proceso creciente es el enfoque más apropiado por dos razones:
1. Por lo general, es imposible calendarizar todos los desarrollos de los diferentes subsistemas de tal forma que éstos se completen al mismo tiempo.
2. La integración creciente reduce el costo en la localización de errores. Si varios subsistemas se integran simultáneamente, se pueden localizar errores que surjan durante las pruebas en cualquiera de estos subsistemas. Cuando un único subsistema se integre a un sistema en funcionamiento, los errores que ocurran estarán probablemente en el subsistema recién integrado o en las interacciones entre los subsistemas existentes y el nuevo subsistema.

e. Instalación del sistema

Durante la instalación del sistema, este se ubica en el entorno en el cual se pretende que opere. Aunque esto puede parecer un proceso sencillo, surgen muchos problemas que implican que la instalación de un sistema complejo puede llevar meses o incluso años.

Algunos problemas típicos pueden ser:
- El entorno en el cual el sistema se instala no es el mismo que el supuesto por los desarrolladores del sistema.
- Los usuarios potenciales del sistema pueden ser hostiles a su introducción, ya que puede reducir su responsabilidad y el número de empleos en la organización.
- Un nuevo sistema puede convivir con uno existente hasta que la organización esté satisfecha con el funcionamiento del nuevo sistema.
- Puede haber muchos problemas en la instalación física.

f. Operación del sistema

Operar el sistema implica:
- Organizar sesiones de entrenamiento para los operadores.
- Cambiar el proceso normal de trabajo con el fin de que la utilización del nuevo sistema sea efectiva.

Los problemas no detectados pueden surgir en esta etapa debido a que la especificación del sistema puede contener errores u omisiones. Aunque el sistema funcione acorde a la especificación, sus funciones pueden no cumplir las necesidades reales de operación. En consecuencia, el modo de utilizar el sistema no es el que sus diseñadores habían previsto.

g. Evolución del sistema

Los sistemas grandes y complejos tienen un período de vida largo. Durante su vida, tienen que evolucionar para corregir errores en los requerimientos del sistema original y cumplir los nuevos requerimientos. Los sistemas de cómputo son reemplazados con nuevas máquinas más rápidas. La organización que utiliza el sistema puede reorganizarse y utilizar el sistema de formas diferentes. El entorno externo del sistema puede cambiar y forzar a cambios en el sistema.

La evolución del sistema, como la del software, es costosa por varias razones:
- Los cambios propuestos tienen que analizarse cuidadosamente desde perspectivas técnicas y de negocios. Dichos cambios deberán ser aprobados por cierto número de personas antes de su realización.

- Debido a que los subsistemas nunca son completamente independientes, los cambios en uno pueden afectar de forma adversa el funcionamiento o comportamiento de otros. Por lo tanto, será necesario cambiar estos subsistemas.
- A menudo las razones del diseño original no son registradas. Los responsables de la evolución del sistema tienen que resolver el por qué se tomaron decisiones particulares de diseño.
- Al paso del tiempo, su estructura se corrompe por el cambio de tal forma que se incrementan los costos en los cambios adicionales.

h. Desmantelamiento del sistema

El desmantelamiento del sistema significa poner fuera de servicio a dicho sistema después de que termina su período de utilidad operativa. Algunas veces esto es directo; sin embargo, algunos sistemas pueden contener materiales que son potencialmente peligrosos para el entorno. Durante la fase de diseño, la ingeniería de sistemas debe anticipar el desmantelamiento y tomar en cuenta los problemas de desecho de los materiales.

En cuanto al software concierne, existen problemas de desmantelamiento. Sin embargo, alguna funcionalidad del software se puede incorporar en un sistema para ayudar al proceso de desmantelamiento. Por ejemplo, el software se puede utilizar para controlar el estado de otros componentes del sistema. Cuando el sistema se desmantela, los componentes en buen estado se pueden identificar y reutilizar en otros sistemas.

Si la organización desea conservar los datos del sistema que se está desmantelando, éstos deben convertirse para utilizarlos en otros sistemas. A menudo esto implica un costo, ya que la estructura de datos puede estar implícitamente definida en el software mismo.

1.1.4. Modelos del ciclo de vida del desarrollo de software

Dentro de los esfuerzos por convertir las tareas de desarrollo de sistemas en una rama de la ingeniería, otra de las contribuciones cruciales ha sido la de crear o definir un esquema del proceso o procedimiento genérico, denominado también ciclo de vida.

El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE - Institute of Electrical and Electronics Engineers) define el ciclo de vida de los sistemas como “Una aproximación lógica a la adquisición, el suministro, el desarrollo, la utilización y el mantenimiento del software” (IEEE 1074).

Mientras que el estándar ISO 12207-1, lo define como “Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la implantación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requerimientos hasta la finalización de su uso”.

En cualquier caso, es claro que un modelo de ciclo de vida define un conjunto de fases, etapas y actividades, a través del cual se desarrollan y mantienen los sistemas, lo cual contribuye a crear un marco de referencia para planificar y administrar los proyectos de desarrollo de sistemas.

En la Tabla 5, se definen algunos modelos de procesos del ciclo de vida del software, por su relevancia y vigencia en la actualidad.

Tabla 5. Algunos modelos de procesos del ciclo de vida del software.

Modelo del ciclo de vida	Descripción
Modelo en cascada	Es tal vez el modelo más ampliamente difundido dentro de los métodos clásicos, y si bien no existe una única formulación del mismo, su característica distintiva es la de llevar a cabo distintas fases de desarrollo en secuencia, comenzando cada una de ellas en el punto en que terminó la anterior.

Modelo del ciclo de vida	Descripción
Modelo en "V"	Es una evolución del modelo en cascada en el que se enfatizan las actividades de verificación y validación.
Modelos de proceso basados en prototipo	Un prototipo software es un modelo ejecutable de un sistema futuro que implementa solo una pequeña parte de la funcionalidad, pero permite a los clientes, usuarios y desarrolladores adquirir experiencia con la arquitectura y la funcionalidad. Hay dos tipos básicos de procesos basados en prototipos: prototipos desechables y prototipos evolutivos.
Modelo en espiral	Este modelo propugna evaluar los riesgos de forma regular en el proyecto y, si procede, tomar acciones para mitigar el impacto de esos riesgos a medida que avanza el desarrollo

1.2. Marco histórico (proceso evolutivo del software)

La premisa principal que sustenta el desarrollo de software es la resolución de problemas a través de herramientas y técnicas.

Foro de discusión

La ingeniería de software y el proceso evolutivo del software

Comenzaremos "calentando motores" reflexionando un poco sobre algunos conceptos elementales de la Ingeniería de software.

Un vistazo rápido a la Ingeniería de software: ¿Quién hace el software?, ¿Por qué es importante?, ¿Cuáles son los pasos a seguir para desarrollar un software?, ¿Cuál es el producto obtenido? y ¿Cómo puede estar seguro que lo ha hecho correctamente?

Sistema de educación a distancia UDB: http://moodle.udb.edu.sv/ead/

Ante la relevancia que ha tomado el software y el nivel de dependencia de la sociedad surge la necesidad de comprender todos los aspectos de la producción del software desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de éste. Por otra parte, "en la sociedad moderna el papel de la ingeniería es proporcionar sistemas y productos que mejoren los aspectos materiales de la vida humana, para que así la vida sea más fácil, segura y placentera" (Richard Fairley y Mary Willshire).

1.2.1. Evolución de la ingeniería de sistemas

En la actualidad, el software tiene un papel dual. Es, a la vez, un producto y un vehículo mediante el cual se entrega un producto.

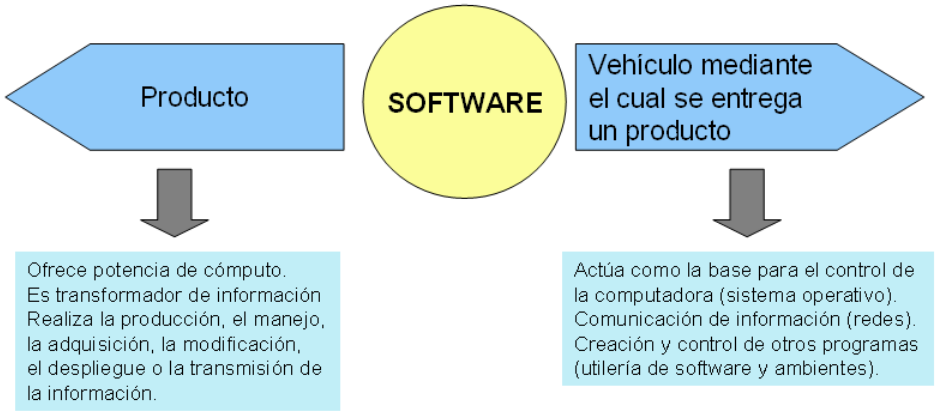

Figura 4. Dualidad del software.

La ingeniería de software y la de sistemas nacieron como disciplinas hacia finales de los años 60, cuando se acuñó el término software engineering para referirse a los procedimientos de diseño y construcción de programas. Desde entonces, las técnicas para desarrollar programas y sistemas han venido evolucionando, guiadas por diferentes propuestas metodológicas: estructuradas, de orientación a objetos, entre otras.

Uno de los aportes cruciales en los inicios de la ingeniería de sistemas lo constituye el trabajo de Edsger Wybe Dijkstra presentado en su célebre artículo Go To Statement Considered Harmful (Las sentencias Go To son consideradas dañinas), publicado en la revista Communications of the ACM, Vol. 11 (1968). En dicho artículo, Dijkstra demostraba que todo programa puede escribirse sin utilizar la instrucción Go To y utilizando únicamente las tres instrucciones de control siguientes:

- El bloque de instrucciones consecutivas, que se ejecutan una después de otra.
- La instrucción condicional conocida normalmente como estructura IF-THEN-ELSE.
- El lazo o loop condicional "WHILE condición, DO instrucción", que ejecuta la instrucción repetidamente mientras la condición se cumpla. En su lugar, se puede utilizar también la forma "UNTIL condición DO instrucción", que ejecuta la instrucción hasta que la condición se cumpla. Los dos loops se diferencian entre sí porque en la forma WHILE la condición se comprueba al principio, mientras que en la forma UNTIL la condición se comprueba al final del loop.

Los programas que utilizan sólo estas tres estructuras básicas o sus variantes (como la instrucción condicional CASE), y no utilizan la instrucción GOTO, se llaman estructurados.

A partir del trabajo de Dijkstra, la "programación estructurada" (llamada también "programación sin GOTO") se convirtió en la forma de programar más extendida y el término "estructurado" pasó a ser una especie de sello de calidad, a tal punto que las diferentes propuestas metodológicas que fueron apareciendo en el mercado, añadían a su nombre el término estructurado, como lo hicieron Edward Yourdon, Tom De Marco y Ganes & Sarson.

Durante los años 80 fue ganando terreno la idea de que la programación y el desarrollo de sistemas debían dejar de ser una "artesanía" para convertirse en una verdadera rama de la ingeniería y, junto con estas ideas, las diferentes metodologías fueron ganando aceptación universal.

A principios de los años 90 las dos metodologías de mayor importancia fueron la Ingeniería de la Información[3], desarrollada por James Martin, y la metodología IBM/AD Cycle, creada por la empresa IBM.

[3] Conocida como "Information Engineering" que es un "grupo de técnicas formales y automatizadas utilizadas para desarrollar modelos de la organización, de los datos y los procesos, las cuales tienen el propósito de crear una base de conocimiento integrada que se utilizará para producir y darle mantenimiento a los sistemas de información" (http://graduado.sagrado.edu/gsi711/cap2tesis_jvega.pdf).

Ambas incluían consideraciones acerca del uso de herramientas CASE (computer aidded software engeeniering) para el desarrollo de sistemas y pasaron a ser el estándar de referencia para los productores de tales herramientas.

La "revolución de los microcomputadores" y las aplicaciones "tipo Windows", permitieron que la década de los 90 nos trajera nuevas maneras de desarrollar aplicaciones -como el enfoque cliente/servidor-, con nuevos lenguajes de programación –Power Builder, Oracle Forms, Visual Basic, C++, Java, entre otros.- y nuevos enfoques para las tareas de desarrollo de software, dando lugar a la aparición de la programación orientada a objetos (OOP - Object Oriented Programming).

La nueva forma de concebir el software incorpora técnicas y metodologías diferentes a las formas utilizadas tradicionalmente para desarrollar sistemas; técnicas que se fundamentan en el denominado paradigma de la POO, el cual, en líneas generales, concibe el universo computacional como un universo poblado por objetos, cada uno responsable de sí mismo y comunicándose con los demás por medio de mensajes.

Al igual que antaño ocurriera con la programación estructurada, a medida que la programación orientada a objetos fue ganando terreno, la industria comenzó a utilizar sus conceptos en otras actividades, más allá de la programación. Ya en 1996, James Martin y James J. Odell, en su obra Object Oriented Methods - Pragmatic Considerations, señalaban que «las nociones empleadas por los lenguajes de programación OO pueden ser aplicadas como una filosofía general para todo el proceso de desarrollo de sistemas».

En el libro citado, sus autores también señalaban que: "Principalmente, esta interpretación más amplia significa que OO es una forma de organizar nuestras ideas acerca del mundo. Esta organización se basa en los tipos de cosas - o tipos de objetos - en nuestro mundo. De esta forma, podemos definir los atributos de estos tipos de objetos, las operaciones que se ejecutan sobre estos tipos de objetos, las reglas basadas en ellos, … En lugar de una unidad física que contiene variables y métodos, un enfoque OO más general nos brinda una forma de organizar conceptualmente nuestro conocimiento. "En el tiempo han ido apareciendo diferentes metodologías OO, muchas de ellas pueden ser catalogadas como orientadas a objetos, porque se basan en el paradigma de orientación a objetos. Entre ellas, las de mayor difusión han sido:

- Object-Oriented Design (OOD), Grady Booch.
- Object Modeling Technique (OMT), James Rumbaugh.
- Object Oriented Analysis (OOA), Coad/Yourdon.
- Hierarchical Object Oriented Design (HOOD), ESA.
- Object Oriented Structured Design (OOSD), Wasserman.
- Object Oriented Systems Analysis (OOSA), Shaler y Mellor.

En la actualidad, las metodologías más importantes para el análisis y diseño de sistemas han confluido en lo que se ha ido convirtiendo en un importante lenguaje estándar para el modelaje de sistemas, denominado UML (Unified Modeling Language – Lenguaje Unificado de Modelaje[4]), bajo el respaldo de la organización OMG (Object Management Group o Grupo de Administración de Objetos).

UML es un lenguaje de modelaje que permite especificar y documentar un sistema y sus componentes. Las herramientas de modelaje que componen UML permiten poner en "blanco y negro" la información sobre la estructura (elementos estáticos) y el comportamiento (elementos dinámicos) de un sistema.

UML no es ni una metodología, ni un lenguaje de programación; existen, sin embargo, diferentes metodologías, como RUP (Rational Unified Process), y sistemas CASE, como Rational y Magic Draw,

[4] UML es, fundamentalmente, la creación de Grady Booch, James Rumbaugh e Ivar Jacobson, conocidos en el medio como "los tres amigos", quienes habiendo desarrollado metodologías para desarrollo de software con la orientación a objetos, tuvieron la oportunidad de reunir esfuerzos y, como consecuencia de la aceptación general que encontró su propuesta, se pudo crear el Object Management Group, como una organización sin fines de lucro que reúne a la gran mayoría de las empresas líderes en el mercado de la informática: IBM, HP, Sun Microsystems, Microsoft, Intellicorp, Oracle, Texas Instruments, Rational, entre otras.

que ofrecen generadores de código a partir de una especificación en UML, para una gran variedad de lenguajes de programación, como Java y C++.

UML es un lenguaje de modelaje visual, de propósito general, para el desarrollo de software orientado a objetos, que permite representar un sistema y sus componentes con la suficiente precisión como para desarrollar esos componentes -generar el código- y, a la vez, con la suficiente "independencia tecnológica" como para permitir que el desarrollador utilice las herramientas de construcción que sean de su preferencia.

El gran valor que tiene UML radica en la estandarización que establece, lo cual facilita su uso universal, así como el desarrollo de productos de software con la capacidad de generar código a partir de una especificación UML, para las más diversas plataformas. De hecho, la mayoría de los productores de sistemas CASE han ido incorporando UML en su conjunto de herramientas.

La OMG, adicionalmente, ha anunciado su Arquitectura Centrada en Modelos (MDA - Model Driven Architecture), señalando que MDA es el siguiente paso en la resolución de problemas de estandarización e integración a través de especificaciones abiertas e independientes de los proveedores. MDA busca lograr la integración y la interoperatividad de todos los componentes, a través del ciclo de vida de los sistemas, desde el modelaje del negocio y las tareas de diseño, hasta la construcción, integración, prueba, implantación y mantenimiento.

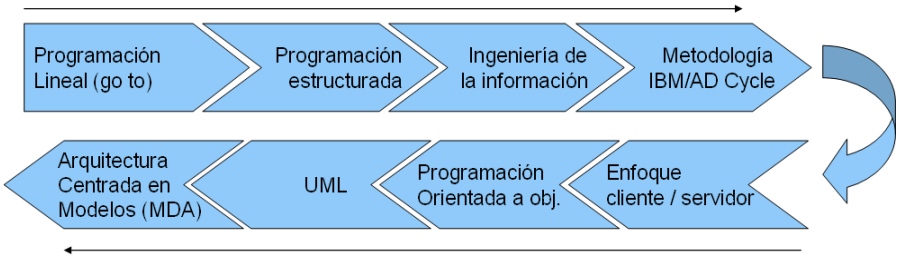

Figura 5. Evolución de la ingeniería de sistemas.

1.3. Áreas de aplicación de la ingeniería de software

La Ingeniería de Software tiene múltiples campos de aplicación. Se citan algunos en la figura 14.

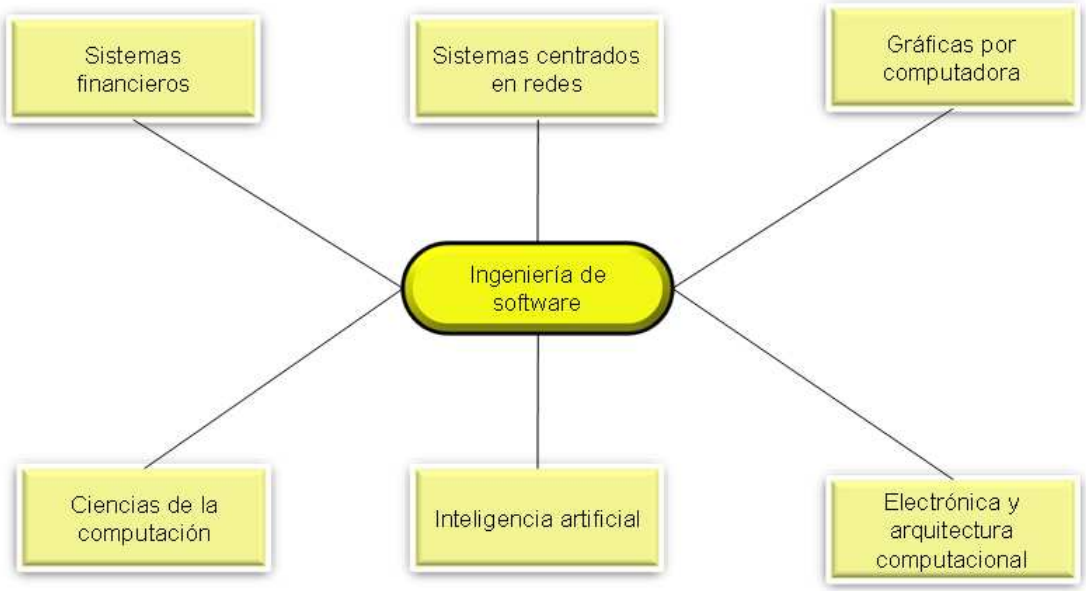

Figura 14. Áreas de aplicación de la ingeniería de software.

1.4. Importancia del software

El software es imprescindible para cualquier sistema informático o basado en informática, puesto que sin él, este no funcionaría.

Una computadora sin software sería simplemente un conjunto de chips, cables, periféricos e interruptores totalmente inerte y sin función alguna. Es el software quien ordena todo ese material, lo reconoce, le asigna una función según sus características, y permite que funcione todo en su conjunto.

La importancia del software[5] radica también en que permite una comunicación entre el usuario y la máquina, e incluso una interacción entre ambos.

1.5. Problemas del software

La Ingeniería de Software se enfrenta con un dilema durante el proceso de desarrollo de software: el software requiere de mantenimiento una vez concluido, pero cómo sabemos que se ha finalizado un software.

Por otra parte el proceso de creación del software implica hacer un levantamiento de requerimientos, lo cual puede ser una función compleja si los usuarios no tienen claridad de qué es lo que quieren sistematizar y automatizar. En este caso habrá que determinar cuánto es la información "suficiente" para emprender el diseño y desarrollo posterior.

Otro dilema con que se enfrentan los Ingenieros de Software es la cuantificación del tiempo requerido para el desarrollo de todos los procesos de creación del software.

La definición del costeo del software es otro problema en tanto no se hagan todas las consideraciones pertinentes al momento de fijar costos. En este contexto no se debe únicamente pensar en el costo de desarrollo, sino también en las capacidades individuales del personal asignado al proyecto y su familiaridad con el área de aplicación; la complejidad y tamaño del producto; el tiempo asignado; el nivel de confiabilidad y el nivel tecnológico utilizado.

[5] Citado de internet, en el URL http://www.importancia.org/software.php, el 07 de julio de 2013.

FUENTES DE CONSULTA

BIBLIOGRAFIA

BRUGGE, BERND (2001). Ingeniería de software orientada a objetos, Editorial Pearson Prentice Hall.

BOOCH, GRADY; RUMBAUGH, JAMES; JACOMSON, IVAR (2007). El lenguaje unificado de modelado, Manual de referencia. Segunda edición, Editorial Pearson Addison Wesley, España.

IEEE (2004). Guide to the Software Engineering Body of Knowledge, 2004 version. A project of the IEEE Computer Society and Professional Practices Committee.

LAWRENCE PFLEEGER, SHARI (2002). Ingeniería de software – Teoría y práctica, primera edición, Editorial Pearson Prentice Hall.

MURCH, R. (2000) Project Management: Best Practices for IT Professionals, Prentice-Hall, Columbus.
PRESSMAN, ROGER S. (2006). Ingeniería de software, un enfoque práctico, sexta edición. Editorial McGrawHill, México (Libro de texto).

SÁNCHEZ, SALVADOR; SICILIA, MIGUEL ÁNGEL; RODRÍGUEZ, DANIEL (2012). Ingeniería del Software. Un enfoque desde la guía SWEBOK, Primera Edición, Editorial Alfaomega.

SOMMERVILLE, IAN (2005). Ingeniería de software, sexta edición, Editorial Pearson Addison Wesley (Libro auxiliar).

SCHACH, STEPHEN R. (2002). Ingeniería de software clásica y orientada a objetos, sexta edición. Editorial McGrawHill, México.

INTERNET

http://graduado.sagrado.edu/gsi711/cap2tesis_jvega.pdf, Análisis y Diseño de Sistemas de Información Computarizados, Prof. Carmen R. Cintrón.