Capítulo 3

Fundamentos teóricos

3.1 Proceso de desarrollo de un sistema informático

Sistema

Un sistema se define como un conjunto de elementos relacionados entre sí de alguna u otra manera para llegar a un fin común. Enfocándonos al área de sistemas informáticos, dichos elementos son el *hardware*, el *software* y los usuarios.

Aunque es muy común usar "sistema informático" y "sistema de información" como sinónimos son términos diferentes. Un sistema de información comprende elementos como personas, datos, técnicas y recursos materiales para la administración de los datos y su posterior aprovechamiento. En el sistema informático se ven involucrados equipos de cómputo que permitirán el almacenamiento, procesamiento y acceso a dicha información.

Los sistemas, aunque tengan diferentes objetivos, comparten elementos de los que dependerá su buen funcionamiento.

Uno de los agentes más importantes que forma parte de un sistema informático son las personas, ya que estas son las que ingresan los datos por medio de periféricos de entrada, dando pie a una serie de procesos y generando la información que puede ser consultada.

Otro de los elementos que integran los sistemas son los procedimientos, los cuales atienden a las actividades del usuario, procesando los datos y generando la información que posteriormente será consultada, limitando este acceso solo a las personas autorizadas.

Un tercer componente que podemos encontrar es el equipo, que hace referencia a cuestiones técnicas como los dispositivos de cómputo y en general, la infraestructura necesaria para que el sistema funcione de manera correcta.

Ingeniería de software

El *software* se puede definir como el conjunto de componentes lógicos (procedimientos, reglas, datos, etc.) que hacen funcionar y realizar tareas específicas a una computadora, pueden ir desde lo que es el sistema operativo hasta aplicaciones específicas como editores de texto, videojuegos, editores de imágenes y más.

En un principio, el software era desarrollado por quien tenía una necesidad específica y lo producía conforme a su experiencia o mejor dicho, de acuerdo a lo que su intuición le dictara, más que nada, la programación era un arte.

Para generar un producto de *software* se requieren de ciertos métodos y técnicas para que el desarrollo sea de calidad, de ahí que haya surgido la Ingeniería de software, rama de la ingeniería que cuida los aspectos mediante la aplicación de procesos previamente corroborados.

La Ingeniería de software se puede definir como el estudio de principios y metodologías para generar el conocimiento necesario para el buen diseño, desarrollo, operación y mantenimiento del *software*.

Algunos problemas que se llegaron a detectar en la producción de *software* incluyen:

a) Retrasos considerables en la entrega.
b) Poca productividad.
c) Elevados costos de producción y cargas de mantenimiento.
d) Baja calidad y fiabilidad del producto.
e) Gran dificultad en el mantenimiento.

A lo anterior se le conoce como crisis del *software* y es mediante el desarrollo de ciertas metodologías y la implementación de nuevas herramientas con las que se pretende evitar problemas.

Lo que se busca con estos cambios es solucionar problemas de administración, calidad, productividad y sobre todo, facilidad de

mantenimiento, actividad que demanda una gran cantidad de recursos de toda índole (humanos, materiales y económicos).

Dentro de la Ingeniería de software se han desarrollado diferentes metodologías que ayudan a mejorar la producción, este proceso se denomina ciclo de desarrollo o ciclo de vida e incluye varias fases que van desde el diseño, pasando por la codificación y pruebas, hasta el mantenimiento del producto.

3.2 Metodología de desarrollo

3.2.1 Ciclo de vida del software

El ciclo de vida del software se utiliza para estructurar las actividades que se llevarán a cabo en el desarrollo de un producto.

En todo sistema se deben considerar tiempo y asignación de recursos para el desarrollo y posteriormente, para el mantenimiento y que el producto siga teniendo una vida útil.

Existen varios modelos de ciclo de vida, dependiendo de esto serán las etapas que lo componen y la forma en cómo se llevará a cabo la realización del proyecto.

Independientemente del modelo que se trate, hay etapas en común:

a) Análisis. Estudio de los requerimientos para determinar su viabilidad.
b) Diseño. Abstracción del sistema y elaboración de la interfaz de usuario.
c) Codificación. Propiamente la elaboración del sistema como tal.
d) Pruebas. Verificación del correcto funcionamiento del producto.
e) Mantenimiento. Correcciones y/o modificaciones al sistema.

Análisis

Tanto el desarrollador como el usuario tienen un papel activo en la Ingeniería de software, es aquí, en esta interacción usuario - desarrollador donde nacen un conjunto de actividades llamadas análisis. El usuario intentará planear un sistema confuso a nivel de descripción de datos, funciones y comportamiento. El desarrollador debe actuar como interrogador, como un consultor que resuelve problemas y como negociador.

El análisis de requisitos permite al ingeniero de sistemas especificar las características operacionales del *software* (función, datos y rendimientos), indica la relación de la interfaz con otros elementos del sistema y establece las restricciones que debe cumplir el producto.

El análisis de requisitos del *software* se enfoca en cinco áreas de trabajo:

a) Reconocimiento del problema.
b) Evaluación del problema.
c) Modelado.
d) Especificación.
e) Revisión.

Diseño

El diseño del *software* es realmente un proceso *multipasos* que se enfoca sobre cuatro atributos distintos del problema:

a) Estructura de datos.
b) Arquitectura del *software*.
c) Representaciones de interfaz.
d) Detalle procedimental (algoritmos).

El proceso de diseño traduce requisitos de una representación del *software* donde se pueda evaluar su calidad antes de que comience la codificación. Si el diseño se ejecuta de una manera detallada, la codificación puede realizarse mecánicamente.

Codificación

El diseño debe traducirse en una forma legible para la máquina, para este fin se utilizan lenguajes de programación que pueden ser procesados a diferentes niveles por los equipos de cómputo. La etapa de codificación puede variar en función de la tecnología en la que se va a implementar la solución de *software*.

Pruebas

Una vez que se ha generado el código, comienzan las pruebas del programa. Las cuales se enfocan en la lógica interna del *software*, asegurando que todas las sentencias se han probado, y sobre las funciones externas. Es decir, se realizan pruebas integrales para asegurar que la entrada definida producirá los resultados que realmente se requieren.

Mantenimiento

El *software*, indudablemente sufrirá cambios después de que se entregue al usuario final. Esto se deberá a diversas razones, por ello, el mantenimiento del *software* se aplica a cada uno de los pasos precedentes.

A continuación se describen algunos modelos de ciclo de vida:

3.2.2 Ciclo de vida en cascada

También se le conoce como modelo lineal secuencial. Es una técnica que se derivó de otras ramas de la ingeniería, tenía como objetivo el mejorar la calidad del *software* y reducir sus costos.

Es el modelo que más ha sido utilizado. La forma en cómo opera consiste en concluir cada etapa con un alto grado de exactitud y verificar si es posible continuar con la siguiente. Una vez que se ha llegado al periodo de mantenimiento es posible regresar a etapas anteriores para hacer cambios, haciendo nuevamente el recorrido hasta llegar al final.

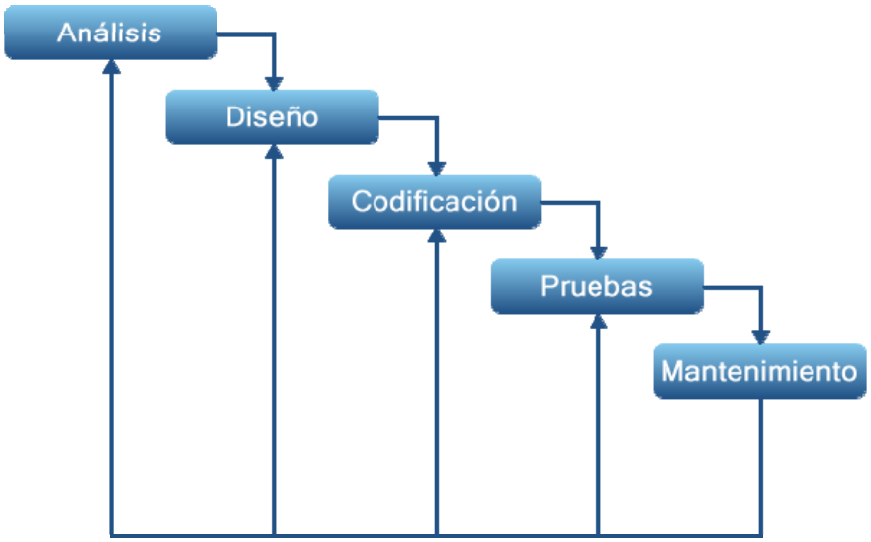

Figura 3.1 Ciclo de vida en cascada

Ventajas:

a) Facilidad en la planificación.
b) Producto resultante de alta calidad.

Desventajas:

a) Disponibilidad de todos los requisitos del sistema.
b) Resultados perceptibles hasta que el proyecto ya se encuentre en una etapa avanzada.
c) Gran impacto de errores cuando el proyecto va avanzado.
d) Repercusión en los tiempos de entrega cuando hay retrasos en etapas tempranas.
e) Mayor costo del desarrollo.

Un ejemplo de aplicación de este modelo lo podemos encontrar en un proyecto de reingeniería, donde ya están establecidas las especificaciones.

3.2.3 Ciclo de vida en V

Está basado en el modelo en cascada y consta de las mismas fases, con la ventaja de que una fase también nos sirve para verificar o validar otras etapas.

En la figura 3.2 podemos ver que el análisis y diseño se encuentran del lado izquierdo y del derecho tenemos las pruebas y el mantenimiento y el vértice lo hace la fase de codificación.

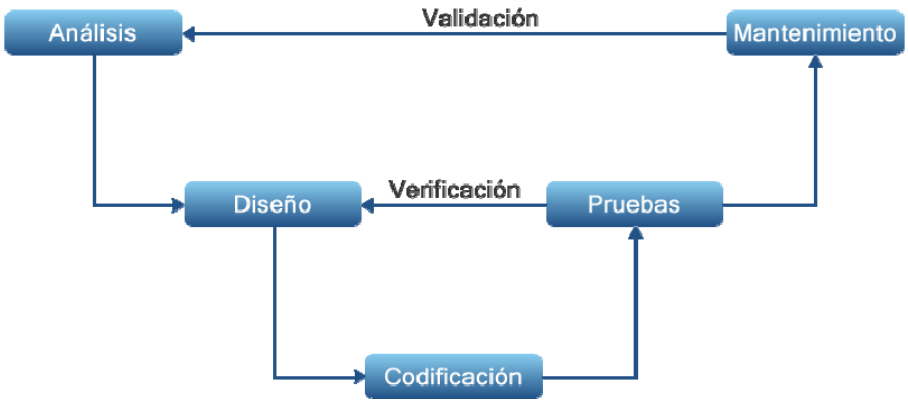

Figura 3.2 Ciclo de vida en "V"

Ventajas

a) Facilidad en la detección de fallos gracias a la relación entre desarrollo y pruebas.
b) El modelo se puede ajustar completamente a las necesidades del proyecto.
c) Ofrece información detallada (instrucciones, recomendaciones, explicaciones, etc.) sobre cómo implementar cada etapa y el camino a seguir.

Desventajas

a) Necesidad de todos los requerimientos por parte del usuario.
b) Producto disponible hasta el final del ciclo de vida.
c) Costo elevado en el periodo de pruebas.

3.2.4 Ciclo de vida por prototipos

Forma parte de los modelos de desarrollo evolutivo y es de gran ayuda cuando los requerimientos no están claramente establecidos, cuando no se sabe cómo implementar la interfaz de usuario o en algún otro caso que implique una gran cantidad de modificaciones y/o interacción con el usuario.

El prototipo nos ofrece un mejor medio para comunicarnos con el usuario y así poder identificar los requerimientos. Gracias a la retroalimentación se afinan detalles para empezar con el desarrollo formal del proyecto.

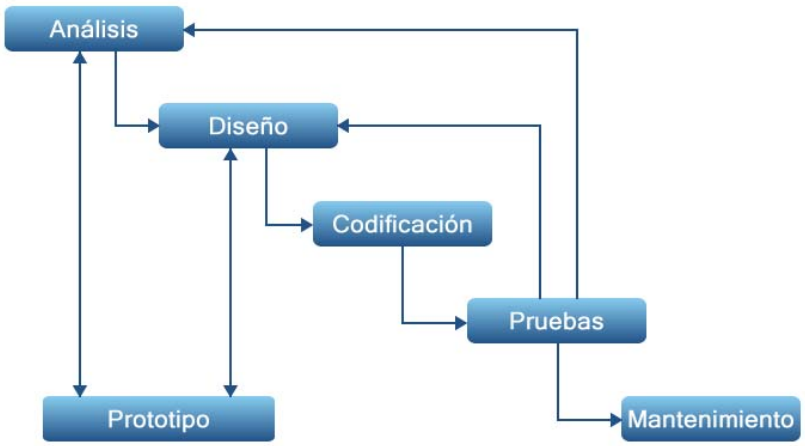

Figura 3.3 Ciclo de vida por prototipos

Ventajas

a) No es necesaria la descripción completa de los requisitos.
b) Se pueden realizar modificaciones en etapas tempranas del desarrollo.
c) Entendimiento más claro por parte del usuario ya que el planteamiento no se encuentra solamente en la documentación.

Desventajas

a) Se puede llegar a extender excesivamente en el tiempo de desarrollo del proyecto.
b) Difícil administración si no se tiene un objetivo claro del *software* a desarrollar.
c) El usuario e incluso los desarrolladores pueden considerar al prototipo como un sistema terminado.

3.2.5 Modelo DRA

El Desarrollo Rápido de Aplicaciones (DRA) es un modelo de proceso del desarrollo del *software* lineal secuencial que enfatiza un ciclo de desarrollo extremadamente corto, como se muestra en la siguiente figura:

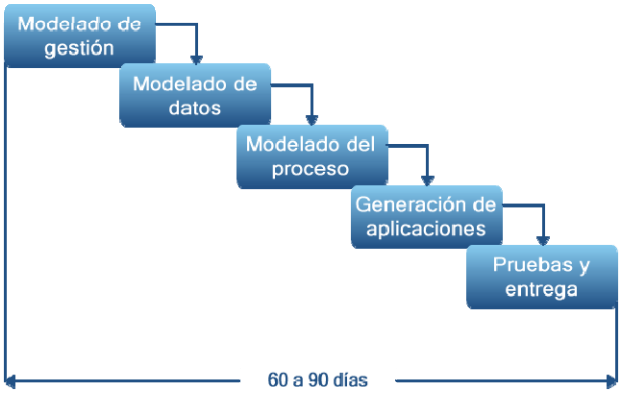

Figura 3.4 Modelo DRA

En el DRA se logra el desarrollo rápido utilizando un enfoque de construcción basado en componentes. Si se comprenden bien los requisitos y se limita el alcance del proyecto, el proceso DRA permite al equipo de desarrollo crear un sistema completamente funcional dentro de periodos cortos de tiempo. Cuando se utiliza principalmente para aplicaciones de sistemas de información, el enfoque DRA comprende las siguientes fases:

Modelado de gestión

El flujo de información entre las funciones de gestión se modela de forma que responda a las siguientes preguntas: ¿Qué información conduce el

proceso de gestión? ¿Qué información se genera? ¿Quién la genera? ¿A dónde va la información? ¿Quién la procesó?

Modelado de datos

El flujo de información definido como parte de la fase de modelado de gestión se refina como un conjunto de objetos de datos necesarios para apoyar la empresa. Se definen las características, llamadas atributos, de cada uno de los objetos y las relaciones entre ellos.

Generación de aplicaciones

El DRA asume la utilización de técnicas de cuarta generación. En lugar de crear *software* con lenguajes de programación de tercera generación, el proceso DRA trabaja para volver a utilizar componentes de programas ya existentes (cuando es posible) o crear componentes reutilizables (cuando sea necesario). En todos los casos se utilizan herramientas automáticas para facilitar la construcción del *software*.

Pruebas de entrega

Como el proceso DRA enfatiza la reutilización ya se han comprobado muchos de los componentes de los programas, esto reduce tiempo de pruebas. Sin embargo, se deben probar los componentes nuevos y se deben ejercitar todas las interfaces a fondo.

3.2.6 Ciclo de vida en espiral

El modelo en espiral consta de una serie de ciclos que se repiten dando como resultado un producto con mejoras respecto del anterior.

Este modelo es similar al de prototipo pero añade un componente denominado análisis de riesgos. Éstos abarcan aspectos como la mala interpretación de requerimientos, errores de diseño, problemas de implementación, etc.

El primer paso es definir los objetivos y posteriormente se realiza el análisis de los riesgos. Si después de haber hecho dicho análisis se

observa que hay incertidumbre sobre el problema, entonces se hace uso de un prototipo con el fin de entenderlo mejor y depurar los requerimientos proporcionados por el usuario.

Una vez terminada esta parte, el usuario evalúa los productos obtenidos y hace la retroalimentación para hacer las modificaciones necesarias y mejorar el desarrollo.

En cada vuelta de la espiral, la actividad de ingeniería se realiza mediante el ciclo de cascada o por medio del modelo de prototipos.

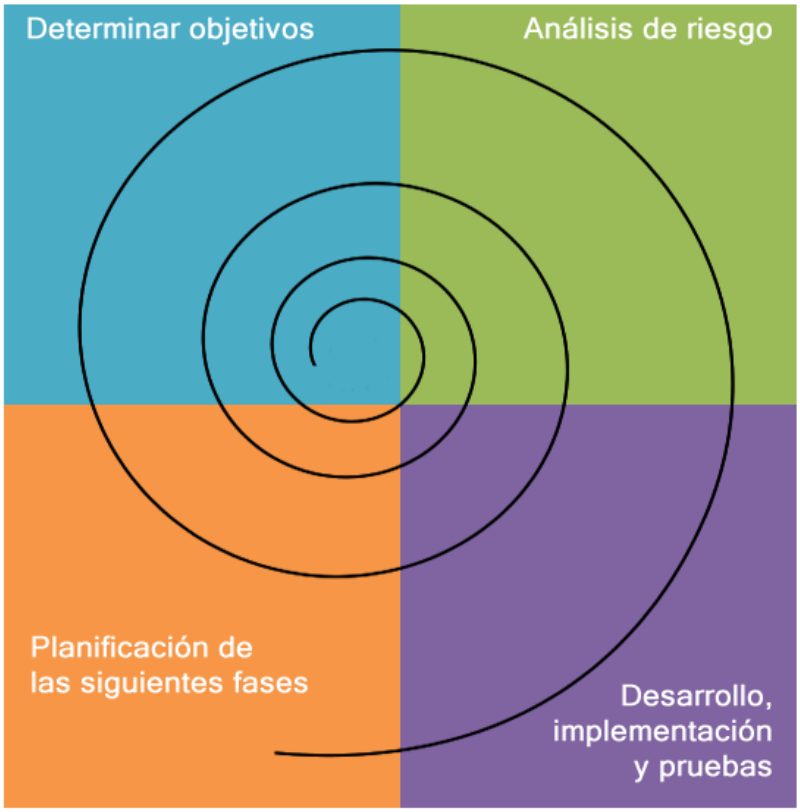

Figura 3.5 Ciclo de vida en espiral

Ventajas

a) No se necesita una definición completa de los requerimientos para iniciar el proyecto.
b) Validación de requerimientos desde etapas tempranas.

c) Facilidad para detectar errores de análisis y diseño.

Desventajas

a) Participación constante por parte del usuario.
b) Mucho tiempo de desarrollo y alto costo.
c) Requiere experiencia en la identificación de riesgos.

3.2.7 Ciclos de vida orientados a objetos

Los objetos tienen como característica que están basados en componentes, permitiendo tanto un alto grado de modularidad como la segmentación en proyectos que pueden llegar a trabajarse como independientes. Al aplicar un ciclo de vida orientado a objetos tenemos la ventaja de que el modelo es iterativo e incremental.

Modelo fuente

El modelo fuente es la forma más utilizada del ciclo de vida orientado a objetos, una de las características principales es que permite que cada clase se encuentre en diferente etapa, teniendo un ciclo de vida para cada una de ellas.

Contempla dos periodos para el desarrollo del proyecto: la etapa de crecimiento donde se construye el sistema y la etapa de madurez, donde se lleva a cabo el mantenimiento y cada mejora tiene un ciclo propio.

Figura 3.6 Ciclo de vida orientado a objetos

Ventajas

a) Mayor productividad debido a que cada clase tiene un ciclo de vida propio.
b) Reducción en el tiempo de desarrollo.

Desventajas

a) Dificultad para aprender y emplear la metodología.
b) Mantenimiento difícil.

3.2.8 Herramientas CASE

Las herramientas CASE (*Computer Aided Software Engineering* - Ingeniería de Software Asistida por Computadora) son un conjunto de utilidades, métodos y técnicas empleadas por los profesionales del desarrollo de *software* para facilitar y agilizar el ciclo de vida en un sistema, ya sea completamente o solo enfocándose en alguna de las etapas que lo componen. Es útil en tareas como el diseño del proyecto,

cálculo de costos, generación de código a partir del diseño, generación automática de documentación, etc.

Actualmente son varias las instituciones públicas que hacen uso de estas herramientas puesto que facilitan las tareas del ciclo de vida, como las etapas de codificación e implementación de pruebas, permitiendo un desarrollo más rápido de los sistemas, a su vez reducen los costos de producción, haciéndolas más productivas y ofreciendo una mayor calidad de servicio al público.

Algunas de las características y ventajas de las herramientas CASE sobre el desarrollo del *software* son:

a) Permiten la comprobación de errores en etapas tempranas del desarrollo.
b) Tienen soporte de reusabilidad.
c) Mejoran la productividad en el desarrollo y mantenimiento.
d) Aumentan la calidad del producto.
e) Reducen el tiempo y el costo de desarrollo mediante la generación de código, documentación, automatización de pruebas, etc.
f) Ayudan a tener una mejor planificación de un proyecto.
g) Incrementan la biblioteca de conocimiento informático de una organización ayudando a la búsqueda de soluciones para los futuros desarrollos.
h) Facilitan el uso de las distintas metodologías propias de la Ingeniería del *software*.

3.3 Técnicas de implementación de sistemas

Para el desarrollo de sistemas hacemos uso de alguna de las metodologías que se han tratado. Una metodología se puede definir como el conjunto de procedimientos necesarios para llegar a un objetivo. Enfocándonos en la parte de desarrollo de sistemas, la metodología es un modo sistemático de producir *software*, donde el proyecto se divide en

etapas y cada una de estas tiene tareas a realizar, además de tener bien definidas las entradas que recibe y las salidas que produce.

Lo que se busca mediante el uso de una metodología son cuatro puntos principales:

a) Satisfacer las necesidades del usuario.
b) Realizar cambios fácilmente una vez que el sistema esté en funcionamiento.
c) Proporcionar una interfaz intuitiva y amigable para usar el sistema.
d) Producir un sistema capaz de funcionar sin fallos y con la mínima cantidad de correcciones posibles.

Al igual que en el proceso de producción de un carro, o en la elaboración de un refresco, en las metodologías podemos encontrar distintos caminos dependiendo de la forma como se quieran satisfacer los requerimientos dados: la metodología estructurada y la metodología orientada a objetos.

3.3.1 Metodología estructurada

La metodología estructurada busca crear modelos de forma descendente, es decir, descompone el sistema de forma que se puedan tener partes lo más pequeñas posibles. La metodología estructurada se divide a su vez en orientada a procesos y orientada a datos.

Existen varias herramientas que se utilizan en la metodología orientada a procesos, entre las que encontramos:

a) Diagramas de Flujo de Datos (DFD): representan la forma en la que los datos se mueven y se transforman e incluye:
    1. Procesos
    2. Flujos de datos
    3. Almacenamiento de datos

Los procesos individuales se pueden a su vez descomponer en otros DFD de nivel superior.

b) Especificaciones de procesos: es una herramienta que nos define las acciones que lleva a cabo una función en el sistema, es decir, lo que hay en medio de las entradas y las salidas.
c) Diccionario de datos: es el conjunto de datos que incluye las características lógicas (si es una cadena, entero, decimal, la longitud, rango de valores, si debe o no ser tener un valor, etc.) para ser utilizados en un sistema.
d) Diagrama de Transición de Estados (DTE): son herramientas que modelan procesos que dependen del tiempo, se basan en el estado que tiene el sistema y en los cambios de estado.
e) Diagramas Entidad - Relación (DER): nos ayuda al modelado de datos mostrando las entidades principales, sus propiedades y la interrelación que hay entre ellas.

3.3.2 Metodología orientada a objetos

Es una metodología de más reciente creación que está enfocada al uso de componentes, teniendo la ventaja de reutilizar código y de que su mantenimiento sea más fácil debido a la mejor localización de los cambios.

El paradigma de orientación a objetos se centra en la entidad principal (objeto), la cual contiene los datos y la forma en cómo estos datos pueden ser manipulados.

El objeto es una abstracción de los entes del mundo real y se puede utilizar por medio de la instanciación de una clase, que no es otra cosa más que la esencia del objeto. En otras palabras, la clase nos define las características del objeto.

Algunos de los rasgos más característicos de la orientación a objetos son:

a) Herencia: como si se tratase de una familia, las clases también pueden heredar métodos y atributos de una clase padre.

b) Polimorfismo: es la capacidad de nombrar a un método (que puede hacer cosas diferentes) de la misma forma para objetos de diferente clase.
c) Encapsulamiento: se le llama así a la propiedad de los objetos que hace que aquellas características no necesarias para interactuar con otros no sean visibles.

3.3.3 Metodología OMT

La metodología OMT (*Object Modeling Technique* - Técnica de Modelado de Objetos) es una práctica que nos sirve para el diseño y modelado de un sistema de *software*.

Algunas de sus características principales es que considera la comunicación con el usuario, muestra la información de una forma diferente y disminuye la complejidad del problema mediante la segmentación de procesos.

La etapa de diseño se descompone en:

a) Modelo de objeto. Representa los artefactos del sistema, es decir las clases y asociaciones con atributos y operaciones.
b) Modelo dinámico. Constituye las interacciones entre los componentes ya mencionados, estas interacciones son eventos, estados y transiciones.
c) Modelo funcional. Hace referencia a los métodos del sistema desde el punto de vista de flujo de datos.

Esta metodología pone especial interés en las etapas de análisis y diseño, siendo la precursora de UML.

3.3.4 Metodología UML

UML (*Unified Modeling Language* - Lenguaje de Modelado Unificado) es un lenguaje de modelado de sistemas de software muy utilizado actualmente y aprobado en el 2005 por la ISO como un estándar.

Este lenguaje busca hacer una descripción completa del modelo y los artefactos del sistema, contempla formas de representación para procesos de negocios, funciones del sistema, expresiones del lenguaje de programación, esquemas de bases de datos, etc.

Existen 2 versiones de UML, la que podemos encontrar actualmente es la 2.3 y cuenta con 13 tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas y se agrupan en diagramas de estructura, comportamiento e interacción.

Los diagramas de estructura enfatizan en los elementos que deben existir en el sistema modelado:

a) Diagrama de clases.
b) Diagrama de componentes.
c) Diagrama de objetos.
d) Diagrama de estructura compuesta.
e) Diagrama de despliegue.
f) Diagrama de paquetes.

Los diagramas de comportamiento hacen hincapié en lo que debe suceder en el sistema modelado:

a) Diagrama de actividades.
b) Diagrama de casos de uso.
c) Diagrama de estados.

Los diagramas de interacción son un subtipo de diagramas de comportamiento, que prestan especial importancia sobre el flujo de control y de datos entre los elementos del sistema modelado:

a) Diagrama de secuencia.
b) Diagrama de comunicación.
c) Diagrama de tiempos.
d) Diagrama global de interacciones.

3.4 Metodología utilizada

Después de haber estudiado y analizado los diferentes ciclos de vida que hay y considerando la forma en cómo se nos presenta la interactividad con el usuario creemos que el modelo que mejor se adapta al desarrollo del sistema SIGED es el ciclo de vida en espiral dado que contempla que el sistema siga en funcionamiento por un largo plazo. En cada iteración, como lo define el modelo, se buscará mejorar la calidad, el funcionamiento y el rendimiento del producto.

Aunado a esto, el ciclo del vida nos ayudará a integrar los módulos necesarios para dotar de una mayor funcionalidad al sistema y así poder responder a las necesidades de otras áreas de la CONADE.