SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

UnADM
Universidad Abierta y a Distancia de México

Desarrollo de Software
3er semestre

Programa de la asignatura:
Análisis y diseño orientado a objetos

Unidad 1. Introducción al análisis orientado a objetos

Clave:

Ingeniería: 15142313

TSU: 16142313

Ciudad de México, noviembre del 2016

Universidad Abierta y a Distancia de México

DCEIT

Índice

Unidad 1. Introducción al análisis orientado a objetos

Presentación de la unidad

Bienvenido(a) a la asignatura Análisis y diseño orientado a objetos. En esta primera unidad se expondrán los principios fundamentales de un buen análisis para hacer un correcto diseño y así poder programar con orientación a objetos, esto permitirá dar cumplimiento al propósito de la unidad.

En primer lugar se expondrán los conceptos básicos de análisis y diseño, como la definición, las características y las ventajas de hacer un análisis previo de la información, en la segunda parte de esta unidad, se abordarán los conceptos de orientación a objetos para que cuando hagas tu diseño sepas darle ese enfoque; también distinguirás las características de este tipo de programación. Finalmente, conocerás el ciclo de vida del software y algunos tipos de ciclos. Toda esta información es el principio básico de un buen análisis de diseño orientado a objetos.

Propósito

Al término de esta unidad lograrás:

- Conocer los atributos de los modelos orientados a objetos para poder establecer las necesidades del diseño de un sistema con estas características, así como ubicar las etapas de vida del software.

COMPETENCIA ESPECÍFICA

- Identificar las etapas de un sistema orientado a objetos para decidir su ciclo de vida, utilizando los conceptos de orientación a objetos.

1.1. GENERALIDADES

El objetivo principal del análisis orientado a objetos (AOO) es desarrollar un software capaz de cubrir las necesidades esenciales del usuario final, y que su diseño se enfoque en los objetos.

El análisis orientado a objetos y su diseño se basan en definir una serie de actividades relevantes al problema que se va a resolver, en donde son comúnmente utilizados las operaciones y atributos asociados. Para cumplir con esto se deben tener en cuenta las siguientes tareas:

1. Debe existir una comunicación sobre los requisitos básicos del usuario ya que será el usuario final del software.
2. Se deben definir los métodos a utilizar para el análisis.
3. Se debe definir la jerarquía de los métodos utilizados para el análisis.
4. Deben existir relaciones de objeto a objeto, así como todas sus conexiones.
5. Debe modelarse el comportamiento del objeto.

Las actividades anteriores se aplican de forma iterativa hasta que el modelo esté completo.

El software orientado a objetos es más fácil de mantener debido a que su estructura es inherentemente poco acoplada. Además, los sistemas orientados a objetos son más fáciles de adaptar y escalables.

El enfoque realizado sobre el desarrollo orientado a objetos no debe implicar hacer las tareas diferentes del enfoque clásico de desarrollo de software porque se desarrollan actividades similares en un proceso evolutivo.

El modelo orientado a objetos tiene como característica el hecho de que un elemento del mundo real se puede representar a través de sus características y de sus comportamientos. Los conceptos como clase, objeto, instancia, atributos y métodos, se hacen cotidianos en el AOO, ya que son parte de su vocabulario. Los conceptos fundamentales que llevan a un diseño de alta calidad son igualmente aplicables a sistemas desarrollados usando métodos orientados a objetos. Por esa razón, un AOO debe exhibir abstracciones de datos y procedimientos que conducen a una modularidad eficaz.

La gestión de un proyecto de software orientado a objetos por lo general implica actividades como:

1. Establecer un proceso común para el proyecto.
2. Usar métricas para desarrollar estimaciones de tiempo y esfuerzo.
3. Especificar productos de trabajo e hitos que permitirán medir el progreso.
4. Tener puntos de comprobación para la gestión de la calidad y control.
5. Controlar cambios que se generan durante el progreso del proyecto.
6. Realizar el seguimiento y control del progreso.

El AOO se basa en un conjunto de principios básicos comúnmente usados:

1. Modelado de la información.
2. Descripción de funciones.
3. Representación del comportamiento del modelo.
4. Desarrollo de modelos de datos, funcional y de comportamiento.

El análisis y desarrollo orientado a objetos puede ser interpretado como el conjunto de disciplinas que desarrollan y modelan un software que facilita la construcción de sistemas de información complejos a partir de la formación de sus componentes. Las técnicas orientadas a objetos proporcionan mejoras y metodologías para construir sistemas de software complejos a partir de unidades de software, el enfoque del AOO debe ser capaz de manipular sistemas grandes y pequeños, que sean flexibles con facilidad de mantenimiento y capaces de evolucionar respecto a las necesidades y requerimientos del usuario final.

1.1.1. DEFINICIÓN

Para comenzar a entender en qué consiste el análisis y diseño de software orientado a objetos es importante mencionar algunos conceptos de la programación orientada a objetos que forman parte del modelado del análisis y diseño.

Objetos	Atributos
Instancia de una clase específica. Los objetos heredan los atributos y operaciones de una clase.	Atributos: "Una colección de valores de los datos que describen una clase" (p. 202). Son los datos a los que se refiere el estado del objeto (Booch-Grady, 1996).
Clase	
Representa a una entidad que tiene un estado interno y un comportamiento característico.	

Los objetos tienen un estado interno y un comportamiento, el estado de un determinado objeto es un conjunto de parámetros con valores que lo caracterizan. Estos parámetros son atributos y representan lo mismo que los campos de una tabla de una base de datos o las variables de un programa estructurado, por ejemplo:

- La edad de una persona
- El nombre de un animal
- La altura de un edificio
- La jornada de un trabajo

El comportamiento de un objeto se forma por el conjunto de operaciones que se pueden realizar. Cada una de esas operaciones recibe el nombre de método. Los métodos podrían ser:

- Dibujar un triángulo
- Asignar laboratorio de trabajo a un grupo
- Prestar un libro de la biblioteca

Cada objeto de una clase tiene sus propios atributos, que describen y caracterizan el estado en cada momento, y un conjunto de métodos sobre los cuales ejecuta las operaciones que manifiesta su comportamiento. El análisis orientado a objetos (AOO) construye un modelo orientado a objetos y a las clases que se basan en la comprensión de los conceptos orientado a objetos (OO).

Rompecabezas

El enfoque orientado a objetos permite que los objetos estén auto-contenidos. Los objetos existen por sí mismos, con una funcionalidad para invocar comportamientos de otros objetos. Por definición son modulares, es decir, pequeñas unidades lógicas de códigos independientes entre sí que se

comunican entre ellas mediante mensajes en su construcción. Esto quiere decir que son entidades completas y, por lo tanto, tienden a ser altamente reutilizables. Las aplicaciones orientadas a objetos se constituyen sobre el paradigma de los mensajes a otros objetos.

Por lo general la mayoría de los programadores desarrolla sobre un lenguaje y sólo utiliza su propio estilo de programación. Desarrollan sobre un paradigma forzado por el lenguaje que utilizan, tienden a no enfrentarse a métodos alternativos de resolución de un problema y como resultado se presentan con la dificultad de ver las ventajas de elegir un estilo más apropiado para el problema a manejar. Autores como Bobrow y Stefik (1986), citados en Booch-Grady (1996) sugieren que existen cuatro clases de estilo de programación:

- Orientada a procedimientos (algoritmos).
- Orientada a objetos (clases y objetos).
- Orientada a la lógica (expresado en cálculo de predicados).
- Orientada a reglas (reglas if-Then).

Análisis y diseño orientado a objetos (ADOO) es un enfoque de la ingeniería de software que modela un sistema como un grupo de objetos que interactúan entre sí. Este enfoque representa un dominio en términos de conceptos compuestos por verbos y sustantivos, clasificados de acuerdo con su dependencia funcional.

En este método de análisis y diseño se crea un conjunto de modelos utilizando una notación acordada, por ejemplo, el lenguaje unificado de modelado (UML). ADOO aplica técnicas de modelado de objetos para analizar los requerimientos en un contexto (un sistema de negocio, un conjunto de módulos de software) y para diseñar una solución para mejorar los procesos involucrados. Este método no está restringido al diseño de programas de computadora, sino que cubre sistemas

enteros de distinto tipo. Las metodologías de análisis y diseño más modernas son casos de uso guiados a través de requerimientos, diseño, implementación, pruebas, y despliegue.

Cubo UML. El lenguaje unificado de modelado se ha vuelto el lenguaje de modelado estándar usado en análisis y diseño orientado a objetos.

Las estrategias que se utilizan para hacer un correcto desarrollo orientado a objetos son: análisis, diseño y programación. En el análisis se consideran las actividades que hay que hacer para desarrollar un programa OO y se identifican los objetos, transformándolos en entidades y operaciones para asociarlos con el problema a resolver. En el diseño, los objetos se relacionan con la solución del problema. Finalmente, en la programación se hace la codificación del problema en algún lenguaje con orientación a objetos.

Para reforzar el conocimiento sobre el concepto del AOO, revisa el documento *U1. Concepto*.

1.1.2. CARACTERÍSTICAS

El modelo AOO se basa en el concepto de objeto. Un objeto tiene estado, comportamiento e identidad. La estructura y el comportamiento de los objetos son similares y están definidos en su clase común. El modelo AOO se basa en el concepto de objeto. Un objeto tiene estado, comportamiento e identidad. La estructura y el comportamiento de los objetos son similares y están definidos en su clase común. De tal modo, los sistemas deben estar funcionando siempre de manera correcta, su complejidad a veces es tan grande que el mismo ser humano o su capacidad intelectual se ve excedida, por lo que resulta imposible comprenderlo en su totalidad

por una sola persona. Así, el software es complejo de forma innata, esto es una característica esencial de él. Es complejo porque hereda la complejidad del problema, la dificultad de gestionar el proceso de desarrollo, la flexibilidad que se puede alcanzar a través del software y los problemas que plantea:

1. **El problema** presenta tantos requisitos que compiten entre sí o se contradicen, y esas contradicciones existen porque los usuarios no saben expresar sus necesidades de forma clara para que las otras personas que participan en el proyecto lo puedan entender.
2. **Los requisitos** son cambiados con frecuencia durante el desarrollo, incluso porque la mera existencia de un proyecto de solución altera al sistema real.
3. **Un sistema grande** debido a la inversión financiera que implica, no puede desecharse y reemplazarse por uno nuevo cada vez que los requisitos cambian, debe evolucionar considerando los siguientes puntos:
    - Evolución del software: responder al cambio de requerimientos.
    - Mantenimiento del software: corregir errores.
    - Conservación del software: emplear recursos para mantener en operación un elemento de software anticuado y decadente.
4. **La principal tarea del grupo de desarrollo** es dar una ilusión de simplicidad para los usuarios de esta complejidad arbitraria del problema, se hace lo posible por escribir menos código pero a veces es imposible y más en un sistema tan grande, por lo que se debe recurrir a la aplicación de varias técnicas de re-utilización de código existente.
5. **Debe también** enfrentarse la existencia de miles de módulos separados y esto implica un grupo de desarrolladores, nunca una única persona.
6. **Un proyecto de software** es muy frecuentemente apoyado en pilares construidos por los mismos desarrolladores, por lo que el desarrollo del proyecto de software sigue siendo una tarea muy laboriosa.

7. En algunos sistemas una pequeña modificación en la entrada provoca una pequeña modificación en la salida. Mientras que en otros, y sobre todo de gran tamaño, se percibe una explosión combinatoria que hace que la salida se modifique enormemente.
8. **Se intenta diseñar los sistemas** con una separación de intereses de forma que el comportamiento de una parte del sistema tenga el mínimo impacto en el comportamiento de otra parte del sistema.
9. **En un sistema de gran volumen** uno debe contentarse con un grado de confianza determinado a lo que refiere su corrección, ya que no puede llevarse a cabo una prueba a fondo exhaustiva por no tener las herramientas matemáticas ni intelectuales para un sistema no continuo.
10. **Las consecuencias de la complejidad ilimitada,** mientras más complejo es el sistema, más abierto está al derrumbamiento total.
11. **Crisis del software,** ha existido tanto tiempo que debe considerarse normal. Es cuando se pretende dominar la complejidad del sistema a un extremo que lleva al presupuesto a niveles excedentes y que transforman en deficiente al sistema respecto a los requerimientos fijados.

1.1.3. VENTAJAS

1. Los beneficios del enfoque OO, de acuerdo con Booch-Grady (1996), son:

- **Primero**, el uso del modelo OO ayuda a explotar el poder expresivo de todos los lenguajes de programación basados en objetos y los orientados a objetos, como Smalltalk, Object Pascal, C++, CLOS, Ada y recientemente Java.
- **Segundo**, el uso del modelo OO alienta el re-uso no sólo del software, sino de diseños completos.

- **Tercero**, produce sistemas que están construidos en formas intermedias estables y por ello son más resistentes al cambio en especificaciones y tecnología.

Se considera que el principal beneficio del AOO, es que da un esquema para formalizar el modelo real.

2. El análisis orientado a objetos (AOO) es un método de análisis que examina los requerimientos desde la perspectiva de las clases y objetos encontrados en el vocabulario del dominio del problema. El diseño orientado a objetos es un método de diseño que abarca el proceso de descomposición orientado a objetos y una notación para representar ambos modelos (lógico y físico), como los modelos estáticos y dinámicos del sistema bajo diseño.

3. Dentro de las metodologías del análisis y diseño orientado a objetos hay una variedad de métodos en la actualidad. Muchos de los métodos pueden ser clasificados como orientados a objetos porque soportan de manera central los conceptos de la orientación a objetos. Algunas de las metodologías más conocidas y estudiadas hasta antes de UML, de acuerdo con Jacobson (1996), citado en Booch-Grady (1996), son:

Metodología	Autor
Diseño orientado a objetos (DOO)	Grady Booch
Técnica de modelado de objetos (TMO)	Rumbaugh
Análisis orientado a objetos (AOO)	Coad/Yourdon
Jerarquía de diseño orientada a objetos (JDOO)	ESA
Diseño estructurado orientado a objetos (DEOO)	Wasserman
Análisis de sistemas orientado a objetos (ASOO)	Shaler y Mellor, entre otros

4. Actualmente las metodologías más importantes de análisis y diseño de sistemas orientados a objetos se han basado en lo que es el UML, bajo el respaldo del grupo administrador de objetos.

El modelo de objetos ha demostrado ser aplicable a una amplia variedad de dominios de problema. En la actualidad, el ADOO puede que sea el único método que logre emplearse para atacar la complejidad innata de muchos sistemas grandes. Sin embargo, puede no ser aconsejable en dominios, no por razones técnicas, sino por cuestiones como la falta de personal con entrenamiento adecuado o buenos entornos de desarrollo.

Lo interesante de la programación orientada a objetos (POO) es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible.

1.2. IDENTIFICACIÓN Y CONCEPTOS BÁSICOS DE MODELOS ORIENTADOS A OBJETOS

1. El análisis orientado a objetos es una forma de hacer frente a la comprensión y solución de problemas, usando modelos a partir de conceptos. La pieza fundamental es el objeto, el cual se combina en una sola entidad.

Para dar énfasis sobre los conceptos en el análisis orientado a objetos, a continuación se detallan los utilizados con mayor frecuencia.

2. Objeto. Es una entidad bien definida, real o abstracta, que tiene sentido sobre el contexto de alguna aplicación y un papel bien definido. A su vez se pueden diferenciar dos tipos:

- Concretos: ejemplo de objetos concreto, una unidad de almacenamiento, un archivo de computadora, un automóvil, un profesor.
- Conceptuales: ejemplo de objetos conceptuales, un programa de computadora, una variable de programación, un pensamiento.

3. Atributo. Es un valor atribuido a un objeto, por ejemplo un alumno es un objeto y sus atributos podrían ser su número de control, su nombre y su calificación. Se entiende que se pueden llegar a tener más atributos, pero si el contexto de la aplicación es sólo obtener un resultado específico, los atributos serán los únicos que son relevantes para esa aplicación.

4. Comportamiento. Es el conjunto de cada acción o transformación que el objeto ejecuta, también podría ser llamado operación y método. Por ejemplo, para el objeto alumno se requiere de algunas acciones y transformaciones: asignar calificación, leer nombre y leer número de control; estas acciones representan el comportamiento del objeto alumno. En el caso de asignar calificación, leer nombre y leer número de control, se refieren a transformaciones, ya que modificarán el valor de la calificación final.

5. Identificación. Cada objeto posee una identificación mediante la cual se puede hacer alusión a él de modo exclusivo.

6. Clase. Describe propiedades importantes para una aplicación y que ignora el resto. La selección de clases es arbitraria y depende de la aplicación.

7. Instancia. Se considera que cada objeto es una instancia de su clase. Toda clase describe un conjunto posiblemente finito de objetos individuales.

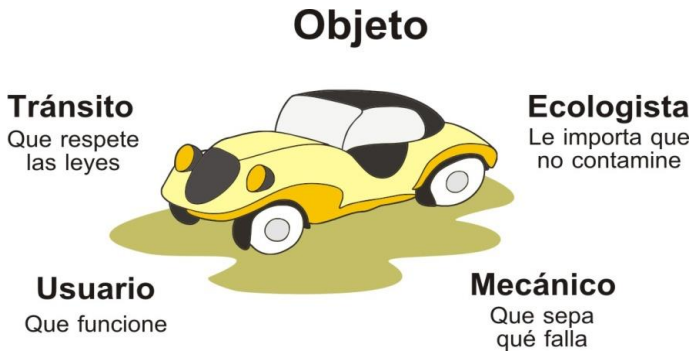
8. Identidad. Se refiere a que cada objeto conserva de manera inherente su propia identidad. O sea, dos objetos son distintos aún si el valor de todos sus atributos es idéntico. Por ejemplo, los ocho peones negros de un juego de ajedrez, son todos negros, tienen las mismas dimensiones, textura, pero todos son diferentes, existen y tienen su propia identidad. Dos gotas de agua es otro ejemplo de la característica de identidad de los objetos.

9. Clasificación. Se refiere a que los objetos con los mismos atributos y comportamiento, métodos, son agrupados en clases. Cada objeto perteneciente a una clase, se dice que es una *instancia* de la clase. Así que una clase, representa a un posible conjunto infinito de objetos individuales. Por ejemplo, a todos los alumnos que aparecerán en la lista de calificaciones finales, se clasificaron en la clase alumno. A todos los amigos que se registran en una agenda, se les puede clasificar en la clase persona. (Kendall, 2005 y Booch-Grady, 1996).

1.2.1. Abstracción

En la abstracción la mente humana modela la realidad en forma de objetos. Para ello busca semejanzas entre la realidad y la posible implementación de *objetos del programa* que simulen el funcionamiento de los *objetos reales*.

Los humanos entienden la realidad como objetos ya definidos y no como un conjunto de situaciones menores que se unen para dar forma a algo. No es necesario conocer detalles del cómo, cuándo, dónde o por qué las cosas, solamente se necesita saber que cuando se quiere caminar se hace y punto.
Es la caracterización de un objeto de acuerdo con las propiedades que interesen en un instante de tiempo. Las características escogidas son relativas a la perspectiva del observador.

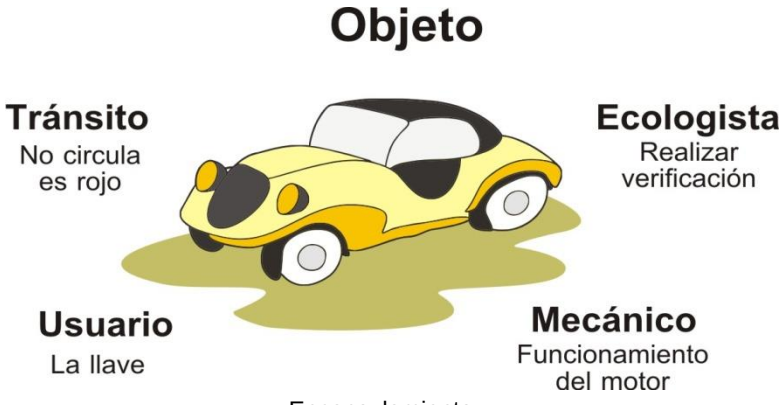


Abstracción

1.2.2. ENCAPSULAMIENTO

Cuando un objeto es encapsulado se tiene la libertad de saber qué información se hace pública o no, para ello se les puede hacer privados e inaccesibles los datos de este objeto a través de otro previamente publicado. Con esto se logra que los datos sólo sean utilizados por su interfaz dejando de lado cómo está implementada, haciendo así, más fácil la utilización del mismo.

Así, la manera de ocultar los detalles de la representación interna de un objeto es presentando sólo la interface para el usuario.



Encapsulamiento

1.2.3. MODULARIDAD

A través de la modularidad, se propone al programador dividir su aplicación en varios módulos diferentes (ya sea en forma de clases, paquetes o bibliotecas), cada uno de ellos con un sentido propio.

Esta fragmentación disminuye el grado de dificultad del problema al que da respuesta el programa, pues se afronta éste como un conjunto de problemas de menor dificultad, además de facilitar la comprensión del programa.

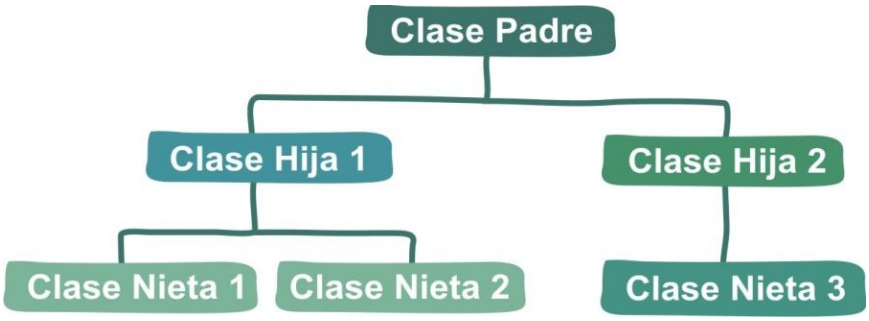
1.2.4. HERENCIA

La herencia se basa en la capacidad para reflejar la abstracción que realiza automáticamente y se refiere a compartir atributos y métodos entre objetos que se relacionan de manera jerárquica durante un proceso de análisis de información.

Se percibe en la realidad como un agregado de objetos relacionados. Estas interrelaciones pueden verse como un conjunto de generalizaciones que se asimilan con el tiempo. De esta manera, la herencia es el mecanismo fundamental de relación entre clases en la orientación a objetos.

Modularidad

Del mismo modo, las distintas clases de un programa se organizan mediante la **jerarquía**. La representación de dicha organización da lugar a los denominados *árboles de herencia*:



Ejemplo de árbol de herencia

La capacidad de descomponer un problema o concepto en un conjunto de objetos relacionados entre sí, y cuyo comportamiento es fácilmente identificable, puede ser muy útil para el desarrollo de programas informáticos. Del mismo modo, relacionar las clases de manera jerárquica; una clase padre o superclase sobre otras clases hijas o subclases.

1.2.5. POLIMORFISMO

> Mediante el denominado **paso de mensajes**, un objeto puede solicitar de otro objeto que realice una acción determinada o que modifique su estado. El paso de mensajes se suele implementar como llamadas a los métodos de otros objetos.
>
> Desde el punto de vista de la programación estructurada, esto correspondería con la llamada a funciones (Del Río San José, 2010, p.30).

Ahora bien, el polimorfismo es una característica de la orientación a objetos, esto significa que un mismo método puede tener diferente manera de realizarse, en las diferentes clases que haya bajo estudio. Cada objeto perteneciente a una clase y "sabe cómo" ejecutar sus propios métodos. Cuando se programa orientado a objetos, el lenguaje de programación automáticamente selecciona el método correcto para efectuar una cierta acción o transformación sobre el objeto al que se

aplica. Por ejemplo, si se cuenta con los objetos bicicleta, carro, barco y se les aplica la operación mover, la acción se ejecuta de manera diferente para cada objeto. Otro ejemplo típico es el de los objetos de la clase figura, por ejemplo, círculo, rectángulo, triángulo, y se le aplica la acción dibujar, cada uno de ellos tendrá su propio método de dibujar definido, ya que la acción debe implementarse de manera diferente para cada objeto.

1.3. CICLO DE VIDA DEL SOFTWARE Y TIPOS DE CICLOS

La metodología para el desarrollo de software tiene pasos establecidos en la realización y administración de un proyecto para llevarlo a cabo con éxito. Para facilitar esto, se debe dividir un gran proyecto en módulos más pequeños llamados etapas. Las acciones que corresponden en cada una de ellas ayudan a definir entradas y salidas para cada una de las etapas, y sobre todo normalizan el modo como se administrará el proyecto.

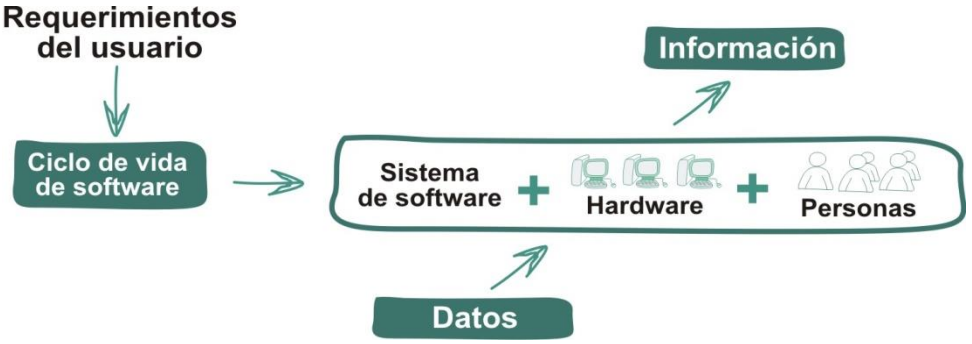
Pasos del software

1.3.1. DEFINICIÓN

Llevar a cabo la metodología para el desarrollo del software requiere seguir puntualmente una serie de pasos o procesos para analizar, diseñar y realizar un producto software, desde que surge la necesidad hasta que se cumple el objetivo por el cual fue creado. Desde un punto de vista puede considerarse que el ciclo de vida de un software tiene capas claramente diferenciadas:

- **Planificación**: planteamiento detallado que los pasos a seguir durante el desarrollo del proyecto, considerando aspectos de tiempo y dinero.

- **Implementación**: decidir las actividades que componen la realización del producto.
- **Producción**: el proyecto es presentado al cliente o usuario final, sabiendo que funciona correctamente y responde a los requerimientos solicitados en su momento.



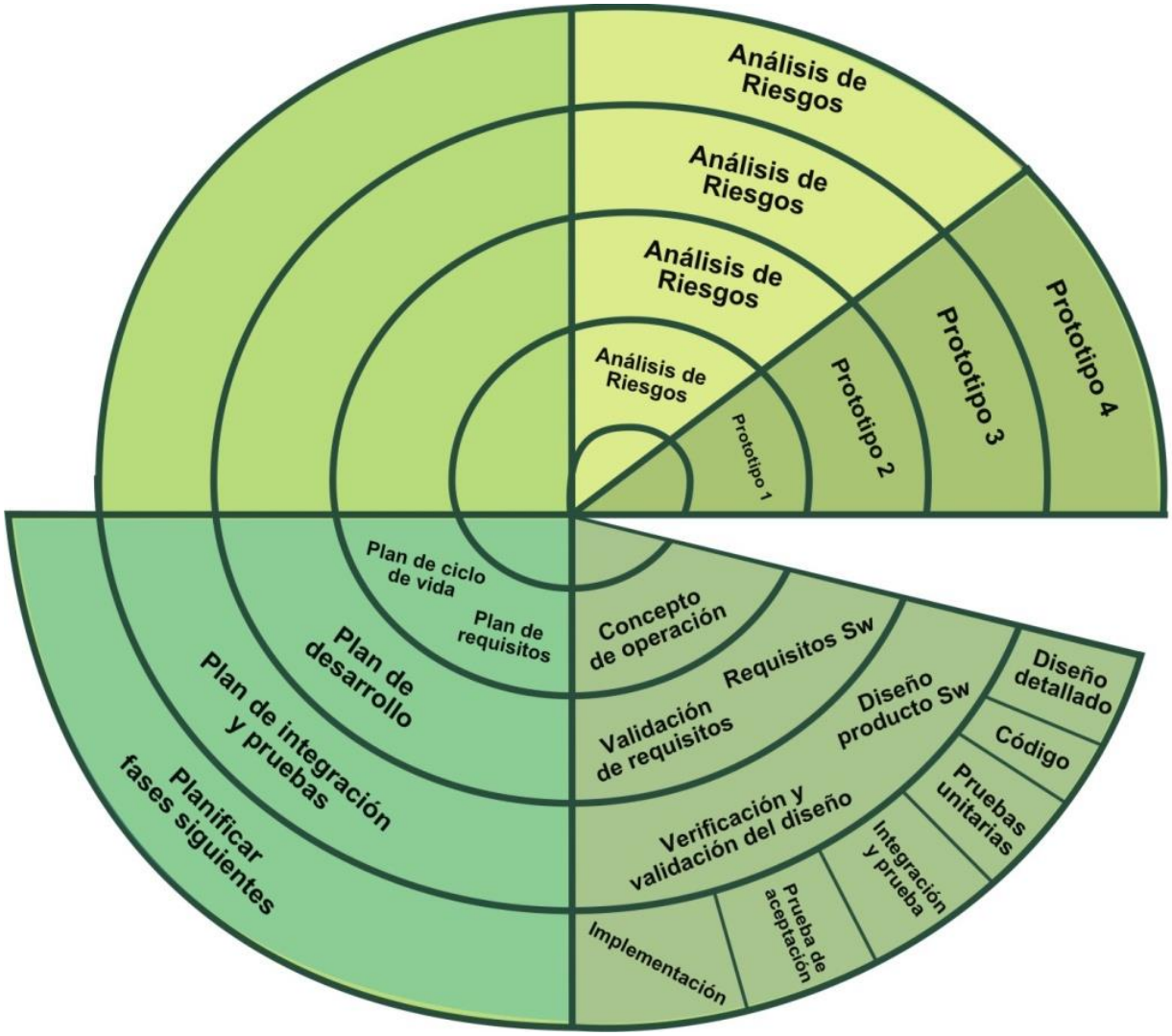
Ciclo de vida del software

1.3.2. ESPIRAL

Este ciclo de vida puede considerarse una variación del modelo prototípico que fue diseñado por Boehm (1988), citado en Kendall (2005). El modelo se basa en una serie de ciclos repetitivos para ir ganando madurez en el producto final. Conforme se va desarrollando el sistema se hace un primer prototipo se presenta al cliente y sobre éste se hacen adecuaciones y nuevos prototipos, de esta manera se tiene un avance en espiral hasta llegar a la perfección de todas las funcionalidades o módulos. En este modelo hay cuatro actividades principales para las etapas:

- **Planificación**: relevamiento de requerimientos iniciales o luego de una iteración.
- **Análisis del riesgo**: de acuerdo con el relevamiento de requerimientos se decide si se continúa con el desarrollo.
- **Implementación**: es un prototipo basado en los requerimientos.

- **Evaluación**: el cliente evalúa el prototipo, si da su conformidad termina el proyecto. En caso contrario se incluyen los nuevos requerimientos solicitados por el cliente en la siguiente iteración.



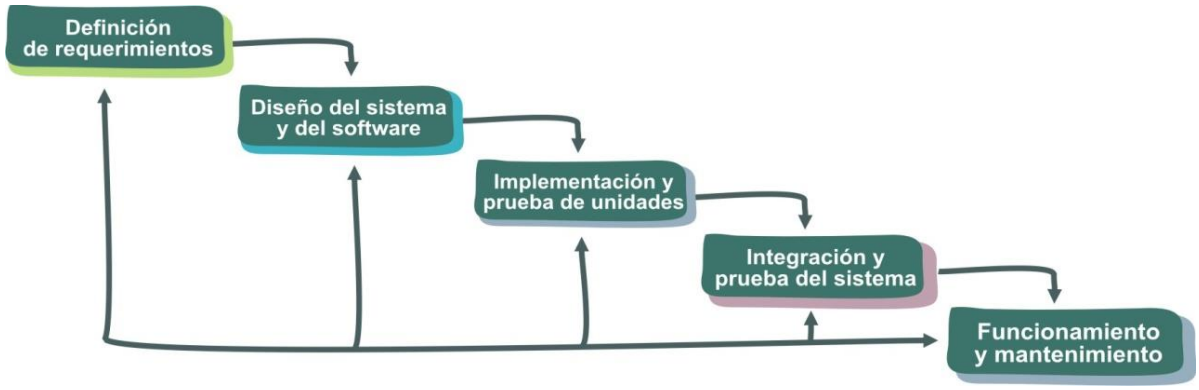
Espiral.

1.3.3. CASCADA

El primer modelo de proceso de desarrollo de software que se publicó, se derivó de procesos de ingeniería de sistemas más generales (Royce 1970, citado en Sommerville, 2005).

A continuación se verá cómo de un diseño previo se deriva otro, dando así su nombre, cascada. Cayendo de una a otra, las etapas de este modelo se transforman en actividades fundamentales de desarrollo:

1. Análisis y definición de requerimientos. Los servicios, restricciones y metas del sistema se definen a partir de las consultas con los usuarios.
2. Diseño del sistema y del software. El proceso de diseño del sistema divide los requerimientos en sistemas hardware o software.
3. Implementación y prueba de unidades. Durante esta etapa, el diseño del software se lleva a cabo como un conjunto o unidades de programas.
4. Integración y prueba del sistema. Los programas o las unidades individuales de programas se integran y prueban como un sistema completo para asegurar que se cumplan los requerimientos del software.
5. Funcionamiento y mantenimiento. Por lo general, aunque no necesariamente, ésta es la fase más larga del ciclo de vida. El sistema se instala y se pone en funcionamiento práctico. El mantenimiento implica corregir errores.

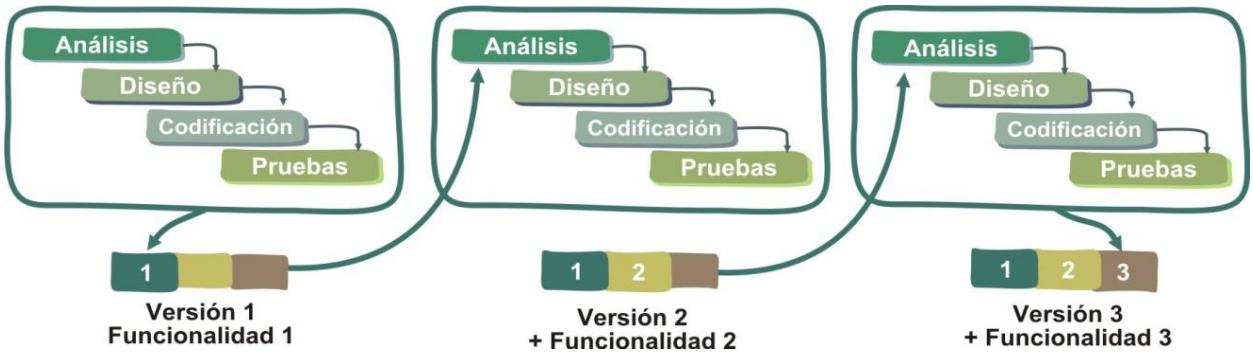


Modelo cascada

1.3.4. INCREMENTAL

Este modelo está basado en varios ciclos cascada realimentados aplicados repetidamente, es decir que va incrementando las funcionalidades del programa. Se realiza construyendo por módulos que cumplen las diferentes funciones del

sistema, lo que permite ir aumentando gradualmente las capacidades del software. Desarrollar un módulo o módulos por separado resulta excelente modelo cuando es desarrollado por varios programadores.



Modelo incremental.

El modelo de ciclo de vida incremental genera algunos beneficios como:

- Construir un sistema pequeño siempre es menos riesgoso que construir un sistema grande.
- Como se desarrolla independientemente las funcionalidades, es más fácil relevar los requerimientos del usuario.
- Si se detecta un error grave sólo se desecha la última iteración.
- No es necesario disponer de los requerimientos de todas las funcionalidades en el comienzo del proyecto, además facilita la labor del desarrollo con la conocida filosofía de "divide y vencerás".

Este modelo de ciclo de vida no está pensado para cierto tipo de aplicaciones, sino que está orientado a cierto tipo de usuario o cliente.

CIERRE DE LA UNIDAD

Has concluido la primera unidad del curso. A lo largo de ésta se revisaron conceptos generales sobre el análisis orientado a objetos, su definición, características y

ventajas. Posteriormente identificaste los conceptos básicos de los modelos orientados a objetos, tales como abstracción, encapsulamiento, modularidad, herencia y polimorfismo, cuyo propósito fue dar un panorama general para identificar un modelo orientado a objetos. De la misma manera, se identificaron los ciclos de vida del software y los tipos de ciclos que existen al diseñar un sistema orientado a objetos.

Es aconsejable que revises nuevamente la unidad en caso de que los temas que se acaban de mencionar no te sean familiares o no los recuerdes, de no ser este tu caso, ya estás preparado(a) para seguir con la unidad 2, en donde abordarás los requerimientos para el análisis del diseño orientado a objetos, realizarás levantamientos de requerimientos y la documentación necesaria, teniendo en cuenta los estándares que deben cumplir y los tipos de modelos para el desarrollo de software.

PARA SABER MÁS

Si deseas saber más acerca de la programación orientada a objetos, visita las siguientes direcciones electrónicas:

- Introducción a la programación orientada a objetos. Recuperado de http://java.ciberaula.com/articulo/tecnologia_orientada_objetos/

- Programación orientada a objetos. Recuperado de http://zarza.usal.es/~fgarcia/doc/tuto2/l_1.htm

FUENTES DE CONSULTA

Bibliografía básica

- Booch-Grady. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. México: Pearson Educación.
- Del Río San José, J. (2010). *Introducción al tratamiento de datos espaciales en hidrología*. Berlín: Bubok.
- Kendall, E. (2005). *Análisis y diseño de sistemas*. México: Pearson Educación.
- Seen, J. (1990). *Análisis y diseño de sistemas de información*. México: McGraw-Hill.

Bibliografía complementaria

- Ciberaula. (2010). *Programación orientada a objetos*. Recuperado de http://java.ciberaula.com/articulo/tecnologia_orientada_objetos/
- Coad, P. y Yourdon, E. (1990). *Object Oriented Programming*. USA: Yourdon Press.
- Fowler, M., y Kendall, S. (2000). *UML gota a gota*. México: Prentice-Hall.
- Fernández, S. (1995). *Fundamentos del diseño y la programación orientada a objetos*. México: McGraw Hill.
- Microsoft Autorized Academic. (2010). *Principles of Components Desing 1518*. USA: Microsoft.
- Sommerville, I. (2005). *Ingeniería del software*. México: Pearson Educación.