12

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

"UML Y RUP COMO ELEMENTOS CLAVE EN EL DESARROLLO DE SISTEMAS ORIENTADOS A OBJETOS"

TESIS PROFESIONAL
QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN INFORMÁTICA

PRESENTAN:
Silvia Jiménez Luna
Claudia Elvira Soriano Monzalvo

ASESOR: Dr. Ricardo Rivera Soler

México, D.F. 2002

TESIS CON
FALLA DE ORIGEN

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

A Dios:

Por darme la vida y su amor infinito.
Por bendecirme con el cariño y el apoyo de mis padres y hermanos.
Por iluminarme en cada paso que doy.

A mi Madre:

Por su amor y apoyo incondicional.
Por su comprensión y sus consejos.
Por sus desvelos y preocupaciones.
Por sus atenciones y cuidados.

A mi Padre(+):

Por ser una inspiración para la realización de este sueño.

A mis Hermanos:

Por su confianza y apoyo.
Por sus palabras de aliento.
En especial a Luz por sus sacrificios, su bondad y apoyo absoluto.

A Daniel Cruz Vargas:

Por brindarme su amor, apoyo y confianza.
Por la seguridad que me hace sentir.
Por comprenderme y alentarme.
Por estar a mi lado incondicionalmente y hacerme pasar los momentos más felices de mi vida.

A mi Asesor Dr. Ricardo Rivera Soler:

Por su tiempo y su paciencia.
Por sus conocimientos y experiencia compartida.
Por su amistad.

A la Universidad Nacional Autónoma de México:

Por la formación y educación que me dio.

A Claudia Soriano Monzalvo:

Por confiar en mí como persona y como compañera de tesis.
Por su dedicación, trabajo y esfuerzos empleados.
Por sus conocimientos y experiencia compartidos.

A mis amigos:

Por sus consejos y palabras de ánimo.
Por los momentos de alegría que me han dado.

Silvia Jiménez Luna

A Dios:

Por darme la oportunidad de estar viva.
Por tener los padres que tengo.
Por darme fuerza para seguir adelante a pesar de los obstáculos.

A mis Padres:

Por apoyarme incondicionalmente.
Por guiarme con sus sabios consejos.
Por su comprensión, cuidados y desvelos.

A mi Esposo:

Por brindarme su apoyo en todo momento.
Por compartir mis desvelos.
Por enseñarme que con AMOR todo es posible.

A mi tía Isabel :

Por ser siempre para mí un ejemplo.
Por brindarme su apoyo incondicional para alcanzar este sueño.
Por enseñarme a valorar lo que tengo.

A mi Asesor:

Por guiarme por el camino correcto.
Por sus sabios consejos.
Por su amistad.

A la Universidad Nacional Autónoma de México:

Por darme la oportunidad de ser una verdadera profesionista universitaria.
Por brindarme valiosos conocimientos.

Claudia Elvira Soriano Manzalvo

ÍNDICE

ÍNDICE

INTRODUCCIÓN

INTRODUCCIÓN

Hoy en día, la necesidad de desarrollar software de alta calidad ha venido creciendo dentro de la industria mundial del software, la preocupación de las empresas desarrolladoras de sistemas por mejorar sus procesos de desarrollo es latente. Por otra parte, encontramos a organizaciones e instituciones con necesidades de información veraz y oportuna, que requieren sistemas seguros, confiables y adaptables en el menor tiempo posible y con el presupuesto estimado.

Un factor que ha dado origen al problema de no desarrollar software de calidad, que cumpla al 100% con los requerimientos de los usuarios, es la serie de dificultades que se tienen dentro de las etapas de Análisis y Diseño de los sistemas, esto se da por no emplear las herramientas adecuadas.

El paradigma orientado a objetos es una nueva y diferente forma de pensar, actualmente no solo se aplica a los lenguajes de programación, sino que también se ha propagado a los métodos de análisis y diseño, así como a otras áreas, tales como las bases de datos, hardware, sistemas operativos, entre otras.

El desarrollo de sistemas puede caracterizarse como la solución de problemas, incluye la comprensión y conceptualización del problema para resolverlo y realizar su solución. *Conceptualizar un problema* involucra comprenderlo y representarlo en alguna forma. *Resolver el problema* involucra manipular construcciones y representaciones desde el dominio del problema de las cuales se derive la solución deseada. *Realizar una solución* involucra implantar dichas construcciones y representaciones en una forma útil.

A través de esta tesis pretendemos dar a conocer los puntos clave que consideramos como pilares fundamentales para lograr el desarrollo de sistemas orientado a objetos exitosos:

- Un *proceso de desarrollo* que defina la relación entre las tareas.
- Una *lenguaje unificado* que permita un análisis y diseño uniforme que mejore a comunicación entre usuarios y desarrolladores.
- *Personas* con formación, capacitación y motivación.

Consideramos que un *lenguaje unificado* que permita mejorar la comunicación entre los usuarios y el equipo de desarrollo, que permita que evolucionen en conjunto y no por separado; un *proceso de desarrollo* que permita aclarar las fases de desarrollo (análisis, diseño, desarrollo, implantación, pruebas), que ayude a los usuarios a evaluar la calidad de las soluciones planteadas por el equipo de desarrollo a sus requerimientos, así como controlar las tareas asignadas a cada uno de los integrantes del equipo de desarrollo; llevados a cabo por parte de un equipo de desarrollo capacitado y motivado, permite desarrollar sistemas orientados a objetos de calidad.

Esta tesis está integrada por cinco capítulos:
En el capítulo I se describe el *Marco Problemático,* el cual incluye los antecedentes, donde planteamos nuestro interés por el tema y vivencias sobre el mismo, también se hizo la identificación del problema basándonos en la experiencia laboral propia y en la de personas conocidos dentro del ámbito informático, así como en el estudio de la situación actual de las organizaciones que desarrollan software. Con el objetivo de afirmar la existencia del problema identificado, se construyó un cuestionario que fue aplicado a personas con conocimientos, teórico o prácticos, relacionado con el tema estudiado. Este estudio nos permitió

plantear nuestra hipótesis preliminar, la cual constituyó nuestra primera aproximación, por último presentamos los objetivos que a lograr con el desarrollo de esta tesis.

El capítulo 2 lo constituye el *Marco Teórico*, que contiene una descripción de los libros, tesis, revistas, conferencias, páginas web y notas que sirvieron de apoyo para documentarnos y ampliar nuestro conocimiento sobre el tema y poder desarrollar esta tesis.

En el capítulo 3 se incluyó el *Marco Conceptual*, el cual está dividido en antecedentes, definiciones, evolución, clasificación, métodos y tendencias futuras de UML (Lenguaje Unificado de Modelado, Unified Modeling Language) y RUP (Proceso Unificado Racional, Rational Unified Process). Como antecedentes se incluyeron las diferentes metodologías de programación que existen, el surgimiento del paradigma orientado a objetos y su uso dentro de las metodologías de análisis y diseño, del hardware, de los sistemas operativos, dentro de las bases de datos y dentro de los lenguajes de programación. En la sección de definiciones se incluyeron las palabras que componen el título de la presente tesis, de las cuales se proporcionó su definición etimológica, informática, en inglés, francés, definiciones propias, así como sinónimos y antónimos de las palabras. Como evolución se incluyeron las metodologías que dieron origen a UML y RUP, se incluyen cuadros que explican cómo fueron evolucionando. Como métodos se explica la notación utilizada por UML y las fases que componen a RUP. Como parte final se encuentran las tendencias futuras y la integración de los tres elementos clave en el desarrollo de sistemas orientados a objetos: UML, RUP y personas capacitadas.

En el capítulo 4 se desarrolló el *Marco Metodológico*, donde se plantean las variables y la hipótesis definitiva a la que se llegó después de haber conocido y desarrollado más sobre el tema, se definió el universo, la muestra y el método de investigación para probar la hipótesis. Se estructuró el cuestionario final aplicado que nos permitió llegar a la aprobación o desaprobación de la hipótesis después del estudio de los resultados y conclusiones obtenidas.

En el capítulo 5, M*arco Instrumental*, se encuentran listadas las diferentes aportaciones que realizamos, entre las que están: el desarrollo de un caso práctico, una propuesta de temario para el plan de estudios de la FCA, el desarrollo de un artículo de revista, una página web acerca de la tesis y material necesario para realizar pláticas y conferencias acerca de UML y RUP, todos éstos trabajos se presentan en la sección de anexos.

CAPÍTULO I

1. MARCO PROBLEMÁTICO

1.1 ANTECEDENTES

El *paradigma orientado a objetos* es una opción que brinda grandes ventajas para desarrollar sistemas, tales como: diseño más rápido, reutilización, mayor calidad, programación más sencilla, mantenimiento, integridad, ciclo de vida dinámico, modelo más realista, mejor comunicación entre analistas y usuarios, etc.

Sin embargo, muchos de los beneficios sólo se alcanzan cuando el análisis y diseño orientado a objetos se realiza correctamente combinando los conocimientos y herramientas brindadas por éste.

Dentro de nuestra experiencia en el desarrollo de sistemas hemos observado que el paradigma orientado a objetos dentro del análisis y diseño de sistemas está teniendo grandes éxitos, ya que es una forma totalmente diferente de analizar, diseñar y desarrollar, en la cual se cuida en gran medida la calidad del software.

Creemos que es de vital importancia el uso de una metodología que contemple todos los elementos necesarios para desarrollar software de calidad. Los elementos que consideramos más importantes dentro de una metodología son: el *proceso, la notación y el equipo de desarrollo*, ya que son piezas fundamentales de una metodología eficiente, razón por la cual nos hemos interesado en su estudio. El llevar a cabo un buen *proceso* asegura que en todo momento se sabrá quién es el responsable de hacer qué actividades, permite cuidar en todo momento la calidad, los tiempos y prototipos desarrollados, una *notación* eficiente cubrirá la parte de documentación dentro del análisis y diseño orientado a objetos y ayudará a mejorar la comunicación entre el *equipo de desarrollo*.

Lo anterior nos ha impulsado a dar a conocer de manera práctica y teórica el análisis y diseño orientado a objetos con procesos y con notación, que permitan definir de forma clara a los desarrolladores lo que tienen que realizar para construir software de alta calidad y de acuerdo a las especificaciones de los usuarios.

Nuestro interés acerca del tema *Análisis y Diseño Orientado a Objetos,* se debe a que nos hemos percatado de que la comunidad informática de nuestro país que programa bajo el paradigma orientado a objetos no lleva adecuadamente un estándar de documentación, así como un proceso de desarrollo eficiente.

Además nos interesa mostrar la importancia que tiene la realización de un adecuado análisis y diseño en un proceso de desarrollo orientado a objetos, ya que esto es la base del éxito de un proyecto, lo que permitirá en un futuro, no muy lejano, reducir la actual crisis del software en nuestro país.

El tema de esta tesis lo hemos denominado **"UML y RUP como elementos clave en el desarrollo de sistemas orientados a objetos"**.

Como alumnas de la máxima casa de estudios hemos participado en difundir este tema, impartiendo el curso de ***"Análisis y Diseño Orientado a Objetos"*** en la Dirección General de Servicios de Cómputo Académico (DGSCA) en la Dirección de Sistemas de la Universidad Nacional Autónoma de México (UNAM).

1.2 IDENTIFICACIÓN DEL PROBLEMA

Un factor que ha dado origen al problema de no desarrollar software de calidad, que cumpla al 100% con las especificaciones de los usuarios en el desarrollo de sistemas orientados a objetos, es la serie de dificultades para elaborar un adecuado análisis y diseño que permita visualizar al sistema como un modelo arquitectónico desde diferentes puntos de vista (hardware, software, funcionalidad, comportamiento, procesos, datos), y que además represente las necesidades reales de los usuarios, ***esto sucede continuamente por no emplear una metodología adecuada como RUP (Proceso Unificado Racional) y una notación eficiente como UML (Lenguaje Unificado de Modelado). Empleando estas técnicas flexibles que permiten documentar el sistema y apoyándose en un equipo de trabajo capacitado, se logra adoptar un proceso de desarrollo unificado de acuerdo a las necesidades del proyecto, sin que esto represente graves afectaciones para el diseño y la programación.***

Este problema se ve reflejado dentro de un entorno informático en las empresas que utilizan estos sistemas y en las encargadas de desarrollarlos por el gran cúmulo de información que manejan, la cual resulta difícil de entender con los métodos tradicionales de análisis y diseño orientado a objetos, ya que los modelos o diagramas se vuelven complejos y voluminosos.

Todo esto trae como consecuencia deficiencias en la comunicación entre usuarios y analistas además pérdidas de recursos, puesto que el tiempo de desarrollo del sistema se prolonga, las inversiones monetarias y el esfuerzo se incrementa; lo cual afecta al usuario, al desarrollador y al cliente.

1.3 DEMARCACIÓN DEL FENÓMENO

Está tesis se enfocará principalmente a empresas del Distrito Federal, que utilicen o se dediquen a desarrollar sistemas orientados a objetos.

1.4 CONOCIMIENTO EMPÍRICO EN EL MEDIO

Las siguientes personas que se entrevistaron poseen grado de pasantes, cuentan con experiencia en el análisis, diseño y desarrollo de sistemas de información, entre sus actividades está la de impartir cursos de actualización de diversos temas de computación.

Ma. Teresa Ventura Miranda
Pasante de la Licenciatura en Informática de la Facultad de Contaduría y Administración, Jefa del departamento de Estructuración de la Dirección de Sistemas en la UNAM. Sus actividades son realizar el análisis y diseño de sistemas orientados a objetos y estructurados, impartir cursos acerca de "Análisis y Diseño Orientado a Objetos" en el Diplomado de Comercio Electrónico de la DGSCA-UNAM.

Adrián Rivera González
Pasante de Ingeniero en Computación de la ENEP Aragón UNAM, labora en la Dirección de Sistemas en la UNAM. Sus actividades son realizar el análisis y diseño de sistemas orientados a objetos, e impartir cursos y conferencias acerca de "Análisis y Diseño Orientado a Objetos" y otros cursos relacionados al cómputo e informática.

1.5 OPINIONES PROFESIONALES

Las personas que se entrevistaron poseen estudios de nivel superior, en su ámbito laboral desarrollan sistemas de información utilizando el paradigma orientado a objetos para el análisis y diseño de dichos sistemas.

Alejandro Talavera Rosales
Actuario por la UNAM, actualmente labora para la Dirección General de Servicios de Cómputo Académico (DGSCA). Sus actividades son desarrollar sistemas orientados a objetos y aplicaciones de multimedia.

Alexei Samuel Dezotti Ruiz
Ingeniero en computación por la UNAM, actualmente labora en la Torre de Rectoría de C.U. como Jefe de Informática, ha tomado diversos cursos en Estados Unidos acerca del análisis y diseño orientado a objetos.

Rita Carolina Rodríguez Martínez
Maestra en ciencias de la computación, administra actualmente un servidor de correos, labora en el Instituto e Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS), posee experiencia en el desarrollo de sistemas.

Alejandro Núñez
Ingeniero en computación, graduado en la UNAM, especialista en seguridad en cómputo, actualmente se dedica a desarrollar software libre.

Ernesto Rubio
Ingeniero en computación, labora en el Instituto de Investigación en Matemáticas Aplicadas y en Sistemas (IIMAS), actualmente se dedica a la programación en paralelo de algoritmos para estimación espectral.

1.6 OBJETIVO QUE SE PRETENDE LOGRAR AL APLICAR EL CUESTIONARIO

Conocer la opinión acerca del proceso de desarrollo empleado para la construcción de sistemas orientados a objetos, además de investigar las técnicas de documentación que utilizan para llevar acabo la documentación para el análisis y diseño de un sistema orientado a objetos antes de la programación.

1.6.1 CUESTIONARIO

A: Usuarios de UML y RUP
B: Usuarios de otras herramientas

Pregunta	Razón de ser	Respuesta esperada
¿Utiliza algún proceso especifico para el desarrollo de sistemas?	Conocer qué proceso utilizan para desarrollar sistemas.	Respuesta abierta
¿Qué notación utiliza para el análisis y diseño de sistemas?	Saber en qué tipo de metodología se basan para realizar el análisis y diseño.	Respuesta abierta
¿La notación utilizada le ha permitido una comunicación eficiente con los usuarios?	Indagar qué tan eficiente es la notación que utilizan para la comunicación entre el equipo de trabajo y el usuario.	A = SÍ B = NO
¿La notación utilizada le ha permitido identificar todos los requerimientos de los usuarios?	Indagar qué tan eficiente es la notación que utilizan para obtener todos los requerimientos de los usuarios.	A = SÍ B = NO
¿El proceso utilizado ha influido en la culminación de los sistemas en el tiempo y costos estimados?	Indagar qué tan eficiente es el proceso que utilizan para controlar el tiempo y costos requeridos.	A = SÍ B = NO
¿El proceso de desarrollo que utiliza le proporciona una comprensión global del sistema a modelar?	Saber si el proceso de desarrollo abarca el diseño del sistema desde diferentes puntos de vista: hardware, software, funcionalidad.	A = SÍ B = NO
¿Cree que el proceso de desarrollo que utiliza es el adecuado para verificar el cumplimiento de la calidad y los requerimientos de los usuarios?	Verificar si el proceso de desarrollo que utilizan verifica la calidad del sistema que desarrollan.	A = SÍ B = NO
La documentación que utiliza, ¿refleja realmente la estructura y el comportamiento del sistema?	Indagar qué tan eficiente es la documentación que utilizan, ya que esto influye en la forma en que se comprende el problema y cómo se le dará solución.	A = SÍ B = NO
La documentación que realiza durante el análisis y diseño, ¿le proporciona una eficiente guía de construcción del sistema?	Conocer qué tan homogénea es la documentación con el lenguaje de programación orientado a objetos con el que se pretende desarrollar un sistema.	A = SÍ B = NO

Pregunta	Razón de ser	Respuesta esperada
¿Considera que la documentación que genera durante la etapa del análisis y diseño, comunica eficientemente la funcionalidad del sistema facilitándole el desarrollo del mismo?	Saber si la metodología empleada en el análisis, diseño y el proceso de desarrollo que utilizan constituyen bases suficientes para el desarrollo de sistemas.	A = SÍ B = NO
¿Considera que la documentación que genera durante el análisis y diseño del sistema es fácil de integrar en la programación?	Indagar qué tan clara resulta la documentación de un sistema para los desarrolladores.	A = SÍ B = NO
¿Cuáles son los causas por las que no se lleva a cabo una adecuada documentación en las etapas de análisis y diseño?	Conocer las posibles circunstancias que dan origen al fracaso de los sistemas.	Respuesta abierta
¿Conoce el término de RUP (Rational Unified Process) y UML (Unified Modeling Language)?	Saber qué tan conocidos y utilizados son el UML y el RUP.	Respuesta abierta

1.6.2 FORMATO DEL CUESTIONARIO

UNAM

FCA

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

CUESTIONARIO

Fecha: ____/___/____

Nombre:_____

OBJETIVO: OBTENER UNA OPINIÓN EMPÍRICA Y/O PROFESIONAL PARA TEMA DE TESIS DE LAS ALUMNAS DE LA FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN.

INSTRUCCIONES: ***Conteste las siguientes preguntas, marcando con una X si ó no. En caso de ser necesario escribir en la parte reversa de la hoja o bien anexar hojas.***

Alumnas que aplican el cuestionario: **Silvia Jiménez Luna y Claudia Elvira Soriano Monzalvo.**

1. *¿Utiliza algún proceso específico para el desarrollo de sistemas?*

_____

_____

2. *¿Qué notación utiliza para el análisis y diseño de sistemas?*

_____

_____

3. *¿La notación utilizada le ha permitido una comunicación eficiente con los usuarios?*

SÍ____ NO____

Comentario_____

_____

4. *¿La notación utilizada le ha permitido identificar todos los requerimientos de los usuarios?*

SÍ____ NO____

Comentario_____

_____

5. *¿El proceso utilizado ha influido en la culminación de los sistemas en el tiempo y costos estimados?*

SÍ____ NO____

Comentario_____

_____

6. *¿El proceso de desarrollo que utiliza le proporciona una comprensión global del sistema a modelar?*

SÍ_____ NO_____

Comentario_____

_____

7. *¿Cree que el proceso de desarrollo que utiliza es el adecuado para verificar el cumplimiento de la calidad y los requerimientos de los usuarios?*

SÍ_____ NO_____

Comentario_____

_____

8. *La documentación que utiliza, ¿refleja realmente la estructura y el comportamiento del sistema?.*

SÍ_____ NO_____

Comentario_____

_____

9. *La documentación que realiza durante el análisis y diseño, ¿le proporciona una eficiente guía de construcción del sistema?.*

SÍ_____ NO_____

Comentario_____

_____

10. *¿Considera que la documentación que genera durante la etapa del análisis y diseño, comunica eficientemente la funcionalidad del sistema facilitándole el desarrollo del mismo?*

SÍ_____ NO_____

Comentario_____

_____

11. *¿Considera que la documentación que genera durante el análisis y diseño del sistema es fácil de integrar en la programación?*

SÍ_____ NO_____

Comentario_____

_____

12. *¿Cuáles son los causas por las que no se lleva a cabo una adecuada documentación en las etapas de análisis y diseño?*

_____

_____

6. ¿El proceso de desarrollo que utiliza le proporciona una comprensión global del sistema a modelar?

SÍ_____ NO_____

Comentario_____

7. ¿Cree que el proceso de desarrollo que utiliza es el adecuado para verificar el cumplimiento de la calidad y los requerimientos de los usuarios?

SÍ_____ NO_____

Comentario_____

8. La documentación que utiliza, ¿refleja realmente la estructura y el comportamiento del sistema?.

SÍ_____ NO_____

Comentario_____

9. La documentación que realiza durante el análisis y diseño, ¿le proporciona una eficiente guía de construcción del sistema?.

SÍ_____ NO_____

Comentario_____

10. ¿Considera que la documentación que genera durante la etapa del análisis y diseño, comunica eficientemente la funcionalidad del sistema facilitándole el desarrollo del mismo?

SÍ_____ NO_____

Comentario_____

11. ¿Considera que la documentación que genera durante el análisis y diseño del sistema es fácil de integrar en la programación?

SÍ_____ NO_____

Comentario_____

12. ¿Cuáles son los causas por las que no se lleva a cabo una adecuada documentación en las etapas de análisis y diseño?

_____

_____

13. *¿Conoce el término de RUP (Rational Unified Process) y UML (Unified Modeling Language)?*

SÍ_____ NO_____

Comentario_____

1.6.3 MATRIZ DE RESULTADOS DE LOS CUESTIONARIOS PROFESIONALES Y EMPÍRICOS APLICADOS

PREGUNTA	1	2	3		4		5		6		7		8		9		10		11		12	13	
Talavera	TSP	UML	SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		SÍ			NO	Falta de experiencia del personal que desarrolla. Reportes erróneos. Requerimientos no claros.	SÍ	
Dezotti	RUP	UML	SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		Falta de experiencia.	SÍ	
Rodriguez	ORIENTACIÓN A OBJETOS	BOOCH	SÍ			NO		NO	SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		La comunicación y los comentarios.	SÍ	
Núñez	ANÁLISIS PLANEACIÓN PROGRAMACIÓN	DIAGRAMAS DE FLUJO		NO		NO		NO	SÍ			NO	SÍ		SÍ		SÍ			NO	La complejidad de los sistemas.	SÍ	
Rubio	NINGUNO	UML	SÍ		SÍ			NO	SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		Por las restricciones del tiempo.	SÍ	
Ventura	CASCADA RUP	UML IDEF-X	SÍ		SÍ			NO	SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		Falta de tiempo.	SÍ	
Rivera	RUP	UML	SÍ			NO		NO	SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		Análisis de requerimientos pobre. Mala planeación.	SÍ	

TOTAL	RUP =3 TSP =1 Orientación a objetos = 1 Ninguno =1 Análisis P. D.=1	UML =5 Booch=1 D. Flujo=1	SÍ 7	NO 1	SÍ 4	NO 3	SÍ 2	NO 5	SÍ 7	NO 0	SÍ 6	NO 1	SÍ 7	NO 0	SÍ 7	NO 0	SÍ 7	NO 0	SÍ 5	NO 2	Falta de experiencia del personal que desarrolla=2 Requerimientos erróneos / pobre =2 Falta de tiempo=2 Mala comunicación=2 Sist. Complejos=1 Reportes erróneos=1 Mala planeación= 1	SÍ 7	NO 0

1.6.4 CONCLUSIONES DEL CUESTIONARIO PILOTO

Por pregunta:

1. ¿Utiliza algún proceso específico para el desarrollo de sistemas?

Esta pregunta no fue del todo comprendida por algunas de las personas entrevistadas, ya que la palabra "proceso" fue muy abierta, e implica una serie de pasos a seguir que pueden variar de acuerdo al sistema que se esté desarrollando, como se observa algunas personas respondieron "orientación a objetos" como un proceso de desarrollo o bien mencionaron los pasos que siguen de acuerdo a su propia metodología. Tres personas del resto de los personas entrevistadas utilizan RUP y otra más TSP, por lo cual concluimos que casi la mitad de las personas coinciden con nuestra idea al considerar a RUP como una opción viable y eficiente en el desarrollo de sistemas.

2. ¿Qué notación utiliza para el análisis y diseño de sistemas?

Cinco de las siete personas entrevistadas utilizan UML, con lo cual reforzamos nuestra idea de que ésta es una de las notaciones que nos permite especificar la estructura o el comportamiento de un sistema, ya que puede ser expresado en diferentes niveles de precisión.

3.¿La notación utilizada le ha permitido una comunicación eficiente con los usuarios?

En esta pregunta observamos que las personas que utilizan UML o la metodología de BOOCH consideran que éste lenguaje de notación les ha permitido una comunicación más eficiente con los usuarios. A diferencia de la persona que utiliza solamente diagramas de flujo.

4.¿La notación utilizada le ha permitido identificar todos los requerimientos de los usuarios?

De cinco personas que utilizan el UML, cuatro de ellas afirman que a través de ésta notación han logrado identificar más fácilmente los requerimientos de los usuarios, y para quienes utilizan Diagramas de Flujo y Booch, no logran identificar con su notación los requerimientos de los usuarios, resultado que apoya nuestra respuesta esperada.

. ¿El proceso utilizado ha influido en la culminación de los sistemas en el tiempo y costos estimados?

En esta pregunta sólo dos personas que utilizan RUP y UML consideran que estos procesos de desarrollo de sistemas influyen de alguna manera en la culminación a tiempo de los sistemas que desarrollan, por lo cual consideramos que en este punto se debería de profundizar más sobre las causas que afectan el tiempo de desarrollo de los sistemas, ya que aunque el proceso de desarrollo empleado sí influye para reducir el tiempo, no es suficiente para lograrlo.

Por pregunta:

6. ¿El proceso de desarrollo que utiliza le proporciona una comprensión global del sistema a modelar?

En esta pregunta todas las personas contestaron afirmativamente, por lo que se intuye que ésta es la razón principal por la cual utilizan el proceso de desarrollo que cada uno mencionó.

7. ¿Cree que el proceso de desarrollo que utiliza es el adecuado para verificar el cumplimiento de la calidad y los requerimientos de los usuarios?

Seis de los siete entrevistados coincidieron en que el proceso de desarrollo que utilizan les es de utilidad para vigilar más la calidad del software que desarrollan y cumplir con todos los requerimientos de los usuarios. La respuesta positiva a esta pregunta coincidió con las personas que utilizan RUP como proceso de desarrollo o bien TSP, por lo que concluimos que RUP es un proceso eficiente y si se lleva a cabo de una manera correcta y controlada se puede obtener un mayor porcentaje de calidad en el software que se desarrolla.

8. La documentación que utiliza refleja, ¿realmente la estructura y el comportamiento del sistema?

Todos los entrevistados contestaron afirmativamente a esta pregunta, por lo que concluimos que el documentar cada una de las fases del desarrollo de un sistema es de vital importancia, ya que trae grandes ventajas a analistas, diseñadores y desarrolladores al proporcionarles un panorama general de la estructura y comportamiento del sistema a crear, entre más detallada y clara sea ésta, mayores beneficios traerá.

9. La documentación que realiza durante el análisis y diseño, ¿le proporciona una eficiente guía de construcción del sistema?

El total de los entrevistados dieron una respuesta afirmativa a esta pregunta, con lo que confirmamos nuestra idea de que la documentación generada durante las fases de desarrollo de un sistema, es una guía muy útil e importante para liberar sistemas con calidad.

10. ¿Considera que la documentación que genera durante la etapa del análisis y diseño comunica eficientemente la funcionalidad del sistema, facilitándole el desarrollo del mismo?

Todos los entrevistados respondieron que *sí*, por lo que concluimos que la documentación generada dentro del análisis y diseño es una herramienta de gran utilidad para visualizar y comunicar correctamente la funcionalidad que debe tener el sistema a desarrollar.

Por pregunta:

11. ¿Considera que la documentación que genera durante el análisis y diseño del sistema es fácil de integrar en la programación?

Seis de las personas entrevistadas contestaron afirmativamente, pues la documentación que generan les ha permitido llevar a cabo la programación de los sistemas de una manera más sencilla. Por el contrario dos de los entrevistados consideran que no siempre es fácil integrar la documentación a la programación, esto nos lleva a la conclusión de que el poder integrar la documentación a la programación tiene sus excepciones, esto se da cuando no se realiza un documentación detallada o que realmente refleje todos los requerimientos de los usuarios.

12. ¿Cuáles son los causas por las que no se lleva a cabo una adecuada documentación en las etapas de análisis y diseño?

Dos de las causas más comunes mencionadas por los entrevistados fueron: la falta de experiencia (en herramientas, metodologías y aplicación de las mismas) y la falta de tiempo. También consideraron la mala planeación de actividades, la complejidad de los sistemas y el análisis de requerimientos pobre, como causas por las que no se crea una documentación eficiente, además de que aún no se le da la importancia que ésta debe tener.

13. ¿Conoce el término de RUP (Rational Unified Process) y UML (Unified Modelimng Language)?

Todas las personas entrevistadas contestaron afirmativamente, por lo que concluimos que el término de RUP y UML está siendo cada vez más difundido entre la comunidad informática. De hecho dos de los entrevistados están comenzando a adoptar ya estas herramientas dentro de sus actividades, y otros dos de ellos tienen ya mayor experiencia con UML.

1.6.5 CONCLUSIÓN GENERAL DEL CUESTIONARIO PILOTO

Respecto a la estructura del cuestionario consideramos que es entendible, ya que ninguno de los entrevistados tuvo problemas con la redacción de las preguntas, sin embargo, respecto a la pregunta uno, tres de los entrevistados no contestaron precisamente cuál es el proceso de desarrollo que utilizan. Se podría mejorar especificando más la pregunta.

Por lo que respecta al resto del cuestionario, se obtuvo la información requerida, ya que la mayoría de los entrevistados coincidió con las respuestas esperadas por nosotras. Los entrevistados que utilizan y/o conocen más acerca de UML y RUP apoyaron la idea que teníamos acerca de que un proceso eficiente y una notación adecuada son de vital importancia para desarrollar software de manera exitosa y con el mayor porcentaje de calidad, a dichas personas les ha funcionado el implantar estas herramientas en sus prácticas laborales, sin embargo, aclaran que si hace falta un poco más de experiencia para poder sacar provecho de todas las ventajas que ofrecen UML y RUP.

Consideramos que la aplicación de este cuestionario fue muy enriquecedor, pues pudimos confirmar la verdadera existencia del problema planteado anteriormente. A través de la información obtenida se constató que las personas que no tienen problemas con la notación empleada para documentar los sistemas es porque utilizan UML en la mayoría de los casos, por otra parte, el proceso de desarrollo más eficiente que se emplea es RUP y TSP.

Algunos de los datos más interesantes que se obtuvieron son las causas por las cuales no se lleva a cabo una adecuada documentación en las etapas de análisis y diseño, entre las cuales destacan: La falta de experiencia en el personal que desarrolla, la escasez de tiempo de desarrollo y la identificación pobre o errónea de los requerimientos de usuarios.

1.7 HIPÓTESIS PRELIMINAR

Análisis de elementos para la composición de la hipótesis preliminar

Variables Independientes	Variables Dependientes
1. *El realizar un análisis y diseño orientado a objetos utilizando una notación que modele al sistema desde una perspectiva clara, global y real*	🗁 Ayuda a visualizar cómo es o queremos que sea un sistema. 🗁 Permite especificar la estructura o el comportamiento de un sistema. 🗁 Proporciona plantillas que nos guían en la construcción de un sistema. 🗁 Documenta las decisiones que hemos adoptado para desarrollar un sistema. 🗁 Ayuda a comprender un problema y dar forma a soluciones eficientes. 🗁 Permite expresar el sistema en diferentes niveles de precisión. 🗁 Coadyuva a crear modelos que estén ligados a la realidad. 🗁 Facilita la identificación de problemas en el modelo. 🗁 Facilita la integración en la etapa de desarrollo. 🗁 Mejora la productividad de desarrolladores. 🗁 Permite realizar mejoras y modificaciones de una manera más sencilla. 🗁 Permite realizar modelos reutilizables dentro de todas las etapas del ciclo de desarrollo del sistema. 🗁 Proporciona flexibilidad en los modelos. 🗁 Permite realizar el mantenimiento de una manera sencilla. 🗁 Minimiza el grado de complejidad. 🗁 Permite representar modelos completos de sistemas de software. 🗁 Permite tener un único lenguaje de comunicación entre usuarios, desarrolladores y computadoras. 🗁 Permite tener una notación entendible y consistente.

Variables Independientes	Variables Dependientes
2. *El usar un proceso de desarrollo iterativo e incremental en el desarrollo de sistemas*	🗁 Ayuda a dar liberaciones tempranas y continuas de aplicaciones. 🗁 Permite crear aplicaciones dinámicas y escalables. 🗁 Coadyuva a liberar los sistemas en el tiempo y con los costos estimados. 🗁 Permite la asignación de tareas y responsabilidades de una forma disciplinada. 🗁 Permite detectar errores e inconsistencias tempranamente. 🗁 Aumenta la productividad del equipo de trabajo. 🗁 Permite una verificación continua de la calidad del software. 🗁 Ayuda a administrar las actividades y responsabilidades. 🗁 Permite el desarrollo de aplicaciones con la funcionalidad requerida por el usuario. 🗁 Representa una guía eficiente para ordenar las actividades de un equipo. 🗁 Ofrece criterios de medición y control de las actividades del proyecto. 🗁 Permite el desarrollo de software repetible, es decir, que pueden estimarse en tiempo y costo.
3. *El contar con un equipo de trabajo capacitado en el desarrollo de un sistema*	🗁 Incrementa la productividad. 🗁 Permite el desarrollo de un producto de software exitoso. 🗁 Facilita la compresión de lo que tiene que hacer cada uno de los miembros del equipo de desarrollo de productos de software. 🗁 Minimiza el tiempo de desarrollo.

1.7.1 CORRELACIONES DE LAS VARIABLES DEPENDIENTES E INDEPENDIENTES PARA LA COMPOSICIÓN DE LA HIPÓTESIS PRELIMINAR

Variable independiente 1:

- *El realizar un análisis y diseño orientado a objetos utilizando una notación que modele al sistema desde una perspectiva clara, global y real,* permite especificar la estructura o el comportamiento del sistema, obteniendo así, plantillas que guíen la construcción del mismo, expresándolo en diferentes niveles de precisión.

- *El realizar un análisis y diseño orientado a objetos utilizando una notación que modele al sistema desde una perspectiva clara, global y real,* facilita la identificación del problema y esto nos ayudará a documentar mejor las decisiones que se adopten durante el desarrollo del mismo.

- *El realizar un análisis y diseño orientado a objetos utilizando una notación que modele al sistema desde una perspectiva clara, global y real,* permite obtener modelos más ligados a la realidad, reduciendo así la complejidad durante la etapa desarrollo, implantación y mantenimiento del modelo, esto se ve reflejado en la productividad de los desarrolladores.

- *El realizar un análisis y diseño orientado a objetos utilizando una notación que modele al sistema desde una perspectiva clara, global y real,* permite construir modelos completos del sistema a desarrollar, basados en un único lenguaje de comunicación entre desarrolladores, usuarios y computadoras.

- *El realizar un análisis y diseño orientado a objetos utilizando una notación que modele al sistema desde una perspectiva clara, global y real,* ayuda a obtener un modelo flexible reutilizable dentro de todas las etapas del ciclo de desarrollo del sistema.

- *El realizar un análisis y diseño orientado a objetos utilizando una notación que modele al sistema desde una perspectiva clara, global y real,* permite obtener una notación expresiva y consistente que influirá en la solución del problema planteado por los usuarios.

Variable independiente 2:

- *El usar un proceso de desarrollo iterativo e incremental en el desarrollo de sistemas,* permitirá crear liberaciones tempranas de aplicaciones dinámicas y escalables en el tiempo y con los costos estimados.

- *El usar un proceso de desarrollo iterativo e incremental en el desarrollo de sistemas,* permite la detección de errores e inconsistencias tempranamente verificando de manera continua la calidad del software con la funcionalidad requerida por el usuario.

- *El usar un proceso de desarrollo iterativo e incremental en el desarrollo de sistemas,* ayuda a administrar las actividades y responsabilidades de forma disciplinada del equipo de trabajo,

contando así con criterios de medición y control de las actividades del proyecto lo que permitirá estimar los costos del mismo.

Variable independiente 3:

- *El contar con un equipo de trabajo capacitado en el desarrollo de un sistema*, se obtiene mayor productividad ya que todo el equipo de desarrollo comprende lo que tiene que hacer para desarrollar un producto de software exitoso en poco tiempo.

1.7.2 HIPÓTESIS PRELIMINAR:

Al emplear una metodología para el análisis y diseño de sistemas orientados a objetos que contemple los elementos clave: proceso RUP, notación UML y equipo de desarrollo capacitado, *permitirá mejorar la comunicación entre el equipo de desarrollo y los usuarios, logrando modelar la estructura y comportamiento del sistema con modelos completos y cercanos a la realidad, además de controlar las responsabilidades del equipo de desarrollo para realizar liberaciones tempranas y continuas que respondan a las necesidades de información de los usuarios, obteniendo así sistemas de información de calidad dentro del presupuesto, tiempo y costo estimados.*

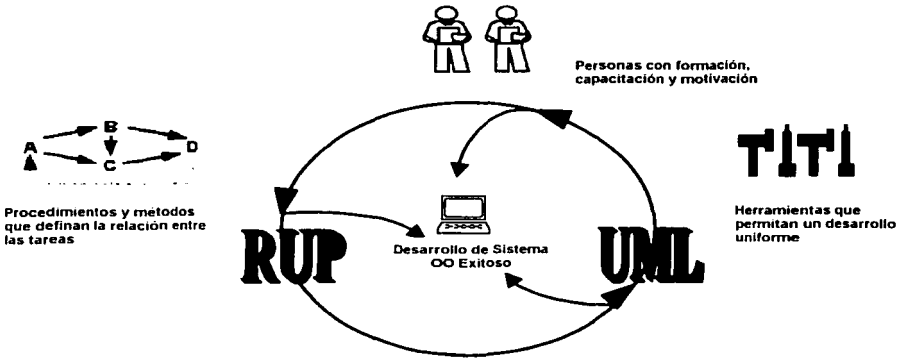


Fig. 1 Personas capacitadas, proceso de desarrollo y notación son elementos clave que permitirán desarrollar un sistema orientado a objetos de calidad que cubra con los requerimientos de los usuarios.
Fuente: Parte de esta imagen fue tomada del material del curso CMM, impartido en la Dirección de Sistemas DGSCA.

1.8 OBJETIVOS PERSONALES Y GENERALES

PERSONALES

- Obtener el título de Licenciado en Informática con opción de Tesis Profesional, de acuerdo al reglamento de Exámenes Profesionales de la Facultad de Contaduría y Administración, en sus artículos 44 a 56, apartado E, el cual está basado en el Reglamento General de Exámenes Profesionales de la Universidad Nacional Autónoma de México.

- Llegar a dominar el paradigma orientadas a objetos para modelar sistemas de información de una manera sencilla y eficiente, que nos permita brindar nuevas soluciones a problemas reales de las organizaciones.

GENERALES

- Aportar a la comunidad informática conocimientos teórico-práctico del modelado de sistemas orientadas a objetos, dando a conocer los conceptos elementales para crear modelos de datos fáciles de comprender, modificar y que permitan identificar problemas para ofrecer soluciones eficientes, sin que esto implique grandes pérdidas de recursos.

- Aportar una nueva visión acerca del proceso de desarrollo de los sistemas orientados a objetos de manera que pueda ser visto como una metodología informática, induciendo a toda persona involucrada en el desarrollo a experimentar los beneficios que ofrece una notación consistente y clara.

ESPECÍFICOS

- Probar la hipótesis que hemos planteado en esta tesis.

- Destacar la importancia que tienen los elementos clave (UML, RUP, quipo de desarrollo eficiente) en el desarrollo de un sistema orientado a objetos.

CAPÍTULO II

2. MARCO TEÓRICO

2.1 ACOPIO BIBLIOGRÁFICO

Para mejor comprensión de los temas "UML" y "RUP", se consultaron diversas fuentes documentales, las cuales se clasifican en:

2.1.1 Lectura total o libros de estudio
2.1.2 Lectura ligera
2.1.3 Lectura rápida
2.1.4 Lectura superficial

LIBROS

2.1.1 Lectura total o libros de estudio

Título: "El lenguaje unificado de modelado"
Autor: Grady Booch, James Rumbaugh, Ivar Jacobson
Editorial: Addison-Wesley, c1999
Edición: Clasific. QA76.76D47 B66418 Dewey 005.1/7 21
ISBN: 84-7829-028-1

DESCRIPCIÓN:

Sección 1: *Introducción*
Describe la importancia del modelado y los principios del mismo, según el paradigma orientado a objetos, ya que el modelado es una parte central de todas las actividades que conducen a la producción de un buen software.

Sección 2: *Modelado estructural básico*
En esta sección se explican los conceptos acerca del modelado con UML como clases, relaciones, diagramas, así como las técnicas comunes de modelado.

Sección 3: *Modelado estructural avanzado*
Se describen las características avanzadas de las clases y las relaciones, muestra además una introducción a conceptos relacionados con interfaces, tipos, roles paquetes e instancias, en esta sección también se muestra a fondo la forma de elaboración del diagrama de objetos.

Sección 4: *Modelado básico del comportamiento*
Se describen términos y conceptos de las interacciones que se dan entre los objetos, así como la elaboración de diagramas de casos de uso, de interacción y de actividades.

Sección 5: *Modelado avanzado del comportamiento*
Se tratan técnicas comunes de modelado de señales y excepciones, dedicando un capítulo al tema de eventos y señales, se describe la técnica de modelado de vida de un objeto en el capítulo denominado "Máquina de estado".

Sección 6: *Modelado arquitectónico*
Se explican algunas técnicas comunes y patrones de modelado para la arquitectura de proyectos de software, como el modelado de bibliotecas, ejecutables, tablas, archivos, a través de la descripción de los diagramas de componentes, despliegue y de colaboración.

Sección 7: *Conclusiones*
Se da una breve descripción de la forma de utilización del Lenguaje de Modelado Unificado, los elementos que integran a este lenguaje, y algunas características de RUP.

Título: "El proceso unificado de desarrollo de software "
Autor: Ivar Jacobson, Grady Booch, James Rumbaugh
Editorial: Addison-Wesley
Edición: 2000
ISBN: 0-201-57169-2

DESCRIPCIÓN:
Parte I *El Proceso de Desarrollo de Software*
Capítulo 1: Describe el proceso unificado de desarrollo de software, se explica que es un proceso dirigido por casos de uso, centrado en la arquitectura, que es iterativo e incremental, que basa gran parte del proyecto de desarrollo en componentes reutilizables, es decir, piezas de software con una interfaz bien definida. Se explica el proceso y cómo se debe hacer uso de UML dentro de él.
Capítulo 2: Es una introducción a las "cuatro P": personas, proyecto, producto y proceso; describe sus relaciones, las cuales son esenciales para la comprensión del resto del libro, además, abarca conceptos básicos como son: artefacto, modelo, trabajador y flujo de trabajo.
Capítulo 3: Trata el concepto de desarrollo dirigido por casos de uso con mayor detalle, ya que los casos de uso son un medio para determinar los requisitos correctos y utilizarlos para conducir el proceso de desarrollo.
Capítulo 4: Describe el papel de la arquitectura en el proceso unificado. La arquitectura establece lo que se tiene que hacer; esquematiza los niveles significativos de organización del software y se centra en el armazón del sistema.
Capítulo 5: Este capítulo enfatiza en la importancia de adoptar una aproximación iterativa e incremental en el desarrollo de software, lo que como se verá, permitirá minimizar riesgos de fracasos al desarrollar productos de software.

Parte II *Los Flujos de Trabajo fundamentales*
Capítulo 6: Describe uno de los flujos de trabajo más importantes que se realizan durante el proceso de desarrollo de software, como lo es la captura de requisitos y la visión general del modelo.
Capítulo 7: Describe la captura de requisitos con casos de uso, explica cómo a través de esta actividad se pueden determinar los actores del sistema, para ir detallando la estructura y funcionalidad del sistema.
Capítulo 8: Describe el flujo de trabajo de análisis, su importancia dentro del proceso y explica por qué este flujo no puede equipararse con ningún otro flujo.
Capítulo 9: Describe algunas de las actividades que se realizan en el flujo del diseño.
Capítulo 10: Explica cuál es el papel de la implantación en el ciclo de vida del software, también describe la realización de pruebas durante esta etapa.
Capítulo 11: Describe las actividades necesarias que se realizan durante el flujo de trabajo de pruebas.

explicando su procedimiento, los planes, componentes y evaluación con mayor detalle.

Parte III *El Desarrollo Iterativo e incremental*
Capítulo 12: Trata sobre el equilibrio que debe existir en el proceso, de manera que no se descuide ninguna parte del mismo; menciona cómo cada fase, durante el proceso de desarrollo, es la primera división del trabajo, además de la planificación y cómo influyen los riesgos en el proyecto, por lo que recomiendan evaluar las iteraciones y las fases.
Capítulo 13: Aborda la fase de inicio, la cual pone en marcha el proyecto, en ésta se expone el establecimiento de los criterios de evaluación y el ajuste del proyecto al entorno de desarrollo.

Capítulo 14: Describe la fase de elaboración, la cual constituye la base de la arquitectura del producto de software, en ésta se menciona la forma en la que se recopilan y refina la mayor parte de los requisitos.
Capítulo 15: Explica la fase de construcción, en la cual se asigna al personal y los criterios de evaluación, del mismo modo se explica la planeación de la próxima fase denominada transición.
Capítulo 16: Trata la última fase del proceso unificado racional de desarrollo, llamada transición, cómo se establecen los criterios de evaluación y la asignación del personal para esta fase.
Capítulo 17: Describe cómo hacer para que el proceso unificado funcione y cómo éste ayuda a manejar la complejidad de los proyectos para obtener los beneficios del proceso unificado.

Título: Developing software with UML Object-Oriented Analysis and Design in Practice (Desarrollo de software con UML, Análisis y Diseño Orientado a Objetos en Práctica)
Autor: Oestereich, Bernard
Editorial: Addison Wesley
Edición: 1999
ISBN: 0-201-36826-5

DESCRIPCIÓN:

CAPÍTULO 1:
En este capitulo se explican las características esenciales del desarrollo orientado a objetos de software, se explica un poco de la historia de la orientación a objetos y sus diferencias con otras metodologías.

CAPÍTULO 2:
Este capitulo proporciona una introducción a los conceptos de la orientación a objetos, como son: clases, objetos, instancias, atributos, operaciones, herencia, clases abstractas, tarjetas CRC, comunicación entre los objetos, mensajes, polimorfismo, persistencia, entre otros.

CAPÍTULO 3:
Este capitulo trata el tema del proceso de desarrollo, se describen brevemente las fases del proceso, del desarrollo iterativo e incremental, trata acerca del análisis de requerimientos, del análisis de dominio del problema y temas de aspectos generales de los sistemas, así como la administración de proyectos.

CAPÍTULO 4:
Este capitulo discute acerca del análisis orientado a objetos, en él se muestra un ejemplo donde se explican aspectos interesantes como: casos de uso, las tarjetas CRC, identificación de clases, entre otros.

CAPÍTULO 5:
Este capitulo trata especificamente la etapa de diseño, aquí se desarrolla un ejemplo. En esta sección se discuten temas como: diseño de componentes, identificación de clases y relaciones, especificación de operaciones, de atributos, modelado de actividades, modelado de estado, interacción entre objetos, ademas de una conexión con la bases de datos.

CAPÍTULO 6:
Este capitulo muestra a detalle todos los elementos de los casos de uso con UML, se explica acerca de los actores de los casos de uso y se muestran ejemplos de cada uno de ellos.

CAPÍTULO 7:
Este capitulo explican los elementos básicos para elaborar los diagramas de clase con UML, se estudian conceptos como: clases, objetos, atributos, métodos, interfaces, interfaces de clase, restricciones, estereotipos, notas, paquetes, entre otros. Para cada uno de estos conceptos se muestran ejemplos muy sencillos y entendibles.

CAPÍTULO 8:
Este capitulo ahonda conceptos muy importantes, necesarios para el diseño de diagramas de clase. En esta sección se muestra la definición, descripción, notación y ejemplos acerca de la generalización, especializacion, asociación, agregación, dependencia de relaciones y refinamiento de relaciones.

CAPÍTULO 9:
En esta seccion se explican a detalle los elementos de UML necesarios para representar los aspectos dinámicos del sistema. es decir, diseñar los diagramas de comportamiento. Se explican los diagramas de actividad, los diagramas de colaboración, los diagramas de secuencia y los diagramas de estado; para cada uno de ellos se da su definición, su descripción, su notación y un ejemplo.

CAPÍTULO 10:
Este capítulo describe los elementos UML usados para representar la parte de implantación del sistema, por lo que se estudian los diagramas de componentes y los diagramas de despliegue.
CAPÍTULO 11:
En este capítulo se explican los elementos más importantes del Object Constraint Lenguaje (OCL), lenguaje de restricción de objetos.

2.1.2 Lectura Ligera

Libro	Descripción
Título: Fundamental of Object-Oriented Design in UML (Diseño Fundamental Orientado a Objetos en UML) ***Autor*** :Page-Jones, Meiler ***Editorial:*** :Addison-Wesley ***Edición:*** 2000 **ISBN:** 0-201-69946-X	Capítulo 2: Es una descripción de la historia del paradigma orientado a objetos, se explican los orígenes, así como sus precursores, entre los que están: Bertrand Meyer, Grandy Booch e Ivar Jacobson. Capítulo 3: En este capítulo se describe la notación UML, explicando algunas de sus características y los términos más utilizados, mostrando así las ventajas de éste lenguaje en el modelado de sistemas. Capítulo 4: En este capítulo se proporciona una introducción a los diagramas de clase, en el cual se muestran tres modelos importantes de construcción de software orientado a objetos, así como las herencias entre las estructuras y los tipos de asociaciones que se dan entre las clases. Capítulo 5: Describe la parte dinámica del modelado de sistemas que propone UML, la cual posee una profunda relación con la parte estática del modelado, pero en ésta se muestra la interacción entre los elementos del sistema, en donde nos proporcionan una idea de cómo se comportaría el sistema cuando esté funcionando. Capítulo 6: En este capítulo se describen principalmente los diagramas de estados, que muestra el comportamiento de un objeto o una clase dada, además se describe la utilidad de este diagrama, ya que de aquí se pueden identificar los principales eventos y transiciones de los mismos. Capítulo 7: Describe, a través de conceptos, el diseño orientado a objetos de un sistema desde el punto de vista de la arquitectura de hardware, software, así como las interfaces con el usuario.
Título: Real-Time UML: Developing Efficent Objects from Embedded Systems (UML en Tiempo Real: Desarrollo Eficiente de Objetos a partir de Sistemas Acoplados) ***Autor***: Douglas, Bruce Powel ***Editorial***: Addison-Wesley ***Edición:*** 2000 ***ISBN:*** 0-201-65784-8	Capítulo 1: Presenta una explicación de la orientación a objetos con UML, el capítulo estudia de manera general los conceptos básicos de la orientación a objetos, así como los diagramas y la notación de UML. Capítulo 2: Es un estudio acerca del análisis de requerimientos de un sistema en tiempo real, en este capítulo se estudió la forma en que se realizan los casos de uso y los escenarios para obtener los requerimientos de un sistema. También se estudiaron los diagramas de secuencia.

	Capítulo 3: Contiene más acerca de los casos de uso, además de estrategias para la identificación de clases, esto para realizar los diagramas de clase.
Título: Patterns in Java (Patrones en Java) ***Autor:*** Grand, Mark ***Editorial:*** John Wiley ***Edición:*** 1999 ***ISBN:*** 0-471-25841-5	Capítulo 2: Es una descripción de UML, en este capítulo se consultaron los diagramas de clase, interfaces y los diagramas de colaboración. Capítulo 3: Realiza un estudio de los casos de uso, describe cómo hacer la especificación de los requerimientos, y de manera muy breve explica el análisis y diseño orientado a objetos.
Título: Applying UML and Patterns, an introduction to Object Oriented Analysis and Design (Aplicando UML y Patrones, una introducción al Análisis y Diseño Orientado a Objetos) ***Autor:***Craig Larman ***Editorial:*** Prentice Hall PTR ***Edición:*** 1998 ***ISBN:*** 0-13-748880-7	Capítulo 1: Trata acerca de la importancia de UML y del proceso de desarrollo dentro del ADOO. Capítulo 2: Estudia acerca del proceso de desarrollo, menciona cómo es que interactúa UML dentro de este proceso y se explica qué actividades deben realizarse dentro de cada una de las fases del proceso.
Título: Using UML, software engineering with Objects and Components (Usando UML, ingeniería de software con Objetos y Componentes) ***Autor:*** Stevens, Perdita, Pooley, Rob ***Editorial:*** Addison Wesley ***Edición:*** 1999 ***ISBN:*** 0-21-64860-1	Capítulo 1: Contiene algunos aspectos importantes acerca de la calidad de los sistemas, se analizaron cuestiones como: ¿Qué es un buen sistema?, ¿Cómo desarrollar un buen sistema?. Capítulo 2: Se estudiaron algunos conceptos interesantes acerca de los objetos, su definición, ejemplos de objetos, mensajes, interfaces, clases, herencia, polimorfismo. Capítulo 3: Es una introducción a los casos de uso, se habla acerca de clarificar los requerimientos, entender el problema real a resolver, se estudia el modelo de casos de uso, el diagrama de clases, relaciones entre clases. Capítulo 4: Trata acerca del proceso de desarrollo, en esta sección se definen algunos términos y se explica cómo puede usarse UML dentro del proceso. Capítulo 5: Es un estudio del diagrama de clase, en él se estudian aspectos como: identificación de clases y objetos, asociación, atributos y operaciones, generalización, tarjetas CRC. Capítulo 6: Es una extensión del capítulo 5, pues se habla de otros aspectos importantes dentro de los diagramas de clase, se trata más a fondo la agregación, las clases, estereotipos.

	Capítulo 7 y 8: Realizan un estudio más profundo acerca de los casos de uso. Capítulo 9 y 10: Tratan acerca de los diagramas de interacción. Capítulo 11 y 12: Estudian acerca de los diagramas de estado y de actividad.
Título: UML y Patrones: Introducción al Análisis y Diseño Orientado a Objetos ***Autor:*** Larman, Craig ***Edición:*** 1999 ***ISBN:*** 970-17-0261-1	Capítulo 1: Trata acerca de UML aplicado al análisis y diseño orientado a objetos. Explica el concepto de análisis y de diseño, primero en su forma general y posteriormente desde la perspectiva orientada a objetos. Además se explican brevemente los casos de uso y los diagramas de clase. Capítulo 2: Es una introducción al proceso de desarrollo y se explica cómo interactúa UML dentro de éste mismo.
Título: "Developing applications with Visual Basic and UML" (Desarrollando aplicaciones con Visual Basic y UML) ***Autor*** : Paul R. Reed, Jr. ***Editorial:*** :Addison-Wesley ***Edición:*** 2000 ***ISBN:*** 2-201-61579-7	Capítulo 1: Explica el desarrollo iterativo e incremental del software, los riesgos del desarrollo, el modelo del proceso iterativo de software, se explica la combinación del concepto iterativo e incremental. También estudia el concepto de UML, sus diagramas, UML dentro del proceso de desarrollo, UML dentro de la arquitectura 4+1. Capítulo 2: Es una breve descripción del diagrama de clases, así como del diagrama de componentes. Capítulo 3: Estudia acerca del modelo de proceso, además de una breve explicación acerca de los actores. Capítulo 4: Este capítulo trata acerca de los casos de uso. Capítulo 5: Realiza un estudio acerca de las clases entity (entidad), de interfaz y control. Además de un estudio acerca las relaciones (de asociación, agregación y generalización). Explica cómo crear clases, cómo identificar atributos y operaciones. Aunado a esto explica la fase de elaboración del proceso de desarrollo. Capítulo 7: Explica brevemente los diagramas de secuencia, de colaboración, de transición de estados y de actividad. Capítulo 9: Explica la fase de construcción del proceso de desarrollo. Cómo realizar el diseño físico, cómo mapear las clases y tablas.

Título: Object oriented systems development (Desarrollo de sistemas orientados a objetos) **Autor:** Ali Bahrami ***Editorial:*** Mc Graw - Hill ***Edición:*** 1999 ***ISBN:*** 0-256-25348-X **Ubicación:** Biblioteca Central UNAM QA76.9S88 B34	Capítulo 1: Este capitulo provee de una apreciación global del desarrollo de sistemas orientados a objetos y muestra un estudio del "Acercamiento Unificado", el cual es una metodología utilizada en este libro para aprender acerca del desarrollo de sistemas orientados a objetos. Capítulo 2: Describe conceptos básicos para entender el paradigma orientado a objetos como: objeto, clase, atributo, mensaje, herencia, relaciones, encapsulación, poliformismo, abstracción, persistencia y meta clase, entre otros. Capítulo 3: Explica el ciclo de vida de desarrollo de sistemas orientados a objetos, lo que en esencia es el proceso de desarrollo, que consiste en las etapas de análisis, diseño, pruebas y refinamiento, transformando asi las necesidades de los usuarios en un producto de software. Capítulo 4: En este capitulo se estudian algunas de las metodologías bien conocidas de orientación a objetos, asi como las técnicas emergentes que pueden ser utilizadas como patrones o marcos de trabajo. Capítulo 5: Este capitulo define los conceptos referentes al Lenguaje Unificado de Modelado, mostrando los diferentes diagramas que se utilizan en el mismo y los beneficios obtenidos al ocupar este lenguaje. Capítulo 13: En esta sección se describe lo que implica realizar un software de calidad desarrollando pruebas para los diferentes casos del proceso y para los planes a realizar en el mismo.
Título: Object oriented methods (Métodos orientados a objetos) **Autor:** Ian Graham ***Editorial:*** Addison - Wesley ***Edición:*** 1996 ***ISBN:*** 0-201-56521-8 **Ubicación:** Biblioteca Central UNAM QA76.64 G73	Capítulo 4: "Aplicaciones". En este capitulo se examinan algunas de las aplicaciones de los sistemas orientados a objetos y las caracteristicas que hacen que las aplicaciones sean adecuadas, también se listan algunas preguntas a las que se enfrentan los administradores de proyectos que utilizan lenguajes de programación orientados a objetos. Capítulo 7: "Diseño Orientado a Objetos". En este capitulo se examina la utilidad e inconvenientes de varios métodos de diseño y análisis orientados a objetos que han sido propuestos y se sugiere un conjunto de requisitos para una técnica de análisis práctica. Capítulo 8: "Análisis orientado a objetos". En esta sección se examina la ingeniería de software en general, detallando

	cada uno de los métodos de análisis orientados a objetos existentes, además de una descripción de las técnicas CASE y los estándares emergentes para el análisis y diseño orientado a objetos. Capítulo 10: "El futuro de los métodos orientados a objetos". En este capítulo se examina de forma amplia el futuro inmediato de la ingeniería de software, abarcando los desarrollos de software esperados, los métodos, hardware y direcciones de investigación.
Título: Análisis y diseño orientado a objetos con aplicaciones **Autor:** Grandy Booch ***Editorial:*** Addison - Wesley ***Edición:*** 1996 ***ISBN:*** 0-201-60122-2 **Ubicación:** Biblioteca Central UNAM QA76.64 B6618	**Sección 1:** *Conceptos* Capítulo 1: "Complejidad". En esta parte del libro se explica cómo el diseño orientado a objetos es el método que conduce a una descomposición del sistema a desarrollar, además de que define una notación y un proceso para construir sistemas de software complejos, sin embargo, existen factores de limitación fundamentales a los cuales se les hace frente mediante la abstracción y la jerarquía. Capítulo 2: "El modelo de objetos". Describe cómo la tecnología orientada a objetos se apoya en los sólidos fundamentos de la ingeniería, cuyos elementos reciben el nombre de "modelo de objetos", el cual abarca los principios de abstracción, encapsulación, modularidad, jerarquía, concurrencia y persistencia. Capítulo 3: "Clases y objetos". Explica cómo y cuándo se deben utilizar los métodos orientados a objetos para analizar o diseñar un sistema de software complejo, los bloques básicos que lo componen son las clases y los objetos, por lo que muestra un estudio detallado de éstos conceptos junto con sus relaciones. Capítulo 4: "Clasificación". Muestra la importancia de la clasificación dentro del diseño orientado a objetos, ya que el reconocimiento de la similitud entre las cosas nos permiten conocer lo que tienen en común. **Sección 2:** *El Método* Capítulo 5: "La notación". Describe cómo al contar con una notación en el desarrollo de un sistema orientado a objetos, logramos tener una visión y los detalles de la arquitectura, el disponer de una notación expresiva y bien definida provee de muchos beneficios para el proceso de desarrollo de software. Capítulo 6: "El proceso". Describe cómo el proceso de

	desarrollo de software no puede ser explicado mediante una receta mágica, aunque está lo bastante bien definido como para ofrecer un proceso predecible y repetible para una organización de desarrollo de software madura. Capitulo 7: "Aspectos pragmáticos". Muestra que no importa lo sofisticado que sea el método de desarrollo, y no importa lo bien fundamentado de sus bases teóricas, ya que no pueden ignorarse los aspectos prácticos del diseño de sistemas para el mundo real, lo que significa que hay que considerar buenas prácticas de gestión del proyecto como son: administración de personal, gestión de versiones y control de calidad.
Titulo: Business Modeling with UML Business Patterns at work (Modelando negocios con UML, patrones de negocio en el trabajo) ***Autor:*** Hans - Erik Eriksson Magnus Penker OMG Press ***Editorial:*** Willey Computer Publishing John Wiley & Sons, Inc. ***Edición:*** 2000 ***ISBN:*** 0-471-29551-5	Capitulo 1: "Modelado de negocio o casos de uso". Presenta los conceptos y propósitos del modelado de negocio. También provee los argumentos del por qué UML es adecuado para modelar sistemas orientados a objetos y qué elementos son requeridos en UML para realizarlo. Capitulo 2: "Primero UML". Muestra un resumen sobre lo que es el lenguaje de UML, sin embargo hay algunas secciones importantes sobre diagramas de actividad, prototipos y extensiones de UML que son relevantes para modelar casos de uso. Capitulo 3: "Modelando la arquitectura del sistema". Define de la mejor manera los conceptos usados en el modelado de negocio como: procesos, metas, recursos y reglas, además muestra una introducción a las extensiones del lenguaje que definen el uso de estándares, extensiones y mecanismos de UML que facilitan el modelado de los sistemas. Capitulo 4: "Vistas de negocios". Define las diferentes vistas de casos de uso que son utilizadas al modelar un sistema. Por lo que se presentan técnicas y diagramas usados de manera especifica explicando todo esto a través de ejemplos. Capitulo 5: "Reglas de negocio". Describe cómo son definidas las reglas de negocio usando OCL (Object Constraint Lenguage), el cual forma parte del estándar de UML. En esta sección se presentan ejemplos mostrando cómo se usan las diferentes categorias de reglas, así como las técnicas para definir reglas. Capitulo 10: "Arquitectura del negocio y arquitectura del software". Muestra discusiones de cómo los modelos de negocio o caso de uso pueden ser usados para producir

	información sobre el sistema, ya que a través de los casos de uso se pueden definir los requerimientos y la arquitectura, además de toda la variedad de información necesaria para integrar más fácilmente el sistema a desarrollar. Capitulo 11: "Ejemplo de un modelo de negocio". En esta sección se muestran las técnicas y notaciones definidas a través de todo el libro y cómo han sido utilizadas para modelar los ejemplos prácticos. Las vistas de negocios, extensiones de negocios, reglas de negocios, patrones de negocio que fueron mostrados en cada caso de estudio.

2.1.3 Lectura Rápida

Libro	Descripción
Titulo: Objects, components, and Frameworks with UML (Objetos, Componentes y Frameworks con UML) ***Autor:*** Desemond Francis D' Souza Alan Cameron Wills ***Editorial:*** Addison – Wesley ***Edición:*** 1999 ***ISBN:*** 0-201-31012-0	Capitulo 1: En este capitulo se proporciona una introducción al desarrollo de componentes con UML y conceptos básicos del modelado de casos de uso, el diseño de componentes, el proceso de desarrollo, la especificación de requerimientos en un modelo. Capitulo 5: Describe la facilidad que se tiene al realizar documentación eficiente con el Lenguaje Unificado de Modelado, lo cual permite reducir el diseño y desarrollo del sistema a desarrollar. Se explica cómo se documentan los casos de negocio. Capitulo 12: Aqui encontramos qué es la arquitectura en un modelo de sistema orientado a objetos, por qué es importante modelar este aspecto en un sistema, cómo se pueden evaluar los diferentes escenarios para modelar la arquitectura, así como la definición de los elementos de construcción. Capitulo 13: En esta parte del libro se muestra un resumen del proceso de desarrollo de sistemas orientados a objetos en donde se tratan conceptos acerca del modelado, implantación y pruebas.
Titulo: Java modeling color with UML (Modelado de Java a color con UML) ***Autor:*** Coad, Peter; Lefebvre, Eric; De Luca, Jeff ***Editorial:*** Prentice hall ***Edición:*** 1999	Este libro contiene varios ejemplos de diagramas de clase y diagramas de secuencia, además se consultaron las ventajas de UML.

ISBN: 0-13-011510-X	
Título: Object – oriented systems análisis and design (Análisis y diseño de sistemas orientados a objetos) **Autor:** Ronald J. Norman ***Editorial:*** Prentice Hall ***Edición:*** 1996 ***ISBN:*** 013122946X ***Ubicación:*** Biblioteca Central UNAM QA76.64 N67	**Parte I** Capítulo 3: Define y describe algunas metodologías tradicionales y estructuradas, información de modelado, así como la clasificación de metodologías orientadas a objetos. **Parte II** Capítulo 8: Contiene información acerca de los antecedentes del diseño de sistemas orientados a objetos, las principales actividades que se realizan, marcando la diferencia entre la transformación y refinamiento del diseño, además hace énfasis en los diferentes tipos de documentación de sistemas. **Parte III** Sección D: Describe el mejoramiento del proceso de software, marcando la diferencia entre una organización de desarrollo de software madura con una inmadura basándose en los cinco niveles del CMM. Sección F: Trata acerca de los recursos necesarios para la administración de proyectos y define varias fallas que suelen cometerse durante la administración de un proyecto de software, muestra la importancia de la utilización de las gráficas PERT y GANTT.
Título: Enterprise Modeling with UML Designing Successful Software Through Business Analysis (Modelado de negocios con UML diseñando exitosamente software a través del análisis de negocio) ***Autor:*** Chris Marshall ***Editorial:*** Adison Wesley ***Edición:*** 2000 ***ISBN:*** 0-201-43313-3	Capítulo 1: Describe la administración y organización de conceptos acerca del modelado de un sistema orientado a objetos, además se especifican conceptos acerca de los procesos, las entidades y lo que provee el contexto necesario para poder modelar sistemas con eficiencia. Capítulo 3: Explica los casos de uso y sus principales objetivos; el proceso de diseño, el esquema y la presentación. Contiene los conceptos, herramientas y los componentes compatibles dentro del proceso de reingeniería. Capítulo 4: En este capítulo se describe cómo pueden ser modeladas las entidades desde varios puntos de vista, lo que permite describir diferentes aspectos de comportamiento que son requeridos durante el proceso de diseño.

2.1.4 Lectura Superficial

Libro	Descripción
Título: The object constraint language : precise modeling with UML (Restricciones del Lenguaje de Objetos: Modelado exacto con UML) **Autor:** Warmer, Jos B. , Anneke G. Kleppe ***Editorial:*** Addison Wesley Longman, c1999 ***Edición:*** 1999 ***ISBN:*** 0201379406	De este libro retomamos conceptos de los capítulos 1, 2 y 4, en los cuales se da una introducción sobre constraints, su diseño y las ventajas de utilizarlos, ya que permiten mejorar la precisión de la documentación. En el capítulo 2 encontramos un ejemplo de aplicación del OCL (Object, Constraint Language), y por último en el capítulo 4 se explica cómo se pueden modelar los constraints (restricciones), de acuerdo al lenguaje UML, mostrando los casos en los que pueden ser usadas las expresiones de OCL.
Título: Análisis y diseño orientado a objetos **Autor:** James Martin y J. Odell ***Editorial:*** Prentice Hall ***Edición:*** 1994 ***ISBN:*** 968-880-362-6 **Ubicación:** Biblioteca Central UNAM QA76.64 M3718	En este libro se consultaron diversos conceptos relacionados con el paradigma orientado a objetos, así como aplicaciones teóricas y prácticas del análisis y diseño de sistemas.
Título: "Diccionario de la Lengua Española Esencial" ***Editorial:*** LAORUSSE ***Edición:*** 1995 ***ISBN:*** 970-607-425-2	Consulta de la definición de algunos conceptos.
Título: "Diccionario Práctico de Sinónimos y Antónimos" ***Autor:*** Fernando Corripio ***Editorial*** : LAORUSSE ***Edición:*** 1995 ***ISBN*** .: 970-607-127-X	Consulta de sinónimos de algunas palabras.
Título: Ensayo de un diccionario Español de Sinonimos y Antónimos ***Autor:*** Sainz de Robles y Federico Carlos ***Editorial*** : Aguilar ***Edición:*** 1971	Consulta de sinónimos de algunas palabras.
Título: Diccionario Etimológico Español e Hispánico Etimológico Español de la Real Academia Española ***Autor:*** Vicente García de Diego ***Editorial*** : S.A.E.T.A. ***Edición:*** 1990	Consulta de la definición de algunos conceptos.

Título: Enciclopedia abreviada de ordenadores ***Autor:*** Philip B. Jordain ***Editorial*** : Aguilar ***Edición:*** 1976 ***ISBN*** : 84-03-19058-1	Consulta de la definición de algunos conceptos.
Título: Diccionario de Informática ***Autor:*** Anthony Chandor ***Editorial*** : Alianza Editorial ***Edición:*** 1989 ***ISBN*** : 84-206-5235-0	Consulta de la definición de algunos conceptos.
Título: Real Academia Española Diccionario de la Lengua Española Tomo 1 y 2 ***Editorial*** : 21° Edición ***Edición:*** Madrid 1992 ***ISBN*** : 84-239-9200-4	Consulta de la definición de algunos conceptos.
Título: Diccionario General de Etimología de la Lengua Española Tomo 1 y 2 ***Autor:*** D. Roque Barcia ***Edición:*** Madrid 1881	Consulta de la definición de algunos conceptos.

2.2 TESIS

Tesis	Descripción
Título: Documentación del análisis y diseño de sistemas orientados a objetos con UML ***Autor:*** Jiménez Herrada, Jenny ***Fecha:*** 2001 ***Carrera:*** Lic. en Informática ***Facultad:*** Facultad de Contaduría y Administración	Esta tesis se enfoca en la importancia de la documentación del análisis y diseño de sistemas, la autora propone que UML es un lenguaje muy útil para generar toda la documentación necesaria, la cual resulta clara, fácil de modificar y permite identificar problemas de una manera sencilla. Contiene 6 capítulos: Marco problemático, teórico, conceptual, metodológico, instrumental y conclusiones. Contiene una amplia bibliografía acerca de libros, revistas e Internet con información de UML.
Título: Aplicación de la tecnología orientada a objetos: el lenguaje de modelado unificado (UML) y el proceso unificado de desarrollo de software en un caso de estudio ***Autor:*** Basto Aguilar, Eddie David ***Fecha:*** 2001 ***Carrera:*** Maestría en Ciencias de la Computación ***Facultad:*** Facultad de Ingeniería	Esta tesis contiene, en su primer capítulo, un panorama general acerca del proceso de desarrollo orientado a objetos, se define la tecnología OO y UML. Se definen los conceptos para la formación del equipo de desarrollo y la planeación de las tareas. En otro de sus capítulos contiene un caso práctico, donde realiza una abstracción del proceso de desarrollo del proyecto SINSAS, donde aplica todo lo explicado en esta

	tesis.
Título: Una guía práctica para la evaluación de plataformas de desarrollo de software orientado a objetos ***Autor:*** Francisco Alejo Ríos Gascon ***Fecha:*** 1995 ***Carrera:*** Maestría en Ciencias de la Computación ***Facultad:*** Colegio de Ciencias y Humanidades - UNAM ***Ubicación:*** Biblioteca Central 001 –0321-P4-1995	Se consultaron los capítulos I, II y V, el primero trata acerca del enfoque orientado a objetos contra la descomposición funcional del software, así como las características deseables que debe poseer. En el segundo capítulo se describen las características de las herramientas CASE y los aspectos indispensables que deben existir en un ambiente de desarrollo orientado a objetos. Por último, en el capítulo V, se definen los parámetros de medición de una plataforma de desarrollo de software orientada a objetos en cuanto a implantación de los esquemas de persistencia de los objetos, desarrollo de interfaz gráfica y evaluación de los costos que se generan.
Título: La metodología de orientación a objetos como una nueva herramienta para el análisis y diseño ***Autor:*** Adriana Santos Rosas ***Fecha:*** 1994 ***Carrera:*** Ingeniería en Computación ***Facultad:*** Facultad de Ingeniería UNAM ***Ubicación:*** Biblioteca Central 01-41132-S2-1994	Esta tesis la encontramos interesante, ya que en el capítulo I se exponen los antecedentes y las características de los métodos de programación más populares y usados en la actualidad, además de incluir algunas definiciones acerca de la programación orientada a objetos. En el capítulo V se presenta una visión panorámica de la situación de la programación orientada a objetos en nuestro país, las perspectivas en el mercado, la problemática que existe en nuestra comunidad para la aceptación de esta innovación, mostrando las ventajas y desventajas.
Título: Propuesta de políticas, normas y procedimientos para la elaboración de sistemas de información con orientación a objetos bajo la norma ISO9000 ***Autor:*** Gustavo Adolfo de la Cruz Garces y Angel César Govantes Saldivar ***Fecha:*** 1997 ***Carrera:*** Ingeniería en Computación ***Facultad:*** Facultad de Ingeniería UNAM ***Ubicación:*** Biblioteca Central 001-01132-C8-1997	De esta tesis consultamos principalmente las normas en ISO 9000 para el análisis, diseño y documentación de un sistema orientado a objetos. En el capítulo I se presentan conceptos básicos sobre políticas, normas y procedimientos, así como los beneficios de las mismas. En el capítulo II se describen las normas en ISO 9000 que se aplican al desarrollo del software. El capítulo IV analiza la cuestión del por qué del paradigma orientado a objetos, realizando una comparación entre las metodologías tradicionales y las orientadas a objetos, mencionando las características y beneficios.

2.3 REVISTAS

DATOS DE LA REVISTA
Título: Computación y Sistemas Revista Iberoamericana de Computación Publicación Trimestral **Periodo revisado:** 1999 al 2001

Artículos localizados y leídos	Descripción
Artículo: Especificación de temporalidad en entornos automáticos de producción de software a partir de modelos conceptuales objetuales ***Autor:*** Oscar Pastor y Carlos Menesses ***Año:*** 2001 ***No.*** 4 ***Vol.*** 4 ***Fecha:*** Abril - Junio ***Ubicación:*** Biblioteca DGSCA-UNAM	El contexto principal de este artículo se basa en cómo representar de manera adecuada los aspectos estáticos y dinámicos de un sistema, acorde con las necesidades de modelado del mundo real, ya que es importante incluir especificaciones que permitan incluir un modelo funcionalmente equivalente al modelo conceptual del sistema estudiado. La idea de "métodos para la especificación de las transacciones en fase de modelado conceptual en ambientes orientados a objetos", se basa en presentar cómo se incluye adecuadamente, en métodos orientados a objetos, las propiedades que permitan capturar la expresividad temporal de modelos conceptuales, preservando la generación automática del correspondiente producto final de software.
Artículo: Extensiones al sistema de clasificación de UML ***Autor:*** José A. Troyano, Manuel Mejías y Jesús Torres ***Año:*** 2000 ***No.*** 3 ***Vol.*** 3 ***Fecha:*** Enero - Marzo ***Ubicación:*** Biblioteca DGSCA-UNAM	Se plantea cómo extender las capacidades taxonómicas del modelo con UML, proponiendo afinar un poco más la semántica que propone este lenguaje al mecanismo denominado "generalización" y a su mecanismo inverso denominado "especialización". Se define a la generalización en UML, como un mecanismo abstracto de ordenación de clases que permite factorizar los elementos comunes de varias clases en una clase más general, sin embargo, la idea de este artículo vincula la factorización a la idea de taxonomía, de manera que el conjunto de objetos de las clases especializadas construyan subconjuntos de la superclase. Por lo que se exponen criterios de clasificación que permitan decidir qué objetos de la clase cumplen las condiciones de cada especialización.

DATOS DE LA REVISTA
Titulo: The journal object oriented programming (El periódico de programación orientada a objetos)(JOOP) **Periodo revisado:** 1998 al 2001

Artículos localizados y leídos	Descripción
Artículo: OPEN and RUP: How do they compare? (OPEN y RUP: ¿Cómo compararlos? ***Autor:*** B. Henderson – sellers, G. Collins y I. Graham ***Año:*** 2000 ***No.*** 4 ***Vol.*** 13 ***Fecha:*** Julio – Agosto ***Ubicación:*** Biblioteca DGSCA-UNAM	En este articulo se presenta la polémica acerca del comportamiento y adaptación racional para el marco de trabajo del desarrollo de sistemas y cómo puede ser éste comparado con el suministro de un proceso orientado a objetos. Adicionalmente, se muestra cómo existen diferentes procesos orientados a objetos y cómo todas las transiciones y elaboraciones que ofrecen estas dos metodologías RUP (Proceso Unificado Racional) y OPEN (Proceso del Ambiente Orientado a Objetos) y podrían ser generadas como un sólo marco de trabajo, conocido como "proceso orientado a objetos", tanto en ambiente como en la notación. Además este artículo explica cómo ha aumentado el interés en RUP, el cual representa una metodología tan flexible como la metodología que ofrece OPEN, por lo que se muestra una comparación entre ambas.
Artículo: Traceability Management from Business Processes to Use Cases with UML (Administración fácil de procesos de negocio para casos de uso con UML) ***Autor:*** Birol Berkem ***Año:*** 1999 ***No.*** 5 ***Vol.*** 12 ***Fecha:*** Septiembre ***Ubicación:*** Biblioteca DGSCA-UNAM	En este artículo se muestra una propuesta para las extensiones de UML, en específico para los diagramas de actividad. La meta principal de este artículo es evaluar, la aplicabilidad de los conceptos para negocios en los diagramas de actividad del Lenguaje Unificado de Modelado, así como las necesidades de modelado, para definir con facilidad y formalidad los procesos de negocio, para los casos de uso, enfocados principalmente hacia los componentes de software. La robustez, las pruebas y la ejecución de las especificaciones, aparecerán como resultados directos de estas extensiones que se proponen al desarrollar un sistema orientado a objetos.
Artículo: Extended Specification of Composite Objects in UML (La Especificación Extendida de Objetos Compuestos en UML) ***Autor:*** S.Vauttier, M. Magnan, C. Oussalah ***Fecha:*** Mayo 1999 ***Ubicación:*** DGSCA	Al inicio de este artículo se da una breve introducción de la estructura de los objetos compuestos, las propiedades y la semántica de las relaciones compuestas. Posteriormente se describe acerca del modelado de la estructura de los objetos compuestos en UML. Se propone una extensión de la estructura del modelo en UML. La segunda parte describe, con la ayuda de algunos modelos, el comportamiento de los objetos compuestos. Después se presenta y discute el modelado dentro del contexto de UML, se exponen los problemas

	generales. Se propone una extensión al UML con respecto al modelo de comportamiento, en base a los conceptos que se originan desde COBALT (Composite Object Behavior Applied Taxonomy). Finalmente se exponen las conclusiones de este trabajo.
Artículo: The Meta Object Facility: The Final Frontier of Modeling (La Facilidad Meta Objeto: La Frontera Final de Modelar) ***Autor:*** Max Tardiveau ***Fecha:*** Junio 1999 ***Volumen:*** 10 ***Número:*** 9 ***Ubicación:*** DGSCA	Este artículo describe qué es MOF (Meta Object Facility), quiénes lo usan, cómo se relaciona con UML y por qué es importante en el modelado.
Artículo: Tailoring Process-Focused OO Methods (Proceso enfocado a la medida hacia metodos Orientados a Objetos) ***Autor:*** Henderson-Sellers y S. J. Mellor ***Fecha:*** Julio-Agosto 1999 ***Volumen:*** 12 ***Número:*** 4 ***Ubicación:*** DGSCA	Explica el proceso basado en métodos orientados a objetos. Se discuten dos tipos de procesos: RUP, el cual describe el manejo de los casos de uso y el Shaler Mellor, el cual se enfoca en la transición de especificaciones precisas del análisis.
Artículo: Use of CS SI UML Development Process on the GEDYS Project (Uso de UML - CS SI para el Proceso de Desarrollo en el Proyecto GEDYS) ***Autor:*** Y. Bernard, A. Canals ***Fecha:*** Julio-Agosto 1999 ***Volumen:*** 12 ***Número:*** 6 ***Ubicación:*** DGSCA	En este artículo se explica que debido a que UML es un lenguaje de modelado y no un método, CS SI diseñó un software de UML dedicado al proceso de desarrollo de software llamado UML-CS SI. En dicho artículo primero se da una breve descripción del UML-CS SI y posteriormente se presenta una aplicación de este proceso de desarrollo para diseñar parte de una GEDYS (Generation Dynamique de Schemas de cablage- Dynamic generation of wiring diagrams- Generación dinámica de cablear diagramas). Finalmente se analiza el uso de UML, las dificultades encontradas, el uso de UML-CS SI respecto al proceso y los beneficios logrados dentro de este proyecto. Como conclusión de este artículo, se menciona que UML-CS SI es un proceso, resultado de la experiencia en varios desarrollos de proyectos, es un proceso efectivo para integrar la tecnología de objetos en un enfoque jerárquico.
Artículo: Development Process for the Creation and Reuse of Object-oriented Generic Applications and Components (El proceso de desarrollo para la creación y reuso de aplicaciones genéricas y componentes	En este artículo se habla acerca del reuso dentro del proceso de desarrollo de aplicaciones. El proceso consiste en dos subprocesos: diseño de aplicaciones y diseño de componentes. Cada subproceso incluye una etapa de diseño general para producir aplicaciones y

orientados a objetos) ***Autor:*** Xavier Castellani y Stephen Y. ***Fecha:*** Junio 1999 ***Volumen:*** 12 ***Número:*** 7 ***Ubicación:*** DGSCA	componentes genéricos. Esto proporciona una estructura y comportamiento comunes para el diseño de otras aplicaciones y componentes similares.
Artículo: Interface Specification, refinement and Design with UML/ Catalysis (Especificación de interfaces, refinamiento y diseño con UML/ Catalysis) ***Autor:*** Desmond D'Souza ***Fecha:*** Junio 1999 ***Volumen:*** 12 ***Número:*** 7 ***Ubicación:*** DGSCA	En este artículo se definen tres conceptos interesantes del diseño que son: Colaboración, tipo, refinamiento. Además de otros conceptos como: paquetes y estructura.
Artículo: Experience report: SSADM-Designed System to Object-Oriented System (Reporte de experiencia: Diseño del sistema SSADM al sistema orientado a objetos) ***Autor:*** Desmond D'Souza ***Fecha:*** Febrero 1998 ***Volumen:*** 10 ***Número:*** 9 ***Ubicación:*** DGSCA	Se muestran las ventajas y desventajas de la reingeniería del proceso, utilizando el esquema de la orientación a objetos en sistemas pequeños. Se describen los pasos de la reingeniería del sistema SSADM implantado en Clipper al método orientado a objetos de Booch, se muestra cómo muchos problemas son resueltos mediante la aplicación de la orientación a objetos.
Artículo: Open relationships-Associations, Mappings, Dependencies, and Uses (Relaciones-Asociación, mapeo, dependencias y usos) ***Autor:*** B. Henderson-Sellers ***Fecha:*** Febrero 1998 ***Volumen:*** 10 ***Número:*** 9 ***Ubicación:*** DGSCA	Se describe la importancia de la relación entre objetos dentro del modelado. Se estudian las asociaciones, la dependencia y referencia, las relaciones simétricas, relaciones recomendadas (relaciones de asociación estáticas). Contiene varios ejemplos de metamodelos con UML y OML (jerarquía propuesta para conexiones, jerarquía unificada con UML, excepciones para el mapeo de OML, excepciones para el mapeo de Kilov y Ross).
Artículo: Our Cases with Use Cases (Nuestros casos con casos de uso) ***Autor:*** Ari Jaa ***Fecha:*** Febrero 1998 ***Volumen:*** 10 ***Número:*** 9 ***Ubicación:*** DGSCA	Este artículo presenta la experiencia con los casos de uso en un proyecto de software real, el cual fue desarrollado en varios sistemas de administración de red y plataformas multimedia. Este sistema utilizó la metodología OMT++. La primera parte de este artículo describe el esquema OMT++ y los casos de uso usados dentro de este método, se enlistan además algunas reglas prácticas. En la parte final del artículo se explican los planes futuros del uso de los casos de uso.

DATOS DE LA REVISTA
Título: e – Week (e – Semanal) Publicación semanal **Periodo revisado:** 1999 al 2001

Artículos localizados y leídos	Descripción
Artículo: Rational revs testing, online resources (Pruebas de la revolución de Rational, recursos en línea) ***Autor:*** Roberto Holland ***Año:*** 2001 ***No.*** 21 ***Vol.*** 18 ***Fecha:*** 28 de Mayo ***Ubicación:*** Biblioteca DGSCA-UNAM	Es una publicación acerca de la suite de Rational, versión 2001, la cual presenta una arquitectura en tiempo real de calidad, ya que puede generar pruebas de código por medio de los componentes UML. Ahora algunos de los productos de Rational incluyen templates, ejemplos de software y capacitación por vía web en cada uno de sus productos.

DATOS DE LA REVISTA
Título: Information Systems an International Journal. (Sistemas de Información un Periódico Internacional) Publicación mensual **Domicilio:** Gran Bretaña **Periodo revisado:** 1999 al 2001

Artículos localizados y leídos	Descripción
Artículo: Method for the analysis and design of class characteristic migrations during object system evolution. (Método para el análisis y diseño de la migración de las características de clases durante la evolución de un sistema objeto) ***Autor:*** Xavier Castellani, Hong Jiang ***Año:*** 2001 ***No.*** 4 ***Vol.*** 26 ***Fecha:*** Junio ***Ubicación:*** Biblioteca DGSCA-UNAM	El propósito de este artículo es mostrar un método que apoye a los desarrolladores en el análisis y diseño para la migración de las características de las clases, métodos y atributos durante la evolución de un sistema orientado a objetos. El método que se propone abarca del análisis a la migración, para identificar la transferencia de ligas que pueden ser usadas para las características de las clases. Además hacen énfasis en la partición de las migraciones obtenidas de cada componente con el propósito de minimizar el trabajo requerido, ya que cada partición implica un número menor de componentes de un sistema orientado a objetos.

Artículo: A form driven object – oriented reverse engineering methodology. (Manejo de la metodología de ingeniería en reversa orientada a objetos) ***Autor:*** Heeseok Lee y Cheonsoo Yoo ***Año:*** 2000 ***No.*** 3 ***Vol.*** 25 ***Fecha:*** Marzo ***Ubicación:*** Biblioteca DGSCA-UNAM	Las aplicaciones legadas son valiosos recursos que deberían ser integrados en los sistemas de negocios en generaciones sucesivas, lo que permitirá tomar las ventajas de estas aplicaciones, viendo progresar a las compañías, ya que mejoran sus operaciones debido a la ingeniería en reversa. El propósito de este artículo es manejar las metodologías de orientación a objetos y la ingeniería en reversa para recuperar a las aplicaciones legadas a través de cinco diferentes fases: Formas de análisis de uso, formas de objetos divididos, estructura del modelado de objetos, escenario de diseño e integración del modelado.

DATOS DE LA REVISTA
Título: Sdmagazine **Domicilio:** http://www.sdmagazine.com/ **Periodo revisado:** Enero 1999-Noviembre 2001

Artículos localizados y leídos	**Descripción**
Artículo: Driving Design with Use Cases (El Manejo del Diseño con Casos de Uso) ***Autor:*** Doug Rosenberg y Kendall Scott ***Fecha:*** Diciembre 2000 ***Ubicación:*** http://www.sdmagazine.com/	Este es el primero de cinco artículos que se realizaron para la prepublicación del libro "Applied Use Case Driven Object Modeling" (Aplicando casos de uso para el manejo del modelado de objetos), el cual contiene un caso de estudio de una tienda de venta de libros por Internet. Este artículo enseña cómo construir un modelo para un sistema de comercio electrónico utilizando algunas de las técnicas de UML como el diseño casos de uso concisos, diagramas de secuencia, diagramas de clase, realizando así un análisis robusto y eficiente, lo cual facilita la fase de programación.
Artículo: How to Avoid Use-Case Pitfalls (Cómo evitar trampas de casos de uso) ***Autor:*** Susan Lilly ***Fecha:*** Enero 2000 ***Ubicación:*** http://www.sdmagazine.com/	Explica los beneficios de los casos de uso dentro del diseño de sistemas, los cuales son una técnica muy valiosa para la documentación de los sistemas y especificación de requerimientos de los usuarios, además de que ayudan a detectar problemas dentro de áreas críticas. El artículo contiene ejemplos de casos de uso para un sistema de venta de boletos de baseball, contiene tips para un diseño correcto de los casos de uso, para evitar

	confusiones o los errores más comunes que se cometen.
Artículo: Top Ten Use Case Mistakes (Top diez de las equivocaciones de los caso de uso) ***Autor:*** Doug Rosenberg y Kendall Scott ***Fecha:*** Febrero 2001 ***Ubicación:*** http://www.sdmagazine.com/	Este articulo es la tercera parte de la prepublicación del libro "Applied Use Case Driven Object Modeling". En esta parte se muestran las equivocaciones más frecuentes y se explica cómo corregirlas. Este articulo trata acerca de la primera fase del proceso ICONIX, es decir, el diseño de casos de uso, fase que ayuda a capturar los requerimientos de los usuarios, esta fase pertenece a la parte dinámica del proceso. La segunda parte del articulo habla acerca del dominio del problema, donde se encuentran y comprenden los objetos. Como última parte se enumeran y describen los diez errores más comunes en el modelado de casos de uso.
Artículo: Selecting Test Cases Based on User Priorities (Seleccionando casos de prueba basados en prioridades del usuario) ***Autor:*** John D. McGregor y Melissa L. Especialice ***Fecha:*** Marzo 2000 ***Ubicación:*** http://www.sdmagazine.com/	Explica la utilidad de los casos de uso para orientar a los diseñadores para probar el funcionamiento correcto de su sistema y evitar que los usuarios encuentran fallas en los mismos. Contiene información y ejemplos de casos de uso, de los actores, así como el diagrama de casos de prueba, que son una extensión de los caso de uso. Se explica cómo aumentar confiabilidad en los sistemas.
Artículo: Components with Catálysis (Componentes con Catálisis) ***Autor:*** Desmond D'Souza ***Fecha:*** Diciembre 1999 ***Ubicación:*** http://www.sdmagazine.com/	Este articulo enfatiza en el diseño de sistemas eficientes como clave para el éxito e integración de las empresas. Explica que los sistemas de software existen para apoyar al negocio, son adquiridos inicialmente por una necesidad del negocio o bien por una oportunidad tecnológica, de cualquier manera el desarrollo del software debe estar ligado fuertemente con los problemas de negocio. Se menciona que al construir componentes distribuidos de misión critica para los sistemas, el método de Catálysis, basado en UML, es eficiente, pues enfatiza en la precisión, en los modelos y en la estructura de reuso. Explica acerca del modelado de la empresa, del modelado de los componentes del negocio y de soluciones de negocio.
Artículo: Distributed Object Design (Diseño Distribuido de Objetos) ***Autor:*** Scott Ambler ***Fecha:*** Junio 1999 ***Ubicación:*** http://www.sdmagazine.com/	Explica nueve pasos del enfoque del diseño distribuido de objeto, los cuales ayudan al lector a construir software distribuido robusto para aplicaciones de misión critica Contiene ejemplos de diagramas de UML para un sistema bancario, como el diagrama de clase, el diagrama de colaboración simplificado, el diagrama de componentes y el de despliegue.

Artículo: Driving Design: The Problem Domain (El Manejo de Diseño: El dominio del problema) ***Autor***: Doug Rosenberg y Kendall Scott ***Fecha:*** Enero 2001 ***Ubicación:*** http://www.sdmagazine.com/	Este artículo es la parte dos de los cinco que se refieren a la prepublicación del libro "Applied Use Case Driven Object Modeling" (Aplicando casos de uso para el manejo del modelado de objetos). En esta parte se habla acerca del dominio del modelado (Domain Modeling) es una parte esencial del proceso ICONIX, en la cual se descubren los objetos (clases) que representan los conceptos encontrados en la fase de dominio del problema. Se mencionan los diez errores mas comunes de la fase de dominio del modelado y se da un ejemplo de un diagrama de clases con dichos errores y otro diagrama donde se corrigen esos errores.
Artículo: Concepts for Simpler Design (Conceptos para un diseño más simple) (El Manejo de de Diseño: La Heredad de Problema) ***Autor***: Tulu Tanrikorur ***Fecha:*** Junio 1999 ***Ubicación:*** http://www.sdmagazine.com/	Este artículo estudia dos conceptos muy importantes para el modelado, estos son los paquetes y las categorías de clase, los cuales ayudan a abordar los desafíos de diseño de sistemas grandes, haciéndolo mas sencillo. Se explica el concepto de paquetes y sus categorías de clase, de análisis, diseño e implantación de clases.
Artículo: Sequence Diagrams: One Step at a Time (Diagramas de secuencia: Uno paso a la vez) ***Autor***: Doug Rosenberg y Kendall Scott ***Fecha:*** Abril 2001 ***Ubicación:*** http://www.sdmagazine.com/	Este artículo es la quinta parte de los artículos de la prepublicación del libro "Applied Use Case Driven Object Modeling" (Aplicando casos de uso para el manejo del modelado de objetos), describe los diagramas de secuencia, sus elementos clave, los cuatro pasos para realizarlos, los diez errores más comunes de estos diagramas; contiene un ejemplo con errores y otro más con dichos errores corregidos. Explica que el modelado iterativo permite dejar fuera el comportamiento detallado de los objetos y permite encontrar los atributos y operaciones apropiados.
Artículo: Effective Software Deployment (Despliegue Efectivo de Software) ***Autor***: Scott Ambler ***Fecha:*** Noviembre 1999 ***Ubicación:*** http://www.sdmagazine.com/	Este artículo explica el ciclo de vida extendido del proceso de desarrollo de forma general, su utilidad en el análisis y diseño de sistemas complejos, explica cada una de las etapas del proceso: la fase de inicio, de elaboración, de construcción y de transición
Artículo: Effective Building Better Business Systems with Scenarios (Construcción efectiva de sistemas mejorados de negocio con escenarios) ***Autor***: Dr. Carlos Jerome ***Fecha:*** Junio 2000 ***Ubicación:*** http://www.sdmagazine.com/	Recomienda el uso de escenarios como otra opción para sondear los problemas en situaciones en las que los casos de uso no se usen por considerarse demasiado onerosos, además de que los escenarios ilustran de manera clara las reglas de negocio. Explica de forma general los beneficios y características de los escenarios, cómo desarrollarlos, cuándo y por qué suplen a los casos de uso.

Artículo: Components, States and Interfaces, Oh My! (Componentes, Estados e Interfaces, Oh Mi!) ***Autor:*** Bruce Powel Douglass ***Fecha:*** Abril 2000 ***Ubicación:*** http://www.sdmagazine.com/	Este artículo trata acerca del comportamiento de los componentes, considerando a estos como objetos a gran escala de los cuales se desea optimizar un aspecto en particular de sus cualidades. Explica acerca de los estados, de las interfaces y de las dependencias.
Artículo: Extreme Lessons Learned (Aprendiendo Lecciones Extremas) ***Autor:*** B Scott W. Ambler ***Fecha:*** Febrero 2001 ***Ubicación:*** http://www.sdmagazine.com/	Se explican algunos de los temas tratados en una conferencia "Software Development East 2000" que se realizó en Washington D.C., la cual trató acerca del desarrollo de un sistema en línea (Hydrocephalus Association's Web), en ella se discutieron las estrategias para desarrollar dicho sistema en tres días. Entre los temas tratados está la descripción de las fases del proceso de desarrollo: la fase de inicio, elaboración, construcción, transición y producción.
Artículo: The Illusion of Simplicity (La ilusión de la simplicidad) ***Autor:*** B Grady Booch ***Fecha:*** Febrero 2001 ***Ubicación:*** http://www.sdmagazine.com/	El artículo explica la importancia de tres conceptos: el modelado, los componentes y los procesos bien definidos, necesarios para mejorar la eficiencia dentro del desarrollo de software.
Artículo: Components: Logical, Physical Models (Componentes: Modelo Lógico, Físico) ***Autor:*** Bruce Powel Douglass ***Fecha:*** Diciembre 1999 ***Ubicación:*** http://www.sdmagazine.com/	Debido a que UML es un lenguaje y no un método, en este artículo se estudia el uso del proceso ROPES (Rapid Object-Oriented Process for Embedded Systems) que divide su arquitectura en las siguientes fases: • La arquitectura lógica • El modelo de coincidencia • El modelo de distribución • La seguridad y modelo de fiabilidad • El modelo de despliegue • El modelo de subsistema Este proceso complementa a UML en el modelado de sistemas.
Artículo: Modeling Web-Tier Components (Modelado de Componentes Web-Tier) ***Autor:*** Jim Conallen ***Fecha:*** Enero 2001 ***Ubicación:*** http://www.sdmagazine.com/	La WAE-Web Application Extension (Extensión de Aplicación Web) es la clave para el modelado exitoso de sistemas complejos basados en web. WAE define cómo podemos expresar el diseño a nivel de construcción en el contexto de un modelo de UML de un sistema. Explica los componentes de una aplicación web (índice, catálogo, categorías, búsquedas, productos), los estereotipos fundamentales (página del servidor, página del cliente, forma HTML) del WAE, contiene un diagrama de clase que muestra las funciones de la página y las asociaciones lógicas, explica las redirecciones en un diagrama de

	clase y por último el manejo de la complejidad.
Artículo: Four-Wheel Drive, Garbage Barges and Objects (Manejo en cuatro ruedas, basura y objetos) ***Autor***: Steve Adolph ***Fecha:*** Junio 2000 ***Ubicación:*** http://www.sdmagazine.com/	Explica los beneficios del diseño orientado a objetos, el cual crea software más robusto y resistente, se habla del DOO para una aplicación en línea, el funcionamiento del servidor y del cliente.
Artículo: Object-Oriented Business Rules (Reglas de negocio orientadas a objetos) ***Autor***: Scott Ambler ***Fecha:*** Junio 2000 ***Ubicación:*** http://www.sdmagazine.com/	Explica la importancia de las reglas de negocio, por lo que deben ser soportadas por los sistemas de software. El paradigma orientado a objetos, específicamente los modelos de UML, soportan de una manera eficiente dichas reglas. Menciona que pueden usarse los diagramas de actividad para documentar las reglas de negocio, así como los casos de uso y los diagramas de clase.
Artículo: Creating Executable Models (Creando Modelos Ejecutables) ***Autor***: Bruce Powel Douglass ***Fecha:*** Septiembre 1999 ***Ubicación:*** http://www.sdmagazine.com/	Este capítulo es una introducción a la programación de sistemas de tiempo real, explica el concepto de la abstracción, qué son los modelos y su importancia, Menciona la importancia del uso de objetos dentro de los modelos, pues una buena abstracción de los mismos es la clave para un análisis y diseño eficiente. El uso de objetos es importante, ya que combinan información (atributos) y operaciones que actúan sobre esa información (implementada como métodos). Un diagrama de clases muestra la estructura de clases y por lo tanto, las instancias de un objeto. También se mencionan los beneficios de ROPES (Rapid Object-Oriented Process for Embedded Systems), el proceso iterativo, dentro del ciclo de vida. Explica que este proceso no solo contempla las fases de Análisis, Diseño, Implantación y Pruebas, sino que además cada una de estas fases se divide en subfases, en el artículo se explica cada una de ellas.
Artículo: Tracing Your Design (Trazando tu diseño) ***Autor***: Scott Ambler ***Fecha:*** Abril 1999 ***Ubicación:*** http://www.sdmagazine.com/	Este artículo trata acerca de la determinación de los requerimientos, los cuales representan un impacto potencial en los cambios de software. Explica el ciclo de vida para un sistema y menciona la importancia de la determinación de los requerimientos en la fase inicial del ciclo, por último se mencionan otros beneficios que se obtienen al realizar una correcta definición de los requerimientos. Explica cuándo hacerlo y cómo hacerlo apoyado con el lenguaje UML e incorporarlos a RUP. Como parte final habla acerca de los secretos para obtener éxito en el modelado de sistemas de software.

Artículo: Capturing Business Rules (Capturando las Reglas de Negocio) ***Autor:*** Ellen Gottesdiener ***Fecha:*** Diciembre 1999 ***Ubicación:*** http://www.sdmagazine.com/	Explica que las reglas de negocio son el núcleo de los requerimientos funcionales pues proporcionan el conocimiento de cada negocio estructura o procese. Habla acerca del uso de los casos de uso como herramienta útil para el análisis y modelado de las reglas de negocio. La importancia de la colaboración con los clientes.
Artículo: Give Them What They Want (Darles lo que ellos quieren) ***Autor:*** Doug Rosenberg y Kendall Scott ***Fecha:*** Junio 2001 ***Ubicación:*** http://www.sdmagazine.com/	Los requerimientos son una parte esencial del proceso de modelado, por lo que en este artículo se explica cómo los casos de uso y el dominio del modelo trabajan juntos encaminados a obtener los requerimientos funcionales de los usuarios. Muchas veces se piensa que los clientes no saben lo que quieren y que sus requerimientos cambian constantemente, sin embargo es de gran importancia que el analista o diseñador trabaje conjuntamente con el cliente para entender y definir los requerimientos del sistema. Se explica la localización de la determinación de los requerimientos dentro del modelo del proceso ICONIX. Como parte final del artículo se explican los diez errores más comunes cuando se realiza la revisión de los requerimientos.
Artículo: Object Testing Patterns (Las pruebas en el modelo de Objetos) ***Autor:*** Scott Ambler ***Fecha:*** Julio 1999 ***Ubicación:*** http://www.sdmagazine.com/	Los modelos de proceso ayudan a probar las aplicaciones orientadas a objeto de una manera más completa. Se menciona que las pruebas son una parte crítica del proceso de software, al igual que la compresión de los requerimientos, el análisis y el diseño, la implementación, el despliegue y la administración del sistema. Las pruebas son de gran complejidad por lo que se requiere grandes habilidades y conocimiento para tener éxito y validar la funcionalidad del sistema para que sea eficiente.
Artículo: Process for Sale (Procese para Venta) ***Autor:*** Andy Barnhart ***Fecha:*** Junio 2001 ***Ubicación:*** http://www.sdmagazine.com/	Este artículo habla acerca de Rational Unified Process (RUP –Proceso Unificado Racional), se dice que ha tenido gran éxito dentro del desarrollo de software. Se recomienda que al evaluar este producto se debe entender qué hace RUP y qué no hace. Se habla acerca de la instalación de RUP, así como su funcionalidad, sobre todo se hace hincapié en la parte de modelado y determinación de requerimientos. Como parte final contiene las características de RUP 2001, así como los requerimientos técnicos para que pueda funcionar.

Artículo: Palm-Sized Process: Point-of-Sale Gets Agile (Proceso Palm-Sized: Punto de Ventas Obtenidas) ***Autor:*** Gary K. Evans ***Fecha:*** Septiembre 2001 ***Ubicación:*** http://www.sdmagazine.com/	En este artículo se habla acerca del gran éxito comercial de Rational Unified Process (RUP), y se predice que seguirá teniendo grandes éxitos. Se explican las funcionalidades RUP.
Artículo: Pal Successful Robustness Analysis (Análisis Exitoso Robusto) ***Autor:*** Kendall Scott y Doug Rosenberg ***Fecha:*** Marzo 2001 ***Ubicación:*** http://www.sdmagazine.com/	Este artículo pertenece a la cuarta parte de una serie de cinco artículos que se realizaron para la prepublicación del libro "Applied Use Case Driven Object Modeling". Primera parte: Manejo del diseño con casos de uso. Este artículo se enfoca al estudio del análisis robusto, técnica simple útil que consiste en analizar el texto de los casos de uso, identificar los primeros objetos que participan en cada caso de uso y posteriormente clasificar dichos objetos en tres tipos (boundary, entity, cotrol); esta técnica vincula el qué de análisis con el cómo del diseño. Se explican las actividades que deben hacerse dentro del análisis, la utilización de los casos de uso, las reglas esenciales del análisis robusto, así como los diez errores más comunes.
Artículo: Distributed Object Transactions (Las Transacciones Distribuidas de Objetos) ***Autor:*** Scott Ambler ***Fecha:*** Julio 2000 ***Ubicación:*** http://www.sdmagazine.com/	Este artículo trata acerca del control de transacciones, las cuales han existido desde los años 60's y que han tenido que controlarse según la tecnología existente en cada época. Estudia acerca de la evolución de los mainframes centralizados hasta el uso de las arquitecturas distribuidas. Contiene ejemplos de diagramas que muestran el funcionamiento de pequeños negocios; un diagrama de secuencia que muestra las transacciones de un banco. Como parte final del artículo lista la terminología utilizada en las transacciones (transacción, transacción distribuida, transacción distribuida de objeto, commit, Rollback, etc.).
Artículo: The Road to UML 2.0: Fast Track or Detour? (El Camino a UML 2.0: ¿La vía rápida o Desvío?) ***Autor:*** Cris Kobryn ***Fecha:*** Abril 2001 ***Ubicación:*** http://www.sdmagazine.com/	Este artículo habla acerca de la evolución del OMG-UML, acerca de la infraestructura RFP UML 2.0, las superestructura RFP UML 2.0, UML 2.0 OCL RFP.
Artículo: UML Testing Framework (UML Estructura de Pruebas) ***Autor:*** Martin Fowler ***Fecha:*** Abril 1999 ***Ubicación:*** http://www.sdmagazine.com/	Cuando se usa UML para explicar cómo trabaja el sistema, la clave para lograr una comunicación clara está en realizar diagramas simples, pero que muestren los aspectos esenciales del sistema. Este artículo explica acerca de la notación UML, los paquetes, los diagramas

	de clase dentro de la estructura de paquetes, los diagramas de interacción y cómo usar estos diagramas dentro del diseño.
Artículo: Persistence Modeling in the UML (Persistencia del Modelado en el UML) ***Autor***: Scott Ambler ***Fecha:*** Agosto 1999 ***Ubicación:*** http://www.sdmagazine.com/	Este capítulo discute la posibilidad de que el OMG debe extender la definición de los diagramas de clase UML existentes para ayudar a desarrollar el mundo real, así como aplicaciones de misión crítica usando objetos y la tecnología relacional. Se propone un estándar, un estereotipo potencial, se explica acerca de tablas, columnas y relaciones, se listan los estereotipos potenciales para un modelo UML persistente.
Artículo: UML and Communication through the Life Cycle (UML y Comunicación mediante el Ciclo de Vida) ***Autor***: Murray Cantor ***Fecha:*** Abril 1999 ***Ubicación:*** http://www.sdmagazine.com/	Este artículo contiene conceptos interesantes para que el uso del UML mejore la comunicación con el cliente. Explica el desarrollo del ciclo de vida, las vistas del sistema (Arquitectura 4+1: Vista de diseño, de proceso, de implementación, de despliegue y la de casos de uso). Menciona la importancia que tiene la comunicación con el cliente dentro de cada una de las fases del RUP (inicio, elaboración, construcción, transición), en la parte final se explican cada una de estas etapas.
Artículo: UML Deployment Modeling and Beyond (UML Modelando de despliegue y mas allá) ***Autor***: Scott Ambler ***Fecha:*** Abril 1999 ***Ubicación:*** http://www.sdmagazine.com/	Estudia la importancia del uso de los diagramas de despliegue de UML dentro del desarrollo de sistema, lo cual incrementa las oportunidades de entregar sistemas exitosos. Los diagramas de despliegue entran dentro de la vista estática, describe los componentes físicos sobre los que corren los módulos, en otras palabras, muestra el hardware del sistema, el software instalado sobre éste, también muestra el middleware que conecta las máquinas separadas.
Artículo: UML Meets EJB and COM+ (UML Encuentro EJB y COM+) ***Autor***: Cris Kobryn ***Fecha:*** diciembre 2000 ***Ubicación:*** http://www.sdmagazine.com/	Este artículo contiene una descripción de lo que es UML y la tecnología de componentes, explica acerca del soporte que proporciona UML 1.3, contiene un ejemplo de diagrama de despliegue que muestra una arquitectura cliente-servidor, el diagrama detalla la lógica de la aplicación, dónde residen las bases de datos.
Artículo: The Power of a Unifying View (El Poder de una Vista Unificadora) ***Autor***: Bertrand Meyer ***Fecha:*** Junio 2001 ***Ubicación:*** http://www.sdmagazine.com/	Es autor explica acerca de la importancia de UML y de la tecnología orientada a objetos, también explica acerca de Eiffel, método sobre el cual ha trabajado, es una notación común que maneja todos las actividades de desarrollo de software, desde la más abstracta a la más concreta, facilita el estudio de requerimientos, el proceso aplicado a este método no es en cascada o espiral, es un bucle continuo. Por último habla acerca de las clases y los

	componentes.
Artículo: A Broader Context for UML (Un contexto amplio para UML) ***Autor:*** Warren Keuffel ***Fecha:*** Mayo 2000 ***Ubicación:*** http://www.sdmagazine.com/	El autor menciona que a pesar de que se tiene la creencia convencional de que el lenguaje UML es el todo dentro de las metodologías de análisis y el diseño, todavía hay mucho que aprender sobre otros enfoques alternativos para crear software. En este artículo el autor nos habla acerca de Objects for Systems (Objetos para sistemas), también conocido como O4S, que es una metodología de alto nivel enfocada al software como única parte del sistema, no reemplaza a UML pero proporciona un contexto más amplio dentro del cual UML puede jugar un papel valioso. Se muestran esquemas del ciclo de vida de O4S.
Artículo: Defining the UML Kernel (Definiendo el núcleo de UML) ***Autor:*** Roger Smith ***Fecha:*** Octubre 2000 ***Ubicación:*** http://www.sdmagazine.com/	Se explica acerca de qué es el núcleo de UML y cómo puede extenderse, se habla acerca de la evolución de UML dentro de la OMG, por último se muestran opiniones que proporcionan diferentes autores sobre algunas cuestiones interesantes de UML.

2.4 CONFERENCIAS

CONFERENCIA	DESCRIPCIÓN
Título: Presentación de la suite de herramientas de Rational Rose ***Organizador:*** ITERA ***Fecha:*** 7 de septiembre del 2001 ***Lugar:*** WTC México ***Expositor:*** José Santos ***Material:*** CD de la presentación	La compañía Itera es la distribuidora de los productos de Rational Rose en México, dichas herramientas se enfocan principalmente Lenguaje Unificado de Modelado (UML) y al Proceso Unificado Racional (RUP). En esta conferencia se expuso: • La descripción de los productos y servicios de Itera. • Calendarios de los cursos sobre metodologías y herramientas. • Presentación en multimedia de cómo las soluciones, herramientas y mejores prácticas de Rational permiten desarrollar software más rápido y con mejor calidad. ITERA e-development process se encuentra en Jose Luis Lagrange 103, 8° piso Los Morales Polanco, México D.F. Para más información consultar http://www.itera.com.mx
Título: Presentación de las herramientas Select Component Factory (Fabricando y seleccionando componentes) ***Organizador:*** Compañía Ultrasist (Ingeniería en Sistemas Abiertos)	La compañía Ultrasist realizó una presentación de las siguientes herramientas las cuales son de gran utilidad en el desarrollo de sistemas orientados a objetos: • Select Enterprise

Fecha: 30 de Octubre del 2001 ***Lugar:*** Auditorio de la DGSCA - UNAM ***Expositor:*** Guadalupe Quijano León ***Material:*** Presentación en Power Point y material impreso	• Select Component Manager • Reviewer for Select Enterprise • Select Synchronizers: JSync, CSync y VBSync Esta compañía hace énfasis en que sus herramientas se desempeñan mejor que las de Rational, ya que administran componentes, reutilizan código, generan documentos o templates, se pueden conectar a Erwin, poseen ayuda en linea y se basan en UML y en RUP. Para más información consultar http://www.ultrasist.com.mx

2.5 CONGRESOS

CONGRESO	DESCRIPCIÓN
Título: iDevelop Oracle, Presentación de herramientas para el desarrollo de software orientado a objetos ***Organizador:*** Oracle ***Programa:*** Secciones de talleres y conferencias de manera simultánea acerca de las nuevas herramientas de desarrollo que distribuye oracle ***Fecha:*** 28 y 29 de agosto del 2001 ***Expositores:*** Blaise Ribet Principal administradora de productos de Java Tools Group) ***Material:*** Diapositivas impresas, folletos	En este congreso se presentaron conferencias y talleres teórico – prácticos de herramientas, como Jdeveloper, que se basan en el Lenguaje Unificado de Modelado (UML) y en Java. Jdeveloper es un leguaje portable para cualquier plataforma, por lo que representa una herramienta muy poderosa junto con el lenguaje UML, el cual ha sido universalmente aceptado. Se mencionaron además algunas de las ventajas de estas herramientas, como son: • Permiten la expresión de ideas en el modelado. • Ahorran tiempo. • Se reutiliza código. • Valida los requerimientos de los usuarios. • Permite la comunicación con otros desarrolladores. • Genera código automáticamente acerca de las bases de datos en Oracle y clases y objetos para Java. • Administración de código.

2.6 CURSOS

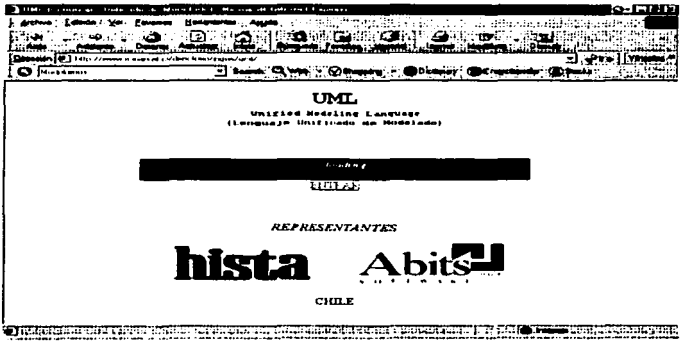
CURSO	DESCRIPCIÓN
Curso: UML ***Dependencia:*** Dirección de Sistemas ***Fecha:*** 28 y 29 de agosto del 2001 ***Instructor:*** Ma. Teresa Ventura ***Material:*** Diapositivas impresas.	Este curso fue impartido dentro del departamento de Integración de Aplicaciones de la Dirección de Sistemas, dicho lenguaje fue utilizado para el análisis y diseño de un sistema por un grupo de trabajo de este departamento, el objetivo de dicho curso fue involucrar a los usuarios con dicho lenguaje de modelado para efectos de recopilación de requerimientos.

2.7 NOTAS DE CURSOS

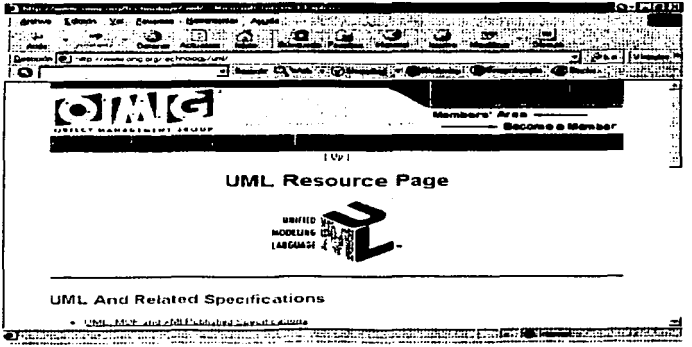
NOTAS	DESCRIPCIÓN
Curso: Análisis y Diseño Orientado a Objetos ***Dependencia:*** Dirección de Sistemas ***Instructor:*** José de Jesús Garcia ***Fecha:*** 16-27 de octubre de 2000	Son una serie de notas tomadas dentro de este curso, el cual fue impartido por la Dirección de Sistemas a la promoción 22 del programa de becas.
Notas: UML y RUP	Son una serie de diapositivas recolectadas y resumidas por Alexei Samuel Dezotti Ruiz.

2.8 DIRECCIONES DE INTERNET

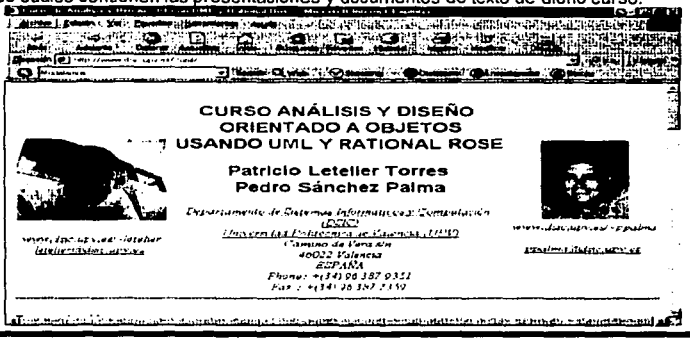
DIRECCIÓN: http://www.e-market.cl/directorio/ingsw/uml/
DESCRIPCIÓN: Muestra la Metodología de Booch, los tipos de diagramas empleados y su utilidad dentro de las etapas del Análisis y Diseño Orientado a Objetos.



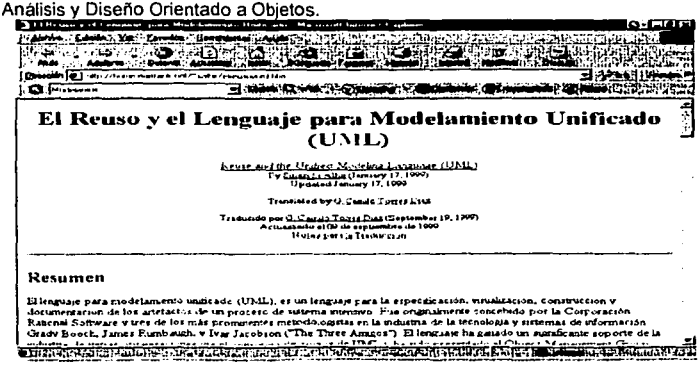
DIRECCIÓN: http://www.omg.org/technology/uml/
DESCRIPCIÓN: Página de UML y OMG, contiene ligas de los avances en UML, a varios artículos e información interesante a otras ligas útiles.



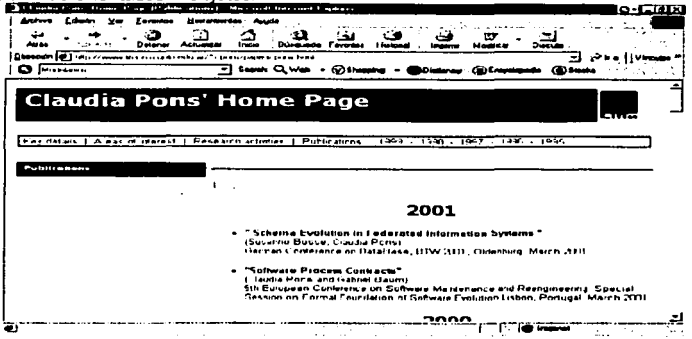
DIRECCIÓN: http://www.dsic.upv.es/~uml/
DESCRIPCIÓN: Esta es una página del Departamento de Sistemas Informáticos y Computación (DSIC) de la Univ. Politécnica de Valencia (UPV), organismo que pone a disposición de los visitantes de la página información acerca de un curso de ADOO con UML y Rational Rose impartido por ellos. La página contiene cuatro documentos, los cuales contienen las presentaciones y documentos de texto de dicho curso.



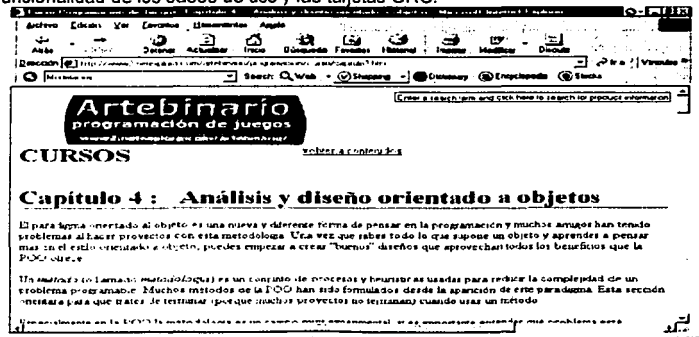
DIRECCIÓN: http://home.earthlink.net/~salhir/elreusoyuml.htm
DESCRIPCIÓN: Muestra brevemente aspectos importantes de UML como: definición, ventajas, importancia de su uso, sobre todo se maneja el concepto de la reusabilidad además de otros elementos generales que deben tomarse en cuenta dentro del Análisis y Diseño Orientado a Objetos.



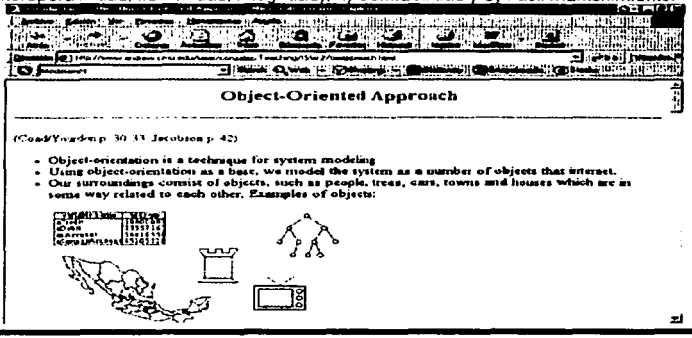
DIRECCIÓN: http://www-lifia.info.unlp.edu.ar/~cpons/paperscpons.html
DESCRIPCIÓN: Contiene publicaciones interesantes relacionadas con el desarrollo de Sistemas Orientados a Objetos.



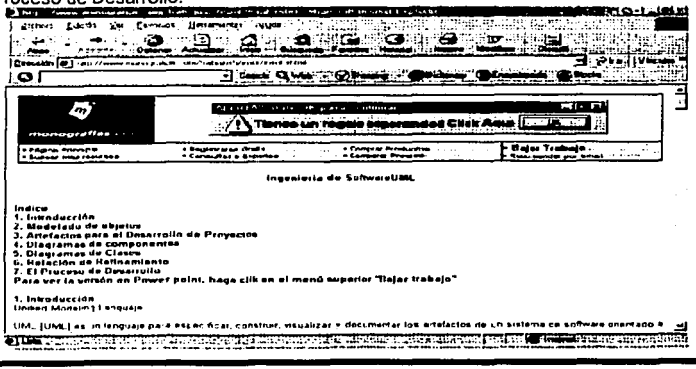
DIRECCIÓN: http://www2.netexplora.com/artebinario/programacion/curso/capitulo3.htm
DESCRIPCIÓN: Contiene un estudio acerca del Análisis y Diseño Orientado a Objetos, muestra los pasos mas importantes que deben realizarse al momento de desarrollar proyectos con este paradigma. Contiene una breve descripción y funcionalidad de los casos de uso y las tarjetas CRC.



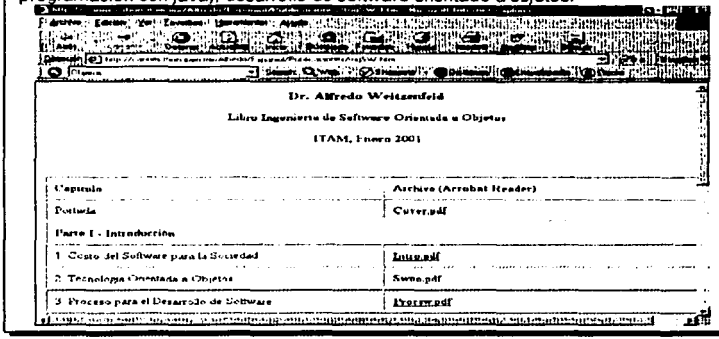
DIRECCIÓN:http://www.andrew.cmu.edu/user/conzalez/Teaching/ISW2/ooapproach.html
DESCRIPCIÓN: Contiene conceptos importantes de la OO, como: abstracción, encapsulamiento, asociación, herencia. Menciona las cualidades de un sistema desarrollado con el paradigma OO: 1) calidad de software (portabilidad, reusabilidad, interoperabilidad, flexibilidad, integridad), 2) Confiabilidad y 3) Fácil mantenimiento.



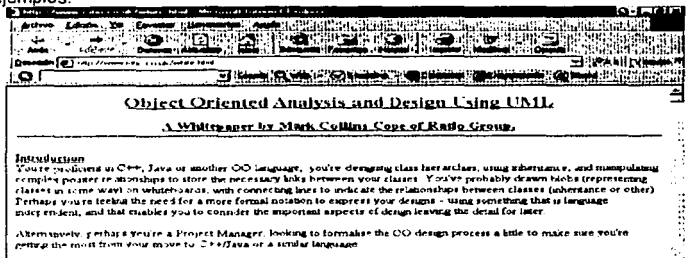
DIRECCIÓN: http://www.monografias.com/trabajos5/insof/insof.shtml
DESCRIPCIÓN: Es un estudio acerca de UML, primero tiene una breve introducción, posteriormente se habla del modelado de objetos, artefactos para el desarrollo de proyectos; explica los diagramas de componentes y el de Clases. También maneja el Proceso de Desarrollo.



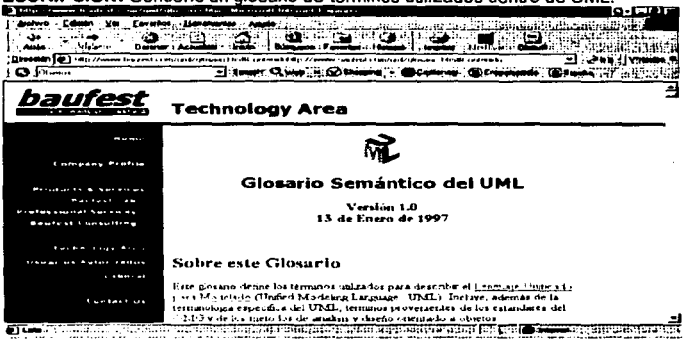
DIRECCIÓN: http://cannes.rhon.itam.mx/Alfredo/Espaniol/Publicaciones/IngSW.htm ()archivo objetos.pdf
DESCRIPCIÓN: Esta página contiene documentos en formato pdf, con temas interesantes acerca del paradigma OO, como: tecnología orientada a objetos, proceso para el desarrollo de software, programación orientada a objeto (modelado con UML, programación con java), desarrollo de software orientado a objetos.



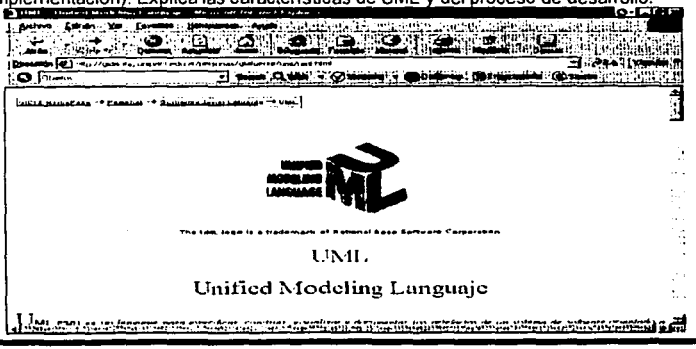
DIRECCIÓN: http://www.ratio.co.uk/white.html
DESCRIPCIÓN: Describe de una manera rápida y entendible el Análisis y Diseño usando UML. Muestra la notación para los objetos, para las clases, contiene ejemplos de código en C++ para definir y manejar las clases. Explica los Casos de Uso, muestra un ejemplo con un sistema bancario. Explica el diagrama secuencia, su notación y ejemplos.



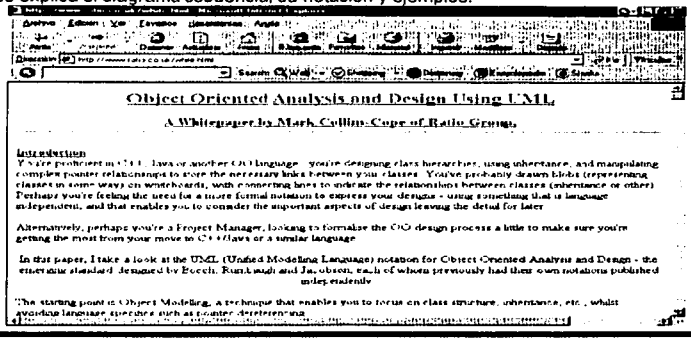
DIRECCIÓN:http://www.baufest.com/uml/glosario.htm#Contenidohttp://www.baufest.com/uml/glosario.htm#Contenido
DESCRIPCIÓN: Contiene un glosario de términos utilizados dentro de UML.



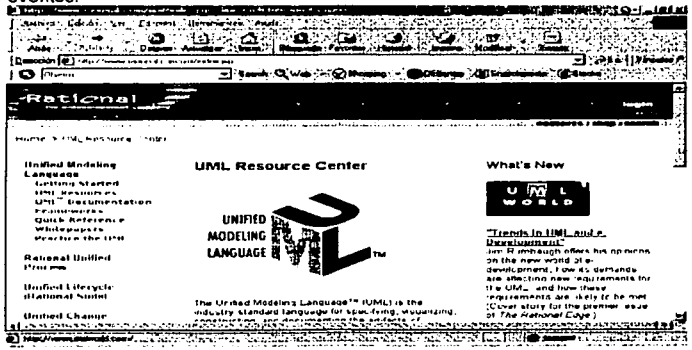
DIRECCIÓN: http://gidis.ing.unlpam.edu.ar/personas/glafuente/uml/uml.html
DESCRIPCIÓN: Contiene información interesante acerca del origen de UML, qué es UML, explica cómo se realiza el modelado de objetos con este lenguaje, cuáles son los artefactos que se usan para el desarrollo de proyectos (diagramas de casos de uso, diagramas de clases, diagramas de comportamiento o interacción, diagramas de implementación). Explica las características de UML y del proceso de desarrollo.



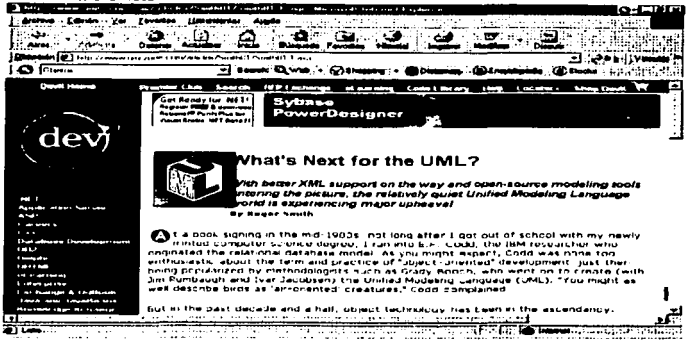
DIRECCIÓN: http://www.ratio.co.uk/white.html
DESCRIPCIÓN: Esta página describe de una manera muy rápida y entendible el Análisis y Diseño usando UML. Muestra la notación para los objetos, para las clases, pequeños ejemplos de código en C++ para definir y manejar las clases. También explica los Casos de Uso y se muestra un ejemplo con un sistema bancario. También se explica el diagrama secuencia, su notación y ejemplos.



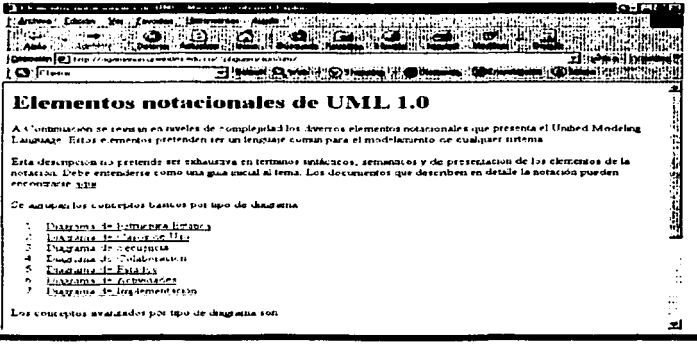
DIRECCIÓN: http://www.rational.com/uml/index.jsp
DESCRIPCIÓN: En es la página de la compañía Rational de los autores de UML en la que se puede encontrar información diversa de UML y RUP así como de los productos y eventos.



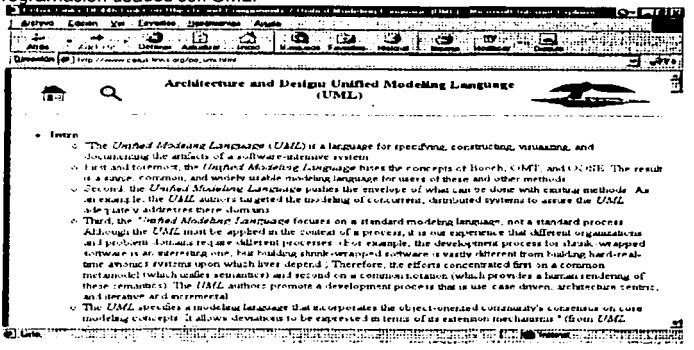
DIRECCIÓN: http://www.uml-zone.com/articles/Smith01/Smith01-1.asp
DESCRIPCIÓN: Esta página contiene noticias acerca de lo nuevo en UML, tiene acceso a las preguntas más frecuentes de UML (UML FAQ), a discusiones de temas diversos relacionados con UML, además de notificar acerca de las nuevas herramientas existentes.



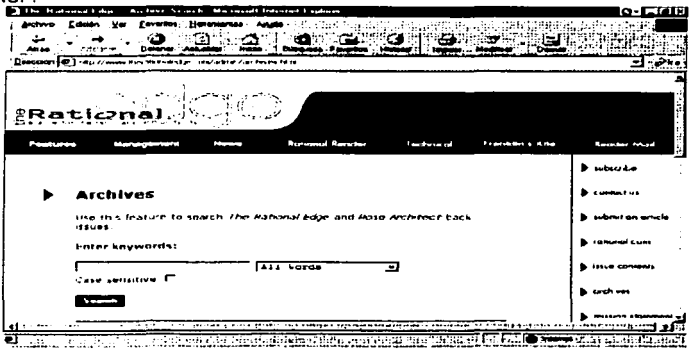
DIRECCIÓN: http://agamenon.uniandes.edu.co/~pfiquero/soo/uml/
DESCRIPCIÓN: Explica los diagramas de UML como son: Diagrama de Estructura Estática, Diagrama de Casos de Uso, Diagrama de Secuencia, Diagrama de Colaboración, Diagrama de Estados Diagrama de Actividades, Diagrama de Implementación, Diagrama de Estructura Estática, Diagrama de Secuencia y Diagrama de Estados.



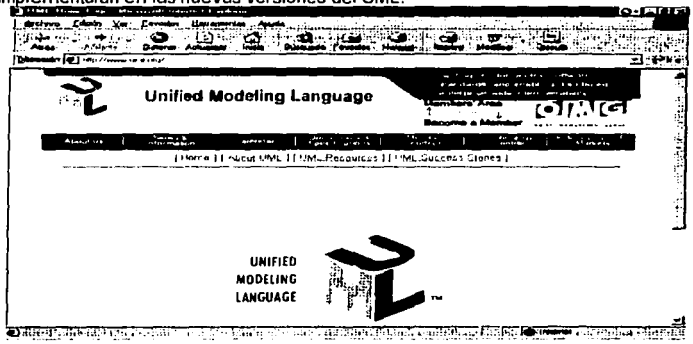
DIRECCIÓN: http://www.cetus-links.org/oo_uml.html
DESCRIPCIÓN: Esta página contiene ligas a temas muy interesantes de UML como son: UML aplicado al ADOO, estandarización de UML, explicación de los diagramas (Casos de Uso, de transición de estados, de secuencia) algunos lenguajes de programación usados con UML.



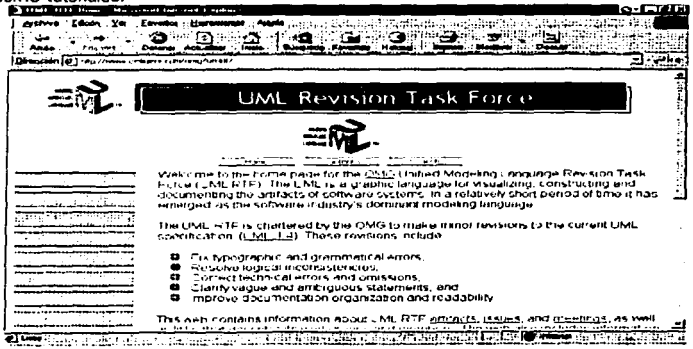
DIRECCIÓN: http://www.therationaledge.com/admin/archives.html
DESCRIPCIÓN: Contiene varios archivos PDF de temas relacionados con UML y RUP.



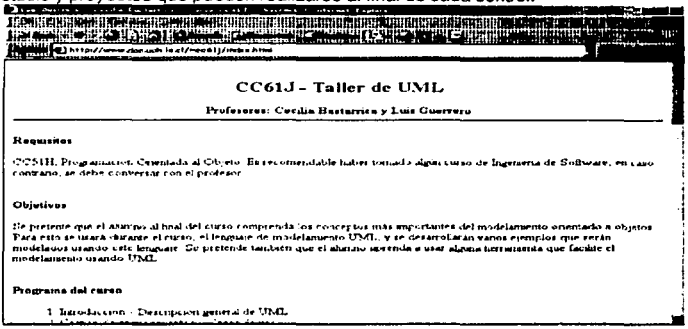
DIRECCIÓN: http://www.uml.org/
DESCRIPCIÓN: Página que cuenta con información acerca de las distintas versiones de UML, así como las más especificaciones importantes y componentes que posee. Podemos encontrar información acerca de los nuevos artefactos que se implementarán en las nuevas versiones del UML.



DIRECCIÓN: http://www.celigent.com/omg/umlrtf/
DESCRIPCIÓN: Esta liga contiene archivos RTF de temas relacionados con UML, así como tutoriales.

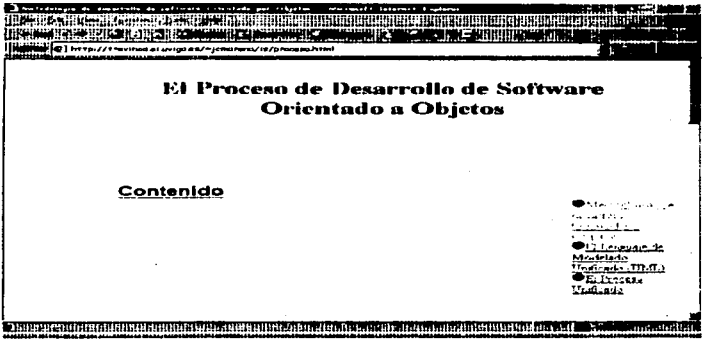


DIRECCIÓN: http://www.dcc.uchile.cl/~cc61j/index.html
DESCRIPCIÓN: Contiene información referente a un taller de UML, dicha información se divide por clases, para cada una de ellas estudia temas como: casos de uso, diagramas de: clases, de interacción, de estado, de componentes, estereotipos, técnicas de modelado, Rational Unified Process (RUP). Presenta varios casos de estudio y proyectos que pueden realizarse al final de cada control.

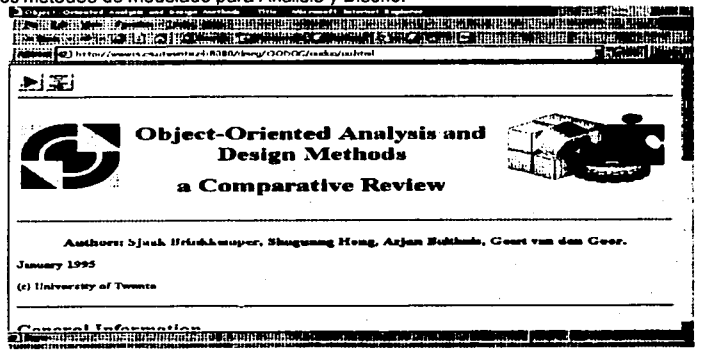


DIRECCIÓN: http://dcx.ucauca.edu.co/pregrado/proy-inv.html
DESCRIPCIÓN: Esta página contiene varios archivos pdf de temas interesantes como: UML, Proceso Unificado para el Desarrollo de Programas, RUP, Aspectos generales de Análisis y Diseño de software, entre otros temas.

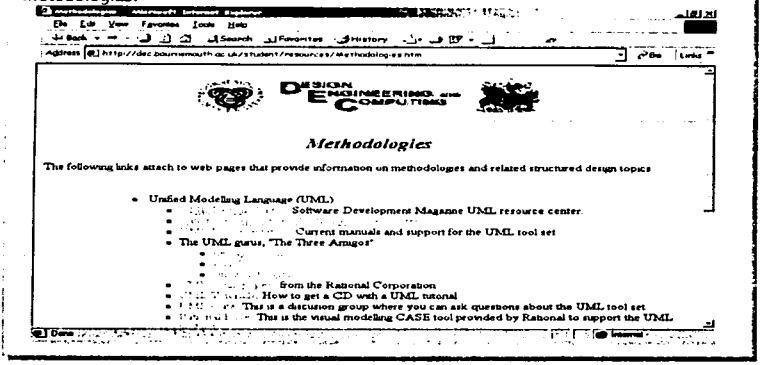
DIRECCIÓN: http://trevinca.ei.uvigo.es/~jcmoreno/is/proceso.html
DESCRIPCIÓN: En esta página se puede encontrar información acerca de: Metodologías de Desarrollo Orientadas a Objetos, Lenguaje de Modelado Unificado (UML) y Proceso Unificado.



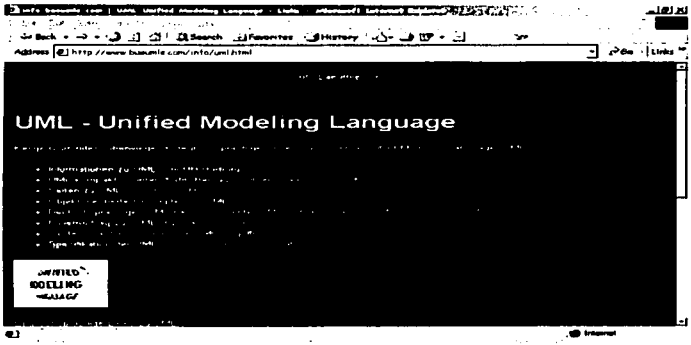
DIRECCIÓN: http://wwwis.cs.utwente.nl:8080/dmrg/OODOC/oodoc/oo.html
DESCRIPCIÓN: Esta página tiene acceso a documentos interesantes acerca de: Métodos orientados a objetos, Análisis y Diseño Orientado a Objetos, tiene información de las metodologías OMT, OOADA, OOSE y de OOSD. Trae información acerca de los métodos de modelado para Análisis y Diseño.



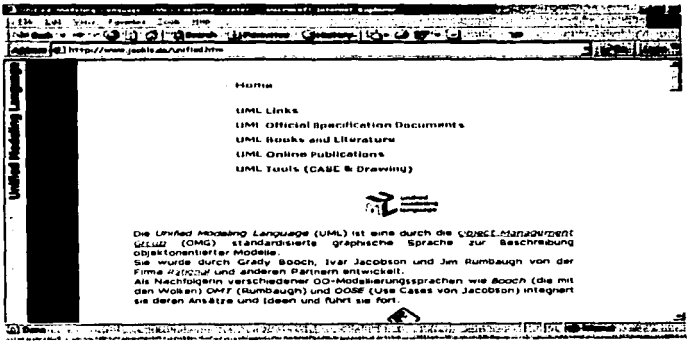
DIRECCIÓN: http://dec.bournemouth.ac.uk/student/resources/Methodologies.htm
DESCRIPCIÓN: Aquí podemos encontrar información acerca de: Documentación de UML, revistas de UML, contiene tutoriales de UML, Técnicas de modelado, información de los autores Grady Booch, Ivar Jacobson y Jim Rumbaugh, algo acerca de la evolución de las metodologías.



DIRECCIÓN: http://www.baeumle.com/info/uml.html
DESCRIPCIÓN: Se encuentran ligas a documentos interesantes como: Diccionarios de UML, Especificación OMG de UML, Herramientas orientadas a objetos para UML, Modelado visual con UML, además de otros Métodos y Herramientas.



DIRECCIÓN: http://www.jeckle.de/uml_pub.htm
DESCRIPCIÓN: Contiene varias secciones como: Ligas de UML, Documentació oficial de la especificación de UML, Libros y literatura de UML, Publicaciones Herramientas.



CAPÍTULO III

3. MARCO CONCEPTUAL

3.1 ANTECEDENTES

Metodologías de Programación
A continuación se presenta un cuadro sinóptico con las metodologías de programación más importantes:

Metodologías de Programación

- Programación no Estructurada
- Programación Estructurada o Modular
- Programación Procedimental
- Programación Orientada a Eventos
- Programación Orientada a Objetos

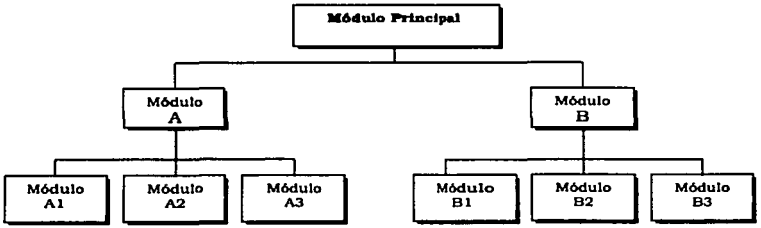
Programación no Estructurada
En esta metodología se empieza escribiendo programas pequeños y sencillos, consistentes en un solo programa principal, el cual contiene una secuencia de comandos o instrucciones que modifican datos, que son a su vez globales en el transcurso de todo el programa.

Esta técnica de programación se vuelve complicada cuando el programa es demasiado extenso. Por ejemplo, si la misma secuencia de instrucciones se necesita en diferentes situaciones dentro del programa, la secuencia debe ser repetida. Esto condujo a la idea de extraer estas secuencias, darles un nombre y poderlas llamar desde el procedimiento principal.

Programación Estructurada o Modular
En programación modular el programa se divide en módulos (partes independientes), cada una de las cuales ejecuta una única actividad o tarea específica, codificada independientemente de otros módulos.
Se tiene, además, un "programa principal", que controla todo lo que sucede; transfiere el control a submódulos, para que se puedan ejecutar sus funciones y después cada submódulo devuelve el control al módulo principal cuando se haya completado su tarea. Si la tarea asignada a cada submódulo es demasiado compleja, éste deberá romperse en otros módulos más pequeños, el objetivo es que cada módulo tenga solamente una tarea específica que ejecutar, esta tarea puede ser entrada, salida, manipulación de datos, control de otros módulos o alguna combinación de éstos. Un módulo puede transferir temporalmente (bifurcar) el control a otro módulo; sin embargo, cada módulo debe, eventualmente, devolver el control al módulo del cual se recibe originalmente el control.

Con esta técnica ya no se tiene un solo programa, sino que se tienen secciones pequeñas de código que interactúan a través de llamadas a procedimientos y que integran el programa en su totalidad.
Cada módulo puede contener sus propios datos, lo que permite que cada módulo maneje un estado interno que es modificado por las llamadas a procedimientos de ese módulo. Sin embargo, solamente hay un estado por módulo y cada módulo existe sólo una vez en todo el programa.

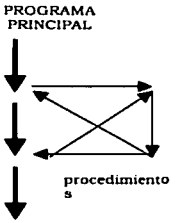
La programación estructurada reduce la complejidad de los programas y los errores, ya que los programas se escriben más fácilmente. Permite la descomposición de un problema en un conjunto de subproblemas independientes entre sí, más sencillos de resolver y que pueden ser tratados de forma separada. Otra de las ventajas es que se pueden probar los subprogramas o módulos de manera independiente, sus errores son depurados antes de su inclusión en el programa principal y almacenarse para su posterior utilización cuantas veces se desee. Dado que los módulos son independientes, diferentes programadores pueden trabajar simultáneamente en diferentes partes del mismo programa, de ésta manera se reduce gran tiempo de diseño del algoritmo y la codificación, además una modificación radical dentro de un módulo no afectará a los demás.
La programación modular es uno de los métodos de diseño más flexibles y potentes para mejorar la productividad de un programa.

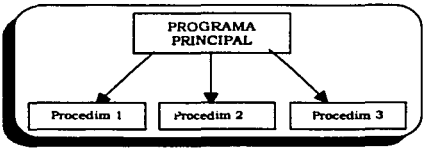


Programación Procedimental
En esta técnica se combinan las sentencias de instrucciones repetibles en un solo lugar (en procedimientos), los cuales se invocan mediante llamadas a procedimientos, después de que la secuencia es procesada, el flujo de control inicia exactamente después de la posición donde la llamada fue hecha.

Al introducir parámetros y subprocedimientos, los programas se escriben en forma más estructurada y libres de errores, cuando un procedimiento ya es correcto siempre producirá resultados correctos. Por consecuencia, en caso de errores, se puede reducir la búsqueda a aquellos lugares que todavía no han sido revisados.

Con esta técnica los programas son vistos como una secuencia de llamadas a procedimientos y el programa principal es responsable de pasar los datos a las llamadas individuales, los datos son procesados por los procedimientos y, una vez que el programa ha terminado, los datos resultantes son presentados. El flujo de datos puede ser ilustrado como una gráfica jerárquica, es decir, como un árbol.





Programación Orientada a Eventos

Consiste en la ejecución del código cuando se lleva a cabo una acción sobre un control (botones, menús, etc.). Las porciones de código a ejecutarse no son controladas por la aplicación en sí, sino por el código asociado a los eventos de un control.

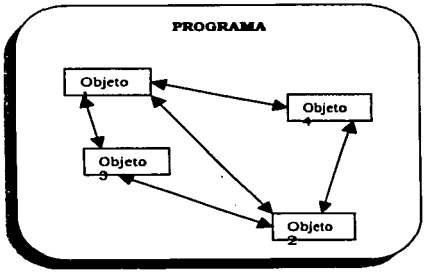
Programación Orientada a Objetos
Es un método en el que los programas se organizan como colecciones de objetos que cooperan entre sí, cada uno de los cuales representan una instancia de alguna clase, las cuales a su vez son miembros de una jerarquía de clases unidas mediante relaciones de herencia.

Esta metodología de programación posee tres elementos importantes:
1) La programación orientada a objetos utiliza objetos.
2) Cada objeto es una instancia de alguna clase.
3) Las clases están relacionadas con otras clases por medio de relaciones de herencia (jerarquía de clases).

La herencia es un concepto muy importante en esta técnica, de manera que la programación sin herencia no es orientada a objetos, es más bien programación con tipos abstractos de datos. Un lenguaje es orientado a objetos, si y sólo si satisface los siguientes requisitos:

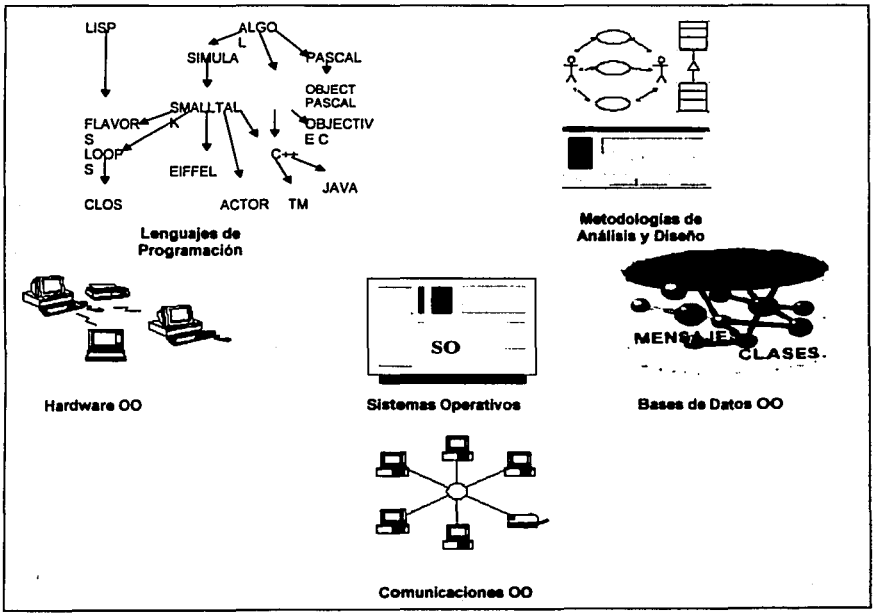
1) Soporta objetos que son abstracciones de datos con una interfaz de operaciones, con nombre y un estado local oculto.
2) Los objetos tienen un tipo asociado (clase).
3) Los tipos de clases pueden heredar atributos de los supertipos (superclases).

Cada objeto es responsable de inicializarse y destruirse de forma correcta. Por consiguiente, ya no existe la necesidad de llamar explícitamente al procedimiento de creación o de terminación.



Paradigma Orientado a Objetos

La complejidad, cada vez más crítica del software, llevó a que el Paradigma Orientado a Objetos se propagara a varias áreas de la informática, entre las que están: los lenguajes de programación, las metodologías de análisis y diseño, hardware, sistemas operativos, bases de datos, las comunicaciones. Para comprender mejor el auge que ha tomado el desarrollo de sistemas de software basados en el paradigma OO, se realizará un estudio de todos los conceptos que están detrás de este paradigma, así como de sus antecedentes históricos y cómo ha venido evolucionando esta tecnología.



Paradigma Orientado a Objetos

El concepto de *objeto* es central dentro del paradigma orientado a objetos, de primera instancia podríamos decir que un objeto es una entidad tangible que muestra algún comportamiento bien definido que sirve para unificar las ideas algorítmicas y de datos. Los objetos tienen propiedades invariantes que los caracterizan así mismos y a su comportamiento, pueden cambiar de estado, actuar, ser manipulados o permanecer en relación con otros objetos, lo único que no se puede hacer a los objetos es violar su integridad.
El término objeto surgió a principios de los 70's, casi simultáneamente, en varios campos de la informática, con el objetivo de manejar la complejidad de sistemas de software, de tal forma que los objetos representaban componentes de un sistema descompuesto en unidades modulares.

Entre los sucesos que contribuyeron a la evolución de conceptos orientados a objetos, fueron:

- Avances en lenguajes de programación: Simula, Smalltalk, CLU y Ada.
- Avances en metodología de la programación, incluyendo la modularización y la ocultación de información.
- Avances en modelos de bases de datos.
- Investigación en inteligencia artificial.

Dijkstra fue el primero en identificar formalmente la importancia de componer sistemas en capas de abstracción. Después Parnas introdujo la idea de ocultación de información y en los años setenta varios investigadores, como Liskov y Zilles, Guttag y Shaw, trabajaron en el desarrollo de mecanismos de tipos de datos abstractos. Hoare contribuyó a esos desarrollos con su propuesta de una teoría de tipos y subclases.

Lenguajes de Programación Orientados a Objetos

1950/60	Los primeros inicios de la programación orientada a objetos se dan cuando Kristen Nygaard desarrolla Simula 1 (orientado a la simulación sistemas).
1960-67	Desarrollo del lenguaje Simula 67, creado en Noruega, por un grupo de investigadores dirigido por Krinsten Nygaard y Ole-Johan Dahl en 1967 en el Centro de Cálculo Noruego. Su finalidad era realizar simulaciones discretas de sistemas reales. En estos tiempos no existían lenguajes de programación que se ajustaran a sus necesidades, se basaron en el lenguaje ALGOL 60 y lo extendieron con conceptos de objetos, clases, subclases, corrutinas, herencia, polimorfismo por inclusión (herencia de clases) y procedimientos virtuales; éstos últimos permiten la sobrecarga de procedimientos, de tal forma que para un solo procedimiento se pueden tener varias implementaciones, dependiendo del nivel de jerarquía de la herencia de clases en el cual está definido un procedimiento. Todos estos conceptos eran nuevos dentro de la programación. El nacimiento de la orientación a objetos en Europa no tuvo mucho impacto comercial, pasó inadvertido para gran parte de los programadores, sin embargo, los conceptos que se definieron en ese entonces se volvieron sumamente importantes para el futuro del desarrollo de software.
1970	Durante esta década pocos eran los sistemas que lograban terminarse con los requisitos iniciales, muchos sistemas no cumplían con los requerimientos o se usaban según lo planificado, representaba un gran problema para los desarrolladores adaptar el software a nuevos requerimientos no planeados inicialmente. La orientación a objetos brindó métodos de experimentación, más que de planeación o previsión. En respuesta al problema anterior, a mitad de los 70's, los científicos del Centro de

Investigación de Xerox en Palo Alto, E.U.A., crearon el lenguaje Smalltalk. Este lenguaje adoptó varios de los conceptos nombrados anteriormente como su fundamento, se considera el primer lenguaje basado en objetos, pues cada elemento del lenguaje fue realizado como un objeto, cada aspecto del lenguaje, el entorno de programación y la cultura que lo rodeaba eran orientados a objetos.
El hecho de haber sido creado en E.U.A. ayudó a que se introdujera a nivel mundial el término de Orientación a Objetos (Object Oriented) y que cobrara importancia entre los diseñadores de lenguajes de programación.
Los puntos importantes de este lenguaje fueron, por un lado, adoptar el concepto de objeto y clase como núcleo del lenguaje y la programación interactiva, incorporando las ideas ya conocidas de lenguajes funcionales. Es decir, que se tuviese un lenguaje interpretado y no compilado.
Después del surgimiento de varias versiones de Smalltalk (72, 74, 76,80) todo llegó a ser instancia de una clase dentro del lenguaje.
Smalltalk fue el primer lenguaje Orientado a Objetos puro de los lenguajes Orientados a Objetos, es decir, únicamente utiliza clases y objetos (Java usa tipos de datos primitivos, o bien los Wrappers que son clases que encapsulan tipos de datos primitivos).

Durante esta década también se desarrollaron lenguajes como Alphard, CLU, Euclid, Gypsy, Mesa y Modula, que soportaban las ideas entonces emergentes de abstracción de datos.
Para este entonces, uno de los defectos más graves de la programación es que las variables eran visibles desde cualquier parte del código y podían ser modificadas incluyendo la posibilidad de cambiar su contenido (no existen niveles de usuarios o de seguridad).

Por lo que D. Parnas propuso la disciplina de ocultar la información. Su idea era encapsular cada una de las variables globales de la aplicación en un solo módulo junto con sus operaciones asociadas, sólo mediante las cuales se podía tener acceso a esas variables.
El resto de los módulos (objetos) podían acceder a las variables sólo de forma indirecta mediante las operaciones diseñadas para tal efecto.

1980 Se da el boom de los lenguajes OO.
Los lenguajes basados en objetos agregan nuevas técnicas de OO con programación estructurada (Orientado Híbrido).

1985

En 1985 Bjarne Stroustrup de los laboratorios de AT&T, extendió el lenguaje de programación C para crear C++ que soporta la programación orientada a objetos.
También por esta fecha se creó desde sus bases el lenguaje EIFFEL por B. Meyer. Ambos manejan conceptos de objetos y herencia de clases. Se introduce la herencia múltiple pensando en dar mayor flexibilidad a los desarrolladores, sin embargo, actualmente este concepto se ha evitado por agregar complejidad a las estructuras de clases.
La adición de mecanismos de programación orientada a objetos al Pascal ha llevado a la aparición del Object Pascal, Eiffel y Ada. Además, hay muchos dialectos del lenguaje Lisp que incorporan las características orientadas a objetos de Simula y Smalltalk, como Flavors, LOOPS y el Common Lisp Object System (CLOS).

Posteriores mejoras en herramientas y lanzamientos comerciales de C++ por distintos fabricantes, justificaron la mayor atención hacia la programación orientada a objetos en la comunidad de desarrollo de software. El desarrollo técnico del hardware y su disminución del costo fue el detonante final. Con más computadoras al alcance de más personas más programadores, más problemas y más algoritmos surgieron.

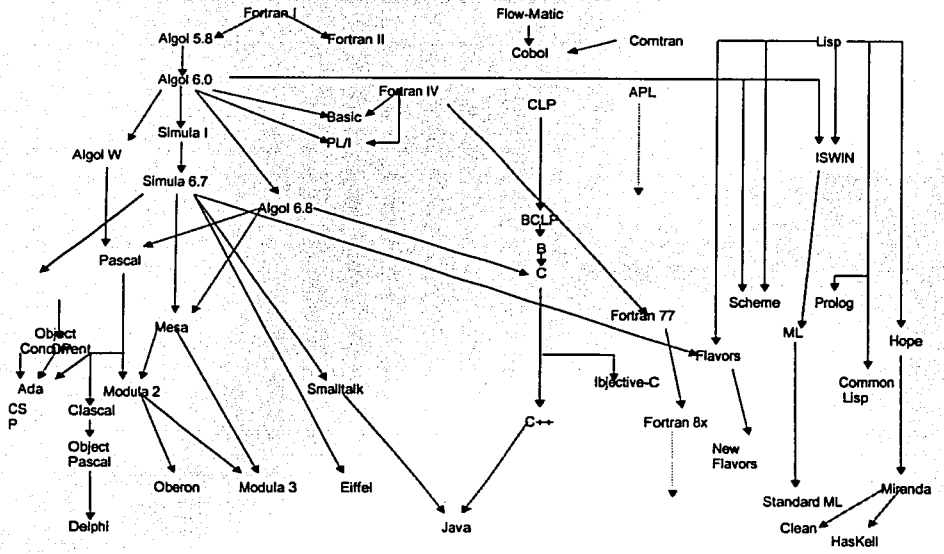
1990

En el inicio de esta década se consolida la orientación a objetos como una de las mejores maneras para resolver problemas. Aumenta la necesidad de generar prototipos más rápidamente (surge el concepto de RAD Rapid Aplication Developments). Sin esperar a que los requerimientos iniciales estén totalmente precisos.
En 1996 surge un desarrollo llamado JAVA (el mas reciente lenguaje OO), desarrollado por SUN, que hereda conceptos de C++, pero los simplifica y evita la herencia múltiple. A cambio se introduce el término de interfaz y la herencia múltiple de interfaces. Su filosofía es aprovechar el software existente. Facilitar la adaptación del mismo a otros usos diferentes a los originales sin necesidad de modificar el código ya existente. Este lenguaje obtuvo una aceptación rápida gracias a los applets, que son unos programas en JAVA insertado en páginas WEB dentro del código HTML. Estos programas pueden viajar a través de la Internet y brindarle al usuario mayor interactividad con las páginas WEB. JAVA introduce también, la programación concurrente y distribuida. El lenguaje es mitad compilado y mitad interpretado dando como resultado la portabilidad a distintas plataformas. JAVA aun sigue evolucionando y se espera que en los próximos años logre la madurez adecuada para volverse un lenguaje de desarrollo de mayor importancia.

En 1997-98 se desarrollan herramientas 'CASE' orientadas a objetos (como el diseño asistido por computadora).

De 1998 a la fecha se desarrolla la arquitectura de objetos distribuidos RMI, Corba, COM, DCOM.

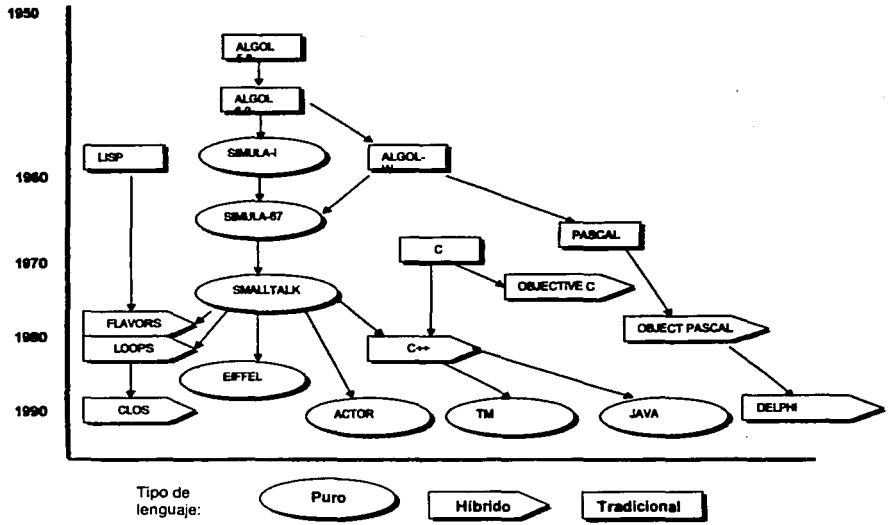
Actualmente la orientación a objetos parece ser el mejor paradigma, no obstante, no es una solución a todos los problemas. Trata de eliminar la crisis del software.



Lenguajes de Programación

A lo largo de la historia han surgido dos grupos principales de lenguajes orientados a objetos a partir de la última década de evolución de lenguajes orientado a objetos:

1. *Lenguajes orientados a objetos puros*: en los cuales casi todo es un objeto, en este grupo se encuentran Smallalk, Actor, Fiel. Estos lenguajes resaltan la exploración y el prototipo rápido.
2. *Lenguajes orientados a objetos hibridos:* son una mezcla entre los lenguajes tradicionales y los orientados a objetos, los miembros de este grupo incluyen a C++, Objective-C, Common Lisp Object System (CLOS), Object Pascal, etc. Estos lenguajes ponen énfasis en la velocidad de ejecución y facilitan la incorporación del programador orientado a procedimientos, incorporando extensiones orientadas a objetos a los lenguajes procedimentales.



Es importante diferenciar entre lenguajes OO y lenguajes que no lo son. Tenemos lenguajes:
Basados en objetos: Lenguajes que no tienen herencia. Presentan un concepto parecido a clase y alguna forma de crear objetos a partir de ésta.
Orientados a objetos: Presentan el concepto de objetos, clases y herencia de clases.
La topología de estos lenguajes es una colección lógica de clases y objetos en lugar de subprogramas, como ocurría en lenguajes anteriores. Si los procedimientos y funciones son verbos y los elementos de datos son nombres, un programa orientado a objetos se organiza alrededor de los nombres, por lo que la estructura física de una aplicación orientada a objetos tiene el aspecto de un grafo, no el de un árbol. Además, existen pocos o ningún dato global. Los datos y operaciones están unidos de tal modo que los bloques lógicos de construcción fundamentales de estos sistemas, ya no son algoritmos, sino clases y objetos.

Los lenguajes orientados a objetos benefician al desarrollador proporcionándole una forma natural de modelar el fenómeno del mundo real.
A continuación haremos un breve estudio de los lenguajes orientados a objetos, como sus antecedentes, concepto y características más importantes:

SIMULA

Antecedentes:
La primera versión de este lenguaje, Simula I, fue diseñada e implantada en el Centro de computación noruego de Oslo entre 1962 y 1964, por Kristen Nygaard y Ole-Johan Dahl. Mas tarde apareció la versión Simula 67.
Simula nació con la idea de crear una herramienta que permitiera describir fácilmente simulaciones computacionales de sistemas de colas, en el contexto de la investigación operativa, a pesar de este objetivo inicial, los investigadores reconocieron que podían convertir a Simula en un lenguaje de programación de propósito general, tarea que se llevó a cabo con éxito.
Simula tomó directamente las estructuras de control y bloques de Algol, lenguaje en el que está firmemente inspirado, sin embargo, incorpora una serie de conceptos novedosos, que constituyen la base de la programación orientada a objetos.

Concepto
Simula (Simple universal language, Lenguaje universal simple), es un lenguaje general orientado a objetos, que se soporta en el sistema de clases (sistema de clases Simset) y el proceso discreto orientado a la simulación (sistema de clases Simulation).

Características
- Clases y objetos como instancias de clases.
- Herencia o extensión de clases.
- Métodos virtuales.
- Co-rutinas.
- Simula puede considerarse el antecesor mas directo del lenguaje Samlltalk, que permite la difusión de la programación orientada a objetos.

OBJECTIVE C

Antecedentes
En Febrero de 1988 se libera Objective-C desarrollado por Brad Cox.

Concepto
Lenguaje de programación C orientado a objetos de The Stepstone Corporation, que corre en PCs y en muchas estaciones de trabajo. Fue la primera extensión comercial orientada a objetos del lenguaje C. Incluye conjuntos de clases empaquetados llamados ICpaks, tanto para la manipulación de estructuras de datos de propósito general como para la construcción de interfaces de usuario icónicas.

Características
- Es un lenguaje orientado objetos.
- Es una extensión del estándard ANSI C, los programas de C pueden adaptarse para usar las estructuras de software sin perder los trabajos desarrollados originalmente.
- Tiene todos los beneficios de C, pero cuando se desea desarrollar la técnica orientada a objetos, se pueden definir clases.
- Objective-C es un language simple. Su sintaxis es pequeña, no ambigua y fácil de aprender.
- Objective-C es el más dinámico de los lenguajes orientados a objetos basado en C, tiene un gran poder y flexibilidad. Permite crear arquitecturas para el uso de interfaces interactivas.
- Permite la construcción de herramientas sofisticadas de desarrollo.

SMALLTALK

Antecedentes

Desarrollado dentro del Grupo de Investigación del Aprendizaje de Xerox en Palo Alto a principios de los 70s'. Las principales ideas de Smalltalk se le atribuyen a Alan Kay, quien se basó en Simula, LISP y SketchPad. Dan Ingalls escribió el código de las primeras ventanas, los pop-up menus y la clase BitBlt.

Adele Goldberg y Dave Robson, miembros clave del equipo de desarrollo, escribieron los manuales de referencia para Smalltalk.

Un programa de licenciamiento de Xerox y Xerox Special Information Systems distribuyó el entorno de desarrollo Smalltalk a un limitado número de desarrolladores y grandes compañías, la distribución generalizada a la comunidad de desarrollo fue hasta la fundación de la compañía ParcPlace Systems Inc., dirigida por Adele Goldberg.

Un segundo Smalltalk (Smalltalk/V) fue desarrollado por Digitalk en los Angeles, California, con financiamiento de Ollivetti y otros clientes. Este Smalltalk estaba dirigido a cubrir la necesidad de un producto pequeño, de alta velocidad, basado en PC. Antes de la adquisición por parte de ParcPlace Systems Inc., Digitalk era el líder de las ventas.

Object Technology International Inc. (OTI) desarrolló un conjunto de herramientas de manejo para todos los Smalltalks llamado ENVY/Developer para proveer el control de versiones y el manejo de configuraciones en grandes proyectos.

OTI desarrolló una máquina virtual de 32-bits para el producto de Digitalk para Apple y participó en una amplia gama de proyectos de investigación, desde herramientas cliente servidor orientadas a objetos, Smalltalk integrado, y procesamiento de imágenes de radar para operaciones militares hasta sistemas con restricciones y generadores de máquinas virtuales portables.

IBM desarrolló la familia de productos Visual Age para Smalltalk en colaboración con Object Technology International Inc.

Hoy, Object Share (antiguamente Parc Place-Digitalk) e IBM permanecen como los distribuidores dominantes de entornos de desarrollo en Smalltalk. Algunos nuevos Smalltalks se hallan en etapa de desarrollo.

Concepto

Es un lenguaje de programación orientado a objetos integrado con un entorno de desarrollo multiventana

Características

- Permite definir clases de objetos, las cuales están organizadas en un árbol de herencia, creando así de nuevos objetos parecidos a los anteriores pero con pequeños cambios. La herencia reduce la cantidad de código y puede permitir el reuso de componentes. No limita a los desarrolladores a un pequeño número de tipos de datos, tales como integer,float, string, etc., puestos que un objeto es una representación de información que consiste en una porción de memoria privada y un conjunto de operaciones para manipular la información almacenada en esa porción de memoria privada. Permite el envío de mensajes a un objeto, que hacen que las clases, objetos o extensiones de las clases básicas cobren vida cuando se les envía un mensaje. Todo es un objeto en Smalltalk, lo cual da gran flexibilidad a los desarrolladores. Los sistemas Smalltalk vienen con una extensa librería de clases para construir a partir de ellas. Smalltalk tiene un número reducido de comandos y es relativamente fácil de aprender comparado con C++ o Ada. El código fuente de Smalltalk tiene formato libre y es fácil de leer. Ejemplo de una sentencia:

```
saldo := estaCuenta saldoActual.fechaExtracción estaDentroLimites
ifTrue: [ saldo := saldo - cantidadExtraccion ]
```

FLAVORS
Antecedentes Lenguaje desarrollado en 1979 como una extensión orientada a objetos del lenguaje LISP. El lenguaje Flavors es descrito en "Object-Oriented Programming with Flavors".
Características • Es uno de los lenguajes de programación que siguieron las premisas de los primeros lenguajes orientados a objetos.

OBJECT PASCAL
Antecedentes Este lenguaje surge a partir del desarrollo del Borland Pascal 7.0, un lenguaje que ocupa un lugar muy importante en la programación de computadoras personales.
Concepto Object Pascal, un lenguaje de programación muy poderoso que está a la altura de C++, incluso lo supera en algunos aspectos.
Características • Object Pascal es totalmente compatible con Borland Pascal 7.0, lo que permite que programas desarrollados en este lenguaje puedan ser convertidos a Delphi. Incluso la biblioteca de clases OWL 1.0 se incluye con el paquete de Delphi. Aspectos nuevos en el Object Pascal en relación a sus predecesores son el Exception-Handling (tratamiento y canalización de errores en run-time), un manejo más sencillo de los punteros con reconocimiento automático y referenciación, las llamadas propiedades de objetos que pueden ser asignadas como las variables. • Object Pascal utiliza sus características y su funcionamiento para describir un objeto, sobre todo a la hora de que el programador crea sus propios componentes.

DELPHI

Antecedentes

Tras 9 versiones de compiladores de Turbo Pascal y Borland Pascal, que fueron extendiendo el lenguaje gradualmente, Borland puso a la venta Delphi en 1995, convirtiendo Pascal en un lenguaje de programación visual. Delphi extiende el lenguaje Pascal de muchas formas, incluyendo muchas extensiones orientadas a objetos que son distintas de otras versiones de Object Pascal, incluidas las del compilador *Borland Pascal with Objects*.

Concepto

Delphi es una herramienta visual para Windows desarrollada por Borland, basada en el lenguaje Pascal.

Características

- Permite la creación de aplicaciones para Windows 3.x, Windows95 y Windows NT.
- Las aplicaciones pueden colocarse de forma muy sencilla en la pantalla ya que dispone de una paleta, llamada VCL (Visual Component Library), dotada de una gran variedad de componentes visuales.
- Sus componentes facilitan la programación, no se requieren las complejas llamadas al sistema de Windows, necesarias dentro de la programación bajo Windows.
- Permite crear nuevos componentes que pueden incorporarse en la paleta con los componentes ya existentes y pueden ser utilizados de la misma forma, la VCL puede estructurarse libremente.
- Contiene características interesantes para aplicaciones de bases de datos. No está limitado a un formato de datos determinado, sino que se tiene acceso a 50 formatos de datos diferentes a través de controladores suministrados por terceros (IDAPI y ODBC). Entre éstos se encuentran todos los estándares importantes de bases de datos en el área del PC como XBase, Paradox, Access, etc. Es posible acceder fácilmente a servidores de bases de datos de otros sistemas, como UNIX, por medio del SQL (Structured Query Language) que constituye un estándar de lenguaje de uso general para consultar y modificar datos administrados por servidores especiales de bases de datos como Oracle, Sybase, Informix o Adabas.
- Dispone del Object Pascal, un lenguaje de programación muy poderoso que está sin dudas a la altura del C++, incluso lo supera en algunos aspectos. Este lenguaje surge a partir del desarrollo del Borland Pascal 7.0. El Object Pascal es totalmente compatible con Borland Pascal 7.0, los programas desarrollados con este último puedan ser convertidos a Delphi. Incluso la biblioteca de clases OWL 1.0 está incluida con el paquete de Delphi. Object Pascal tiene características interesantes como el Exception-Handling (tratamiento y canalización de errores en run-time), para el manejo más sencillo de los punteros con reconocimiento automático y referenciación, las llamadas propiedades de objetos que pueden ser asignadas como las variables, etc.
- Para aplicaciones terminadas se pueden crear archivos ejecutables (.exe) que pueden utilizarse solos y sin bibliotecas adicionales, la velocidad con la que pueden ejecutarse los programas creados es muy alta. Excepcionalmente, si se incluyen llamadas a VBX, o DLLs, éstas se deben incluir junto con el ejecutable. También es necesario incluir el BDE (Borland Database Engine) en las aplicaciones de bases de datos, por lo cual no se incluye con analogia.exe.
- Delphi es una "Two-Way-Tool", es decir, una herramienta de dos direcciones, porque permite crear el desarrollo de programas de dos formas: una de forma visual en la pantalla, por medio de las funciones de Drag & Drop (Arrastrar y colocar) y la otra a través de la programación convencional, escribiendo el código. Ambas técnicas pueden utilizarse de forma alternativa o simultánea.

C++
Antecedentes Desarrollado a principios de la década de los 80's por Bjarne Stroustrup de AT&T Bell Laboratories, basado en el lenguaje C. Stroustrup necesitaba un lenguaje que le permitiera aplicar mejor la Inteligencia Artificial, su equipo de trabajo utilizaba C, por lo que lo más sencillo fue diseñar un lenguaje que aceptara la mayor parte del código y construcciones del lenguaje C pero orientado a objetos. A principios de los 80's, Stroustrup, diseñó una extensión del lenguaje C, llamándolo "C con Clases", el término clase provenía de Simula 67, y servía para entender mas el comportamiento del mundo real y llevarlo a los códigos, ocultando los detalles de su implantación. En 1984, C con Clases fue rediseñado en un compilador y se le denominó C ++. En 1985 estuvo disponible la primera versión del lenguaje C ++ y Struostrup realizó el libro The C ++ Programming Languaje (El lenguaje de programación C++), en 1986. En las primeras implantaciones no se tenía un verdadero compilador de C++, sino un traductor de C++ a C (el CFRONT), después salieron compiladores de C++ que se unieron a las características del C++ de AT&T. El nombre de C ++, fue porque éste último era una variante del C original. En el lenguaje C, el operador ++ significa, incrementar la variable, se eligió en nombre C ++, debido a que éste agregaba al C original el término de Programación Orientada a Objetos (POO), basadas en Simula 67. En 1990, el lenguaje fue descrito por Stroustrup y Ellis en el libro Annotated C ++ Reference Manual (C ++ Manual de Referencia con anotaciones se publicó), su versión en español se publicó en 1994. Hoy en día, Borland ofrece el compilador de C++ en la versión 5.5 de forma gratuita.
Concepto El lenguaje de programación C++ (C plus plus o C más más) es una evolución del lenguaje C, que soporta la programación orientada a objetos. Casi todas las características y construcciones del C están disponibles en C++, sin embargo, C++ es un lenguaje por sí mismo y no una simple extensión del lenguaje C.

Características

- Lenguaje híbrido que ha adoptado todas las características de la programación orientada a objetos, esto no perjudica la efectividad de C, más bien mejora las capacidades de C, lo cual dota a C++ de una gran potencia, eficacia y flexibilidad que lo convierten en un estándar dentro de los lenguajes de programación orientados a objetos.
- Es un lenguaje robusto y bien diseñado, expresa las ideas en programas de una manera cómoda y concisa, soporta la abstracción de datos.
- Es un lenguaje de programación de propósito general basado en el lenguaje de programación C, su uso está muy extendido, además existen compiladores para muchos sistemas operativos.
- Comparte la eficiencia del código objeto del lenguaje C.
- Es un lenguaje muy extenso, utilizado sobre todo para programar en la plataforma Windows. Es el precursor de lenguajes tan completos y modernos como Java.
- La mayoría de las librerías portables de interface de usuario son implantados como colección de objetos C++.
- Al ser C ++ una variación de C, los programas codificados en C pueden correr fácilmente en C ++.
- Existe una gran cantidad de juegos comerciales escritos en C++.
- Es mejor que C para escribir programas largos.
- Contiene librerías de estructuras de datos comunes, como listas enlazadas y arreglos dinámicos, que evitan la carga de lidiar con detalles de bajo nivel.
- En ocasiones suele ser extremadamente largo, complicado, o ser mas lento que C. No muchos compiladores implementan el lenguaje correctamente.

EIFFEL

Antecedentes

Eiffel fue creado por Bertrand Meyer y desarrollado por su compañía, Interactive Software Engineering (ISE) of Goleta, CA (Diseño interactivo del software (ISE),de Goleta, CA.)

La experiencia extensiva del Dr. Meyer en la programación orientada a objetos, particularmente con Simula, le permitió agregar conceptos importantes desde su trabajo académico sobre comprobación de software y definición de lenguajes de computadora.

El diseño de Eiffel dirigió muchos intereses prácticos en la ingeniería del software complejo. Eiffel ha evolucionado continuamente desde su concepción del 14 de Septiembre de 1985 y su primera introducción en 1986.

Posteriormente Eiffel se llamó el Gustave Eiffel.

Características

- Eiffel es un lenguaje de programación orientado a objetos avanzado que enfatiza en el diseño y construcción de alta calidad y la reusabilidad del software.
- No es un superset ni la extensión de ningún otro lenguaje.
- Fomenta fuertemente la programación OO y no permite prácticas peligrosas de otros lenguajes previos de generación de interfase con otros lenguajes tal como C y C++.
- Soporta el concepto de "Diseño por Contrato" para mejorar corrección de software.
- Más que un lenguaje, puede verse como un método de construcción de software.
- Es un gran elemento para la educación software.

CLOS
Antecedentes CLOS fue el resultado de la fusión de cuatro ampliaciones al LISP orientadas a objetos a mediados de los años ochenta [STEELE y GABRIEL 1993], New Flavors, CommonLoops, Object LISP y Common Objects.
Concepto CLOS (Common LISP Object System), es una extensión que tiene las características de LISP. Las extensiones del CLOS frente al Common LISP permiten ilustrar el uso de la herencia como mecanismo básico de las representaciones basadas en grafos que se estudian en las clases de teoría. El lenguaje de base LISP no incluye características de abstracción de datos; sin embargo, el Common LISP incluye CLOS, el Common LISP Object System (Sistema de Objetos de Common LISP).
Características • Se maneja herencia múltiple • Funciones genéricas. • Metaclases y metaobjetos • Una técnica de creación e inicialización de objetos que permite al usuario controlar el proceso.

ACTOR
Concepto Lenguaje de programación orientado a objetos para PC, de The Whitewater Group Inc.
Características Corre bajo Microsoft Windows y posee una sintaxis parecida a Pascal para facilitar la transición a los lenguajes orientados a objetos.

JAVA
Antecedentes Antes de que surgiera Java, la única forma de realizar una página web con contenido interactivo era mediante la interfaz CGI (Common Gateway Interface), que permite pasar parámetros entre formularios definidos en lenguaje HTML y programas escritos en Perl o en C, esta forma de programar resultaba muy incómodo y limitada. Java fue pensado originalmente para utilizarse en cualquier tipo de electrodomésticos pero la idea fracasó. Uno de los fundadores de Sun Microsystems Inc. rescató la idea y la enfocó al ámbito de Internet. El propósito era crear un lenguaje que pudiera funcionar en redes computacionales heterogéneas (redes de computadoras formadas por más de un tipo de computadora, ya sean PC, MAC's, estaciones de trabajo, etc.), y que fuera independiente de la plataforma. De esta manera convirtieron a Java en un lenguaje potente, seguro, universal y gratuito. Uno de los primeros triunfos de Java fue que se integró en el navegador Netscape y permitía ejecutar programas dentro de una página web.
Concepto JAVA es un lenguaje de programación orientado principalmente a la programación en Internet o intranets, es un lenguaje de programación para la creación de programas seguros, robustos, multitarea y lo más importante de todo, orientado a objetos. Permite desarrollar aplicaciones muy rápidamente y obtener resultados satisfactorios con muy poco esfuerzo.

Características

- Simple: Es un lenguaje fácil de aprender, aún más que C y C++.
- Robusto: Proporciona numerosas comprobaciones en compilación y ejecución, maneja la memoria de la computadora, manejo de apuntadores, etc., olvidándose el programador de estas tareas.
- Seguro: Java tiene ciertas políticas que evitan que se pueda codificar virus con este lenguaje. Existen restricciones, que limitan lo que se puede y no puede hacer con los recursos críticos de una computadora.
- Portable: Como el código compilado de Java (byte code) es interpretado, un programa compilado de Java puede ser utilizado por cualquier computadora que tenga implantado el interprete de Java. Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, por los que los programas son iguales en todas las plataformas. Estas dos características se conocen como la Máquina Virtual Java (JVM).
- Independiente a la arquitectura: Al compilar un programa en Java, el código resultante es un código binario (byte code), un formato intermedio que es interpretado por diferentes computadoras de igual manera, con arquitecturas y sistemas operativos diversos, diseñado para transportar el código a múltiples plataformas hardware y software, solo se debe implementar un intérprete para cada plataforma.
- Multithreaded (Multihebra): Puede ejecutar diferentes líneas de código al mismo tiempo, esto es de gran utilidad en la creación de aplicaciones de red distribuidas..
- Interpretado y compilado a la vez: JAVA es interpretado y compilado, esto se debe a que únicamente cerca del 20% del código en JAVA es interpretado por el browser, la seguridad de JAVA y la capacidad de correr sobre distintas plataformas se dan por el hecho de que los pasos finales de compilación son manejados localmente. Primero se compila el código fuente en JAVA, se obtiene el "bytecode", de carácter binario y neutral (o independiente de la plataforma), este proceso se hace con el compilador de JAVA. Sin embargo, el bytecode no se encuentra completo hasta que se pone junto con el ambiente de ejecución o runtime, que es usualmente un browser. Los bytecode pueden ser interpretados por la plataforma especifica y el producto final puede ser ejecutado para dicha plataforma.
- Dinámico: Java no requiere que se compilen todas las clases de un programa para que funcione. Si se modifica una clase, Java realiza un Dynamic Bynding o un Dynamic Loading para encontrar las clases.
- Orientado a objetos: Los objetos se agrupan en estructuras encapsuladas (datos y métodos).Utiliza la herencia simple. Todo el código usado por JAVA está dividido en clases, normales y abstractas (interfaces), que definen el comportamiento de un objeto por medio de un conjunto de métodos.
- Distribuido: Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.
- Produce applets: Java puede ser usado para crear dos tipos de programas: aplicaciones independientes y applets. Las aplicaciones independientes se comportan como cualquier otro programa escrito en cualquier lenguaje, como por ejemplo el navegador de Web HotJava, escrito íntegramente en Java. Los applets son pequeños programas que aparecen embebidos en las páginas Web, como aparecen los gráficos o el texto, pero con la capacidad de ejecutar acciones muy complejas, como animar imágenes, establecer conexiones de red, presentar menús y cuadros de diálogo para luego emprender acciones, etc.

Metodologías para el análisis orientado a objetos

Metodologías AOO

- OOSA de Shlaer/Mellor
- Coad/Yourdon
- Rumbaugh - OMT
- Martin/Odell - Ptech
- Objectory CASE
- DESFRAY Método de relaciones de clases
- OORASS
- OSA
- Systems Enginecring 00
- TEXEL
- BON - Nerson
- Fusión - Coleman
- OBA

OOSA (*Object Oriented System Analysis*)
Uno de los primeros ejemplos del análisis orientado a objetos se debió a Shlaer y Mellor (1988), pero este método no podía ser considerado realmente como orientado a objetos por varias razones, una de ellas es la ausencia del concepto de herencia. Este método era poco más que una extensión basada en objetos del modelado de datos. Sin embargo, en 1991 esta metodología presentaba la herencia (subtipos de entidades) y la idea de que se podían descubrir los métodos modelando los ciclos vitales de entidades con STD y las reglas de normalización (reglas de atribución), aplicada a los objetos como si éstos fuesen poco más que tablas relacionales.

El primer paso del método de Shlaer y Mellor es la definición de objetos y de sus atributos. La notación de modelado de entidades desciende de la notación de Ward/Mellor.

Este método se puede clasificar como trilateral y se desarrolla creando un modelo de información (o de datos) que muestra los objetos, atributos y relaciones. Un modelo de estado describe los estados de los objetos y las transiciones entre estados y un DFD define el modelo de procesos. Este modelo tiene grandes influencias sobre el modelo relacional, los objetos se encuentran en la primera forma normal y la identidad de los objetos no es una característica natural en los diseños.
Se trata de un método complejo que trata el análisis y el diseño, y que ha tenido una historia controvertida, pero que se utiliza con cierta amplitud.

Este método tiene apoyo, en cierta parte, de la herramienta CASE llamada teamwork.

Coad/Yourdon

Existe una aproximación que surgió de Yourdon, y que debe mucho a la tradición de modelado de entidades y relaciones; esta aproximación se resume en Coad y Yourdon (1991) y resultó especialmente interesante al ser la primera descripción ampliamente difundida de un método de análisis y una notación de apoyo razonablemente completos, prácticos, orientados a objetos y adecuados para proyectos comerciales. Coad y Yourdon presentan una notación mejor a la de Booch, Shlaer/Mellor o la mayoría de las aproximaciones de diseño orientado a objetos. El énfasis se desplaza mucho hacia el análisis, por oposición al diseño.

Las estructuras de datos de Ada, la descomposición en módulos, o cualquier otra estructura del nivel del lenguaje, no forman parte de este método, que es notable por su sencillez, aunque resulte incompleto en algunos aspectos. Sin embargo, las ideas resultan suficiententemente útiles como para formar la base de un método que, aun cuando vaya evolucionando a lo largo del tiempo, podrá proporcionar beneficios verdaderos en términos de especificaciones reutilizables y extendibles.

Una de las características más notables de la metodología Coad/Yourdon es el que los atributos resultan completamente explícitos, esto viola el principio tradicional orientado a objetos que consistente en que los objetos se especifican solamente mediante sus métodos.

Coad y Yourdon sugieren que el análisis se produce en cinco fases, a las que le dan los nombres siguientes:

- Temas: El área problema se descompone en "temas", que se corresponden con la noción de "niveles" o "capas" en los diagramas de flujos de datos, que equivalen a los subsistemas o categorías de clases de Booch. Estos temas deberían ser de un tamaño tratable, aproximadamente entre cinco y nueve objetos.
- Objetos: Posteriormente se identifican los objetos con detalle, buscando entidades de negocios de forma muy parecida a aquella en que se realizaría un análisis de datos, Coad y Yourdon proporcionan pocas ideas adicionales sobre la forma de llevar a cabo esta tarea.
- Estructuras: Se identifican dos tipos de estructuras: las de clasificación (árboles de especialización y generalización) y las estructuras de composición, ambas manejan el concepto de herencia. Esta metodología no dice mucho acerca de la forma en que se procesan estos árboles, ni tampoco acerca de la forma en que interactúan entre sí.
- Atributos: Los atributos son detallados y se especifican las relaciones de modalidad y de multiplicidad empleando una versión del Análisis Relacional Extendido (ERA). Podemos pensar que es aquí donde se produce un desvío con respecto a otros métodos de diseño orientados a objetos, que solamente especifican métodos.
- Servicios: Esta es la palabra que emplean Coad y Yourdon para las operaciones. Cada tipo de objeto debe estar equipado con métodos para crear o borrar instancias, para tomar o poner valores y también deberá disponer de métodos especiales que caractericen el comportamiento del objeto.

La aproximación de Coad/Yourdon se encuentra muy influenciada por las bases de datos relacionales, razón por la cual sobresalen algunos otros defectos, se insiste en que los atributos deben ser atómicos y esto no es correcto, ya que la orientación a objetos trata de modelar objetos complejos que no tienen que ser atómicos.

Rumbaugh - OMT
La Técnica de Modelado de Objetos (OMT) es considerada como una de las metodologías más completas que se publicaron. Procede del trabajo de James Rumbaugh 1991 y de sus colaboradores en General Electric. Tienen fuertes raíces en los métodos estructurados tradicionales y ofrece una notación extremadamente rica, pero extremadamente complicada y detallada.

La complejidad se compensa, ya que la metodología permite generar código automáticamente. El método es relativamente independiente del lenguaje, aun cuando normalmente está más asociado con desarrollos en C++ que con desarrollos en Smalltalk.

OMT consta de tres fases o actividades principales: análisis, diseño de sistemas y diseño de objetos. El análisis supone que existe una especificación de los requisitos y se desarrolla construyendo tres modelos distintos mediante el uso de tres notaciones diferentes. Lo primero que construye es el Modelo Objeto (OM) que consta de diagramas similares a los de Coad/Yourdon, y de un diccionario de datos. La notación es, fundamentalmente, la del modelado ER, con operaciones y otras anotaciones que se han añadido.

Posteriormente se construye, para cada objeto, un Modelo Dinámico (DM), que consta de STD dibujados en una notación de Harel extendida, y de diagramas de flujo globales de sucesos. El tercer paso no es muy utilizado, la mayoría de sus usuarios parecen emplearlo a un nivel alto, aproximadamente de la misma forma en que utilizaría un analista convencional los diagramas de contexto (DFD de contexto a nivel cero). Se trata del Modelo Funcional (FM).

Una vez completados los tres (o los dos) modelos, las operaciones descubiertas se copian en el OM y se puede pasar al diseño de sistemas y de objetos.

El Diseño de Sistemas se realiza organizando los objetos en subsistemas, identificando la concurrencia a partir del DM, asignando subsistemas a procesadores o tareas, decidiendo si los datos deben o no estar almacenados en archivos, en memoria o en un sistema de administración de bases de datos, decidiendo acerca del uso de periféricos y de recursos globales.

El Diseño de Objetos implica transformar la información del DM y del FM en operaciones de OM.

Dada la importancia que tuvo esta metodología la consideramos como una de las precursoras más importantes de UML, razón por la cual se describirá a detalle más adelante.

Martin/Odell - Ptech
Ptech, creado por Associative Desigit Technology, es un conjunto propio de métodos y herramientas que abarca tanto el análisis como el diseño. Posee una cantidad de características comunes con las aproximaciones orientadas a objetos. La herramienta CASE de Ptech también genera código para C++.

Las ideas que nacen en Ptech están basadas en la metáfora de la ingeniería de procesos como producción de sistemas, ensamblando componentes reutilizables, una aproximación que separa el diseño lógico y el análisis de la realización. El hincapié se hace, por tanto, en lo que se hace, y no en cómo se hace. En ese sentido, esta metodología está orientada a procesos, más que estrictamente a objetos, aun cuando las aproximaciones pueden poseer muchas características comunes.

Ptech hace hincapié en la realización de prototipos a lo largo del desarrollo, su sistema notacional consta de tres tipos de diagramas: *Diagramas de conceptos* (que equivalen a los diagramas extendidos de entidad y relación en el estilo genérico de Bachman) y llamados esquemas de objetos o esquemas de conceptos; *Diagramas de sucesos o esquemas* (que desempeñan el papel de diagramas de transición de estado); y *Diagramas de actividad/función* (cuyo propósito es muy parecido al de los DFD). Las dos primeras notaciones de generación de diagramas son admisibles para los lenguajes formales, para el cálculo de clases y para el cálculo de sucesos.

Los diagramas de concepto muestran los aspectos estáticos de los procesos y los diagramas de los sucesos muestran su dinámica. Es posible ilustrar la forma en que se crean y destruyen las instancias, también la forma en que éstas entran y salen de las clases empleando un diagrama combinado de conceptos y sucesos. Los diagramas de actividad proporcionan una visión funcional de alto nivel que se parece, en cierto modo, a la que proporciona un diagrama de flujo de datos. Los conceptos se consideran como conjuntos y se utiliza la teoría de conjuntos para describir los conceptos empleando operaciones sencillas de conjuntos, tales como la unión, la intersección y la diferencia.

Objectory CASE (1992)
También conocida como Ingeniería del Software Orientada a Objetos (OOSE), es un método para el análisis y diseño orientado a objetos que surgió a partir de las ideas del método Objectory de Jacobson. Objectory es un método propio y OOSE es una versión simplificada.

Objectory es poco usual entre los métodos OO, por cuanto intenta resolver todo el desarrollo de software. Se derivó originalmente de una experiencia en la construcción de sistemas de control telefónicos en Ericsson utilizando técnicas de diseño por bloques, es uno de los métodos orientados a objetos más antiguos.

Esta metodología sostiene que las herramientas proporcionan apoyo a las actividades en tres categorías: arquitectura, métodos y procesos.

Una *Arquitectura* representa la selección de técnicas que hay, el *método* hace explícitos los procedimientos que deberán seguirse, el *proceso* proporciona el avance del método. El desarrollo es programado e implica ciclos iterativos de análisis, construcción y pruebas. Una iteración representa un cambio realizado en un sistema que lo hace evolucionar, implica el control de la versión y la documentación en los niveles y módulos de los sistemas.

OORASS (Análisis, Síntesis y Estructuración de Papeles Orientada a Objetos)
Desarrollada por Reenskaug (1989, 1990) y otros en Taskon AS, Noruega y sigue siendo propio de la empresa. Abarca el análisis y diseño, es poco habitual, hace hincapié en los papeles desempeñados por los objetos y resuelve varias partes del ciclo vital evolutivo.

El análisis comienza con el descubrimiento de áreas de interés, sus efectos, funciones de negocio de alto nivel y la unidad para formar subsistemas coherentes.
Posteriormente cada área se modela utilizando agentes y objetos que colaboran y que pueden adoptar varios papeles. Los papeles se obtienen a partir de todos los objetos que posean la misma posición dentro de la estructura de una zona de interés. Se interpretan los papeles de forma similar a los papeles

desempeñados por los agentes de Objectory, pero la idea se extiende hasta las interioridades del sistema en lugar de restringirse a agentes externos.

Los objetos piden adoptar distintos papeles en diferentes contextos. Los papeles poseen requisitos de recursos, competencias, deberes y derechos. El papel X puede conocer al papel Y, esto es, puede contener una referencia que haga posible enviarle mensajes. Estos enlaces cliente/servidor se refinan mediante símbolos de puerto. El puerto de un solo círculo indica que el papel X conoce entre cero y una instancia del papel Y.

Cada símbolo de puerto puede tener asociado un conjunto de operaciones denominadas contrato, que definen lo que puede ofrecer el papel visible. OORASS es neutro acerca de la notación que debe emplearse para la representación de objetos.

El modelado queda complejo cuando los analistas comprenden en su totalidad la zona de interés y el modelo es estable. En caso contrario, o bien si el área no se descompone, el analista deberá redefinir el área y repetir la actividad.

Los tipos de objetos, se sintetizan a partir de papeles, esto define el comportamiento visible de algunos objetos. La síntesis crea nuevos objetos que heredan su comportamiento de otros más sencillos.

Desde el punto de vista de la terminología, en OORASS se definen las clases como la realización de tipos y admite que las redes de tipos difieran de las redes de clases. Se admite la herencia múltiple.

OORASS es apoyado por una herramienta CASE denominada OORAM, que también está disponible en Taskon. El método está orientado a la construcción de sistemas, haciendo especial hincapié en el paso y distribución de mensajes.

DESFRAY Método de relaciones de clases
Desfray (1992) describe el método de relaciones de clases para el análisis y diseño orientado a objetos que, en la actualidad, es comercializado por una compañía francesa, Softeam, en el contexto de proyectos C++, para el cual ha sido diseñado específicamente. El método se caracteriza por el uso de una notación derivada de los modelos de relación de entidad de Chen y por tanto, por su potencial compatibilidad con Merise, un método estructurado de amplia utilización en Francia. También resulta notable, porque presta atención a los métodos formales mediante el uso de afirmaciones parecidas a las de Eiffel. Está disponible una herramienta CASE, que genera códigos en C++.

El método de relaciones de clases es una aproximación "trilateral" mientras existen tres modelos distintos para cada sistema: un modelo de objetos/entidades, un modelo de transiciones de estados y un modelo de flujo de datos. Se hace hincapié en el concepto de encapsulación mucho más que en OMT. Desfray sugiere que el modelo de transición de estados se podría mejorar mediante la introducción de estados específicos y de un mecanismo para la síntesis de estado. En términos de lo que abarca el método, los modelos de entidad y relación contienen un apoyo importante para la herencia, además de las ya estructuras familiares para el modelado de datos. Aun cuando las relaciones se pueden considerar como objetos, Desfray hace una distinción importante entre clases y relaciones, éstas últimas nunca poseen métodos, su única misión es la de conectar clases.

Una relación es un enlace dirigido entre dos clases, que permite a una clase saber acerca de las instancias de la otra clase. No se recomienda la utilización de relaciones bidireccionales. También afirma que una relación es parte de la definición del concepto que representa la clase.
Se considera que las clases están formadas por una composición o agregación de sus atributos, considerados como tipos de objetos. Una regla general en el modelado de entidades en que los únicos atributos autorizados son aquellos que pertenecen a una clase base. Esto ayuda a evitar la confusión que surge de mezclar dominios, o de la herencia múltiple, dentro de este método no se permite que un elemento forme parte de dos clases.

OSA (Análisis de Sistemas Orientados a Objetos)
Metodología desarrollada en Hewlett-Packard, es parecida al OMT de Rumbaugh. Este método sigue la división tripartita del análisis en tres actividades separadas, con una notación independiente para cada una.

OSA comienza mediante una notación de entidades que permite la descripción de atributos, la clasificación y agregación de estructuras así como la semántica de datos, en forma de asociaciones. Las restricciones más generales se escriben en los diagramas en forma de notas, junto a los objetos con los cuales están relacionadas, así las ligaduras no poseen relación con herencia.

La notación de transición de estados permite al analista describir el comportamiento de cada objeto y el paso de mensajes se describe en un modelo de interacción parecido al del flujo de datos (OIM). La ventaja con respecto a OMT es que este modelo resuelve de forma más sencilla el paso de mensajes que los flujos de datos no temporales.

Systems Engineering 00
Creado por LBMS, también conocido como SEOO, es el método propuesto por la compañía inglesa LBMS. Posee cuatro aspectos:

1) Desarrollar la descomposición de estructuras y técnicas
2) Un método compartido para el modelado de objetos
3) Técnicas de diseño específicas para GUI
4) Enlaces con las bases de datos relacionales

Se considera que un método es un modelo de proceso más técnicas con sus reglas y productos finales. Propone un desarrollo estándar rápido, hace especial hincapié en las etapas finales, es decir en los ejecutables que puede ser evaluados por los usuarios.

Una de las grandes ventajas de un sistema convencional de administración de base de datos es la separación de los datos y los procesos, lo cual proporciona una noción de independencia de datos y se beneficia de la flexibilidad y aislamiento del cambio, porque la interfaz con los datos compartidos es estable.

El modelo de datos debe ser considerado como un modelo del aspecto estático de la aplicación. Mediante la orientación a objetos se integran los procesos y la administración de datos, parece razonable entonces, adoptar el punto de vista consistente en que existen dos clases de objetos, *objetos de dominio y objetos de la aplicación.* Los primeros representan los aspectos del sistema que son relativamente estables o

genéricos, visión que posteriormente incluirá, también, limitaciones, reglas y dinámica. El objetivo es hacer que la interfaz de esta parte del modelo sea lo más estable posible. Los objetos del dominio forman el modelo compartido de objetos. La mayor parte de la interacción entre componentes, se realiza a través de este modelo y por esta razón, SEOO los denomina objetos compartidos.

Este tipo de modelado de datos resulta interesante porque:

- Se utiliza el análisis de datos como herramienta de modelado. Los analistas utilizan modelos de datos como base para comprender la funcionalidad. El modelo de datos es estable con respecto a los requisitos de información que apoya.
- Los componentes de procesamiento, ya sean subsistemas completos o transacciones individuales, interactúan entre sí a través de los datos compartidos descritos por el modelo de datos y contenidos en la base de datos. El aislamiento y estabilización de la mayor parte de las interfaces importantes del sistema, desde una etapa tan temprana, es un gran beneficio.
- Existe un camino directo, que partiendo del modelo de datos, llega al diseño preliminar de la base de datos.

TEXEL
Método de análisis y diseño orientado a objetos con fuertes raíces de desarrollo en Ada. El método está soportado por P.P. Texel y la compañía New Jersey y por su creador Putman Texel.
Este método es utilizado en aplicaciones militares o de tiempo real, se dice que las aplicaciones de los MIS (Managament Information System, Sistemas de Información Administrativos) están soportadas por estos métodos.
En esta metodología se nominan, en primer lugar, los tipos de objetos candidatos, los atributos y así sucesivamente esto puede admitir como entrada una especificación escrita (documentos del sistema) tales como los DFD o entrevistas con expertos del dominio. En esta etapa se escriben absolutamente todos los nombres, verbos, etc. No se recomienda ninguna técnica para la entrevista, ni tampoco para la identificación de objetos.

Telxel se clasifica como un sistema de modelado temario con modelos ER, STD y de flujo de mensajes independientes. Sus raíces en el tiempo real se deben a la importancia que se asigna al uso de STD, en lugares en que normalmente no está permitido definir operaciones antes de obtener un modelo de estados. Este método está íntimamente relacionado con la aproximación de Shlaer/Mellor y se basa en Booch. Al igual que la mayoría de los métodos, no abarca todos los aspectos del modelo referencial OMG y nada se dice acerca de la identificación de objetos, de la administración de componentes, de las métricas y de las comprobaciones, salvo por el uso de matrices de objetos/requisitos parecidas a CRUD, a las cuales se asignan como anotaciones posteriores.

El problema principal de este método consiste en que es difícil ver cómo podría resultar útil cuando se efectúen diseños para un lenguaje que sea distinto de ADA, tal como Smalltalk, Eiffel o algún lenguaje 4GL orientado a objetos.

Texel posee el apoyo de una herramienta CASE que adopta la forma de una herramienta monousuario denominada Texel SF y que ha sido construida en el sistema meta - CASE denominado VFS.

BON (Notación de Objetos Mejorada)
Descrita por Nerson en 1992, es uno de los pocos métodos en que se da importancia a las reglas de negocio y a los invariantes de clases. Esta es una derivación de la escuela de pensamiento de Eiffel.

Esta metodología posee influencias de Booch, Coad/Yourdon, Page-Jones y Constantine, Shlaer/Mellor, OMT, OOSD, CRC lo cual hace que BON sea un híbrido.

Las actividades contenidas en BON son:
- Definir los límites del sistema
- Identificar los candidatos de clases
- Agrupar las clases en agrupamientos
- Definir los candidatos de clases en términos de preguntas, órdenes y limitaciones
- Definir el comportamiento de cada clase en términos de sucesos, protocolos de comunicación de objetos y cartas de creación de objetos
- Definir características de las clases, invariantes y relaciones contractuales
- Refinar las descripciones de clases
- Desarrollar estructuras de clasificación

BON posee varias ventajas que no se encuentran en muchos otros métodos, tales como la administración de componentes en las reglas, BON tiene similitudes a la métodología SOMA.

Fusión - Coleman
Este es un método híbrido desarrollado por un equipo lidereado por Derek Coleman en los laboratorios Hewlett - Packard. HP realizó una investigación de los métodos y el principal requisito identificado era el de un modelo de utilización que poseyera una notación sencilla.

Las principales influencias de esta metodología son la notación OMT y el modelado de interacción CRC, ideas de Booch'91 y algunas ideas de las escuelas formales Z y VDM. Los diagramas de flujo de datos se utilizan para el modelo funcional, porque resultan demasiado operacionales, en esta metodología no se presentan los métodos mientras no se alcanza la fase de diseño.

OBA
Análisis de Comportamiento de Objetos, ha sido descrito por Rubin y Goldberg 1992. OBA es un método que empieza por obtener guiones procedentes de entrevistas o de documentos desarrollando con esto un modelo de contexto, por lo que se puede empezar a identificar a los participantes, sus responsabilidades y se pone de manifiesto los indicadores.

Se utilizan CRC modificadas para registrar los detalles de los objetos y para descubrir estructuras de clasificación y utilización y otras asociaciones, aunque no se hace especial énfasis en la agregación. Se desarrollan modelos de transición de estados de Harel para cada objeto.

FOA o Análisis Orientado a Marcos (1992)
Es un método que se deriva del modelado conceptual en aplicaciones avanzadas de bases de datos e inteligencia artificial y hace especial hincapié en el uso de redes semánticas. Abarca gran parte del ciclo de vida, desde los requisitos hasta la comprobación y es poco habitual por cuanto hace especial énfasis en las reglas de negocio. En este método un marco es un componente principal de un sistema de información identificado mediante una red semántica que consta de objetos y asociaciones.

La ventaja del FOA es la importancia que le da a las reglas de negocios y el uso de limitaciones de bases de datos y de activadores para representarla. El método resulta especialmente adecuado para la construcción de sistemas de bases de datos, en las cuales debe utilizarse C++ como lenguaje de desarrollo.

CGI Yourdon
Metodología que constituye la versión moderna de la familia de métodos original de Yourdon, basada en DFD, que es mantenida en la actualidad por CGI. Ha propuesto una extensión orientada a objetos de los métodos estructurados de Yourdon, esta metodología se basa fuertemente en la utilización de la notación DFD para describir el comportamiento interno de los objetos.

El punto de vista externo del objeto se representa empleando una notación que muestra la identidad del objeto y sus operaciones visibles. La herencia y las asociaciones se representan mediante diagramas similares a los de Chen.

Henderson - Sellers
Notación que se ve influida por la Notación Uniforme de Objetos, la cual es más sencilla. El método es un híbrido para el análisis orientado a objetos, hace especial hincapié en incorporar los métodos estructurados existentes siempre que sea posible. Proporciona una notación para todas las características estructurales importantes de los sistemas orientados a objetos.

Este método incluye líneas generales para la realización en varios lenguajes orientados a objetos, este método ha sido refinado y extendido hace poco tiempo con el nombre de MOSES.

MOSES (Metodología para la Ingeniería de Software Orientada a Objetos)
Metodología desarrollada por Henderson - Sellers y Edwards en 1993, está basada en una extensión de la notación uniforme orientada a objetos. Proporciona un extenso modelo del ciclo vital, hace hincapié en la continuidad de la representación, en las líneas generales para la orientación de proyectos y extensibilidad.

ADM3
Desarrollada por Firesmith en 1993, es una extensión de un modelo anterior orientado a Ada, enfatiza en el desarrollo de sistemas de tiempo real. Consta de notaciones para el modelado de estados, para el modelado de control y para el modelado temporal, utiliza ideas procedentes de las redes semánticas.

BERARD (1993)
Modela el ciclo de vida orientado a objetos razonablemente extenso que simula en algunos aspectos a MOSES. Una vez más está presente la influencia de las redes semánticas en la forma de las especificaciones de objetos de clases de Berard, éstas constan de una descripción breve y concisa, distintas representaciones gráficas que incluyen redes semánticas y diagramas de transición de estados.

SYNTROPY
Es un método propio desarrollando por Steve Cook, John Danielsd y los colegas de Object Designers Limited, el método afirma ser independiente de la notación, pero utiliza la notación de OMT, hace hincapié en la aproximación orientada al comportamiento, éste método es clasificado como híbrido.

SOMA
Método rico de análisis orientado a objetos, posee el apoyo de objetos difusos y la herencia.

Las siete actividades de las que consta son:
- Identificar capas
- Identificar objetos
- Identificar estructuras de utilización, clasificación y composición
- Definir la semántica de datos y las asociaciones
- Añadir atributos a los objetos
- Añadir operaciones a los objetos
- Añadir semántica declarativa de objetos

Existen otros métodos tales como : COOSD, ORCA, DONT, ALEX, OBJ, MOOD, OSDL, OSMOSYS, SYS-P-O.

Metodologías para el diseño orientado a objetos

Metodologías DOO

- GOOD
- HOOD
- OOSD
- OODLE
- JSD y OOJSD
- Booch (1991)
- CRC y RDD

GOOD (General Orientado a Objetos)
Método de diseño desarrollado en la NASA por Seidewitz y Stark en 1983. Trata la especificación de requisitos y el diseño de proyectos en Ada. El método se desarrolla partiendo de un conjunto preliminar de diagramas de flujo de datos y de control. Las clases se descubren examinando el flujo de datos y de control.

Con el estudio de los procesos principales se obtiene un modelo abstracto del funcionamiento del sistema, así como las entradas y salidas de flujos asociados a estos procesos, se puede construir un conjunto de diagramas por capas. Estos diagramas hacen especial énfasis en las relaciones de control y de datos entre entidades. De esta manera, las entidades y los agrupamientos de entidades, pasan a ser los objetos y las transformaciones de los datos originales pasan a ser los métodos.

HOOD
Utiliza una notación jerárquica de prioridades y está muy orientado al desarrollo en Ada, fue desarrollado en la Agencia Espacial Europea, posee la influencia directa de GOOD, se basa en el método de máquinas abstractas. Hace énfasis en las jerarquías de composición (parte-de), no trata nada acerca de la clasificación de jerarquías (herencia). En HODD los objetos se clasifican en "pasivos" o "activos".

HOOD es un método de refinamiento progresivo que se basa en la descomposición de un objeto del máximo nivel y posteriormente en la descomposición de los objetos resultantes.
El método HOOD utiliza un cierto número de pasos para descomponer cada objeto, comenzando por un paso de diseño básico que podría implicar técnicas de realización de diagramas procedentes de otros métodos de análisis y diseño estructurado propuesto por Yourdon.

HOOD a pesar de ser adecuado para muchas aplicaciones militares de tiempo real, no lo es para las aplicaciones comerciales. La falta de apoyo para la herencia lo hace basado en objetos, más no orientado en objetos. Por lo tanto se apoya en la reutilización, pero no en la extensibilidad.

OOSD (Método de Diseño Estructurado Orientado a Objetos)
Presentado por Wasserman, Picher y Muller (1990). OOSD no es un método sino una notación a la cual se le pueden agregar reglas metodológicas. Esta notación es, probablemente, la más próxima a la orientación a objetos, en cuanto admite la herencia, así como la abstracción.

OOSD es una notación no propietaria para el diseño arquitectónico, combina el diseño por refinamiento progresivo estructurado y el diseño orientado a objetos su objetivo es apoyar las metas de reutilización, la modularidad, la extensibilidad y la representación de la herencia y de la abstracción.

Esta metodología apoya a la representación visual de interfaces entre componentes del diseño, a la generación del código, a la independencia del lenguaje, a la comunicación entre diseñadores y usuarios. OOSD emplea una notación que se deriva de la de Booch.

En OOSD, los métodos se pueden añadir a la subclase o se pueden invalidar, sin embargo, no pueden ser eliminados.

OOSD es una de las notaciones de diseño más avanzadas híbrido orientado a objetos de bajo nivel. Parece improbable que sea posible extenderla hasta una notación de análisis congruente, debido a la dificultad que surge al tratar un gran número de métodos y a la ausencia de una forma en la que se puedan tratar estructuras y atributos de datos muy complejos. Por otra parte, resulta más adecuada para el diseño arquitectónico o lógico que para el diseño físico.

JSD (Diseño Estructurado de Jackson)
Surge como resultado del trabajo desarrollado por Jackson en 1983, es un método basado en objetos más que un método completamente orientado a objetos. Los modelos JSD se descomponen en términos de sucesos o de acciones y de sus dependencias temporales.

Dentro de estos sucesos, la aproximación JSD define, en primer lugar, los objetos. El paso siguiente construye una especificación en términos de procesos secuenciales que se comunican y pueden acceder unos al estado de otros. De esta manera, se viola el principio de ocultamiento de información.

Es posible identificar un cierto número de similitudes entre JSD y los métodos de diseño orientados a objetos. JSD contiene técnicas útiles para la identificación de entidades y de métodos dentro de su fase de modelado. Además, la técnica de análisis por ordenación temporal de JSD se puede considerar como una forma de documentar las clases. JSD y el diseño orientado a objetos utilizan el concepto de los objetos de forma similar, aun cuando sus terminologías son distintas.

Se podría esperar, como consecuencia, que JSD evolucione hacia en estilo más basado en objetos, aunque no orientado a objetos. Ciertamente, las ideas de Jackson han tenido gran influencia en el desarrollo de otros métodos orientados a objetos.

Booch (1991)
La notación y el método de diseño de Booch constan de cuatro actividades principales y seis notaciones. Los primeros pasos tratan los aspectos estáticos del sistema, tanto en su aspecto lógico como en el aspecto físico. La dinámica se modela con los diagramas de transición de estados.

Las técnicas de análisis estructurado convencionales o el análisis orientado a objetos son precursores admisibles del diseño orientado a objetos, pero se advierte a aquellos desarrolladores que utilizan el análisis estructurado deben resistir la tentación de volver a caer en la concepción mental de diseño estructurado.

Booch aportó una notación bastante rica para las relaciones entre clases, utilizó distintas tipos de líneas para indicar la utilización, la herencia y otras relaciones. Cada clase, en el método de Booch, se describe rellenando una plantilla estándar que contiene su identidad, atributos, métodos.

Los diagramas de clases de objetos, así como las plantillas asociadas, describen el diseño lógico o estático del sistema. Booch describe la parte dinámica del sistema de dos maneras: con los diagramas de transición de estados muestran la dinámica de clases, técnica compartida con varios de los métodos de análisis orientados a objetos. La dinámica del nivel de instancias se muestra mediante diagramas temporales que se han tomado del campo del diseño de hardware.

El método de diseño de Booch es uno de los mejor desarrollados, se considera superior a GOOD y a HOOD, puesto que no está relacionado con Ada.

OODLE (Lenguaje de Diseño Orientado a Objetos)
Es un componente específico de diseño del método Shlaer/Mellor. Esta metodología utiliza cuatro tipos de diagramas, está interrelacionado mediante un esquema de capas que ayuda con la documentación.
En OODLE la notación es diferente del lenguaje, aun cuando se aprecian ciertos aspectos de ADA en la notación. Los tipos de diagramas que utiliza son los siguientes:

Diagramas de dependencia
Diagramas de clase
Diagramas de estructuras de clase
Diagramas de herencia

OODLE se basa en la idea de recursividad, esta idea ha sido motivada por varios problemas, ya que con frecuencia las clases no encajan bien entre sí, a pesar del hecho consistente de la reutilización, está basada solamente en interfaces externas. El diseño por tanto debería ser más recursivo que iterativo y deben realizarse utilizando un método con semántica precisa, siempre que sea posible.

El método Booch es uno de los más importantes que surgieron, por lo que lo consideramos uno de los antecedentes más importantes que dieron origen a la notación UML, por lo que se describirán más adelante las aportaciones de Booch para la evolución de UML.

CRC (Tarjetas de clase, responsabilidad y colaboración) y RDD
Las tarjetas CRC, desarrolladas por Beck y Cunningham (1989), descritas en su totalidad en Wirfs-Broock, son una herramienta útil para documentar diseños orientados a objetos y también para enseñar los conceptos básicos; esta técnica también es conocida como RDD (Diseño Controlado por responsabilidades). Posee una gran ventaja, ya que no es caro, sólo utiliza fichas de cartón como herramientas CASE, aun cuando la idea original fuera la de utilizar un sistema de hipertexto.

La aproximación CRC presupone la existencia de especificaciones en forma escrita y se desarrolla empleando un análisis textual, similar al de Abbott, para dar nombre a los objetos claves. Para cada objeto se representa una clase, se prepara una tarjeta que contiene el nombre de la clase y una lista de superclases y de miembros. Posteriormente se hace un análisis textual para conocer las responsabilidades o métodos que necesita cada clase.

Las responsabilidades se dividen en atributos que representan el estado del objeto y las operaciones que puede llevar a cabo cada objeto. CRC es un método simple pero muy práctico, gran parte de su notación es bastante inadecuada y no abarca todo, sin embargo resultan de gran utilidad para la identificación de las clases, motivo por el cual se detallará un poco más sobre el tema más adelante.

Hardware Orientado a Objetos
Tuvo sus inicios hace más de veinte años, comenzando con la invención de arquitecturas basadas en descriptores y, posteriormente, arquitecturas basadas en capacidades. Estas arquitecturas representaron una ruptura con las arquitecturas clásicas de Von Neumann, y surgieron como consecuencia de los intentos realizados para eliminar el hueco existente entre las abstracciones de alto nivel de los lenguajes de programación y las abstracciones de bajo nivel de la propia máquina. Según sus promotores, las ventajas de tales arquitecturas son muchas: mejor detección de errores, mejora en la eficiencia de la ejecución, menos tipos de instrucciones, compilación más sencilla y reducción de los requisitos de almacenamiento.
Entre los computadores que tienen una arquitectura orientada a objetos pueden citarse el Burroughs 5000, el Plessey 250 y el Cambridge CAP, SWARD, el Intel 432, el COM de Caltech el IBM.System/38, el Rational R1000, y el BiiN 40 y 60.

Sistemas operativos orientados a objetos
El trabajo de Dijkstra con el sistema de multiprogramación THE fue el primero que introdujo la idea de construir sistemas como máquinas de estados en capas. Otros sistemas operativos pioneros en orientación a objetos son Plessey/System 250 (para el multiprocesador Plessey 250), Hydra (para el de CMU), CALTSS (para el CDC 6400), CAP (para el computador Cambridge CAP), UCLA Secure UNIX (para el PDP 11/45 y 11/70), StarOS y Medusa (para el Cm de CMU), iMAX (para el Intel 432), el proyecto Cairo de Microsoft y el proyecto Pink de Taligent.

Bases de Datos orientadas a objetos (BDOO)
Las BDOO son un modelo de bases de datos que soportan el paradigma orientado a objetos, por lo que almacenan y manipulan información, la cual puede ser digitalizada y representada por objetos, proporcionando una estructura flexible, con acceso ágil, rápido y una gran capacidad de modificación, lo que permite al usuario olvidarse de los detalles de implantación de bits y bytes, dedicándose a pensar más, en función de objetos reales y operaciones sobre los mismos.

Las BDOO surgieron como resultado de la convergencia de dos disciplinas de investigación: El modelo de datos semántico y los lenguajes orientados a objetos; así como por las limitaciones del modelo relacional.

Las bases de datos orientadas a objetos surgen, en un principio para soportar la programación orientada a objetos. Las BDOO se volvieron persistentes para ciertos tipos de aplicaciones con datos complejos como CAD (Diseño a Asistido por Computadora) y CAE (Ingeniería Asistida por Computadora).

Las bases de datos orientadas a objetos parecen ser el futuro, pero posiblemente la mejor respuesta a los requerimientos debido al aprovechamiento conjunto de ambas tecnologías.

Estas disciplinas se desarrollaron independientemente pero en años recientes se han ido mezclando con importantes implicaciones para el procesamiento de BD.

El desarrollo del modelo semántico de datos fue enunciado por Hull y Hang en 1987 con el propósito de incrementar la efectividad y exactitud del diseño de las BD. El modelo semántico resultó ser apropiado para manejar varios problemas de los usuarios y podían ser fácilmente convertidos a registros basados en la implantación de modelos tales como BD jerárquicas, de redes y relacionales. Este modelo estuvo interesado, primeramente, en las estructuras de los datos.

Por otra parte los desarrolladores de lenguajes de programación orientados a objetos estuvieron más interesados en la forma en que los lenguajes manipulaban los datos (consultas, cálculos, actualizaciones), más que en la estructura de los datos.
Posteriormente los investigadores empezaron a aplicar los conceptos de los lenguajes OO a las estructuras semánticas de datos, el resultado de esta convergencia fue la noción de una Base de Datos Orientadas a Objetos. En esta fusión de disciplinas, la terminología orientada a objetos ha predominado, por lo que se habla de objetos mas que de entidades, como si estuviéramos usando la terminología semántica.

Los DBMSOO (Data Base Managament System Oriented Object, Sistema Manejador de Base de Datos Orientada a Objetos) nacieron al inicio de la década de los 80's, en ese entonces tenían varias características que limitaban su empleo en el área comercial. En primer lugar, parecía que el usuario ejecutaba un número limitado de transacciones, en comparación con las transacciones frecuentes y de gran volumen que permitían muchas aplicaciones comerciales. En segundo lugar, los usuarios de ingeniería y los programadores, accedían a los objetos a través de un browsing pero los usuarios comerciales requerían utilidades de consulta fáciles de utilizar como SQL.
Los DBMSOO son básicamente similares a las generaciones previas de DBMSs (Data Base Managament System, Sistema Manejador de Base de Datos), sin embargo, tienen la ventaja de almacenar objetos, lo que les permite un mejor manejo de las estructuras de datos.
Los vendedores de DBMSOO proponían alternativas al SQL sin tener éxito, ya que SQL se ha convertido en el estándar del área relacional; finalmente, los primeros DBMSOO eran de bajo rendimiento, haciendo las bases de datos inadecuadas para aplicaciones de gran escala y de gran volumen.
Con el tiempo varios productos nuevos y mejoras en las versiones iniciales han corregido muchas de estas limitaciones, evolucionado hasta nuestros días, ofreciendo las ventajas de la orientación a objetos aunadas con las que ofrecen los DBMS tradicionales. Una BDOO almacena, modifica y recupera objetos por orden de un programa de aplicación, dichos objetos pueden ser tan simples como una cadena de caracteres, números o bien tan complejas como las especificaciones completas de una tarjeta de circuitos, o una aplicación multimedia (sonido, gráficos y video). Una de las principales ventajas que tiene una BDOO sobre sus predecesoras es la capacidad de poder representar como objeto cualquier entidad del mundo real.
Con el tiempo las BDOO han ido evolucionando cada vez más retomando lo mejor de sus predecesoras (relacional, jerárquica, de redes) para formar así, una mejor BD que se ha ido introduciendo poco a poco en el mercado comercial.

Ventajas:

- Gran flexibilidad para definir nuevos tipos de objetos: Un objeto puede contener otros objetos en cualquier nivel de anidamiento.
- El manejo de objetos permite un fácil desarrollo de aplicaciones: todos los datos de una entidad están localizados en un sólo lugar, el desarrollador no necesita buscar a través de múltiples tablas relacionadas para determinar dónde están almacenados ciertos objetos.
- Los objetos pueden almacenar relaciones (enlaces a otros objetos) y pueden almacenar el comportamiento (métodos).
- Las BDOO soportan tipos de datos más variados que el modelo relacional.
- Los objetos representan entidades u objetos del mundo real, objetos en el sentido de combinaciones encapsuladas de estructuras de datos (atributos y propiedades) y procedimientos asociados (métodos) que describen su comportamiento.

- El mapeo uno a uno reduce la distancia semántica entre el mundo real y el modelo utilizado para representarlo dentro de la computadora. Más aún, asociado a un estilo de programación 00 el DBMSOO reduce la diferencia semántica entre el programa y la BD que lo soporta.
- Los objetos hacen que las aplicaciones se ejecuten rápidamente: Los datos de una entidad están relacionados lógicamente, la BDOO tiene los medios para optimizar su localización física, las aplicaciones son capaces de leer menos archivos para recuperar todos los datos relevantes.
- El diseño entero y todos sus componentes puede almacenarse en un objeto, reduciendo grandemente el número de operaciones de archivo necesarias para acceder al diseño.
- Permiten conjuntos arbitrarios de objetos, dichos conjuntos de objetos pueden manipularse por el usuario o por la aplicación, bloqueados para la administración de transacciones, o agrupados para optimizar el rendimiento y facilidad de acceso.

Metodología 00ADA (Object Oriented Analysis and Design with Aplications (Análisis y Diseño Orientado a Objetos con Aplicaciones)

Grady Booch (M. Ing. Computación, 1979), desarrolló el método *OOADA*, también llamado *Método Booch*, es considerado como método líder en el análisis y diseño orientado a objetos.

Booch modela un diseño orientado a objetos desde dos puntos de vista:

1. **Lógico:** Donde define clases, objetos y sus relaciones.
2. **Físico:** Define la arquitectura de módulos y procesos.

OOADA se divide en un micro y macro proceso. El macro proceso controla al micro proceso, ya que direcciona actividades para todo el equipo de desarrollo en la escala de semanas o meses. Al inicio del proceso, el desarrollador debe realizar ciertas actividades diarias (no secuenciales):

- Identificación de clases y objetos a un nivel dado de abstracción.
- Identificación de la semántica de estas clases y objetos.
- Identificación de las relaciones entre clases y objetos.
- Implantación de las clases y objetos.

Actividades del macro proceso:

- Conceptualización. Se establecen los requerimientos.
- Análisis. Desarrollo del modelo del comportamiento deseado.
- Diseño. Se crea la arquitectura.
- Evolución. Desarrollo de la implantación.
- Mantenimiento. Se maneja la evolución de las entregas de versiones futuras.

Booch sostiene que el proceso de diseño orientado a objetos de un sistema se compone de incrementos e iteraciones, es decir, es un proceso evolutivo, incremental e iterativo.

Diagramas utilizados en esta metodología:

1. Diagrama de Clases: Se trata de una variación de los diagramas de entidad relación en los que se añaden nuevos conceptos como clases, clases paramétricas, utilidades y metaclases. Los tipos de relaciones son asociaciones, herencia, uso, instanciación y metaclase. Permite agrupar las clases y relaciones en categorías para diagramas demasiado complejos.

2. Especificación de Clases: Captura toda la información importante acerca de una clase en formato texto.
3. Diagrama de Categorías: Muestra clases agrupadas lógicamente bajo varias categorías.
4. Diagramas de transición de estados. Permiten definir cómo las instancias de las clases pasan de un estado a otro a causa de ciertos eventos y qué acciones se desencadenan de esos cambios de estado.
5. Diagramas de Objetos: Muestran la existencia de objetos, su comportamiento y sus relaciones de forma dinámica mostrando la forma en la que los objetos se pasan mensajes entre ellos. Representan la visibilidad de los objetos, la cual determina qué objetos se pueden comunicar con otros. Pueden ser diagramas de escenario, que muestran cómo colaboran los objetos en cierta operación o diagramas de instancia, que muestra la existencia de los objetos y las relaciones estructurales entre ellos.
6. Diagramas de Tiempo: Muestran la secuencia temporal de creación y destrucción de objetos. Suelen ir acompañados de pseudocódigo en el que se explica el flujo de mensajes de control entre los objetos del sistema.
7. Diagramas de módulos y de procesos: En la fase de implementación, es posible representar mediante estos gráficos la parte física del sistema, es decir, podemos mostrar cómo se van a almacenar internamente las clases y objetos, relaciones entre módulos en tiempo de compilación, procesos, dispositivos y las comunicaciones entre ellos.
8. Subsistemas: Es una agrupación de módulos, útil en modelos de gran escala.

Descripción de las Etapas y Productos generados

Primera Etapa: *Análisis de requerimientos*

Se define qué quiere el usuario del sistema, se identifican las funciones principales del sistema, el alcance del modelamiento, los procesos principales y las políticas a soportar. No se definen pasos formales, éstos dependen de qué tan nuevo es el proyecto, la disponibilidad de expertos, usuarios y documentos adicionales.

Productos:

- *Funciones primarias del sistema:* Principales entradas y salidas del sistema, referencias a políticas, sistemas existentes o procedimientos.
- *Conjunto de mecanismos claves que el sistema debe proveer:* estado de entrada, estado de salida y estados esperados.

Segunda Etapa: *Análisis de Dominio*

Se define de manera concisa y precisa la parte del modelo del sistema.
Productos:

- *Diagrama de clases:* con las clases del dominio claves y sus relaciones de contingencia.
- *Especificación de las clases:* atributos, operaciones.
- *Vistas de herencia.* Diagramas de clases con este tipo de relaciones.
- *Diagramas de escenarios de objetos.*
- *Especificación de objetos:* Relacionan objetos y sus clases.
- Validar e iterar sobre el modelo.

Tercera Etapa: *Diseño*

Proceso que determina una implantación efectiva y eficiente que realice las funciones y tenga la información del análisis de dominio.

Productos:

- *Descripción de arquitectura inicial:* Con las decisiones más importantes de diseño como: conjunto de procesos, manejadores de bases de datos, sistemas operativos, lenguajes, etc.
- *Descripción de prototipo:* Describe las metas y contenido de las implantaciones sucesivas de prototipos, su proceso de desarrollo y la forma de probar requerimientos.
- *Diagramas de Categorías.*
- *Diagramas de clases (diseño lógico)*: Detallan las abstracciones de análisis con características de implantación.
- *Diagramas de objetos en diseño*: Muestran las operaciones necesarias para desarrollar una operación.
- *Nuevas especificaciones.*
- *Especificación de clases corregidas:* Especificación completa de los métodos con algoritmos complicados, la implantación de relaciones y el tipo de atributos.
- *Implantación física:* interfaz a dispositivos o características propias de la implantación.
- *Refinamiento del diseño:* Para incorporar el aprendizaje debido a los prototipos y cumplir con requerimientos de desempeño.

Ventajas:

- Es una metodología de propósito general.
- Herramientas y notaciones comprensibles.
- Proporciona una gran cantidad de información sobre las semánticas de los objetos.

Desventajas:

- Notaciones poco precisas.
- Herramientas orientadas al texto más que a los gráficos.
- Su notación es molesta y hay pocas herramientas disponibles.
- Poca ayuda para el desarrollador novel.

Metodología OMT (Object Modeling Technique, Técnica de Modelado de Objetos)

Jim Rumbaugh (Dr. en C. C.), desarrolló OMT, método para análisis y diseño orientado a objetos. OMT modela un sistema desde tres puntos de vista diferentes donde cada uno se lleva a cabo por separado describiendo un aspecto diferente del sistema y su unión lo describe de forma completa. OMT divide el proceso de desarrollo en tres partes aisladas: análisis, diseño e implementación. A su vez cada una de estas fases consta de otras subtareas como son los modelos de objetos, dinámico y funcional.

1. Modelo de objetos: Describe los aspectos estáticos y estructurales "datos" del sistema, etapa inicial que ayuda a entender el problema, proporciona la estructura de datos esencial en la que los modelos dinámico y funcional pueden basarse y operar. La entrada para esta fase es la especificación de requerimientos o la definición del problema. Se forma una primera imagen del modelo de clases del sistema con sus atributos y las relaciones entre ellas, usando para ello un diagrama entidad relación modificado que representa los objetos, las clases, sus relaciones (generalización, agregación, asociación, instanciación) y los métodos.

2. Modelo dinámico: Describe las relaciones temporales entre objetos y el comportamiento del sistema, básicamente la secuencia de interacciones. Captura los aspectos del sistema relacionados con el tiempo y los cambios. Define la estructura de control, es decir, los aspectos que describen la secuencia de operaciones que ocurren en respuesta a estímulos externos, sin considerar lo que hacen las operaciones (modelo funcional), en lo que operan (modelo de objetos) o cómo se implementan.
*Diagramas utilizados en este modelo: 1)*Diagrama de transición de estados, para cada objeto que sea afectado por los eventos (estímulos externos), mostrando los mensajes que fluyen, las acciones que son realizadas y los cambios en el estado (valores y ligas a un objeto) de los objetos que tienen lugar cuando ocurren los eventos, así como la transición de estados para las clases; 2)Diagramas de estado múltiples, uno por cada clase con comportamiento dinámico, 3)diagrama de flujo de eventos global, para identificar las secuencias de eventos dentro del ámbito del problema.

3. Modelo funcional: Describe qué es lo que el sistema ha de hacer, las relaciones funcionales entre valores. Muestra las funciones de los valores, sin indicar cómo (implantación), cuándo (modelo dinámico) o por qué se cambian los valores.

Diagramas utilizados en este modelo:
1)Diagrama de flujo de datos: muestra la forma en que los valores de salida son computados a partir de los valores de entrada, consta de: procesos, transformación de datos, actores, procesadores y consumidores de valores, almacenamiento de datos, creadores de tardanza entre creación y uso de datos, flujos de datos, relaciones entre procesos.

Este modelo puede revelar nuevos objetos y métodos que se pueden incorporar en los dos modelos anteriores. Por eso se dice que el método OMT es iterativo.

Mediante estas tres fases de construcción de modelos, se consigue una abstracción de la realidad con sus principales características, dichos modelos permiten probar más fácilmente los sistemas que modelan y determinan los errores, según se indica, permiten una mejor comunicación con el cliente por distintas razones:

- Es posible enseñar al cliente una aproximación de lo que será el producto final.
- Proporcionan una primera aproximación al problema que permite visualizar cómo quedará el resultado.
- Reducen la complejidad del original en subconjuntos que son fácilmente tratables por separado.

Se deben obtener solamente los aspectos importantes del problema y omitir el resto para tener un modelo completo.

Al final los tres modelos se juntan en la implantación, que contiene estructuras de datos (modelo de objetos), secuencias en el tiempo (modelo dinámico) y operaciones (modelo funcional).

Descripción de las Etapas y Productos generados

Primera Etapa: *Análisis*

Se realiza el modelo de objetos y procedimientos, actividades que se realizan:

- Identificación de objetos.
- Identificación de clases de objetos.
- Identificación de asociaciones entre objetos (adición, generalización).
- Identificación de atributos de los objetos.
- Uso de herencia para organizar y simplificar la estructura de clases.
- Organización de las clases y asociaciones estrechamente acopladas dentro de módulos.
- Descripciones textuales breves de cada objeto.

Segunda Etapa: *Diseño del sistema*

Actividades de diseño:

- Organizar el sistema en subsistemas y ordenarlos en capas y divisiones.
- Identificar la concurrencia inherente en el problema.
- Asignar subsistemas a los procesadores y tareas.
- Definir la estrategia de almacenamiento.
- Identificar las fuentes globales y definir el mecanismo para controlar el acceso a ellos.
- Elegir un enfoque para la implantación de control del software.
- Considerar las condiciones de los límites (fronteras).
- Establecer cambios fuera de las prioridades.
- Diseñar implantación de asociaciones.
- Determinar la representación de los atributos de las clases.

- Agrupar clases y asociaciones en módulos.

Productos:
- Documento de diseño, que incluye versiones detalladas de los modelos de objetos, dinámico y funcional.

Tercera Etapa: *Implantación*
- Diseño de bases de datos.
- Codificación.

Ventajas:
- Proporciona una serie de pasos perfectamente definidos al desarrollador.
- Tratamiento especial de la herencia.
- Facilita el mantenimiento dada la gran cantidad de información que se genera en el análisis.
- Su notación es simple pero muy representativa.
- Ha sido aplicado en varios proyectos por sus autores.
- Tiene un alto grado de madurez.

Desventajas:
- Falta de técnicas para integrar los modelos de objetos, los modelos dinámicos y los modelos funcionales.
- La interacción de objetos no es soportada explícitamente en ninguna herramienta gráfica.
- Al ser un análisis iterativo es difícil saber cuándo comenzar con el diseño.

Metodología OOSE (Object Oriented Software Enginnering, Ingeniería de Software Orientada a Objetos)

Ivar Jacobson desarrolló la metodología OOSE, la cual tiene un enfoque orientado a casos de uso. Un modelo de casos de uso sirve como modelo central del que todos los otros modelos se derivan, pues describen la funcionalidad completa del sistema identificando cómo todo lo que está fuera del sistema interactúa con el.

El modelo de caso de uso es la base de las fases de análisis, construcción y prueba. El objetivo del análisis es entender el sistema de acuerdo a sus requerimientos funcionales, en esta etapa los objetos se encuentran organizados y se describen las interacciones del objeto, su funcionamiento y su vista interior. La construcción abarca el plan y aplicación del código fuente. Es importante que los objetos en la fase de análisis puedan encontrase de nuevo durante la construcción. Los componentes (piezas definidas de código fuente) son muy importantes en esta metodología.

OOSE combina tres técnicas diferentes que se han usado durante mucho tiempo:

1. Programación orientada a objetos: De la cual utiliza los conceptos de encapsulación, herencia y relaciones, principalmente entre las clases y casos.

2. Trazado conceptual: Lo usa para crear los diferentes modelos del sistema a ser analizado. En OOSE ellos están extendidos con los conceptos orientados a objetos y con la posibilidad de modelar la conducta dinámica. Con estos modelos se obtiene una arquitectura del sistema bien definida.

3. Plan del hardware en el área de las telecomunicaciones: Modela varios módulos que tienen su propia funcionalidad y se conectan a las interfaces. Esta visión también tuvo que ser aplicada en el plan del software, porque los errores en un programa pueden hacer caer el sistema entero.

Diagramas utilizados en esta metodología:

- *Modelo de casos de uso:* Se basa en la descripción de elementos o usuarios externos al sistema (actores) y funcionalidad del sistema (casos de uso). Representa un servicio particular que los actores esperan del sistema de software.
- *Modelo de objetos:* Representa la estructura estática de objetos. Puede contener objetos entidad, de interfaz y de control, entre otros tipos y relaciones de herencia.

- *Diagrama de interacción:* Muestran la secuencia de eventos entre paquetes u objetos, necesarios para realizar un caso de uso.
- *Diagrama de estado:* Muestra los estados internos de un objeto complejo.

Descripción de las Etapas y Productos generados

Primera Etapa: *Análisis de Requerimientos o Modelo de Requisitos*
Este modelo delimita el sistema y define su funcionalidad. Consiste en tres partes:

1. *Modelo de casos de uso:* Describe actores y casos del uso. Los actores definen los papeles que los usuarios pueden jugar intercambiando la información con el sistema y los casos del uso representan la funcionalidad dentro del sistema.
1. *Modelo de objeto de dominio:* Para desarrollar una vista lógica del sistema que puede usarse para hacer una lista que especifique los casos del uso.
2. *Descripción de las interfaces:* Los usuarios deben involucrarse en la descripción de las interfaces detalladas, esto debe hacerse en una fase temprana. La interface tiene que capturar la vista lógica del usuario del sistema, porque el interés principal es la consistencia de esta vista lógica y la conducta real del sistema. Aunque se comienza el diseño de la interfaz, no se ahonda en el cómo y en el qué se realizará.

La meta de dichos modelo es especificar los requisitos del cliente, adaptar los puntos de vista y términos de acuerdo a los usuarios, se debe lograr que la participación de los usuarios sea activa en este modelo.

Aunque OOSE no menciona el propósito del sistema este está implícito, ya que es lo primero que el analista debe hacer junto con el usuario final para establecer su comprensión común correctamente.

Productos

- Modelo de casos de uso: discusión y validación de requerimientos, identificación detallada de cada caso de uso, describiendo la funcionalidad, las posibles variantes y errores.
- Definición de un borrador de la interfaz al usuario del sistema, que muestre cómo se verían los distintos casos de uso.
- Modelo de objetos del dominio.

Segunda Etapa: *Análisis de Estructura o Modelo Ideal*
Define la estructura lógica del sistema, independiente de la aplicación. Se extiende la conducta que se planea en los casos de uso entre los objetos en el modelo del análisis.
Este modelo busca la construcción del sistema, trabajar sobre todo lo que conlleva la aplicación y establecer la robustez del sistema. Procura reconocer los objetos, las asociaciones y estructuras de objetos, asignar atributos a los objetos, asociar un objeto a sus atributos y dividir el sistema en subsistemas (para crear los paquetes).

Se deben identificar:

- *Los objetos de control:* Que controlan la conducta del sistema en la primera aproximación, pueden derivarse de los casos del uso. Se modelan cuando el sistema es lo suficientemente complicado para tener funcionalidad que no corresponde a ningún objeto de interfaz ni a ningún objeto entidad.
- *Los objetos entidad:* Modelan la información y comportamiento que el usuario necesitará por largo tiempo.

- *Los objetos de la interface:* Presentan el sistema en la primera aproximación, pueden derivarse de las asociaciones de la interacción. Expresan toda la funcionalidad que es dependiente del entorno del sistema. Cada objeto de interfaz traduce acciones de los actores en eventos dentro del sistema, se traducen los eventos del sistema en algo visible por el actor.

Productos

- Modelo de análisis con objetos entidad, de interfaz y de control.

Tercera Etapa: *Modelo de Plan o Modelo Real*
Es un refinamiento y formalización del modelo de análisis. Su principal objetivo es adecuar el modelo de análisis al ambiente de implantación. Actividades:

- Agrupar en paquetes las clases existentes. Cada paquete deberá estar relacionado a un solo actor. Las clases con una relación mutua fuerte deben estar en el mismo paquete.
- Refinar las clases de análisis para incluir detalles de implantación.
- Desarrollar el código de los métodos de los objetos.
- Realizar los diagrama de interacción para cada caso de uso concreto, muestran cómo interactúan los distintos paquetes en el desarrollo de un caso de uso, expresa los guiones conductuales. El diagrama de interacción hace que OOSE pueda involucrar alternativas o iteraciones, ya que ellos son basados en la descripción de un caso de uso.
- Desarrollar diagramas de estado para los objetos complejos, éstos diagramas se usan para describir la conducta del objeto y los estímulos (mensajes) que puede recibir o causar.

Este modelo es un refinamiento y formalización del anterior, sus principales objetivos son:

- Modelar el sistema adaptándolo al ambiente de aplicación real. (Considerar componentes como: DBMS, lenguaje de programación, etc.)
- Procurar que el sistema sea tolerante a los cambios en el ambiente de aplicación.
- Establecer interfaces de objetos para que el desarrollo extenso pueda realizarse en paralelo.
- Establecer los requisitos en la arquitectura, actuación, la memoria, la distribución etc.
- Reconocer los objetos reales.

Productos:

- Dibujar diagramas de interacción (guiones de casos de uso particulares).
- Construir los gráficos de estado-transición (conducta de objetos reales).
- Modelo de paquetes. Definición de módulos en la implantación y detalle de las clases de diseño en cada uno de ellos.

Cuarta Etapa: *Implantación o Modelo de Aplicación*
Implementar las clases de diseño definidas. Una clase de diseño puede corresponder a una o más clases en implantación, dependiendo de su complejidad, de su dependencia del ambiente de desarrollo, etc. Las clases en implantación deben tener las siguientes características:

- Robustas y alto grado de reusabilidad.
- No deben ofrecer funcionalidad similar, a menos que estén relacionadas por herencia.
- Funcionalidad interna altamente relacionada (cohesivas).

En esta etapa se inicia la codificación del sistema, tanto el desarrollo de las bases de datos como de las distintas aplicaciones con las que contará. Aquí los paquetes, antes mencionados, pasan a ser clases.

Los objetivos principales de este modelo son:

- Diseñar clases que sean robustas y altamente reusables.
- Crear los objetos reales en un lenguaje de programación.
- La traceabilidad (que es la característica que permite a las clases poder comunicarse y relacionarse con otras clases).

Productos:

- Código de las clases, organizado por paquetes.

Quinta Etapa: *Modelo de Pruebas o Comprobación*
En esta etapa se procede a probar las aplicaciones, el funcionamiento de las clases y la robustez del sistema.

Se recomienda comenzar por los niveles mas bajos del sistema, es decir:

- Módulos de objetos.
- Casos de uso .
- El desarrollo de la aplicación.

Se deben analizar varias cuestiones, comprobar si hay faltas en el programa, dónde las hay, o si el sistema ha sido creado correctamente.

Entre las fases que deben probarse están:

- La comprobación del modelo de requisitos, ya que si se cumplen todos los requisitos allí especificados, pasa la aprobación. Comprobar la robustez del sistema, lo que ayuda a localizar faltas.
- Comprobación de la traceabilidad, lo que ayuda al movimiento dentro del modelo del plan (a donde la falta será corregida).
- Comprobación de la unidad.
- Comprobación de la integración.
- Comprobación del sistema.
- Comprobación de la especificación.
- Comprobación de la estructura.
- Comprobación de la integración.

Productos:

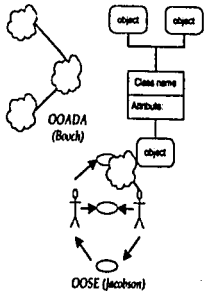
- Definición de las pruebas realizadas.

Cuadro sinóptico de las metodologías

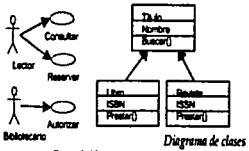
Metodología	OOADA (Object Oriented Analysis and Design with Aplications)	OMT (Object Modeling Technique)	OOSE (Object Oriented Software Enginnering (Ingeniería de Software Orientada a Objetos)
Autor	Grady Booch	Jim Rumbaugh	Ivar Jacobson
Año	1991	1994	1994
Objetivo	Modelar un diseño orientado a objetos desde dos puntos de vista: *Lógico*: Define clases, objetos y sus relaciones. *Físico:* Define la arquitectura de módulos y procesos.	Modelar el sistema desde tres puntos de vista diferentes (análisis, diseño e implementación), cada uno realizado por separado y describe un aspecto diferente del sistema, su unión lo describe de forma completa.	Enfocar su atención en los casos de uso, ya que son la base para las fases de análisis, construcción y prueba, pues deben entenderse bien los requerimientos funcionales del sistema.
Características	Se divide en un micro y macro proceso.	Se divide en tres partes aisladas: análisis, diseño e implementación, cada una de ellas compuesta por: *Modelo de objetos:* Aspectos estáticos y la estructura de datos del sistema, ayuda a entender el problema. *Modelo dinámico:* Relaciones temporales entre objetos y el comportamiento del sistema (secuencia de interacciones). *Modelo funcional:* Describe qué hará el sistema, las relaciones funcionales entre valores, sin indicar cómo, cuándo o por qué se cambian los valores.	Combina tres técnicas: A) Programación orientada a objetos: de la cual usa los conceptos de encapsulación, herencia, relaciones y casos. B) Trazado conceptual: usado para crear los diferentes modelos del sistema a ser analizado, definiendo así una arquitectura bien definida. C) Plan del hardware en el área de las telecomunicaciones: Modela varios módulos con su propia funcionalidad y se conectan a las interfaces. Esta visión también es aplicada en el plan del software.
Diagramas	1) D. de clases 2) Especificación de clases 3) D. de categorías 4) D. de transición de estados 5) D. de objetos 6) D. de tiempo 7) D. de módulos y de procesos 8) Subsistemas	1) D. Entidad relación modificado (muestra objetos, clases, relaciones, métodos) 2) D. de transición de estados 3) D. de estados múltiples 4) D. de flujos de eventos 5) D. de flujo de datos	1) D. del modelo de casos de uso 2) D. del modelo de objetos 3) D. de interacción 4) D. de estado

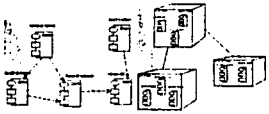
Metodología	OOADA (Object Oriented Analysis and Design with Aplications)	OMT (Object Modeling Technique)	OOSE (Object Oriented Software Enginnering (Ingeniería de Software Orientada a Objetos)
Etapas	1) Análisis de requerimientos 2) Análisis de dominio 3) Diseño	1) Análisis 2) Diseño del sistema 3) Implantación	1) Análisis de Requerimientos o Modelo de Requisitos 2) Análisis De Estructura o Modelo Ideal 3) Modelo de Plan o Modelo Real 4) Implantación o Modelo de Aplicación 5) Modelo de Pruebas o Comprobación
Ventajas	• Es de propósito general. • Herramientas y notaciones comprensibles. • Gran cantidad de información de las semánticas de los objetos.	• Serie de pasos perfectamente definidos. • Tratamiento especial de herencia. • Facilita el mantenimiento por la gran cantidad de información generada en el análisis. • Capacidades simples pero representativas de notación. • Aplicado en varios proyectos por sus autores. • Madurez.	• Facilidad de entendimiento. • Ayuda a recopilar los requerimientos de los usuarios. • Ayudan a identificar claramente la relación entre el sistema y su entorno.
Desventajas	• Notaciones poco precisas. • Herramientas orientadas al texto más que a los gráficos. • Notación molesta • Pocas herramientas disponibles • Poca ayuda para el desarrollador	• Falta de técnicas para integrar los modelos de objetos, dinámicos y funcionales. • La interacción de objetos no es soportada explícitamente en ninguna herramienta gráfica. • Al ser un análisis iterativo es difícil saber cuándo comenzar con el diseño.	

ANTECEDENTES UML

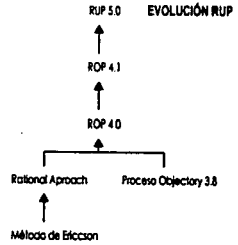


NOTACIÓN UML PARTE ESTÁTICA

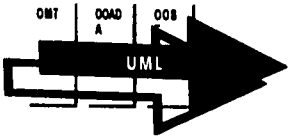




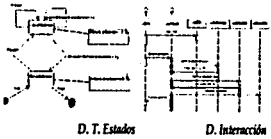
EVOLUCIÓN RUP



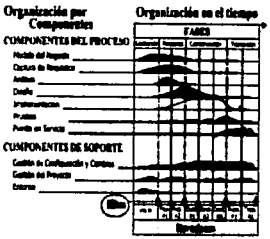
EVOLUCIÓN UML



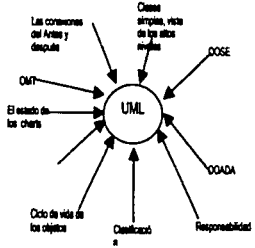
NOTACIÓN UML PARTE DINÁMICA



FASES DE RUP

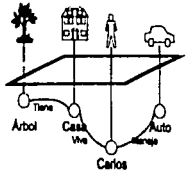


APORTADORES



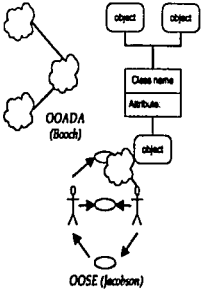
PARADIGMA OO

PARADIGMA ORIENTADO A OBJETOS

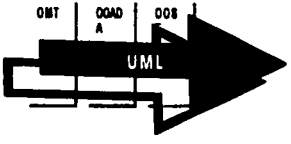


UML

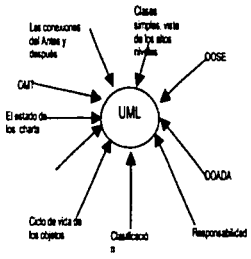
ANTECEDENTES UML



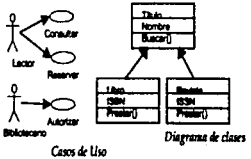
EVOLUCIÓN UML



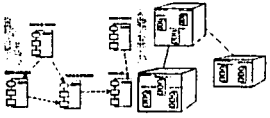
APORTADORES



NOTACIÓN UML PARTE ESTÁTICA



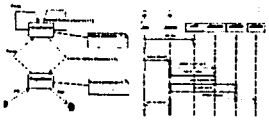
Casos de Uso *Diagrama de clases*



D. Componentes *D. Distribución*

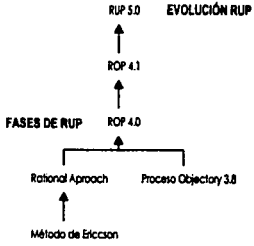
NOTACIÓN UML PARTE DINÁMICA

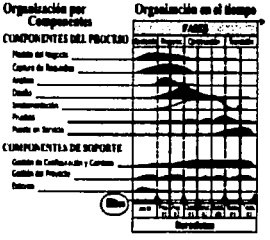
D. Actividad *D. Colaboración*



D. T. Estados *D. Interacción*

RUP





Antecedentes UML

Considerando la importancia y el auge de las metodologías OOADA (Object Oriented Analysis and Design with Aplications) de Grady Booch, OMT (Object Modeling Technique) de Jim Rumbaugh y OOSE (Object Oriented Software Enginnering) de Ivar Jacobson, se consideran como los antecedentes más importantes de la notación UML, pues la conjunción de las mejores técnicas de estas metodologías dieron origen a una notación completa y unificada llamada Unified Modeling Language (UML).

Evolución UML

Desde 1988 aparecieron varios libros clave sobre análisis OO y métodos de diseño. Surgieron metodologías de diferentes autores, quienes no se preocuparon por estandarizarlas en una sola que ofreciera los beneficios de todas ellas; Grady Booch fue el primero en intentarlo pero no tuvo éxito. Posteriormente Jim Rumbaugh (creador de OMT) se unió a él en Rational Software Corporation y comenzaron a unificar sus dos metodologías OMT y OOADA. En 1995 realizaron la primera publicación de su método integrado: la versión 0.8 de la documentación del Unified Method, Método Unificado, considerada como la primera versión de UML.

A finales de 1995 Ivar Jacobson se unió a ellos con su metodología OOSE, Rational hizo el lanzamiento del documento preliminar del 0.8.

En 1996 los tres construyeron su método, nombrándolo UML (Unified Modeling Language, Lenguaje Unificado de Modelado), versiones 0.9 y 0.91. Posteriormente en OMG (Object Managament Group) procuraron la estandarización de los métodos.

Los tres amigos comenzaron a recibir aportaciones de grandes compañías, a mediados de 1997 las propuestas de OMG convergieron sobre la versión 1.1 de UML, adoptada formalmente por OMG a finales de 1997. Un grupo de trabajo trajo la versión 1.2, para unificar las numeraciones, sin embargo no hay cambios importantes desde la versión 1.1 a 1.2., adoptada como estándar en enero de 1999.

Diagramas UML (Parte estática)

Diagramas de casos de uso, de clases, de componentes y el de distribución.

Diagramas UML (Parte estática)

Diagramas de actividad, de colaboración, el de estados y el de interacción.

Antecedentes RUP

A raíz de varios estudios que revelaron el estado tan deficiente de la calidad y confiabilidad del software, así como el asimilar las grandes cantidades de dinero gastadas en reparar los errores de los sistemas desarrollados, fueron dos causas importantes que pusieron en alerta a varios autores y se preocuparon por desarrollar un "proceso efectivo" para el desarrollo de software, un proceso que proporcionara normas para el desarrollo de software de calidad y que permitiera la aplicación de las mejores prácticas que la tecnología permite y que redujera el riesgo, haciendo los proyectos más predecibles.

Así comenzó a diseñarse el proceso de desarrollo iterativo RUP, el esfuerzo de Walker Royce, Rich Reitman, Grady Booch y Philippe Kruchten se unió para lograr la planificación detallada de las fases y la ordenación de las iteraciones dentro de éstas.

Evolución RUP

Los inicios de RUP datan desde 1967 con el Método Ericsson y fue hasta 1981 cuando surgieron libros, artículos y documentos dentro de Rational.

A partir de 1987, Ivar Jacobson inicia más formalmente sus trabajos en el proceso Objectory y comenzó a popularizar el concepto de "casos de uso" destacando la importancia de los mismos. Durante este lapso y hasta 1990 se escriben varios artículos que dieron énfasis a la arquitectura y al desarrollo iterativo. En 1995 surge la primera versión interativa Objectory 3.8. de Rational Software Corporation, como complemento de los trabajos de Objectory. Posteriormente, en 1997, surge el Rational Objectory Process (ROP). En 1998 Rational compró o se fusionó a otras empresas fabricantes de herramientas (Requisite Inc, SQA Inc., Pure-Atria, Performance Awareness, Vigortech), las cuales hicieron importantes aportaciones al ROP, para junio Rational publicó una nueva versión cambiando su nombre por el de Proceso Unificado de Racional (RUP) 5.0.

Fases del RUP

El Proceso RUP puede ser descrito en dos ejes:

El horizontal que representa el tiempo y muestra el aspecto dinámico del proceso y es expresado en términos de ciclos, fases, iteraciones y milestones. Fases: Concepción, Elaboración, Construcción y Transición.

El eje vertical representa el aspecto estático del proceso y es descrito en términos de actividades, artefactos y flujos de trabajo.

120

3.2 DEFINICIONES

Definiciones Etimológicas

LENGUAJE	Sonidos con los que el hombre manifiesta lo que piensa: del provenzal *lenguatge.* *LENGUA:* órgano situado dentro de la boca, del latín *lingua.* *LINGUA:* lengua, órgano de la boca castellano *lingua* 'id' portugés, gallego.
MODELADO	Modelar, dar forma a una materia plástica: de modelo. *MODELO:* ejemplar o tipo elegido, del italiano *modello;* del latín *modelus* por *modulus.*
PROCESO	Progreso, desarrollo, del latín *processus.*
DESARROLLO	Desarrollar, deshacer, del *des* y *arrollar.*
UNIFICADO	Reducir a una cosa, de latín *unus, uno* y *ficáre* hacer. *UNUS:* uno, un tónico: uno castellano; u catalán; atóno: un castellano. Catalán.
RACIONAL	Relativo a la razón del latín *rationalis.*
ELEMENTOS	Principio integrante de un cuerpo, del latín *elementum.*
CLAVE	Aplicación de los signos contenidos para escribir en cifra, del latín *clavis* llave. *CLAVIS:* llave, instrumento de cerrar y abrir. Castellano chave 'llave' portugués gallego; cllav 'id' ribag. Ferraz; clav 'id' aragonés. Catalán.
SISTEMA	Conjunto de principios enlazados entre sí, del latín *systema.*
ORIENTADO	Orientar, colocar algo respecto a los puntos cardinales: de oriente. *ORIENTE:* nacimiento del latín *oriens – enlis.*
OBJETOS	Materia de conocimiento o sensibilidad, del latín *objetus.*

Definiciones de Diccionario

LENGUAJE	1. Conjunto de caracteres, símbolos y reglas que permiten escribir las instrucciones que se dan a una computadora. 2. Cualquiera de los sistemas que emplea el hombre para comunicar sus sentimientos e ideas. 3. Facultad humana que sirve para la comunicación. 4. Manera de expresarse. 5. Conjunto de señales que dan a entender una cosa. 6. Sonidos articulados con los que las personas manifiestan lo que piensan o sienten. 7. Sistema de caracteres y reglas con los que se programa una computadora. 8. Combinación de dígitos binarios, mediante la cual una computadora

	funciona correctamente.
MODELADO	1. Limitar, ajustarse a un modelo. 2. Dar forma artística a una sustancia plástica. 3. Técnica que consiste en modelar figuras en una materia: *modelado artesanal del vidrio.*
PROCESO	*1.* Conjunto de operaciones a las que se someten los datos para conseguir un determinado fin. *2.* Conjunto de fases sucesivas de un fenómeno natural o de una operación artificial. 3. Método o sistema usado para llegar a un fin. 4. Transcurso del tiempo. 5. Conjunto de operaciones lógicas y aritméticas ordenadas, cuyo fin es la obtención de unos resultados determinados.
DESARROLLO	1. Realización de una idea, proyecto, etc. 2. Hacer que un organismo progrese. 3. Acrecentar el valor, riqueza o poder de algo. *4.* Crecimiento o mejora de un aspecto físico, intelectual o moral. *5.* Explicación y ampliación de una teoría o idea. *6.* Progreso de una comunidad humana. 7. Dibujo en el que se muestran simultáneamente las diversas partes de algo que en la realidad no puede ser abarcado en su totalidad desde una perspectiva fija o única.
UNIFICADO	1. Igualar. 2. Reunir varias cosas o personas para formar un todo homogéneo. 3. Hacer de muchas cosas una un todo, uniéndolas, mezclándolas o reduciéndolas a una misma especie.
RACIONAL	1. Dotado de razón: animal racional. 2. Filosofía del conocimiento basada en la razón. 3. Organizar todo haciéndolo más eficaz y menos costoso.

	4. De la razón o relativo a ella. 5. Conforme a la razón.
ELEMENTOS	1. Fundamento, móvil o parte integrante de una cosa. 2. Principio físico o químico de los cuerpos. 3. Sustancia formada por átomos que tienen el mismo número de protones nucleares. 4. Fundamentos y principios de las ciencias y de las artes. 5. Fuerzas naturales capaces de alterar las condiciones atmosféricas o climáticas. 6. Medios, recursos.
CLAVE	1. Básico, fundamental, decisivo. Suele usar en aposición a otro sustantivo. 2. Noticia o idea por la cual se hace comprensible algo. 3. Conjunto de los signos convenidos para escribir en cifra: libro de claves. 4. Contraseña, combinación de signos que sirven para abrir o hacer funcionar ciertos aparatos. 5. Signo que se pone al comienzo del pentagrama para determinar la entonación y el nombre de las notas que contiene: clave de sol.
SISTEMA	1. Sistema de información: Conjunto de elementos que están organizados para procesar información y cumplir un objetivo predefinido. 2. Conjunto de elementos que, ordenadamente relacionadas entre sí, contribuyen a determinado objeto. 3. Conjunto de reglas o principios sobre una materia, estructurados y enlazados entre sí. 4. Conjunto de ideas o principios que conforman una teoria coherente y completa. 5. Conjunto de órganos que intervienen en alguna de las principales funciones vegetativas y animales. 6. Medio, modo o manera usados para hacer algo o lograr un objetivo. 7. Sistema de numeración posicional, que permite representar cualquier número con un conjunto limitado de símbolos o dígitos, que toman distinto valor en función de la posición que ocupan; las posiciones correlativas corresponden a sucesivas potencias de la base de numeración del sistema, que es también el número de símbolos distintos que emplea.

	8. Sistema operativo. Conjunto de programas para el funcionamiento y explotación de un ordenador, encargado de controlar la unidad central, la memoria y los dispositivos de entrada y salida.
ORIENTADO	1. Dirigir alguien su interés, su conducta o sus acciones hacia un objetivo determinado. 2. Colocar una cosa en una posición determinada respecto a los puntos cardinales. 3. Determinar la posición o dirección de una cosa respecto a un punto de cardinalidad. 4. Informar a uno de lo que ignora acerca de un asunto o negocio, o aconsejarle sobre la forma más acertada de llevarlo a cabo.
OBJETOS	1. Lo que posee carácter material e inanimado; cosa. 2. Todo lo que puede ser conocido o sentido por el sujeto, incluso él mismo. 3. Lo que sirve de materia al ejercicio de las facultades mentales. 4. Fin o intento a que se dirige una acción u operación. 5. Materia y sujeto de una ciencia.

Definiciones de Diccionarios Especializados en Informática

LENGUAJE	1.Conjunto de palabras y normas destinadas a la construcción de tendencias que pueden utilizarse para la comunicación de información. 2. Un lenguaje de programación diseñado para manipulación de problemas de ciertas categorías de aplicaciones relativas, por ejemplo, a sistemas matemáticos, científicos, inteligentes o a sistemas comerciales. Se trata, en esencia, de lenguajes que permiten al usuario redactar sentencias en forma que le sea familiar, por ejemplo, la notación matemática o el inglés.
MODELADO	1. Cosa para representar o describir otra. El objetivo de un modelo en el proceso de datos es proporcionar el aprendizaje sobre la cosa que se está modelando. 2. Modelo o representación de un sistema, dispositivo o proceso en forma matemática.
PROCESO	1. Operación que implica entradas, funciones internas y salidas. En contraste con el procedimiento, un proceso no necesita tener un comienzo y un fin (aunque puede tenerlos). 2. Término general que se le da a cualquier operación que una computadora lleve a cabo sobre los datos. 3. Conjunto de programas de usuario, de software de sistemas y datos que el sistema operativo hace conjuntamente para llevar a cabo una tarea especifica .
DESARROLLO	1.Mostrar al completo sin modificadores ni operaciones auxiliares la totalidad de instrucciones utilizadas durante un bucle de instrucciones, el desarrollo puede ser efectuado por la computadora.
ELEMENTOS	1.Circuito o dispositivo que realiza alguna función elemental y específica de proceso de datos. En general, un elemento y específicamente de proceso de datos. En general, un elemento transforma una señal de entradas en una de salida, con arreglo a ciertas normas prescritas. 2.Miembro de una colección de objetos que no puede ser subdividido en partes constituyentes susceptibles a su vez de ser consideradas partes de colección. 3.Todo circuito puede ser considerado como una función y aunque puede ser combinado con otros elementos para realizar funciones más complejas pero no es descomponible en componentes individuales.
CLAVE	1. Item de datos que sirven para identificar única y exclusivamente un registro de datos. La clave puede estar situada en el interior del registro, preceder inmediatamente al mismo o estar separada del registro. 2. Key dígito o grupo de dígitos que permiten localizar o identificar un registro.

SISTEMA	1.Plan, diseño o método de organización que tiene por objeto la realización de algún propósito. Un sistema de ordenador suele ser un conjunto de elementos de equipo físico, relacionados entre sí, y pueden representar dos significados distintos, según sea un sistema general o específico. 2.Diseñado con el fin de realizar una tarea bien determinada, se utiliza este término para distinguir un sistema de usuario (sistema de aplicación) de los diversos compiladores y del soporte lógico de gestión de sistemas, con los cuales ha de coexistir una aplicación en un ordenador.
ORIENTADO	1.Reconocimiento óptico de caracteres, la posición determinada que los elementos lineales de un documento fuente de entrada parecen en paralelo con el borde de entrada del documento.

Definición Propia

*"El **Proceso Unificado Racional** es un conjunto de actividades y responsabilidades necesarias para transformar los requisitos de un usuario en un sistema de software, siendo este un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes organizaciones y diferentes tamaños de proyectos, cuyo propósito principal es la producción de software de alta calidad y de acuerdo a las necesidades de los usuarios."*

*"El **Lenguaje Unificado de Modelado** es un lenguaje de modelado y no un método, que permite mostrar el diseño y los requerimientos de un sistema creando modelos precisos de desarrollo, lo que permite mejorar la comunicación con los usuarios y el equipo de trabajo, ya que es una herramienta útil para generar documentación detallada de la arquitectura del sistema a desarrollar."*

Características de UML
Visualizar: la solución y facilitar la comunicación.
Especificar: significa construir modelos precisos, no ambiguos y completos.
Construir: UML no es un lenguaje de programación pero sus diagramas pueden servir como código base a una gran variedad de lenguajes de programación.
Documentar: la arquitectura, requerimientos, pruebas, procesos de negocio.

Sinónimos

LENGUAJE	Lengua, idioma, habla, dialecto, expresión, jergacaló, jerigonza, frase, cenismo, germanía.
MODELADO	Prototipo, tipo, ejemplar, dechado, ejemplo, muestra, espécimen, molde, forma, relieve, formar, esculpir, presentar, ajustarse.
PROCESO	Juicio, pleito, causa, vista, sumario, procedimiento, desarrollo, paso, sucesión, evolución, marcha, atestado, transcurso, carrera.
DESARROLLO	Auge, progreso, prosperidad, crecimiento, expansión, adelanto, evolucionar, perfeccionar, desenvolverse, aumento, amplificación, dilatación, propagación, medio, crecimiento, incremento, desenvolvimiento, progreso, ramificación, difusión, explicación, explanación.
UNIFICADO	Unir, aunar, adunar, agrupar, centralizar, juntar.
RACIONAL	Lógico, coherente, sensato, correcto, inteligente, humano, superior, razonado, razonable, derecho, justo, fundado, procedente, plausible, probable, ecuánime, incuestionable, equitativo, lógico, bueno, cierto, exacto.
ELEMENTOS	Rudimentos, nociones, bienes, recursos, principios, fundamentos, bases, medios.
CLAVE	Clavicordio, clavicímbalo, cifra, contracifra, quid, secreto, explicación, anagrama, jeroglífico.
SISTEMA	Regla, método, plan, norma, procedimiento, conjunto, régimen, ordenanza, técnica, gobierno.
ORIENTADO	Situación, colocación, disposición, informe, instrucción, consejo, adiestramiento, encauzamiento, encarrilamiento, guía, enderezamiento, encaminamiento.
OBJETOS	materia, cuerpo, cosa, substrato, elemento, masa, concreción, asunto, sustancia, esencia, idea, fin, termino, finalidad, intente, propósito, intención.

Antónimos

LENGUAJE	Callar, omitir
MODELADO	Destruir, copiar
PROCESO	Estancamiento
DESARROLLO	Retraso, retroceso
UNIFICADO	Distinto o disparejo
RACIONAL	Irracional
ELEMENTOS	Totalidad
CLAVE	Enigma
SISTEMA	Disfuncional
ORIENTADO	Desorientar
OBJETOS	Conglomerado

Definiciones en Inglés

LENGUAGE	1. The words, their pronunciation, and the methods of combining them used and understood by a community. *(Las palabras, su pronunciación, y los métodos de combinación usados y entendidos por una comunidad).* 2. Audible, articulate, meaningful sound as produced by the action of the vocal organs. *(Sonido, articulaciones, sonidos claros producir por la acción de los órganos vocales).* 3. A systematic means of communicating ideas or feelings by the use of conventionalized signs, sounds, gestures, or marks having understood meanings. *(Medio sistemático de comunicar ideas o sentimientos por el uso de señales convencionalismos, sonidos, gesticulaciones o marcas para entender significados).* 4. A formal system of signs and symbols (as FORTRAN or a calculus in logic) including rules for the formation and transformation of admissible expressions. *(Sistema formal de señales y símbolos (como FORTRAN o un cálculo en la lógica) incluyendo reglas para la formación y transformación de expresiones admisibles).*
MODELING	1. Serving as or capable of serving as a pattern. *(Usado o capaz de ser usado como un patrón).* 2. Being a usually miniature representation of something. *(Comúnmente es una miniatura representación de algo).*
PROCESS	1. A natural phenomenon marked by gradual changes that lead toward a particular result. *(Fenómeno natural marcado por cambios graduales que conducen hacia un resultado particular).* 2. A natural continuing activity or function.*(Actividad natural continua o función).* 3. A series of actions or operations conducing to an end; especially : a continuous operation or treatment especially in manufacture. *(Serie de acciones u operaciones finalizadas; especialmente : una operación continua o tratamiento especial en manufactura).* 4. The whole course of proceedings in a legal action. *(Curso entero de procedimientos en una acción legal).*
DEVELOPMENT	1. Growth, progress: the industrial development of the area. *(Crecimiento, progreso: el desarrollo industrial del área).* 2. The act, process, or result of developing. *(El acto, proceso o resultado de desarrollo).*

UNIFICADO	1. To make into a unit or a coherent whole : UNITE. *(Hacer una unidad o una totalidad coherente : UNIDAD).*
RATIONAL	1. Having reason or understanding. *(Tener razón o entender).* 2. Relating to, based on, or agreeable to reason : REASONABLE. *(Relacionando con, basado en, o deleitoso con razonar : RAZONABLE).*
ELEMENT	1. One of the factors determining the outcome of a process *(Uno de los factores que determinan el resultado de un proceso).* 2. Mean one of the parts of a compound or complex whole. *(Significa una de las partes de un compuesto o totalidad compleja).* 3. Applies to any such part and often connotes irreducible simplicity. *(Aplica a cualquier parte y frecuentemente implica simplicidad irreductible).* 4. One of the necessary data or values on which calculations or conclusions are based. *(Uno de los valores o datos necesarios sobre los cuales se basan los cálculos o conclusiones).*
KEYSTONE	1. Something on which associated things depend for support *(Algo sobre lo que se asocian cosas dependiendo para el apoyo).*
SYSTEM	1. A group of body organs that together perform one or more vital functions *(Un grupo de órganos de cuerpo que junto desempeñan uno o funciones más vitales).* 2. A group of devices or artificial objects or an organization forming a network especially for distributing something or serving a common purpose *(Un grupo de dispositivos u objetos artificiales o una organización que forma una red especialmente para distribuir algo o sirviendo un propósito común).* 3. An organized or established procedure b : a manner of classifying, symbolizing, or schematizing *(Un procedimiento organizado o establecido : una manera de clasificar, simbolizar, o schematizing).*
ORIENTED	1. To cause to face or point toward the east; specifically : to build (a church or temple) with the longitudinal axis pointing eastward and the chief altar at the eastern end *(Para ocasionar cara o indicar hacia el este; especificamente : para construir (una iglesia o el templo) con el eje longitudinal que indica este y el altar principal al fin oriental).* 2. To set right by adjusting to facts or principles b : to acquaint with the existing situation or environment *(Para colocar derecho por ajustar a hechos o principios b : para enterar con el ambiente o situación existente).*
OBJETS	1. Something material that may be perceived by the senses *(Algo material que puede ser percibido por los sentidos).*

	que puede ser percibido por los sentidos).

Definiciones en Francés

LANGAGE	1. Jointure de caracteres, symboles et règles qui permet écrire les instructions qui sont donné à un ordinateur. *(Conjunto de caracteres, símbolos y reglas que permiten escribir las instrucciones que se dan a una computadora).* 2. Le jeu de mots et la procédure destinée pour la construction des tendances qui peuvent être employées pour la communication d'information. *(Conjunto de palabras y normas destinadas a la construcción de tendencias que pueden utilizarse para la comunicación de información).* 3. Une langue de programmer conçu pour manipulation des problèmes des certaines larges catégories de parent de demandes(applications), par exemple, à systèmes mathématiques, scientifiques, intelligents ou à systèmes commerciaux. C'est une question, dans l'essence, de langues qu'ils permettent à l'utilisateur d'écrire des jugements en forme qui lui (elle) est un parent (familière, la famille), par exemple, la notation mathématique ou l'Anglais. *(Lenguaje de programación diseñado para la manipulación de problemas de ciertas áreas de aplicaciones relativas, por ejemplo, a sistemas matemáticos, científicos, inteligentes o a sistemas comerciales. Se trata, en esencia, de lenguajes que permiten al usuario redactar sentencias en forma que le sea familiar, por ejemplo, la notación matemática o el inglés).*
MODèLE	1. La chose arrête de représenter ou décrire d'autre. Le but (la lentille) d'un modèle dans le traitement de données est de fournir l'étude sur la chose qui est le modelage lui-même. *(Cosa para representar o describir otra. El objetivo de un modelo en el proceso de datos es proporcionar el aprendizaje sobre la cosa que se está modelando).* 1. Border, s'adapter à un modèle. *(Limitar, ajustarse a un modelo).*
PROCESSUS	1. L'opération qui implique le revenu, des fonctions internes et bombantes. Par contraste avec la procédure, un processus ne doit pas avoir un commencement et une fin (le but) quoiqu'il (elle) puisse les avoir. *(Operación que implica entradas, funciones internas y salidas. En contraste con el procedimiento, un proceso no necesita tener un comienzo y un fin, aunque puede tenerlos).* 2. Le terme général (la fin) qui lui (le) donne à lui (vous, eux) à n'importe quelle opération qu'un ordinateur effectue sur l'information. *(Término general que se le da a cualquier operación que una computadora lleve a*

	cabo sobre los datos). 3. On est employé dans la relation avec un système en vigueur pour se rapporter spécifiquement plus à un jeu des programmes d'utilisateur, de logiciel de système et l'information que le système en vigueur fait ensemble pour effectuer une tâche spécifique. *(Se utiliza más especificamente en relación con un sistema operativo para aludir a un conjunto de programas de usuario, de software de sistemas y datos que el sistema operativo hace conjuntamente para llevar a cabo una tarea especifica).* 4. Le jeu des opérations qui livrent l'information pour obtenir une certaine fin (le but). *(Conjunto de operaciones a las que se someten los datos para conseguir un determinado fin).*
DéVELOPPEMENT	1. Pour montrer au complet sans modificateurs ou des opérations auxiliaires la totalité des instructions employées pendant une frisette d'instructions, le développement peut être effectuée (effectuée) par l'ordinateur. *(Mostrar al completo sin modificadores ni operaciones auxiliares la totalidad de las instrucciones utilizadas durante un bucle de instrucciones, el desarrollo puede ser efectuado por la computadora).* 2. Accomplissement d'une idée, projet, etc. *(Realización de una idea, proyecto, etc.).*
UNIFIER	1. Assembler plusieurs choses ou personnes pour former tout à fait homogène. *(Reunir varias cosas o personas para formar un todo homogéneo).* 2. Faire de beaucoup de choses un ou un tout, les joignant(rejoignant), les mélangeant ou les réduisant à la même espèce (sorte) : unifier critères. *(Hacer de muchas cosas una o un todo, uniéndolas, mezclándolas o reduciéndolas a una misma especie: unificar criterios).* 3. Être égal. *(Igualar).*
RAISONNABLE	1. Doté avec raison. *(Dotado de razón).* 2. Organiser tout le faisant plus efficace et moins coûteux. *(Organizar todo haciéndolo más eficaz y menos costoso).* 3. De la raison ou quant à la : comportement raisonnable. *(De la razón o relativo a ella: comportamiento racional).* 4. *Conformément à la raison (*Conforme a la razón*).*
éLéMENT	1. Le membre d'une collection(ramassage) des objets qui ne peuvent pas être subdivisés dans des parties constitutives (des rapports) capables à son tour de parties (les rapports) de collection(ramassage) étant considérée. *(Miembro de una colección de objetos que no puede ser subdividido en partes constituyentes susceptibles a su vez de ser*

	consideradas partes de colección). 2. Je base, le portable ou la partie intégrante (le rapport) d'une chose. *(Fundamento, móvil o parte integrante de una cosa).*
CLEF	1. De base, fondamental, décisif. il (elle) est dans l'habitude d'utilisation dans l'apposition à un autre nom. *(Básico, fundamental, decisivo. Suele usar en aposición a otro sustantivo).*
SYSTÈME	1. Dise ñ L'embarras pour comprendre un certain bien la tâche, ce terme (la fin) est employée pour distinguer le système de l'utilisateur (le système de demande(application)) des collectionneurs divers et de l'appui logique de gestion de système, avec laquelle une demande(application) doit coexister dans un ordinateur. *(Diseñado con el fin de realizar una tarea bien determinada, se utiliza este término para distinguir un sistema de usuario (sistema de aplicación) de los diversos compiladores y del soporte lógico de gestión de sistemas, con los cuales ha de coexistir una aplicación en un ordenador).* 2. Système d'information : le Jeu des éléments qui sont organisés pour essayer l'information et accomplir un but prédéterminé (la lentille). *(Sistema de información: Conjunto de elementos que están organizados para procesar información y cumplir un objetivo predefinido).*
ORIENTER	1. Pour adresser(diriger) quelqu'un son (son) intérêt, son (votre, leur) conduite ou sa (son, elle) exécutent à un certain but (la lentille). *(Dirigir alguien su interés, su conducta o sus acciones hacia un objetivo determinado).*
OBJET	1. Ce qui possède le caractère matériel(substantiel) et inanimé; chose. *(Lo que posee carácter material e inanimado; cosa).* 2. Tout que l'on peut connaître ou senti par le sujet, ci-joimment cela (il) lui-même. *(Todo lo que puede ser conocido o sentido por el sujeto, incluso él mismo).* 3. La fin (le but) ou moi essayons à ce (que) une action ou l'opération vont : l'objet de ce voyage est de négocier un accord de coopération. *(Fin o intento a que se dirige una acción u operación: el objeto de este viaje es negociar un tratado de cooperación).*

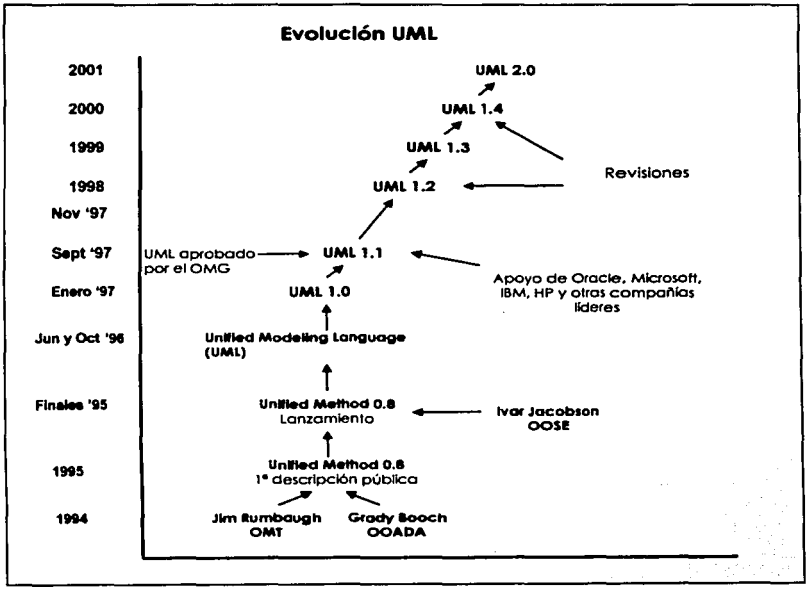
3.3 EVOLUCIÓN UML

Año	Acontecimiento
1988-1992	Durante este periodo aparecieron varios libros que difundieron conocimiento acerca del análisis orientado a objetos y métodos de diseño, entre los que podemos mencionar: • Se dio origen al enfoque de diseño recursivo con dos libros sobre análisis y diseño (1989 y 1991) escritos por Sally Shlaer y Steve Mellor. • Origen del enfoque hacia los métodos ligeros orientados a prototipos de Coad con los libros escritos por Peter Coad y Ed Yourdon dieron. • La comunidad Smalltalk de Portland, Oregon, aportó el Diseño Guiado por la Responsabilidad (Responsibility-Driven Design) y las tarjetas CRC (Clase-Responsabilidad-Colaboración, Class-Responsibility-Collaboration). • Grady Booch desarrolló sistemas en Ada en la empresa Rational Software. En sus libros se daban buenos ejemplos de métodos. • Jim Rumbaugh dirigió un equipo en los laboratorios de investigación de General Electric, el resultado fue un popular libro sobre su método OMT, Técnica de Modelado de Objetos (Object Modeling Technique). Jim Odell y James Martin escribieron juntos acerca de su experiencia en los sistemas de información de negocios y de ingeniería de información, su libro es considerado como el más conceptual de todos. • Ivar Jacobson escribió sobre su experiencia en conmutadores telefónicos para Ericsson, en el primero de sus libros introdujo el concepto de Casos de Uso (use cases).
1994	Durante el OOPSLA'94 (en Pórtland), los métodos estaban muy divididos y competidos. Muchos autores dirigían grupos de profesionales que seguían sus ideas. Todos los métodos eran muy similares, los básicos tenían denominaciones diferentes, lo cual confundía a los clientes. Algunos autores comenzaban a hablar de estandarización, otros se oponían por completo a la idea de estándares para los métodos, otros se interesaban por la idea pero no hacían nada para lograrlo. Un equipo del OMG comenzó a considerar la estandarización, sin embargo, sólo logró una carta abierta de protesta de parte de los métodos más importantes. Grady Booch, autor de la metodología OOADA, intentó organizar una reunión informal para abordar el problema, pero tampoco tuvo éxito. Durante la OOPSLA'94, se dio a conocer la noticia de que Jim Rumbaugh (creador de OMT) había dejado General Electric para unirse a Grady Booch en la empresa Rational Software Corporation, donde comenzaron a unificar sus dos metodologías OMT y OOADA, ambas eran ya reconocidas como líderes a nivel mundial.
1995	Grady y Jim consideraban haber terminado con la guerra de métodos, consideraban ser los ganadores. Otro grupo de metodólogos sugirió la idea de formar una coalición en contra de éstos dos autores. Para la OOPSLA'95 (octubre de 1995) Grady y Jim habían preparado la primera descripción pública de su método integrado: la versión 0.8 de la documentación del Método Unificado (Unified Method), la que se considera como la primera versión del UML. A finales de este año se anunció la compra de Objectory por parte de Rational Software, motivo por el cual Ivar Jacobson se unió al equipo unificado con su metodología OOSE (Object Oriented Software Engineer), con lo que Rational celebró el lanzamiento del documento preliminar de 0.8. Estos tres autores consideraron importante unificar sus trabajos en un sólo lenguaje de modelado,

	debido a que los métodos se estaban desarrollando separada e independientemente, lo mejor era unificar una metodología y evitar diferencias innecesarias que confundieran a los usuarios. Pretendían estabilizar el mercado de la orientación a objetos para que los desarrolladores pudieran enfocarse a sus productos desarrollados más que a los elementos que deberían usar para su desarrollo. Esperaban que su colaboración mejorara todos los métodos anteriores, pretendían: • Crear un lenguaje que permitiera modelar sistemas utilizando los conceptos de la orientación a objetos, pudiera ser utilizado por máquinas y por hombres. • Pretendían establecer un acoplamiento entre conceptos y objetos ejecutables. • Querían manejar los problemas típicos de los sistemas de misión crítica. El diseño de una notación para análisis y diseño orientado a objetos, no fue sencillo, tuvieron que resolver varias cuestiones, determinar si la notación debía incluir la especificación de requerimientos, determinar si la notación debía extenderse a nivel de un lenguaje de programación visual, determinar el grado de funcionalidad y simplicidad de la notación, pues si era demasiado simple podría limitar el rango de problemas a resolver y si era muy compleja, podría llegar a abrumar al desarrollador.
1996	Booch, Rumbaugh y Jacobson, conocidos posteriormente como "Los tres amigos", construyeron, en junio y octubre de 1996, su método y le pusieron otro nombre: Unified Modeling Language (UML), Lenguaje Unificado de Modelado, versiones 0.9 y 0.91. Muchos actores importantes de la comunidad de métodos orientados a objetos no estaban dispuestos a permitir que UML fuera la última palabra. Los tres amigos invitaron a toda la comunidad a hacer aportaciones, recibieron muchas aportaciones que se incorporaron y varias organizaciones consideraron al UML como una estrategia para su negocio. Se creó una fuerza de trabajo en OMG (Object Managament Group) para llevar a cabo la estandarización en el área de los métodos. La OMG lanzó su RFP (Request for Proposal -Requerimiento de Propuestas) para que las organizaciones que lo desearan enviaran sus propuestas y contribuyeran a la definición de la versión 1 0. Con la unión de los tres metodologistas en Rational la estandandarización de UML iba teniendo éxito. Los tres amigos comenzaron a abrirse a otras vistas y recibieron muchos respaldos de grandes compañías como Hewlett-Packard, Microsoft, Orcle y Texas Instruments.
1997	En enero de este año, IBM & ObjectTime, Platinum Technology, Ptech, Taskon &, Reich Technologies y Softeam enviaron respuestas separadas RFP a OMG, estas compañías se unieron a los participantes del UML para contribuir con sus ideas. No todos los metodologistas contribuyeron al UML, surgieron disidentes como Don Firesmith (OPEN consortium), Ian Graham y Brian Henderson, quienes sostuvieron que UML estaba demasiado arraigado en técnicas de modelado de datos anticuadas, inusables para el modelando de objetos, y propusieron el OML - Open Modeling Language, (Lenguaje de Modelando Abierto), sin embargo UML dominó el panorama. Como su propuesta al OMG, Rational liberó la versión 1.0 de la documentación del UML, un lenguaje de modelado bien definido, expresivo, poderoso y aplicable de forma general. A mediados de 1997 todas las propuestas de OMG convergieron sobre UML versión 1.1, esta versión fue formalmente adoptada por el OMG a finales de 1997. El enfoque del UML 1.1

	muchos cambios que resultaron del proceso de normalización del OMG, tenía como objetivo mejorar la claridad de las semánticas del UML 1.0 e incorporar contribuciones de nuevos participantes. Poco después surgió un complicado problema de numeración de versiones, cuando se lanzó la nueva versión de UML, se le llamó 1.1 que se derivaba de la 1.0, lo cual era lógico, embargo el OMG tomó la versión 1.1 de Rational y la llamó versión 1.0. Por tanto la versión 1.1 de UML es la versión 1.0 de UML del OMG. Actualmente un grupo de trabajo trajo la versión 1.2, para unificar las numeraciones, sin embargo, no hay cambios importantes desde la versión 1.1 a 1.2., adoptada como estándar un año después (Enero 1999).

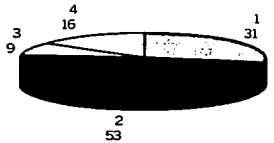
Varias compañías ya están adoptando el UML, como estándar en sus procesos de desarrollo, usando las disciplinas de modelado de negocio, administración de requerimientos, análisis y diseño, programación y pruebas.


Evolución UML
2001
2000
1999
1998
Nov '97
Sept '97
Enero '97
Jun y Oct '96
Finales '95
1995
1994
UML 2.0
UML 1.4
UML 1.3
UML 1.2
Revisiones
UML aprobado por el OMG
UML 1.1
UML 1.0
Apoyo de Oracle, Microsoft, IBM, HP y otras compañías líderes
Unified Modeling Language (UML)
Unified Method 0.8
Lanzamiento
Ivar Jacobson
OOSE
Unified Method 0.8
1ª descripción pública
Jim Rumbaugh
OMT
Grady Booch
OOADA

Antecedentes de RUP (Rational Unified Process, Proceso Unificado Racional)
El Proceso Unificado Racional (RUP), es el proceso de desarrollo vinculado a UML, desarrollados por la misma empresa y equipo de desarrollo. Mencionaremos a continuación cómo se originó este proceso.

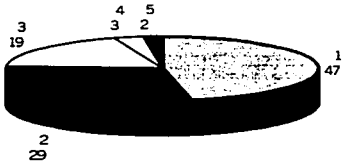
En junio de 1996, un artículo de Fortune titulado "The Trouble with Software Is... it Sucks" (El Problema con el Software es ...) revela el pobre estado de la calidad y confiabilidad del software. Posteriormente, un estudio del Standish Group hecho sobre 352 compañías de software, donde se estudiaron más de 8,000 proyectos de software, los cuales revelaron lo siguiente:

Estadísticas del software



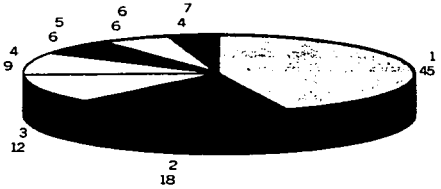
1. Proyectos cancelados antes de terminarse (81 billones dólares perdidos).
2. Proyectos con un costo 189% mayor de lo estimado.
3. Proyectos terminados a tiempo y dentro del presupuesto (compañías grandes).
4. Proyectos terminados a tiempo y dentro del presupuesto (compañías pequeñas).

Estadísticas del software



1. Software pagado nunca entregado.
2. Software entregado nunca usado.
3. Software abandonado o reformado.
4. Software que se usa después de reformado.
5. Software que se usa tal como se entrega.

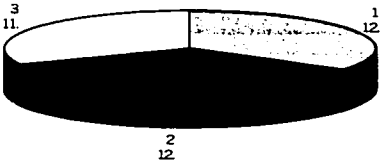
Costo de mantenimiento del software



1. Cambios en los requisitos del usuario.
2. Cambios en el formato de los datos.
3. Arreglos de urgencia.
4. Depuración de rutina.
5. Cambios en el hardware.
6. Documentación.
7. Mejoras de rendimiento.
8. Otros.

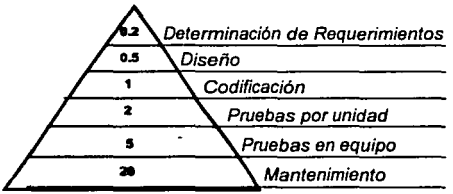
A raíz de estos datos, se preguntó a las empresas sobre las causas de estos problemas. Las tres principales razones expuestas fueron las siguientes:

Razones de la falta de calidad



1. Falta de información por parte de los usuarios.
2. Especificaciones y requerimientos incompletos.
3. Cambios en las especificaciones y requerimientos.

El análisis de éstos y otros estudios han llegado a la conclusión de que: Si el esfuerzo requerido para detectar y reparar un error durante la etapa de codificación tiene costo de 1 unidad, entonces el costo para detectar y reparar un error durante la etapa de determinación de requerimientos es entre 5 y 10 veces menor, por el contrario, el costo de detectar y reparar un error durante la etapa de mantenimiento es 20 veces más alto.



Esto demuestra que los errores cometidos durante la fase de requerimientos detectados en las fases terminales del desarrollo del sistema resultan extremadamente caros de reparar. En proyectos grandes, este tipo de error es muy frecuente.

Debido a estas contrariedades (falta de calidad, tiempo y costos elevados), varios autores se preocuparon por desarrollar un "proceso efectivo" para el desarrollo de software, entendiendo por "proceso efectivo" aquél que proporciona normas para el desarrollo eficiente de software de calidad, además de permitir la captura y aplicación de las mejores prácticas que la tecnología permite y por lo tanto, reduce el riesgo y hace el proyecto más predecible.

El proceso de desarrollo iterativo comenzó a diseñarse para estructurar mejor y controlar el proceso durante las iteraciones. El esfuerzo de Walker Royce, Rich Reitman, Grady Booch y Philippe Kruchten se unió para lograr la planificación detallada de las fases y la ordenación de las iteraciones dentro de las fases.

Evolución de RUP (Rational Unified Process, Proceso Unificado Rational)

Año	Acontecimiento
1967	Las primeras bases de RUP las encontramos en las propuestas metodológicas de Ivar Jacobson en Ericsson. Este método identificaba subsistemas en los casos de uso. También usaba los diagrama de secuencia o bien un diagrama de colaboración, diagramas de estado (con estados y transiciones) y los diagramas de transición de estados (una versión simplificada de los diagramas de actividad de UML). Los productos primordiales de este método eran: *la descripción de la arquitectura software* y la *biblioteca de mensajes*, documentos fundamentales que guiaban el trabajo de desarrollo y eran de gran utilidad para presentar el sistema a los clientes. En aquellos momentos (1968) los clientes no estaban acostumbrados a que les presentaran los productos de software como eran presentados los proyectos de ingeniería, de manera complicada. Además, este método permitían crear nuevas configuraciones de los sistemas, sólo se tenían que intercambiar unos bloques por otros que proporcionaran las mismas interfaces.
1981	Para este año comienzan a escribirse muchos libros, artículos y documentos que detallan los desarrollos de Rational. Rational se dispuso a crear un entorno iterativo que mejoraría la productividad en el desarrollo de grandes sistemas software. Se mencionaba la importancia del diseño orientado a objetos, la abstracción, la ocultación de la información, la reutilización y el prototipado.
1987	En este año Jacobson dejó Ericsson y fundó Objectory AB en Estocolmo. Durante los siguientes ocho años, él y sus colaboradores desarrollaron un proceso denominado Objectory (abreviatura de "Object Factory", fábrica de objetos), Objectory puede considerarse una extensión o reelaboración del método Ericsson. El uso de este nuevo proceso se extendió a otras industrias y países. El concepto de caso de uso se presentó formalmente en la conferencia OOPSLA de 1987, dichos diagramas comenzaron a destacar satisfactoriamente, se hicieron más claros y comenzaron a concebirse como la arquitectura que conduce a los desarrolladores y los comunica con los usuarios. Los requisitos, análisis, diseño, implementación y pruebas se representaron en una serie de modelos dirigidos por casos de uso. Los desarrolladores se guiaban por un caso de uso a través de la secuencia de modelos hasta el código fuente o bien, cuando surgían problemas, volvían hacia atrás.
1988	Surge la versión Objectory 1.0, el desarrollo de este proceso llegó a ser visto como el desarrollo de un sistema, se desarrollaba una nueva versión de Objectory a partir de una anterior. Este modo de desarrollar Objectory hizo más fácil ajustarlo para cubrir las necesidades específicas de diferentes organizaciones de desarrollo. Objectory aportó ideas sobre cómo diseñar los procesos generales sobre los cuales opera un negocio.
1990	Se dio mucho énfasis a la arquitectura y al desarrollo iterativo, estos dos conceptos fueron dos de las contribuciones más importantes que se hicieron al proceso. En este año Mike Devlin escribió un artículo sobre un proceso de desarrollo iterativo dirigido por la arquitectura. Philippe Kruchten, a cargo de la división de Prácticas de Arquitectura en Rational, firmó artículos sobre la iteración y la arquitectura.

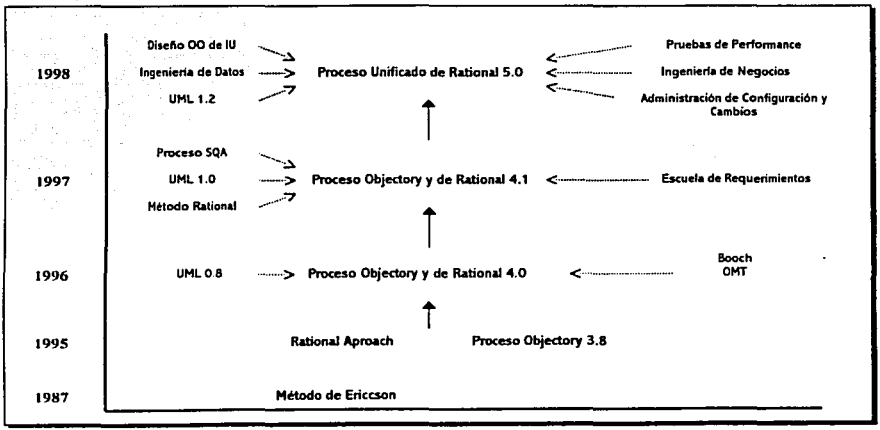
	También surgió un artículo sobre la representación de la arquitectura con cuatro vistas: la vista lógica, la de procesos, la física y la de desarrollo, más una vista adicional, de casos de uso o escenarios, que ilustraba a las primeras cuatro. La idea de este conjunto de vistas nació de la experiencia de Kruchten. Las vistas múltiples permitieron a los usuarios y a los desarrolladores encontrar lo que necesitaban para sus diferentes objetivos con la vista adecuada.
1995	Surge la primera versión interativa Objectory 3.8 . Rational Software Corporation compró Objectory AB a finales de este año teniendo como objetivo primordial unificar los principios básicos subyacentes en los procesos de desarrollo existentes. Rational había desarrollado algunas prácticas de desarrollo de software, la mayoría de ellas complementarias a las contenidas en Objectory.
1995-1997	Para ésta fecha, Objectory 3.8 demostró que se podía crear y modelar un proceso de desarrollo de software como si fuese un producto, había identificado un conjunto de modelos que documentaban el resultado del proyecto. Este proceso no tocaba el tema de administración del proyecto ni de la configuración, distribución y preparación del entorno de desarrollo, razón por la cual se le añadieron la experiencia y prácticas de Rational para formar el Process Objectory de Rational, (Rational Objectory Process, ROP) con la versión 4.1 (1997), al cual se añadieron las fases y la aproximación iterativa controlada. Se desarrolló una definición precisa de la arquitectura, considerada como la parte más significativa de la organización del sistema. Representaba la arquitectura como vistas arquitectónicas de los modelos. Se amplió el desarrollo iterativo, de un concepto relativamente general pasó a ser un método dirigido por los riesgos que consideraba la arquitectura en primer lugar. En1996 Booch, en uno de sus libros, mencionó dos "principios fundamentales" sobre la arquitectura y la iteración: • "Un estilo de desarrollo dirigido por la arquitectura es normalmente la mejor aproximación para la creación de la mayoría de los proyectos complejos basados en el software. • "Para que un proyecto orientado a objetos tenga éxito, debe aplicarse un proceso iterativo e incremental." UML se encontraba en fase de desarrollo y se incorporó como el lenguaje de modelado del ROP. El equipo de desarrollo del proceso liderado por Philippe Kruchten corrigió algunas de las debilidades del ROP, reforzó, por ejemplo, la gestión del proyecto, basada en aportaciones de Royce.
1998	Rational compró o se fusionó a otras empresas fabricantes de herramientas. Cada una de ellas aportó a la mezcla su experiencia en áreas del proceso que ampliaron más el ROP. • Requisite Inc. aportó su expenencia en gestión de requisitos. • SQA Inc. había desarrollado un proceso de prueba para acompañar a su producto de pruebas y lo añadió a la experiencia de Rational en este campo. • Pure-Atria añadió su experiencia en gestión de configuración a la de Rational. • Performance Awareness añadió las pruebas de rendimiento y las de carga. • Vigortech añadió su experiencia en ingeniería de datos.

Otra de las aportaciones al proceso fue un nuevo flujo de trabajo para el modelado del negocio basado en, que se utiliza para obtener los requisitos a partir de los procesos de negocio que el software va a cubrir. También se extendió con diseño de interfaces de usuario dirigido por los casos de uso (basado en el trabajo de Objectory AB).

A mediados de 1998 el Proceso Obectory de Rational se había convertido en un proceso capaz de soportar el ciclo de vida del desarrollo en su totalidad, pues integraba una amplia variedad de aportaciones de muchas fuentes sobre las cuales se basaron Rational y UML.
En junio, Rational publicó una nueva versión del producto, el Proceso Unificado de Rational 5.0, se pusieron, por primera vez, a disposición del público en general muchos elementos de ese proceso.

El cambio de nombre refleja el hecho de que la unificación ha tenido lugar en muchas dimensiones: unificación de técnicas de desarrollo, a través del Lenguaje Unificado de Modelado y unificación del trabajo de muchos metodologistas, no sólo en Rational sino también en las oficinas de los cientos de clientes que llevaban utilizando el proceso muchos años.

El Proceso Unificado es un producto estable, ha funcionado a lo largo de tres décadas de desarrollo y uso práctico. Su desarrollo ha recibido influencias de muchas fuentes.



APORTADORES

Grady Booch

Graduado en la academia de la fuerza aérea los Estados Unidos como Licenciado en Ciencias de la Computación en 1977, estudió una maestría en Ingeniería en Computación en la Universidad de California en Santa Barbara en 1979. Reconocido internacionalmente por su trabajo innovador sobre la arquitectura de software, el modelado y la ingeniería de procesos de software. Sus aportaciones han mejorado la eficacia de desarrolladores de software a nivel mundial. Ha estado en Rational Software Corporation como científico principal desde su fundación en 1980. Grady es uno de los desarrolladores originales de UML, también ha desarrollado varios productos de Rational incluyendo Rational Rose. Ha servido como arquitecto en numerosos sistemas de software complejo alrededor el mundo. Es el autor de seis libros, entre los que están: "*UML User Guide*" (UML Guía de Usuario) y "*Object-Oriented Analysis and Design with Applications*" (Análisis y Diseño Orientado a Objetos con Aplicaciones). También ha publicado cientos de artículos técnicos sobre ingeniería de software desde principios de los 80's, los cuales originaron el término y práctica del diseño orientado a objetos. Es miembro de la ACM, Association for Computing Machinery (Asociación para la Maquinaria Computacional), del IEEE, Institute of Electrical and Electronics Engineers (Instituto de Ingenieros Eléctricos y Electrónicos), de la AAAS, American Association for the Advancement of Science (Asociación Americana para el Adelanto de la Ciencia) y del CPSR, Computer Professionals for Social Responsibility (Computadoras Profesionales para la Responsabilidad Social). Es un seguidor de ACM y de Rational.

Dr. Ivar Jacobson

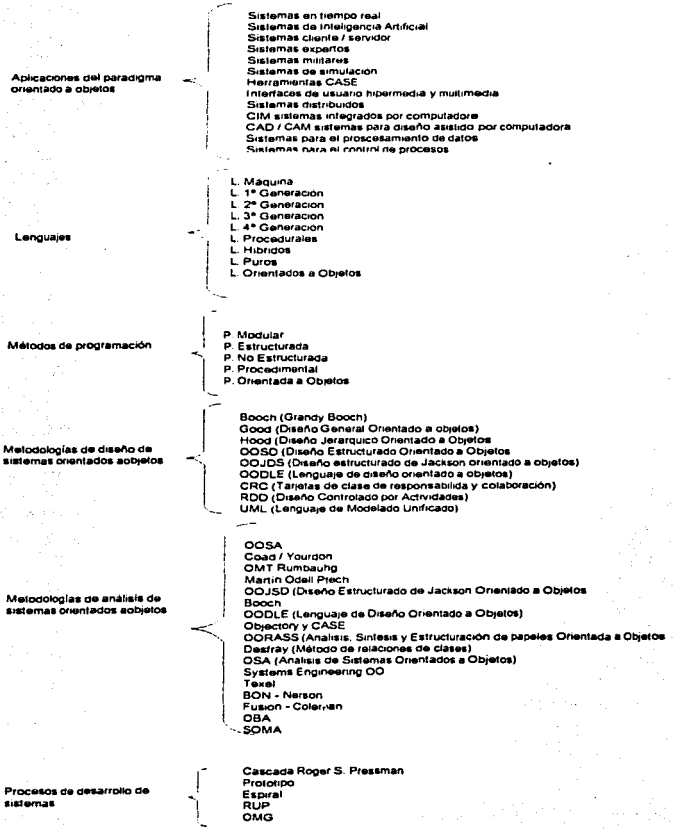
Es uno de los metodologistas principales de desarrollo de software en el mundo. Conjuntamente con Grady Booch y Jim Rumbaugh, Ivar desarrolló UML. Como vise-presidente de e-development Rational, Ivar es responsable de ayudar a Rational a definir estrategias de productos en el área de e-development· Trabaja conjuntamente con Booch y Rumbaugh para refinar y mejorar el UML. Antes de pertenecer a Rational, Ivar estuvo con Objectory AB en Suecia, compañía fundada por él, se unió a Rational Software en 1995. Entre sus contribuciones más importantes podemos mencionar: el desarrollo del método Ericsson a principios de 1967, basado en subsistemas que interactaban en colaboraciones, enfoque adoptado por el estándar de telecomunicaciones SDL. También hizo énfasis en los casos de uso para conducir el desarrollo y el proceso, a principios de 1990 extendió Objectory para incluir la ingeniería de negocio, mencionaba que es mejor entender el contexto del negocio que capturar los requerimientos. Apoyado en Rational, hizo que su proceso evolucionara al the Rational Unified Process, RUP (Proceso Unificado Rational) en 1998. Ivar es el autor principal de tres libros muy importantes: "*Object-Oriented Software Engineering--A Use Case Driven Approach*" (Ingeniería de Software Orientada a Objetos-Manejando el enfoque de Casos de Uso), "*The Object Advantage--Business Process Reengineering with Object Technology*" (La ventaja de los Objetos – Reingeniería de procesos de negocio con la Tecnología de Objetos), y el de "*Software Reuse: Architecture, Process, and Organization for Business Success*" (Reuso del Software: Arquitectura, Proceso, y Organización para el éxito del negocio). Su libro más nuevo es "*The Unified Software Development Process*" (El Proceso Unificado de Desarrollo de Software). También es autor de varios papeles ampliamente recomendados sobre la tecnología de objetos, ha participado en varias conferencias sobre la tecnología de objetos alrededor del mundo.

Dr. James Rumbaugh

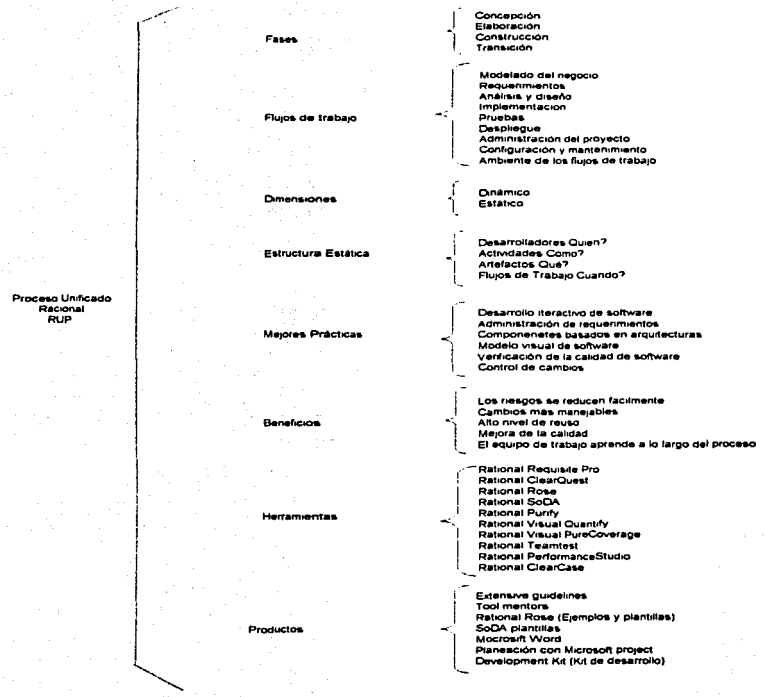
Es uno de los principales metodologistas de desarrollo de software en el mundo. Conjuntamente con sus colegas de Rational Grady Booch e Ivar Jacobson, desarrolló UML. Ha sido un líder en el desarrollo continuo de UML como representante de Rational en el OMG, contribuyó con muchos conceptos de UML. Ha trabajado con otros líderes de software en Rational en áreas como RUP y metodologías de desarrollo en tiempo real. Ha trabajado en metodologías de software, herramientas y conceptos por más de 30 años. Fue el desarrollador principal del Object Modeling Technique (OMT), método de análisis y diseño orientado a objetos, por lo que se le considera uno de los principales aportadores de UML. Antes de unirse a Rational Software Corporation en 1994, trabajó por más de 25 años en el General Electric Research and Development Center (Centro de Investigación y desarrollo de General Electric), en Schenectady, Nueva York; donde desarrolló el DSM (lenguaje de programación orientado a objetos), el árbol de estados del modelo de control, el OMT, notación de modelado orientada a objetos y el editor gráfico Object Modeling Tool (Herramienta de Modelado de Objetos). Anteriormente trabajó en diversas aplicaciones como un sistema VLSI CAD, algoritmos para explorar tomografías y en uno de los primeros sistemas operativos de tiempo compartido. Fué uno de los inventores del flujo de datos de la arquitectura de una computadora en su PhD trabajando con el Prof. Dennis en el MIT. Ha trabajado en un gran número de áreas de computación, incluyendo la semántica de cómputo, herramientas para programar la productividad, usando aplicaciones con algoritmos y estructuras de datos complejas. Trabaja continuamente sobre formas efectivas de desarrollo y de mantenimiento de sistemas grandes de software en ambientes complejos. Jim tiene un B.S. en la física desde MIT, un M.S. en astronomía de Caltech, y un Ph.D. en ciencias de la computación del MIT.

3.4 CLASIFICACIÓN

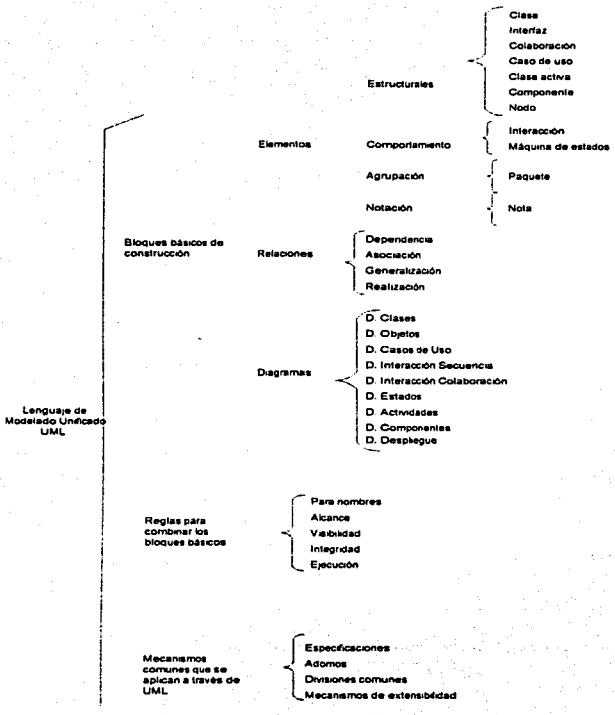
CUADRO SINÓPTICO DE EL PARADIGMA ORIENTADO A OBJETOS



CUADRO SINÓPTICO DEL PROCESO DE DESARROLLO UNIFICADO



CUADRO SINÓPTICO DEL LENGUAJE UNIFICADO DE MODELADO



Conceptos de Orientación a Objetos

OBJETO: Representa un elemento identificable con ciertas características (atributos) que puede realizar un conjunto de acciones (operaciones). Es la instancia de una clase.
CLASE: Es un conjunto de objetos que comparten una estructura común y un comportamiento común.
ABSTRACCIÓN: Es omitir las propiedades y acciones de un objeto y dejar sólo aquellas que nos interesan.

CLASIFICACIÓN: Los objetos con la misma estructura y comportamiento son agrupados en clases.

POLIMORFISMO: El comportamiento de una clase puede variar de acuerdo a ciertas circunstancias, ya que en ocasiones una operación tiene el mismo nombre en diferentes clases.
HERENCIA: Las clases son organizadas jerárquicamente. Las clases hijas conservan la estructura y comportamiento de las clases padres.

ENCAPSULAMIENTO: Característica que permite ocultar los detalles de la implementación de un objetos, los objetos ocultan la funcionalidad interna de sus operaciones, de otros objetos y del mundo exterior.

MENSAJES: Para que los objetos de un sistema trabajen en conjunto, un objeto envía a otro un mensaje para realizar una operación y el objeto receptor ejecutará la operación. Los mensajes son la forma con la cual se comunican los objetos.

REUSABILIDAD: Es la característica de los objetos, ya que pueden ser utilizados cuantas veces sea necesario independientemente del lugar o de las veces que sea referenciado.

VARIABLES DE INSTANCIA: Características o propiedades propias de un objeto, es decir, están en el interior del objeto, sin embargo pueden ser comunes a una clase de objetos.

MÉTODOS: Acciones o funciones que definirán el comportamiento exterior (mensajes) que alterarán las variables de instancia de un objeto lo que modificará su estado. Los métodos son propios de objetos, y pueden ser comunes a un clase de objetos.

CONCURRENCIA: Propiedad que distingue a un objeto activo de otro inactivo.

PERSISITENCIA: Propiedad de un objeto cuya existencia trasciende el tiempo o el espacio, por ejemplo un objeto continua existiendo a pesar de que su creador deja de existir y la ubicación de un objeto se mueve a un espacio de direcciones diferentes de aquellas donde fue creado.

OPERACIONES: Acción que puede ser realizada por un objeto.

ATRIBUTOS: Características o valores de datos asociados a los objetos de una clase a la cual pertenecen, cada atributo que se identifica debe ser atómico y debe ser tratado como una unidad. Los atributos pueden basarse en definiciones de tipos de atributos lo cual provee de una definición estándar de los mismos como longitud, dominio de valores, formato etc.
Para que un sistema sea considerado orientado a objetos es necesario que cumpla con un conjunto de características:

- Objetos
- Identidad
- Encapsulamiento
- Herencia
- Poliformismo
- Clases

3.5 MÉTODOS

Generalidades de UML

Siglas	UML - Unified Modeling Language (Lenguaje Unificado de Modelado).
Concepto	Estándar de modelado para especificar, construir, visualizar y documentar planos de software, arquitectura de hardware de un sistema de software orientado a objetos, así como para modelado de procesos de negocio u otros sistemas no-software.
Características	• Registra el análisis resultante y diseña decisiones. • Define una notación gráfica de diseño. • Es un lenguaje de modelado, no un método. • Es independiente del proceso. • Es el causahabiente a OOA&D. • Es muy expresivo y cubre con todas las vistas necesarias para desarrollar los sistemas. • Es fácil de aprender y de utilizar. • Es un modelo explícito. • Es un lenguaje gráfico muy expresivo, sin ambigüedades. • Cubre la especificación de todas las decisiones de análisis, diseño e implementación que deben realizarse al desarrollar y desplegar un sistema con gran cantidad de software. • Detrás de cada símbolo de su notación hay una semántica bien definida. • No está limitado al modelado de software, también sirve para modelar sistemas que no son software, como flujos de trabajo (workflows) en el sistema jurídico, estructura y comportamiento de un sistema de vigilancia médica de un enfermo y el diseño de hardware.
Utilidad	• Ayuda a mantener una buena comunicación con los usuarios, expertos en el dominio. • Es de gran utilidad para el desarrollo en equipo. • Ayuda a comprender el dominio del problema, aún de sistemas complejos. • Es una técnica que ayuda a mostrar los requerimientos y el diseño. • Permite crear modelos precisos, no ambiguos y completos. • Permite aplicar la ingeniería directa, es decir, generar código en algún lenguaje de programación a partir de un modelo UML. • Sus modelos pueden conectarse directamente a una gran variedad de lenguajes de programación, a pesar de que UML no es un lenguaje de programación visual. • Se pueden establecer correspondencias de un modelo UML a un lenguaje de programación o base de datos orientada a objetos, incluso a tablas en una base de datos relacional. • Se puede aplicar la ingeniería inversa, construir un modelo UML a partir de alguna serie de líneas de código. • Permite la ejecución directa de modelos, la simulación de sistemas y la instrumentación de sistemas en ejecución. • Útil para generar documentación detallada de la arquitectura de un sistema. • Útil para modelar las actividades de planificación de proyectos y gestión de

<table>
<tr><td></td><td colspan="4">versiones.
• Su vocabulario y reglas indican cómo crear y leer modelos bien formados</td></tr>
<tr><td>Propósito</td><td colspan="4">Disponer de un lenguaje de modelado estándar que ayude a los desarrolladores a modelar sus sistemas de software antes de construirlos. (Grady Booch)
Facilitar la comunicación entre usuarios y el equipo de desarrollo.
Diseñar todos los puntos clave dentro del desarrollo.</td></tr>
<tr><td>Autores</td><td colspan="4">Grady Booch, Jim Rumbaugh, Ivar Jacobson, Martin Fowler, Buschmann, Robert Martin, Cook and Daniels, Goldberg & Rubin, McConnell, y Graham.</td></tr>
<tr><td>Utilidad en la programación</td><td colspan="4">• Los diferentes diagramas de UML son útiles en diferentes etapas de la codificación.
• El primer paso en el proceso de codificación es comenzar con un modelo conceptual que describa los conceptos del dominio del negocio.
• Los diagramas, especialmente el diagrama de clases y el diagrama de interacción, ayudan a ordenar las ideas y a codificar más fácilmente.
• Los diagramas son los prototipos rápidos.
• Una vez completado el código, los diagramas sirven como documentación del sistema para el desarrollador o para otros personas.</td></tr>
<tr><td>¿Dónde se utiliza?</td><td colspan="4">Pensado principalmente para sistemas con una gran cantidad de software, en ámbitos como:
• Sistemas de información de empresas.
• Bancos y servicios financieros.
• Transporte.
• Defensa/industria aeroespacial.
• Electrónica médica.
• Ámbito científico.
• Servicios distribuidos basados en la web.</td></tr>
<tr><td>Elementos principales del modelo conceptual de UML</td><td>I. Bloques básicos de construcción de UML</td><td>1. Elementos</td><td>A. Estructurales:</td><td>Clase
Interfaz
Colaboración
Caso de uso
Clase activa
Componente
Nodo</td></tr>
<tr><td></td><td></td><td></td><td>B. De comportamiento:</td><td>Interacción
Máquina de estados</td></tr>
<tr><td></td><td></td><td></td><td>C. De agrupación:</td><td>Paquete</td></tr>
<tr><td></td><td></td><td></td><td>D. De anotación:</td><td>Nota</td></tr>
<tr><td></td><td></td><td>2. Relaciones</td><td colspan="2">Dependencia
Asociación
Generalización
Realización</td></tr>
</table>

		3. Diagramas	Diagrama de clases Diagrama de objetos Diagrama de casos de uso Diagrama de secuencia Diagrama de colaboración Diagrama de estados Diagrama de actividades Diagrama de componentes Diagrama de despliegue
	II. Reglas que dictan cómo se pueden combinar estos bloques básicos	Para nombres De alcance De visibilidad De integridad De ejecución	
	III. Mecanismos comunes que se aplican a través de UML	Especificaciones Adornos Divisiones comunes Mecanismos de extensibilidad	

Importancia de los modelos

Los modelos dentro de la ingeniería del software representan un papel muy importante, ya que proporciona una visión general del problema a resolver, ayudando a entender mejor el problema que se está desarrollando. Los modelos son una simplificación de la realidad, abstraen los elementos más importantes sin importar el resto.
Proporcionan los planos (detallados o generales) del sistema. Un buen modelo incluye los elementos que tienen una gran importancia y omite los elementos no relevantes.

Ventajas:

1. Ayudan a visualizar y comprender cómo es el sistema.
2. Ayudan a especificar la conducta de la estructura del sistema.
3. Son una guía base para la construcción del sistema.
4. Documentan las decisiones que se deben tomar.
5. Permiten realizar estimaciones razonables de recursos (tiempo, dinero, personas) a utilizar.
6. Ayudan a comprender sistemas complejos. ya que sin ellos sería imposible comprender el sistema en su totalidad.
7. Ayudan a comunicar las ideas con otras personas.
8. Son una buena técnica para atacar los problemas, puesto que se divide el problema general en pequeños subproblemas de manera que se puedan resolver.
9. Permite al equipo de trabajo desarrollar más rápidamente el sistema y a construir el software adecuado.

Características:

1. *Los modelos deben tener una profunda influencia al enfocar el problema y la forma de solución.* Se deben elegir los modelos adecuados, que permitan comprender los problemas más difíciles, como

consecuencia de esto se desarrollará software de confianza, que haga lo que esperan los usuarios. Por el contrario unos modelos erróneos sólo harán que uno se centre en aspectos irrelevantes.

2. *Deben poder ser expresados en diferentes niveles de precisión.* En ocasiones, un pequeño y sencillo modelo ejecutable de la interfaz es lo que se necesita; otras veces hay que bajar a nivel de los bits. Los mejores tipos de modelado son los que permiten elegir el grado de detalle dependiendo de quién está viendo el sistema y por qué necesita verlo.

3. *Deben estar conectados a la realidad para considerarlos modelos factibles.* Los modelos deben tener una clara conexión con la realidad, todos los modelos simplifican la realidad, sólo debe cuidarse que esas simplificaciones no ocultan detalles importantes.

4. *Deben integrarse en un conjunto de modelos, un modelo simple no es suficiente.* Un único modelo no es suficiente, se requiere de varios modelos conectados entre si, sobre todo en sistemas que se tornan complejos. Se deben tener modelos que se puedan construir y estudiar separadamente, sin que esto afecte su interrelación.

Diagramas utilizados en la notación UML:

Parte estática:

- Diagrama de casos de uso
- Diagrama de clases
- Diagrama de componentes
- Diagrama de despliegue

Parte Dinámica:

- Diagrama de secuencia
- Diagrama de colaboración
- Diagrama de actividades
- Diagrama de estados

DIAGRAMA DE CASOS DE USO

Ivar Jacobson, uno de los "Tres amigos" precursores de UML, describió la importancia de los casos de uso, los introdujo como elemento primario del desarrollo del software y diseñó un diagrama para su representación gráfica.

Un diagrama de casos de uso está compuesto por un conjunto de *actores*, *casos de uso* y *relaciones* entre ellos, donde los *actores* representan los diferentes roles desempeñados por los usuarios del sistema (personas, dispositivos de hardware, sistemas mecánicos o un sistema externo) interesados en obtener alguna respuesta observable y de valor por parte del sistema. Los *casos de uso* representan la funcionalidad que el sistema ofrece a sus usuarios. Las *relaciones* son una secuencia de mensajes intercambiados entre los casos de uso y uno o más actores.

Este diagrama describe una secuencia de operaciones (normales y excepcionales) desarrolladas por un sistema en respuesta a un evento iniciado por un actor sobre el sistema. Indican lo que se espera del sistema, en términos de entradas y salidas, pero no se describe la forma en cómo se implementan éstas, sólo expresa requerimientos.
Estos diagramas se obtienen hablando con los usuarios habituales y analizando con ellos lo que desean que haga el sistema. Se debe abordar cada requisito, darle un nombre, escribir un texto descriptivo breve y un contenido gráfico particular que lo distingue de los otros.

Se puede realizar un caso de uso por cada objetivo del usuario, cada uno por si solo debe representar un sólo aspecto del sistema y el conjunto de ellos es la representación gráfica de la vista estática completa de casos de uso de un sistema. Los casos de uso pueden ser pequeños o grandes y su nivel de detalle depende del nivel de especificación de requerimientos funcionales, entre mayor es el nivel de detalle de las especificaciones, mejor es la planificación y control del proyecto; obteniendo asi un sistema correctamente diseñado.

Los casos de uso pueden asociarse posteriormente a los diagramas de interacción o bien a los diagramas de estados para describir el comportamiento representado por un caso de uso.

Utilidad del diagrama:

- Permiten descubrir y tener claros los objetivos reales del usuario, lo cual sirve para considerar las diferentes formas de cumplirlos.
- Representan los requisitos funcionales del sistema.
- Proporcionan un medio para que los desarrolladores, usuarios finales y expertos en el dominio comprendan de una forma común el sistema, coadyuvando a que pueden intercambiar opiniones. Los usuarios comunican su vista externa del sistema a los desarrolladores y estos construyen la vista interna.
- Ayudan a los desarrolladores a comprender y abordar problemas, aún de sistemas complejos (con muchas operaciones, objetos, relaciones, etc), con los casos de uso se puede abordar directamente cada elemento, de acuerdo con el modo en el que los usuarios utilizarán el sistema.
- Modelan la vista de casos de uso estática de un sistema, la cual cubre el comportamiento del sistema (servicios visibles que proporciona el sistema en el contexto de su entorno).
- Capturan el comportamiento deseado de un sistema (subsistemas, clases o interfaces individuales), sin especificar cómo se implementa ese comportamiento, para visualizar, especificar, construir y documentar el comportamiento esperado del sistema. Denotan sólo los comportamientos esenciales, se centran en las cuestiones más importantes para el usuario final, por lo que no deben ser excesivamente genéricos o demasiado específicos.
- Permiten que los desarrolladores especifiquen a los usuarios finales las diferentes formas en que se debe utilizar el sistema y a verificar el sistema conforme se desarrolla (validación de la arquitectura).
- Producen algo de valor para un actor, por ejemplo, el cálculo de un resultado, la generación de un nuevo objeto o el cambio de estado a otro objeto.
- Se conjunta el comportamiento común en un sólo casos de uso que pueden incluir los demás casos de uso que lo necesiten, sin necesidad de describirlo nuevamente.
- Se centran las variantes en un solo caso de uso, colocando ese caso de uso en otros que lo extienden.

- Describen el flujo de eventos de forma clara para que alguien externo al sistema lo entienda fácilmente.
- Sirven de base para establecer casos de pruebas para cada elemento del sistema durante el desarrollo, según vayan evolucionando. Aplicados a la totalidad del sistema representan una fuente excelente de pruebas y de integración del sistema.
- Ayudan al proveedor a obtener bases medibles y acotadas para una correcta planificación, pues demarcan objetivamente lo que se espera del sistema, ayudan a clasificar los requerimientos y asignarles alguna complejidad y estimaciones de esfuerzo.
- Son una herramienta poderosa dentro de procesos de desarrollo iterativos, ya que sobre cada caso de uso se puede realizar el análisis, diseño, construcción y pruebas. Las etapas clásicas de análisis, diseño y construcción quedarían incluidas dentro de cada caso de uso. Tanto el modelo conceptual de clases (análisis), el modelo de diseño de clases, los diagramas de interacción de UML y la base de datos(diseño), como el producto de software probado (construcción) se van completando a medida que se avanza sobre los casos de uso y se tiene una versión final de éstos sólo al término del desarrollo.

Elementos del diagrama:

- Actores.
- Casos de uso.
- Relaciones (dependencia, generalización y asociación).
- Pueden contener notas y restricciones.
- Pueden contener paquetes para agrupar elementos del modelo en partes mayores.
- Pueden incluir instancias de casos de uso, especialmente cuando se quiera visualizar un sistema específico en ejecución (escenarios).

Actores:

Es una entidad externa que representa un rol de algún usuario específico del sistema que interactúa directamente con él participando en casos de uso específicos. Los roles representados pueden ser de personas, dispositivos de hardware, sistemas mecánicos o un sistema externo que necesite información del sistema a desarrollar. Un mismo actor puede jugar dos o más roles. Una instancia de un actor, representa una interacción individual con el sistema de una forma específica.

Se pueden definir categorías generales de actores (como estudiante) y especializarlos (estudiante preparatoria) a través de relaciones de generalización. Los actores llevan a cabo uno o más casos de uso.

En sistemas complejos puede resultar difícil obtener una lista de todos los casos de uso, por lo que puede resultar más fácil definir la lista de actores y después determinar los casos de uso de cada actor.

Existen situaciones en las que los actores desempeñan un papel importante, por ejemplo, en la configuración de un sistema con varios tipos de usuarios, donde cada usuario tendría una lista asociada de los actores y los casos de uso que puede ejecutar cada uno. El saber cuáles son los casos de uso relacionados a cada actor es una gran utilidad. También son útiles para especificar políticas de seguridad.

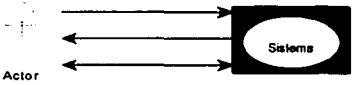
Los actores pueden tener varios papeles con respecto a un caso de uso. Pueden obtener un valor del caso de uso o sólo participar en él. Dependiendo de cómo se utilice la relación entre los actores será la importancia de los papeles que desempeñen los actores.

Notación:
Se representan como personajes

Nombre del Actor

Los actores pueden interactuar con el sistema de 3 formas:

1. Proporcionar información al sistema
2. Recibir información del sistema
3. Proporcionar y recibir información del sistema.



Caso de uso:
Es la descripción de un conjunto de acciones (normales y excepcionales), que ejecuta un sistema, un subsistema, una clase o una interfaz, para producir un resultado tangible de valor para un actor, sin embargo, no especifica cómo lo hace. Se pueden tener casos de uso que son versiones especializadas de otros casos de uso, casos de uso incluidos como parte de otros, casos de uso que extienden el comportamiento de otros casos de uso básicos. Se puede describir el comportamiento común y reutilizarlo en un conjunto de casos de uso. Lo importante es comprender los casos de uso y los objetivos del usuario que satisfacen.

Cada caso de uso debe tener un nombre representativo que los distingue de otros casos. Al nombre solo se llama *nombre simple*, éstos son más comunes, también existen los *nombres de camino*, que constan del nombre del caso de uso precedido del nombre del paquete en el que se encuentra.

Validar clave Seguridad: Validar clave

Nombre simple y Nombre de camino

El nombre de un caso de uso puede constar de texto con cualquier número de letras, números y la mayoría de los signos de puntuación, excepto los dos puntos (:), ya que se utiliza para separar el nombre de un caso de uso del nombre del paquete que lo contiene. Los nombres de los casos de uso describen algún comportamiento del vocabulario del sistema que se está modelando.

Notación:
Se representan como óvalos o elipses

Nombre del Caso de Uso

Todos los diagramas de casos de uso deben documentarse, se describe el flujo de eventos de forma textual, de forma clara para que alguien ajeno al sistema lo entienda fácilmente. Cuando se escribe este flujo de eventos se debe incluir el nombre del caso de uso, cómo y cuándo empieza y acaba el caso de uso, cuándo interactúa con los actores y qué objetos se intercambian, las precondiciones, el flujo principal o básico, los flujos alternos del comportamiento, las excepciones y las postcondiciones. Conviene separar el flujo principal de los flujos alternativos porque un caso describe un conjunto de secuencias, no una única secuencia. Cada una de las variantes se puede expresar en una secuencia diferente, donde cada una representa un posible flujo a través de todas las variantes.

Plantilla para documentar los casos de uso:

Nombre del Caso de Uso:
El cual debe ser representativo.
Actores:
Lista de los actores que participan en el caso de uso.
Breve descripción:
Describe brevemente las operaciones que implica el caso de uso.
Precondiciones:
Es una lista de los pasos que deben darse antes del caso de uso.
Flujo Principal:
Es la descripción del flujo principal, el cual siempre debe cumplirse.
Flujos Alternos:
Son posibles actividades que pueden llevar a cabo los actores.
Excepciones:
Lista una serie de condiciones por las que no se da el caso de uso.
Postcondiciones:
Describe en qué características se queda el caso de uso.

Ejemplo:

Nombre del Caso de Uso: Utilizar cajero automático.
Actores: Cliente
Breve descripción: El caso de uso inicia cuando el cliente desea sacar dinero del cajero.
Precondiciones: Que el cliente tenga una cuenta de ahorros en el banco.
Flujo principal: El caso de uso inicia cuando el sistema pide al Cliente un número de identificación personal (NIP, Número de identificación personal). El Cliente puede introducir un NIP a través del teclado. El Cliente acepta la entrada pulsando el botón Enter. El sistema comprueba este NIP para ver si es válido (E1). El clinte podra realizar las siguientes operaciones:

SUBFLUJO	ACTIVIDADES
A1. Retirar dinero	El cliente retira una cantidad en efectivo.
A2. Consultar saldo	El cliente consulta su saldo en pantalla.
A3. Imprimir saldo	El cliente puede imprimir su saldo.
A4. Cancelar	El cliente cancela su operación.

Flujos Alternos:
A1. Retirar dinero: El cliente introduce la cantidad a retirar (E2)(E3)(E4). El cajero procesa la operación y proporciona la cantidad indicada. El sistema disminuye la cantidad retirada al saldo disponible y muestra el nuevo saldo en pantalla, el cajero pregunta si se desea imprimir el ticket, el cliente puede aceptar presionando Si o No para rechazar.
A2. Consultar saldo: El cajero muestra el saldo del cliente en pantalla. No se efectúa ningún cambio a la cuenta del cliente.
A3. Imprimir saldo: El cajero muestra el saldo en pantalla y pregunta al cliente si desea imprimirlo, el cliente presiona enter para confirmar, el cajero proporciona un ticket con el saldo disponible del cliente. No se efectúa ningún cambio a la cuenta del cliente.
A4. Cancelar: El cliente puede cancelar su transacción en cualquier momento pulsando el botón Cancelar, reiniciando de esta forma el caso de uso. No se efectúa ningún cambio a la cuenta del cliente.

Excepciones:
E1. Si el NIP no es válido, el cliente podrá teclear nuevamente su número de NIP. El cajero proporciona tres oportunidades para introducir un NIP correcto, si no se proporciona un NIP válido el caso de uso termina.
E2. Si la cantidad es mayor a 3000, el cajero mostrará un mensaje de error en el que indique que no puede retirar una cantidad mayor a 3000.
E3. Si la cantidad tecleada es superior al saldo disponible del cliente, el cajero mostrará un mensaje de error indicando que no se dispone de la cantidad indicada.
E4. Si la cantidad tecleada no es un múltiplo de 50, el cajero indicará que se debe introducir una cantidad correcta, en múltiplos de 50.

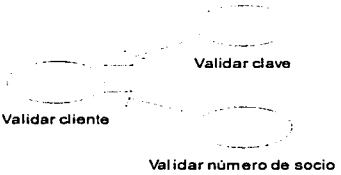
Postcondiciones:
El cajero queda disponible para procesar la siguiente operación.

Relaciones:
Los actores se conectan a los casos de uso a través de *relaciones (asociaciones)*. Una asociación entre un actor y un caso de uso indica que el actor y el caso de uso se comunican entre sí, y cada uno puede enviar y recibir mensajes.

Tipos de relaciones:
1. Generalización: Donde el caso de uso hijo hereda el comportamiento y el significado del caso de uso padre; el hijo puede agregar o redefinir el comportamiento del padre, el hijo puede colocarse donde aparezca el padre (ambos pueden tener instancias concretas).

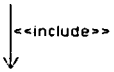
Notación: Se representa con una linea continua con una punta de flecha vacia.

Ejemplo: Se tiene el caso de uso *Validar cliente*, responsable de verificar la identidad del cliente. Además, podria haber dos hijos especializados de este caso de uso: *Validar clave* y *Validar número de socio*, los cuales se comportarian como Validar cliente y pueden aparecer dondequiera que este aparezca, aunque ambos tienen su propio comportamiento, el primero comprueba una clave textual, el segundo comprueba el número que tiene como socio.

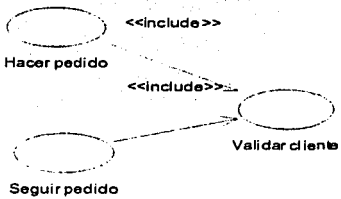


2. Inclusión: Esta relación se utiliza cuando se tiene un comportamiento similar en varios casos de uso y no se quiere copiar la descripción en todos ellos. Su uso evita describir un mismo flujo de eventos repetidas veces, se pone el comportamiento común en un caso de uso aparte (caso de uso común) y se permite que otros casos de uso base los incluyan (tomen), siempre que necesiten usar esa funcionalidad. Este tipo de relación ayuda a identificar los comportamientos comunes y evita repetirlos.

Notación: Una relación de inclusión se representa como una dependencia, usando la palabra *include*.



Ejemplo:

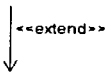


En este ejemplo, los casos de uso *Seguir pedido y Hacer pedido* necesitan de la funcionalidad que tiene el caso de uso *Validar cliente.*

En este tipo de relaciones no hay un actor asociado con el caso de uso común. Incluso si lo hay, no se considera que esté llevando a acabo los demás casos de uso.

3. Extensión: Esta relación se usa cuando se tiene un caso de uso que es similar a otro, pero hace un poco más. Se tiene un caso de uso que extiende al caso de uso base.
Esta relación se utiliza para modelar la parte de un caso de uso que el usuario puede ver como comportamiento opcional del sistema, separándolo del obligatorio. También se puede utilizar para modelar un subflujo separado que se ejecuta sólo bajo ciertas condiciones o para modelar varios flujos que se pueden insertar en un punto dado, controlados por la interacción explícita con un actor. La relación de extensión ayuda a distinguir las variantes del sistema.

Notación: La relación de extensión se representa como una dependencia con la palabra *extend.* Los puntos de extensión del caso de uso base se pueden listar en un comportamiento extra. Estos puntos de extensión sólo son etiquetas que pueden aparecer en el flujo del caso de uso base. En este tipo de relaciones se supone que un actor dado se encargará tanto del caso de uso base como de todas las extensiones.



Ejemplo: E*stablecer prioridad* es un punto de extensión, si se trata de una instancia de un pedido con prioridad, se desarrolla el flujo para este caso base y en el punto de extensión se ejecutará el comportamiento del caso de uso que extiende *Hacer pedido urgente* y después se continuará con el flujo normal. Si hay varios puntos de extensión, el caso de uso que extiende al caso base mezclará sus flujos en orden. En circunstancias normales, el caso de uso base se ejecuta sin importar la prioridad del pedido.

Un caso de uso puede tener más de un punto de extensión y puede aparecer más de una vez identificándolos por el nombre.

Para identificar los casos de uso de tipo extend, se puede obtener primero, el caso de uso normal, en cada paso de ese caso de uso, preguntarse qué puede fallar, cómo podria funcionar de modo diferente, entre otras cuestiones y posteriormente dibujar todas las variaciones como extensiones del caso de uso dado. Con frecuencia habrá un buen número de extensiones, pueden listarse por separado para que sean fáciles de entender.

Se pueden dividir un caso de uso complejo en un caso de uso normal y unas cuantas extensiones, luego se construye el caso de uso normal en una sola iteración y las extensiones como partes de una o varias iteraciones posteriores.

El organizar los casos de uso con relaciones de inclusión y de extensión ayudan a desarrollar el sistema de una forma sencilla, equilibrada y comprensible. Ambos implican la factorización de comportamientos comunes de varios casos de uso, dejando un solo caso de uso común que es empleado, o extendido, por otros caso de eso. La intención es la que cambia.

Escenarios

Es una secuencia especifica de acciones que ilustra un comportamiento especifico del sistema bajo una combinación particular de condiciones, son una instancia de los casos de uso, en ocasiones es usado como sinónimo de caso de uso. Un sistema puede tener varios casos de uso que capturen su comportamiento, y cada caso de uso puede expandirse en varias decenas de escenarios, es decir un caso de uso de uso puede tener muchas realizaciones (diferentes maneras de llevar a cabo un caso de uso). Para cada caso de uso, puede haber escenarios principales (definen secuencias esenciales) y escenarios secundarios definen secuencias alternativas. Se pueden tener varias realizaciones, con el fin de discutirlas y determinar con cuál se va a trabajar.
Es conveniente guardar información acerca de las realizaciones descartadas, incluyendo las notas de por qué se descartaron.

Paquetes

A medida que vayan creciendo los modelos, los casos de uso pueden agruparse en paquetes. Se pueden organizar especificando relaciones de generalización, inclusión y extensión entre ellos. Estas relaciones se utilizan para factorizar el comportamiento común (extrayendo ese comportamiento de los casos de uso en los que incluye) y para factorizar variantes (poniendo ese comportamiento en otros de uso que lo extienden).

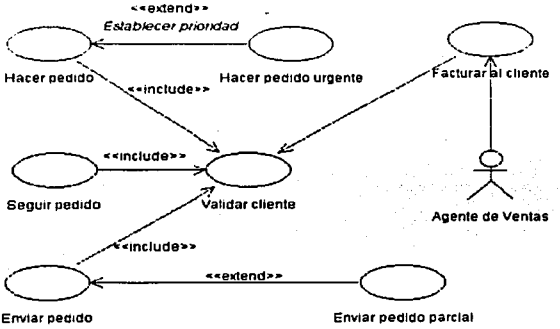
Modelado del diagrama:
Los casos de uso pueden utilizarse para:

I. Modelado de comportamiento de un elemento

- Se deben identificar los actores que interactúan con el elemento. Los actores candidatos pueden requerirse directa o indirectamente para ejecutar las funciones del elemento, en ocasiones los actores deberán tener un comportamiento especifico para poder ejecutar sus tareas.
- Organizar los actores identificando los roles más generales y los más especializados.
- Considerar las formas más importantes que tiene cada actor de interactuar con el elemento, considerando las interacciones que implican el cambio de estado del elemento o su entorno.
- Considerar las formas excepcionales en las que cada actor puede interactuar con el elemento.
- Organizar estos comportamientos como casos de uso, utilizando las relaciones de inclusión y extensión para factorizar el comportamiento común y distinguir el comportamiento excepcional.

Ejemplo
Un sistema de ventas que interactúa con clientes, los cuales efectúan pedidos y pueden hacer un seguimiento de sus propios pedidos. El sistema envía los pedidos y las facturas a los clientes. En ocasiones, según la urgencia de los clientes, se puede adelantar parte del pedido (pedidos parciales).

Se pueden tener los casos de uso Hacer pedido, Seguir pedido, Enviar pedido, Facturar cliente. El comportamiento común en este caso es el caso de uso Validar cliente y las variantes están representadas con el caso de uso Enviar pedido parcial. Cada caso de uso debe incluir una especificación de su comportamiento.



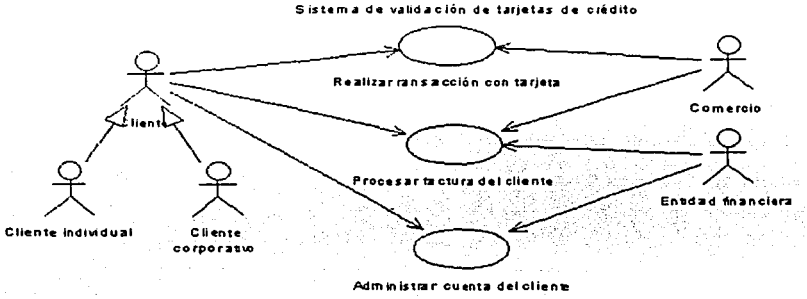
II.- Modelar el contexto de un sistema.

1. Se identifican los actores en torno al sistema, considerando qué grupos requieren ayuda del sistema para llevar a cabo sus tareas, qué grupos son necesarios para ejecutar las funciones del

sistema, qué grupos interactúan con el hardware externo o con otros sistemas software, y qué grupos realizan funciones secundarias de administración y mantenimiento.
2. Se especifica el significado de los roles de los actores.
3. Se organizan los actores similares en jerarquías de generalización/especialización.
4. Se proporciona un estereotipo para entender el sistema.
5. Se introducen los actores en un diagrama de casos de uso y se especifican las vías de comunicación de cada actor con los casos de uso del sistema.

Ejemplo

Sistema de validación de tarjetas de crédito. Se tienen dos categorías de clientes: clientes individuales y corporativos. También hay actores que representan otras instituciones, como Comercio, con el cual el cliente realiza una transacción para comprar un artículo o servicio, y Entidad financiera, que por lo general es una entidad bancaria donde el usuario tiene la tarjeta de crédito.



III. Modelar los requisitos de un sistema

Un requisito es una característica de diseño, una propiedad o un comportamiento de un sistema. Los requisitos se pueden expresar de varias formas, desde texto sin estructura hasta expresiones en un lenguaje formal. La mayoría de los requisitos funcionales de un sistema, si no todos, se pueden expresar con casos de uso, los diagramas de casos de uso son fundamentales para manejar esos requisitos.

Para este tipo de modelado se especifica qué debería hacer el sistema, independientemente de cómo se haga, se especifica el comportamiento deseado del sistema. Se ve el sistema entero como una caja negra, se ve su entorno (elementos externos) y cómo reacciona ante ellos, pero no se ve cómo funciona por dentro.

Se determina qué incluir como actor, se debe restringir el entorno del sistema para que incluya sólo los actores necesarios en la vida del sistema, se establece qué se espera que haga el sistema, sin importar

cómo lo hace. Al construir un sistema es importante que al comenzar exista un acuerdo sobre qué debería hacer el sistema con total seguridad, la comprensión de los requisitos evolucionará conforme se vaya implementando el sistema de manera iterativa e incremental. En ocasiones no se tendrán vínculos claros de los casos de uso con actores específicos, sin embargo se puede conocer primero cómo se comporta el sistema para saber cómo y quién los utilizará de manera correcta.

Pasos para modelar los requisitos de un sistema:

1. Establecer el contexto del sistema, identificando los actores a su alrededor.
2. Considerar el comportamiento que cada actor espera del sistema o requiere que éste le proporcione.
3. Nombrar esos comportamientos comunes como casos de uso.
4. Factorizar el comportamiento común en nuevos casos de uso que puedan ser utilizados por otros; y factorizar el comportamiento variante en nuevos casos de uso que extiendan los flujos principales.
5. Modelar esos casos de uso, actores y en un diagrama de casos de uso.
6. Adornar esos casos de uso con notas que enuncien los requisitos no funcionales; puede que haya que asociar varias de estas notas al sistema global.

Ejemplo del diagrama:

Este casos de uso omite las relaciones entre los actores y los casos de uso, pero añade casos de uso adicionales que son invisibles para el cliente normal, aunque son comportamientos fundamentales del sistema. Este diagrama es importante porque ofrece un punto de partida común para los usuarios finales, expertos del dominio y los desarrolladores para visualizar, especificar, construir y documentar sus decisiones sobre los requisitos funcionales del sistema.

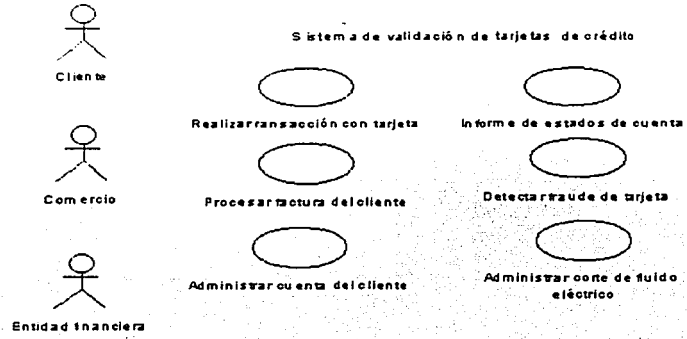


DIAGRAMA DE CLASES
Muestra un conjunto de elementos estáticos como clases, interfaces, relaciones y colaboraciones, los cuales modelan los requisitos funcionales del sistema, los servicios que éste debe proporcionar a los usuarios, así como los detalles del vocabulario del sistema.

Utilidad del diagrama:
- Es una abstracción precisa del sistema.
- Ayudan a especificar, visualizar y documentar los elementos estáticos.
- Ayudan a relacionar los diagramas de componentes y los de despliegue.
- Contiene un conjunto bien definido de responsabilidades.
- Representación clara de la especificación de abstracciones y su implementación.
- Es comprensible, sencillo, extensible y adaptable para describir el problema o la solución.
- Modelan el vocabulario del sistema.
- Muestra los diversos tipos de relaciones estáticas entre las clases, se consideran una parte importante de los métodos OO.

Elementos del diagrama:
- Clases
- Interfaces
- Relaciones de dependencia, generalización y asociación
- Colaboraciones
- Pueden contener notas, restricciones, paquetes o subsistemas

Clases:
Cada clase representa un conjunto de objetos que comparten las mismas propiedades (atributos), comportamiento (operaciones), relaciones (asociaciones, dependencia, agregaciones) y significado. Las clases representan las características relevantes del vocabulario propio del sistema a desarrollar o problema estudiado, como personas, cosas, software, hardware o cosas conceptuales. Cada clase debe estar bien delimitada conceptualmente y tener bien definidas sus responsabilidades. Los sustantivos (cosas, personas, hechos) pueden ser las clases del modelo, posteriormente, para cada clase se puede agrupa una lista de atributos y operaciones.

Componentes de la clase:
Nombre: Es el nombre asignado a cada clase que la distinga de las demás, debe ser un nombre singular que caracterice de la mejor forma la abstracción. Los nombres deben venir directamente del vocabulario del sistema, es decir, si en un sistema de inscripción a cada uno de las materias impartidas se les conoce como asignaturas, Asignatura debe ser el mejor nombre para la clase y no el de Materia.

Atributo: Cada atributo representa una propiedad de la clase, se identifica mediante un nombre. Los atributos son sustantivos especificos que toman algún valor, por ejemplo, el atributo Semestre podria tomar los valores 1, 2 , 3, 4, 5, ó 6, dependiendo del semestre en el que se encuentre cierto alumno. En un momento dado, el objeto de una clase tendrá valores especificos para cada uno de los atributos de su clase.

Operación: Es una abstracción de algo que se puede hacer a un objeto, también es conocido como método. Es un servicio que proporciona la clase a otras clases que lo soliciten.
Representa el comportamiento de una clase que es compartido por todos los objetos de la clase.

Estereotipos
Cada clase tiene como máximo un estereotipo, los cuales pueden ser:

Clase Interfaz: Modelan la interacción entre el sistema y sus actores, es decir, recibir o presentar información, peticiones de y hacia los usuarios o a sistemas externos. Modelan las partes del sistema que dependen de los actores, pues representan abstracciones de ventanas, formularios, paneles, interfaces de comunicación, impresoras, sensores, terminales.

Clase Entity o Entidad: Modela información y comportamiento asociado, generalmente persistente. Pueden representar tareas internas del sistema, fenómenos de la vida real. Los valores de sus atributos se proporcionan generalmente por un actor.

Clase Control: Representan coordinación, secuencia, transacciones y control de otros objetos; se usan con frecuencia para encapsular el control de un caso de uso en concreto. O bien para representar derivaciones y cálculos completos, como lógica del negocio que no pueden asociarse con ninguna información concreta, de larga duración, almacenada por el sistema.

***Notación*:**
Una clase se representa por un rectángulo con tres secciones:

Sección 1: Indica el Nombre de la clase y opcionalmente el estereotipo de la clase entre << >>. Es necesario que cada clase tenga su propio nombre, diferente al de otras, que la distinga. El nombre de las clases se indica con sustantivos singulares de longitud variada, sin embargo, se recomienda que sean nombres cortos, extraídos del vocabulario del sistema que se está modelando. Los nombres compuestos de palabras múltiples se ponen juntas y la primera letra de cada palabra adicional se escribe en mayúsculas. Ejemplo: Alumno, Profesor, SistemaCobro
El nombre de la clase puede ser simple, sólo el nombre de la clase, por ejemplo Alumno o puede ser nombre de camino, el cual también indica el nombre del paquete en el que se encuentra la clase, por ejemplo, Alumno:Persona.

Sección 2: En esta sección se coloca el listado de los atributos que tiene la clase. Puede listarse el nombre del atributo y el tipo de dato, por ejemplo: FechaNacimiento:Date. También puede indicarse algún valor inicial (default). Se recomienda que el nombre del atributo inicie con una letra en minúscula, en caso de tener mas de una palabra deberán ser identificadas con una letra mayúscula al principio de cada una de ellas; por ejemplo: aluApPaterno, aluApMaterno.

Sección 3: Se colocan las operaciones o métodos de la clase. Se puede colocar sólo el nombre de la operación, que consiste en un verbo corto representativo del comportamiento de la clase. El nombre puede contener más de una palabra, cada una de ellas debe empezar con mayúscula, por ejemplo: ObtenerCalificacion. Las operaciones pueden especificarse más, agregando por ejemplo, parámetros, en caso de que los requiera y en el caso de las funciones un valor de retorno, por ejemplo: ObtenerCalificacion(ClaveAlumno:int, ClaveAsignatura:int)double).

La visibilidad de los atributos y operaciones se puede indicar mediante los símbolos "+" para el caso de los públicos, "-" si son privados, o "#" si son protegidos (Rational Rose utiliza su propia notación gráfica).

No es necesario indicar las tres secciones, los atributos y las operaciones pueden suprimirse, aunque esto no quiere decir que la clase no tenga atributos ni operaciones, en realidad los tiene solo que no se representan en el diagrama.

Tarjetas CRC

Una herramienta importante para poder identificar los atributos y operaciones, aparte del análisis de casos de uso, son las tarjetas CRC, las cuales describen las responsabilidades y colaboraciones entre las clases, son tarjetas de papel de 4 x 6 pulgadas, y se dividen en tres secciones:

Nombre de la Clase	
Responsabilidad	**Colaboración**

Una vez identificadas las clases, se especifican sus responsabilidades y se listan en la sección correspondiente en la tarjeta, dichas responsabilidades se traducirán en un conjunto de atributos y operaciones. Posteriormente se identifican las clases con las que colaborará la clase para cual se está elaborando la tarjeta.

Relaciones:

Las relaciones muestran las colaboraciones entre clases, son líneas que representan las conexiones entre clases. Existen diferentes tipos de relaciones:

1. Dependencia

Estas relaciones indican que un cambio en la especificación (interfaz o comportamiento) de una clase puede afectar a otra que la utilice como argumento, dicho en otras palabras, cualquier cambio en las clases independientes afecta a las clases dependientes, esto no sucede necesariamente a la inversa.

Notación:

Se representa mediante una línea punteada con una flecha apuntando hacia la clase independiente, puede etiquetarse con un estereotipo que identifique el tipo de dependencia.

Ejemplo:

Supongamos que en un sistema de inscripciones en línea se desea que las pantallas de inscripción que se muestren dependan del tipo de alumno que vaya a inscribirse, es decir, si es una alumno de 1er. semestre se muestren ciertas asignaturas, si el alumno se inscribe a 2do. semestre, que el sistema muestre en la

pantalla las materias correspondientes a éste semestre, y así sucesivamente. Esto podría representarse de la siguiente manera:

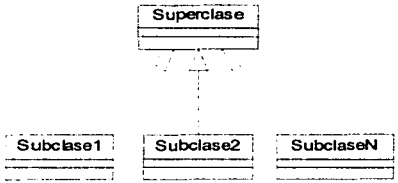
Donde la pantalla depende del tipo de alumno que ingrese al sistema.

Las relaciones de dependencia pueden tener un nombre, es recomendable usarlo sólo si es necesario, es decir, si se tiene un modelo con muchas dependencias y tengan que distinguirse.

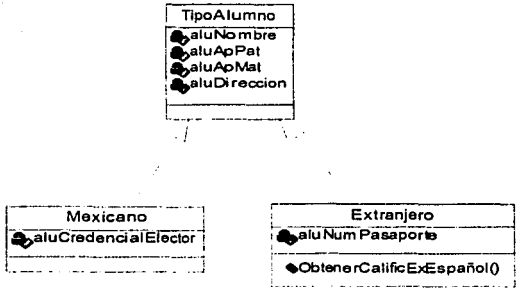
2. Generalización

Es una relación entre una clase más general (superclase o padre) y una más especializada (subclase o hija). Sirve para representar la herencia entre clases, donde la subclase hereda la estructura y el comportamiento (atributos y operaciones) de la superclase, y puede además agregar otros atributos y operaciones propias. La superclase puede tener varias subclases. Las clases hijas tendrán sus atributos y operaciones propios, también pueden redefinir las de su padre. Una operación de un hijo con la misma asignatura que una operación del padres se conoce como *polimorfismo*.
También suele llamarse relación "es-un" o "es-una", ya que es la expresión que se usa entre las clases relacionadas.

Notación:
Se utiliza una línea continua con una punta de flecha vacía apuntando hacia el padre.



Ejemplo:
Siguiendo con el ejemplo de inscripción de alumnos, supongamos que existen dos tipo de alumnos, mexicanos y extranjeros, y que los mexicanos presentan su credencial de elector como identificación oficial y para los extranjeros se almacena su número de pasaporte como identificación. También podemos decir que los alumnos extranjeros deben presentar y aprobar un examen de español para poder inscribirse. Este esquema podría modelarse de la siguiente manera:



3. Asociación:
Indica los enlaces entre los objetos de las clases relacionadas. Los objetos de una clase tienen referencias a los objetos de otra clase, ya sea para acceder a los atributos o para utilizar las operaciones que ofrecen. Esta relación especifica que los objetos de una clase están conectados con los objetos de otra, una vez que se tiene la relación de asociación entre dos clases, se puede navegar de un objeto de una clase hasta un objeto de la otra clase, y viceversa. Dado un objeto de la clase, se puede conectar con otros objetos de la misma clase. Una asociación que conecta exactamente dos clases se dice que es binaria. Aunque no es frecuente, se pueden tener asociaciones que conecten más de dos clases, estas son asociaciones n-arias.

Notación:
Se representa como una linea continua que conecta la misma o diferentes clases. Aparte de esta linea, hay otros elementos que se aplican a las asociaciones, como:

Nombre: Describe la naturaleza de la relación. Se le puede dar una dirección al nombre por medio de una flecha que apunte en la dirección en la que se pretende que se lea el nombre.

Rol: Expresan el papel que cumplen las clases en la asociación, se indican mediante nombres ubicados en el extremo de la clase correspondiente.

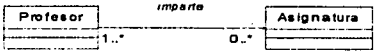
Multiplicidad del rol de la asociación: Es un rango de valores o un valor explicito que determina cuántos objetos pueden participar en la asociación, se indica en los extremos. Puede indicarse una multiplicidad de:

Exactamente 1	1
Cero o uno	0..1
Muchos	0..*
Uno o más	1..*
Un número exacto.	3

Se pueden especificar multiplicidades más complejas utilizando una lista, como 0..3 , 6..*, lo que significa cualquier número de objetos excepto 4 y 5.

Navegabilidad: Sentido en el que se recorre la asociación.

Ejemplo:

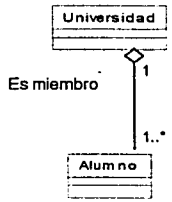


Tipos de asociación:
Asociación binaria: Asociación que navega en un solo sentido, se agrega una flecha en el extremo de llegada.
Se representa mediante una linea sólida entre las clases relacionadas.

Asociación terciaria o de mayor orden: Cuando la asociación incluye a más de dos clases.
Se representa mediante un diamante al que se conectan con lineas las clases relacionadas.

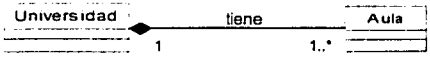
Agregación: Es una forma especial de asociación en la cual especifica una relación entre las clases, donde el "agregado" indica el todo y el "componente" es una parte del mismo. Es una relación de "todo/parte", en la cual una clase representa algo grande (el todo), que consta de elementos más pequeños (las partes),es una relación entre el todo y sus partes.

Notación:
Se representa agregando un diamante vacio en el extremo que corresponde al todo.



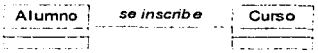
Composición: Es otro caso especial de agregación, también modela relaciones "todo/parte", la diferencia es que representa una relación más fuerte entre el "todo" y sus "partes", éstas últimas pueden crearse después del todo y sólo tendrán sentido como parte del todo, por lo que son destruidas junto con el todo, pueden eliminarse antes. El tiempo de vida de la clase que son parte del todo es determinado por el tiempo de vida de la clase que representan el todo

Notación:
Se utiliza un diamante lleno en el extremo que corresponde al todo.

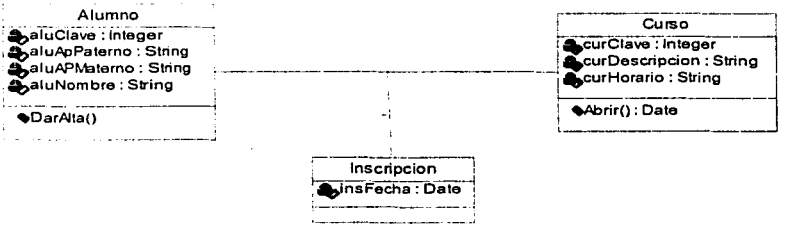


Perspectivas bajo las cuales se pueden dibujar o leer los diagramas de clase:

Conceptual: Se dibujan los diagramas con el objetivo de que demuestren el vocabulario propio del sistema que se está estudiando. Los conceptos se relacionan de manera natural con las clases que los implementan, sin embargo, no hay una correlación directa. Bajo esta perspectiva los diagramas de clase se dibujan sin importar el software en el que se implementará, son independientes del lenguaje. Bajo esta perspectiva el diagrama de clases podría quedar:

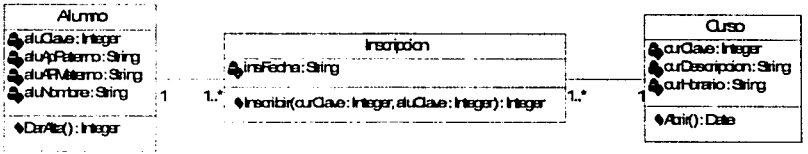


De Especificación: Se toma en cuanta el software, pero sólo interesan las interfaces (tipos), no su implementación (clases). Un tipo representa una interfaz que puede tener diversas implementaciones.
Esta perspectiva suele ser un poco difícil, ya que el concepto de clase en un lenguaje OO combina las interfaces y las clases (implementación de esas interfaces).



De Implementación: Se tienen clases y se implementan por completo. Esta es una de las perspectivas más empleadas, sin embargo, suele ser mejor adoptar la perspectiva de especificación.

La perspectiva bajo la cual se lea o dibuje un diagrama de clases es de vital importancia, se deben dibujar bajo una perspectiva única y clara. Las perspectivas utilizadas pueden especificarse de la siguiente manera: Las clases se marcan con <<clase de implementación>> para mostrar la perspectiva de implementación y con <<tipo>> para el caso de la perspectiva de especificación y la conceptual.



Con el diagrama de clases se puede modelar:

1. El vocabulario de un sistema:
Se modelan los elementos que son importantes para los usuarios. Las tarjetas CRC y el análisis de casos de uso son herramientas de gran ayuda para esta tarea.
Para modelar el vocabulario del sistema se debe:
Identificar las cosas que utilizan los usuarios o programadores para describir el problema o la solución. Para cada abstracción se identifica un conjunto de responsabilidades, cada clase debe estar claramente definida y tener un conjunto de responsabilidades bien delimitadas, posteriormente se proporcionan los atributos y operaciones necesarios en cada clase para cumplir con esas responsabilidades.

2. Modelado de la distribución de responsabilidades en un sistema:
En este tipo de modelado, las clases deben tener un conjunto equilibrado de responsabilidades, no se deben tener clases demasiado grandes, ya que estas ocasionan modelos dificiles de cambiar y no muy reutilizables, tampoco se deben tener clases demasiado pequeñas, porque resultarán muchas abstracciones dificiles de manejar o comprender, cada clase debe representar bien una única cosa.
Para el modelado de este tipo se debe:
Identificar un conjunto de clases que colaboren entre ellas para llevar a cabo algún comportamiento, posteriormente se identifica el conjunto de responsabilidades para cada una de esas clases.

3. Modelado de cosas que no son software
También se pueden modelar cosas que no son software, por ejemplo personas o robots, para los cuales el sistema necesitará almacenar información de ellos.

4. Modelado de tipos primitivos
También pueden modelarse cosas que pueden extraerse del lenguaje de programación utilizado para la implantación de la solución. Los tipos primitivos se refiere a : enteros, caracteres, cadenas o tipos enumerados que uno mismo puede crear.
A continuación mostraremos un ejemplo más completo de un diagrama de clases.

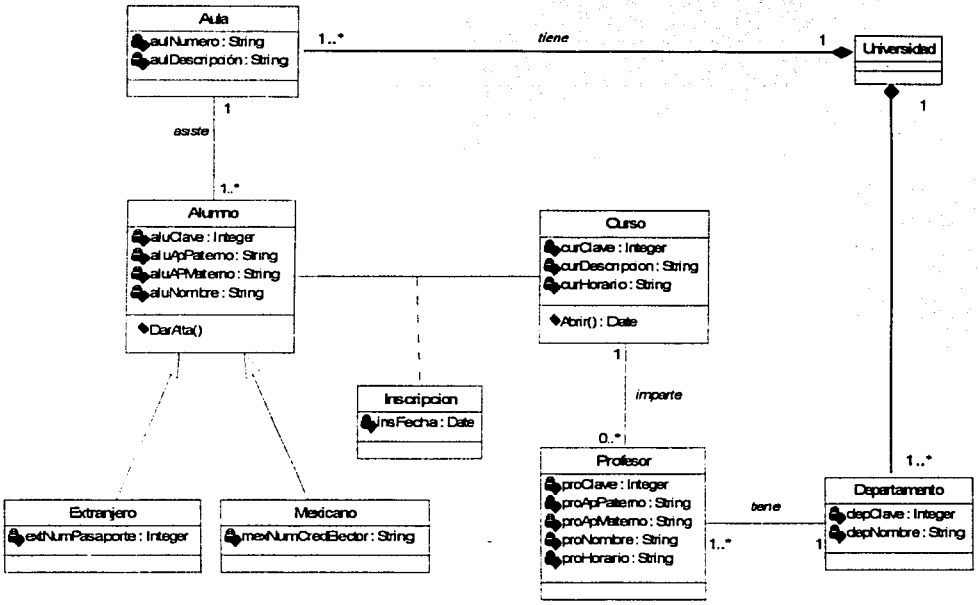

Aula
aulNumero : String
aulDescripción : String
1..*
tiene
1
Universidad
1
asiste
1
1..*
Alumno
aluClave : Integer
aluApPaterno : String
aluAPMaterno : String
aluNombre : String
DarAlta()
Curso
curClave : Integer
curDescripcion : String
curHorario : String
Abrir() : Date
1
imparte
0..*
Inscripcion
insFecha : Date
Profesor
proClave : Integer
proApPaterno : String
proApMaterno : String
proNombre : String
proHorario : String
1..*
tiene
1
Departamento
depClave : Integer
depNombre : String
1..*
Extranjero
extNumPasaporte : Integer
Mexicano
mexNumCredElector : String

DIAGRAMA DE COMPONENTES

Representa los componentes software, como código fuente, binario o ejecutable, unidos por medio de relaciones de dependencia (que representa generalmente compilación). Una relación de dependencia indica que un componente utiliza otro, por lo cual depende de él.

Es la implementación física de la funcionalidad descrita en la arquitectura lógica (clases, colaboraciones).

Si se tiene un sistema sencillo cuya implementación consta de un único archivo ejecutable no es necesario hacer un modelado de componentes. Si se trata de un sistema con varios ejecutables y bibliotecas de objetos asociados, el modelado de componentes ayudará a visualizar, especificar, construir y documentar las decisiones que se han tomado sobre el sistema físico.

Algunos de los componentes tendrán que ser construidos desde cero pero otros serán reutilizados.

Utilidad del diagrama:

- Permite el modelado de base de datos (tablas, triggers, procedimientos almacenados), archivos y documentos, así como las relaciones entre estos y los demás ejecutables, bibliotecas e interfaces del sistema. Aunque los componentes primero mencionados no son ejecutables ni bibliotecas, son importantes para el despliegue físico del sistema, por ejemplo el modelado de archivos de datos, documentos de ayuda, guiones (scripts), archivos diarios (log), archivos de inicialización y de instalación/desinstalación es importante para controlar la configuración del sistema.
- Modelan la vista de implementación estática de un sistema, vista que se ocupa de la administración de configuraciones de las partes de un sistema, formada por componentes que pueden ensamblarse de varias maneras para producir un sistema ejecutable.
- Permiten hacer ingeniería directa e inversa de una manera fácil y directa, pues los componentes son cosas físicas (ejecutables, bibliotecas, tablas, archivos y documentos) y, por lo tanto, son cosas cercanas al sistema en ejecución. Se realiza ingeniería directa al modelar un componente que representa el código fuente, una biblioteca binaria o un ejecutable para una clase o colaboración. La ingeniería inversa se hace a partir de código fuente, bibliotecas binarias o ejecutables, que corresponden con clases o colaboraciones. En ocasiones el aplicar ingeniería inversa no es del todo recomendable, ya que en ocasiones existe pérdida de información.
- Representan el empaquetamiento físico de componentes.
- Describe conjuntos de clases relacionadas técnicamente.
- Puede ser usada para varios propósitos, como los paquetes, sólo que desde la perspectiva física.
- Muestra la organización y las dependencias entre un conjunto de componentes.
- Es un tipo especial de diagrama de clases que se centran en los componentes de un sistema.
- Son importantes para visualizar, especificar y documentar sistemas con una gran cantidad de componentes.
- Ayudan a controlar las versiones y a administrar las configuraciones de esas partes conforme evoluciona el sistema.
- Permite modelar las interfaces para programación de aplicaciones (APIs, Application Programming Interfaces). Las API's representan de programación del sistema que se pueden modelar mediante interfaces y componentes. Una API es una interfaz realizada por uno o más componentes.
- Útil para el modelado de la configuración y dependencias de compilación de los archivos de código fuente, los cuales almacenan los detalles de las clases, interfaces, colaboraciones y otros elementos, son un paso intermedio para crear los componentes binarios físicos.
- Se utilizan para visualizar los aspectos estáticos de los componentes físicos y sus relaciones para especificar sus detalles la construcción.
- Agrupa elementos del modelo del sistema en bloques mayores.

Elementos del diagrama:

- Componentes.
- Relaciones dependencia, generalización, asociación y realización.
- Interfaz.
- Pueden contener notas y restricciones.
- Pueden contener paquetes o subsistemas.

Componentes:

Parte física y reemplazable de un sistema, representa una unidad de código (fuente, binario o ejecutable), muestra las dependencias en tiempo de compilación y ejecución. Puede tener distintas etapas de desarrollo: compilación, link o ejecución.
Están fuertemente relacionados con el lenguaje de programación y las herramientas utilizadas. Los componentes conforman con un conjunto de interfaces y proporciona la realización de esas interfaces.
Se dice que los componentes son reemplazables porque pueden ser sustituidos con otros que conformen con las mismas interfaces. Este mecanismo de insertar o reemplazar componentes es transparente al usuario del componente y es posible gracias a modelos de objetos (como COM+ y Enterprise Java Beans) que requieren poca o ninguna intervención de herramientas que automatizan el proceso.
Un componente representa una parte estructural y/o de comportamiento de un sistema más grande, cada componente es un bloque fundamental sobre el cual se pueden diseñar y construir sistemas.

UML define cinco estereotipos estándar que se aplican a los componentes:

Executable	Componente que se puede ejecutar en un nodo.
Library	Biblioteca de objetos estática o dinámica.
Table	Componente que representa una tabla de una base de datos.
File	Componente que representa un documento que contiene código fuente o datos.
Document	Especifica un componente que representa un documento.

Caracteristicas:

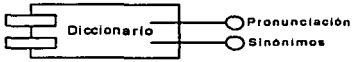
- Tienen un nombre, conformado por una cadena de texto de cualquier longitud, puede incluir letras, números y signos de puntuación, excepto los dos puntos (:). Se pueden emplear los *nombres simples* o *nombres de camino* (nombre seguido de dos puntos y el nombre del paquete que lo contiene).
- Puede realizar un conjunto de interfaces.
- Puede tener instancias y pueden anidarse.
- Tiene operaciones a las cuales pueden acceder otros componentes a través de sus interfaces.

Como puede observase tiene algunas similitudes con las clases, sin embargo tienen algunas diferencias.

Notación:

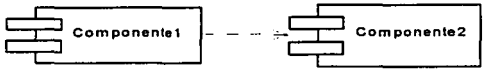
Son representados mediante un rectángulo con dos rectángulos pequeños del lado izquierdo. El nombre del componente y si es necesario el tipo dentro del componente. Puede además tener otros elementos como: objetos, nodos, componentes.

Pueden mostrar también las interfaces que implementan y los objetos que contienen.
Ejemplo:



Relaciones:
De dependencia: Significa que un componente necesita de otro para completar su definición.

Notación:
La dependencia entre dos componentes se muestra como una flecha punteada.

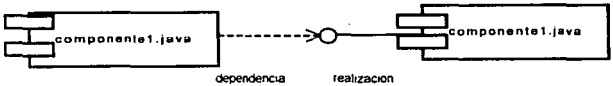


También se pueden organizar especificando entre ellos relaciones de dependencia, generalización, asociación (incluyendo agregación) y realización.

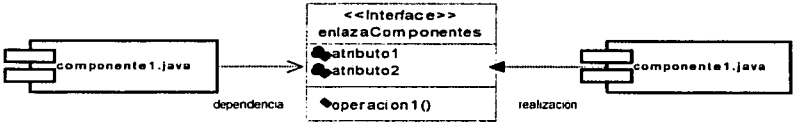
Interfaz:
Conjunto de operaciones utilizadas para especificar un servicio de una clase o de un componente, enlazan a los componentes entre si.
Primero se especifican las interfaces y posteriormente se proporcionan componentes que realicen dichas interfaces, y otros componentes accesaran a dichos servicios a través de la interfaz.
Las interfaces son elementos clave en la creación o reconstrucción de un sistema, permiten añadir nuevos componentes y reemplazar otros existentes. Se especifica una interfaz y se integra en el sistema cualquier componente que conforme o proporcione esa interfaz. Se puede extender el sistema haciendo que los componentes proporcionen nuevos servicios a través de otras interfaces, las cuales a su vez, pueden ser utilizadas por otros componentes. A este proceso se le conoce como sustitución binaria.

Notación:
Se puede mostrar la relación entre un componente y sus interfaces de dos formas:

1. Forma icónica: Se representa a la interfaz mediante un icono en su forma contraida. El componente que la realiza se conecta a la interfaz con una relación de realización simplificada.



2. Forma expandida: Se representa a la interfaz en su forma expandida, mostrando sus operaciones, el componente que realiza la interfaz se conecta a ella con una relación de realización completa.



En ambos casos el componente que accede a los servicios del otro componente a través de la interfaz se conecta a ella con una relación de dependencia.

Tipos de interfaz:
Interfaz de exportación: Es una interfaz que el componente ofrece corno servicio a otros componentes (interfaz realizada por un componente).
Interfaz de importación: Interfaz utilizada por un componente, sobre la cual el componente puede basarse para construir.
Un componente puede importar y exportar muchas interfaces.

Mostrar sólo aquellas interfaces necesarias para comprender el significado de ese componente en el contexto dado.

Paquetes o subsistemas:
Los componentes se pueden organizarse agrupándolos en paquetes de la misma forma en que se organizan las clases.

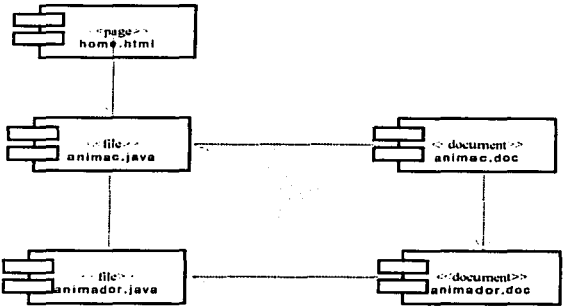
Modelado del diagrama:
Con este diagrama pueden modelarse los siguientes componentes:

1. Archivos Fuente (Componentes producto del trabajo): Este tipo de componentes son los productos finales del proceso de desarrollo (programación), son archivos de código fuente de una o más clases y archivos de datos. Se modelan estos componentes como archivos junto a sus relaciones de dependencia. Estos componentes no participan directamente en un sistema ejecutable, pero son los productos del trabajo de desarrollo que se utilizan para crear el sistema ejecutable.
El modelar estos componentes permite organizar los archivos de código fuente en grupos más grandes. También permite controlar las nuevas versiones incrementales de archivos de código fuente que se vayan produciendo, esto ayuda a visualizar la historia de las versiones producidas.
Permite aplicar la ingenieria inversa sobre un conjunto de archivos de código fuente para visualizar la red de dependencias de compilación.

Modelado:

- Primero se debe identificar el conjunto de código fuente y se modelan como componentes estereotipados como archivos. En sistemas grandes suelen utilizarse paquetes para mostrar los grupos de archivos de código fuente.

- Se muestra un valor etiquetado que indique información como: el número de versión del archivo de código fuente, el autor y la fecha de la última modificación. *
- Se modelan las dependencias de compilación entre estos archivos mediante dependencias. *



2. Ejecutables (Componentes de despliegue): Como producto final, debe entregarse un sistema ejecutable a los usuarios, es decir, una versión final, la cual está compuesta por un conjunto de partes consistentes y completas, tales como bibliotecas dinámicas (.dll**) y ejecutables (.exe), componentes necesarios y suficientes para formar un sistema ejecutable. Estos componentes se generan a partir de código objeto y elementos de bibliotecas dinámicas.
Este tipo de modelado es muy útil cuando se desarrollan aplicaciones complejas, que por lo general se componen de un ejecutable principal (un archivo .exe), de partes auxiliares como bibliotecas (archivos .dll, .class y .jar), bases de datos, archivos de ayuda y de recursos. Un ejemplo de sistema con esas características son los sistemas distribuidos, en los cuales hay varios ejecutables y otras partes distribuidos entre varios nodos, o sistemas que compartan componentes entre varias aplicaciones.

El modelado de estos componentes ayuda a visualizar, especificar construir y documentar la configuración de las versiones ejecutables, incluyendo los componentes de despliegue que cada versión y las relaciones entro esos componentes. Se pueden utilizar para hacer ingeniería directa (con un nuevo sistema) e ingeniería inversa (con un sistema existente).

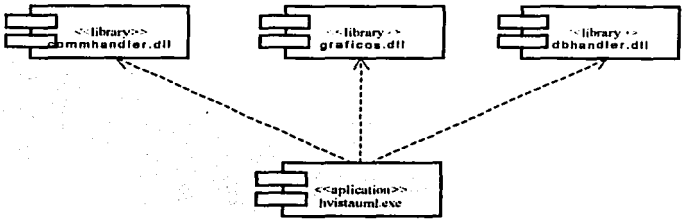
Modelado:

- Identificar el conjunto de componentes a modelar.
- Considerar el estereotipo de cada componente de este conjunto: ejecutables, bibliotecas, tablas, archivos y documentos.
- Identificar las relaciones entre componentes. En ocasiones esto incluirá las interfaces que son exportadas (realizadas) por ciertos componentes, e importadas (utilizadas) por otros. Si se quiere

* Se pueden utilizar herramientas para manejar este punto.

** .dll, Dinamic Link Libraries

mostrar las líneas de separación del sistema, hay que modelar explícitamente las interfaces. Para un nivel mayor de abstracción, se omiten esas relaciones, mostrando sólo las dependencias entre los componentes.



3. Bases de datos físicas: El modelo de una base de datos física representa el almacenamiento de la información en las tablas de una base de datos relacional o en las páginas de una base de datos orientada a objetos.
El diagrama de componentes de UML es útil para modelar las bases de datos físicas (almacenamiento de información en una base de datos relacional, orientada a objetos o híbrida*, para una recuperación posterior), así como para los esquemas lógicos de bases de datos (captura el vocabulario de los datos persistentes y la semántica de las relaciones de un sistema).

La correspondencia entre un modelo lógico de base de datos con UML y una base de datos orientada a objetos es directo, sin embargo con de datos relacional no es tan directo ni simple. En el aspecto de la herencia, hay que tomar decisiones acerca de cómo hacer corresponder las clases con tablas. Puede aplicarse tres estrategias:

1. Definir una tabla separada por cada clase, este es un enfoque sencillo pero poco práctico, porque introduce grandes problemas de mantenimiento cuando se crean nuevas clases hijas o se modifican clases padre.
2. Dejar las jerarquías de herencia de forma que todas las instancias de cualquier clase en una jerarquía tengan el mismo estado. El inconveniente de este enfoque es que se acaba almacenando información superflua en muchas instancias.
3. Separar el estado relativo a las clases padre e hija en tablas diferentes. Este enfoque se corresponde mejor con la jerarquía de herencia, su inconveniente es que la navegación por los datos requiere muchas operaciones de unión (joins) en tablas.

Al diseñar una base de datos física, se deben tomar decisiones acerca de cómo asociar las operaciones definidas en el esquema lógico de la base de datos. Las bases de datos orientadas a objetos hacen que esta asociación sea bastante transparente. Pero con las bases de datos relacionales hay que tomar

*Híbrida objeto-relacional

algunas decisiones acerca de cómo se implementarán estas operaciones lógicas. Para ello existen varias alternativas:

1. Las operaciones ABMC (Altas, Bajas, Modificaciones, Consultas) se pueden implementar con llamadas estándar SQL u ODBC.
2. Los comportamientos más complejos (como las reglas del negocio) se pueden asociar a disparadores o procedimientos almacenados.

Modelado:

- Identificar las clases del modelo que representan el esquema lógico de la base de datos.
- Seleccionar una estrategia para hacer corresponder estas clases con tablas, considerando la distribución física de las bases de datos. La estrategia de correspondencia se verá afectada por la localización en la que se quiere que se encuentren los datos en el sistema desplegado.
- Hay que crear un diagrama de componentes que contenga componentes estereotipados como tablas.
- Utilizar herramientas que ayuden a transformar el diseño lógico en un diseño físico.

4. Sistemas adaptables: Sirven para modelar el comportamiento (estático o dinámico) de los sistemas. En los sistemas distribuidos y complejos es necesario modelar vistas dinámicas, que contengan agentes móviles, componentes que migran de un nodo a otro para llevar a cabo una transacción, para ello será necesario utilizar una combinación de diagramas de componentes, diagramas de objetos y de interacción.

Modelado:

- Considerar la distribución física de los componentes que pueden migrar de un nodo a otro. La localización de la instancia de un componente se puede especificar marcándola con un valor etiquetado de localización en un diagrama de componentes (técnicamente hablando, un diagrama que contiene instancias es un diagrama de objetos).
- Si se quiere modelar las acciones que hacen que migre un componente, hay que crear el correspondiente diagrama de interacción que contenga instancias de componentes. Un cambio de localización se puede describir dibujando la misma instancia más de una vez, pero con diferentes valores para su valor etiquetado de localización.

DIAGRAMAS DESPLIEGUE

Cuando se desarrolla un sistema es importante diseñarlo lógica y físicamente. En la parte no tangible se pueden diseñar clases, interfaces, colaboraciones, iteraciones y estados. En la parte tangible se encuentran componentes, es decir, empaquetamientos físicos de esos elementos lógicos y nodos que representan básicamente el hardware sobre el que se despliega y ejecutan esos componentes.

Los componentes que se desarrollan o se reutilizan como parte de un sistema con gran cantidad de software deben desplegarse sobre algún hardware para su ejecución. Los nodos al igual que los componentes, pertenecen al mundo material y son un bloque de construcción importante en el modelado de aspectos físicos del sistema. Un nodo representa un recurso computacional que generalmente tiene algo de memoria y capacidad de procesamiento, un nodo representa un procesador o dispositivo sobre el que se pueden desplegar los componentes.

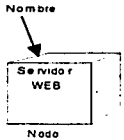
Utilidad del diagrama

- Se utiliza para definir la distribución de los componentes de un sistema
- Se utilizan para el modelado de los procesadores y los dispositivos que conforman al sistema
- Se utilizan para representar los empaquetamientos físicos de componentes
- Los nodos se utilizan para modelar la topología del hardware
- Se utiliza para representar los empaquetamientos físicos de esos elementos lógicos

Elementos
- Nodos
- Componentes
- Conexiones

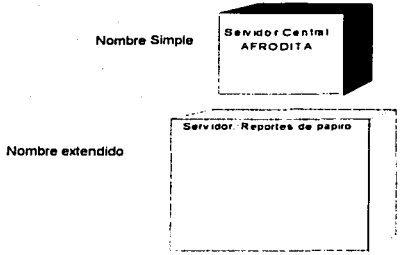
Nodo:
Es un elemento físico que existe en tiempo y ejecución representa un recurso computacional (hardware) que por lo general tiene una memoria y capacidad de procesamiento, se utiliza para representar procesadores y dispositivos.

Generalmente se representa como un cubo:

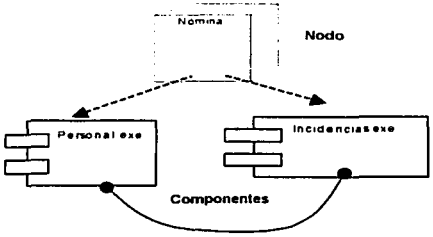


Cada nodo debe tener un nombre que lo distinga , ese *nombre puede ser simple* y *o bien el nombre de camino* que consta del nombre del nodo precedido por el nombre del paquete en el que se encuentra , auque por lo general solo se escribe su nombre. El nombre de un nodo puede estar formado por texto

cualquier número de letras, números y ciertos signos de puntuación excepto los dos puntos que se utilizan como separador entre el nombre del nodo y el paquete que lo contiene.



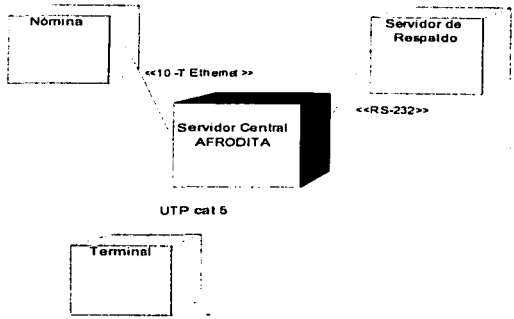
Componentes:
Los componentes son la parte intangible, es decir, parte del sistema que se desarrolla, el cual se despliega sobre un hardware (Nodo). Los componentes se parecen a los nodos, ya que ambos tienen nombre y pueden participar en relaciones de dependencia, generalización y asociación, se pueden anidar y tienen instancias.
Dos diferencias importantes que existen entre los nodos y componentes son:
Los componentes son la parte intangible que se ejecuta sobre los nodos, los cuales representan la parte fisica .
Un componente se convierte en un conjunto de elementos lógicos como clases y colaboraciones. Una clase puede ser implementada por dos o más componentes y a su vez un componente puede desplegarse sobre uno o más nodos.



Se le llama *unidad de distribución* al conjunto de componentes asignados a un nodo como un grupo. Los nodos pueden ser organizados en paquetes.

Conexiones:
Una relación entre nodos se denomina asociación y esta representa una conexión física entre nodos, como puede ser una conexión Ethernet una línea o bus compartido. Se utiliza el nodo sombreado para resaltar los procesadores o dispositivos centrales.

Ejemplo de un diagrama de despliegue:



NOTA: los nodos son similares a las clases, ya que se les pueden especificar operaciones y atributos , y disponen de las asociaciones, esto incluye roles, multiplicidad y restricciones.
Un procesador es un nodo que tiene la capacidad de ejecutar un componente y un dispositivo es un nodo sin capacidad de procesamiento, para modelar los procesadores y dispositivos se debe tener en cuenta:

- Hay que identificar los elementos computacionales que integrarán la plataforma del sistema.
- Si estos procesadores y dispositivos son genéricos hay que especificarlos como tal y de la misma forma se especificarán como tal los procesadores y dispositivos especiales.

Puntos que se deben considerar para modelar la distribución de un sistema:
- Se debe desplegar el conjunto de componentes que residen en cada nodo
- Este diagrama se debe descomponer sólo hasta el nivel que sea necesario
- Debe proporcionar una abstracción bien definida del hardware
- Mostrará aquellos atributos y operaciones relevantes
- Los nodos deberán estar conectados entre si reflejando la topología del sistema del mundo real
- Hay que mostrar los atributos u operaciones que sean necesarios para comprender el significado de cada nodo en el contexto dado

- Se pueden establecer dibujos o iconos apropiados dentro de este diagrama para proporcionar más señales visuales
- Hay que asignar a cada componente del sistema a un determinado nodo
- Hay que considerar que se pueden duplicar algunos componentes como ejecutables y bibliotecas, ya sea dejándola como parte especifica del modelo, conectando cada nodo con el componente que despliega

DIAGRAMA DE ACTIVIDADES

Los diagramas de actividad no surgieron del trabajo de "Los tres amigos", sino que son una combinación de varias técnicas, como el diagrama de eventos de Jim Odell, las técnicas de modelado de estados de SDL y las redes de Petri.

Son similares a los diagramas de flujo, pues describen el flujo de control de una secuencia de acciones, ya sea un algoritmo de la operación de una clase, la interacción de un grupo de objetos, la especificación de un caso de uso, las actividades de un procedimiento, etc. Además tiene algunos elementos adicionales para expresar conceptos como la concurrencia y la división del trabajo, se describen las actividades realizadas y su orden.
Es un caso especial del diagrama de estados en el cual todos o la mayoría de los estados, son estados de actividad (identifican qué acción se ejecuta al estar en él) y casi todas las transiciones son enviadas al terminar la acción ejecutada en el estado anterior, de hecho, sus símbolos son los mismos que los del diagrama de estado, sin embargo la semántica es diferente, ya que los diagramas de actividad están orientados a mostrar las acciones y pueden involucrar objetos de varias clases, por el contrario, los diagramas de estados están centrados en los estados y solo describen el comportamiento de objetos de una clase especifica.

Los diagramas de actividad pueden dar detalle a un caso de uso, un objeto o un mensaje en un objeto, generalmente modelan los pasos de un algoritmo.

Se debe comenzar por modelar el flujo principal y posteriormente las bifurcaciones, concurrencia y los flujos de objetos.

Utilidad del diagrama:

- Representan transiciones internas, sin hacer mucho énfasis en transiciones o eventos externos.
- Permiten especificar la división de trabajo o de responsabilidades entre objetos de un sistema o secciones de una organización.
- Modelan el flujo de trabajo o una operación.
- Ayuda en el modelado de los aspectos dinámicos de un sistema, estos aspectos pueden involucrar la actividad de cualquier tipo de abstracción en cualquier vista de la arquitectura de un sistema, incluyendo clases, interfaces, componentes y nodos.
- Se modela el flujo de un objeto conforme pasa de estado a otro en diferentes puntos del flujo de control.
- Ayudan a visualizar, especificar, construir y documentar la dinámica de una sociedad de objetos.
- Ayudan a modelar el flujo de control de una operación.
- Destacan el flujo de control entre actividades.
- Ayudan a construir sistemas ejecutables a través de ingeniería directa inversa.
- Capturan el camino crítico del flujo de trabajo.

- Se centran en las actividades que tienen lugar entre los objetos.
- Son similares a los diagramas Pert. Es un diagrama de flujo que destaca la actividad que tiene lugar a lo largo del tiempo.
- Muestra las operaciones que se pasan entre los objetos.
- Muestra el flujo de actividades.
- Pueden utilizarse para modelar aspectos dinámicos de un sistema global, un subsistema, una operación o una clase, también pueden asociarse a casos de uso (para modelar escenarios) y a las colaboraciones (para modelar los aspectos dinámicos de una sociedad de objetos).
- Modelan el comportamiento en paralelo, donde el orden de las acciones no importa, pueden realizarse en forma simultánea o intercalada. Esto es de gran utilidad para el modelado de flujos de trabajo y para la programación multihilos, sin embargo no dejan muy claros los vínculos entre acciones y objetos.
- Permite seleccionar el orden en que se llevarán a acabo las acciones o actividades, dicen las reglas esenciales de secuenciación a seguir, así como de los procesos paralelos. Permite evitar secuencias innecesarias y realizar procesos en paralelo, lo que mejora la eficiencia y capacidad de respuesta de los procesos de negocio.
- Son útiles para los diagramas concurrentes, ya que pueden plantear gráficamente cuáles son los hilos y cuándo necesitan sincronizarse.
- Son útiles para describir métodos complicados.
- Son útiles para el análisis de casos de uso, para comprender qué acciones deben ocurrir. Cuando los casos de uso interactúan entre ellos, los diagramas de actividad son de gran utilidad para representar y entender este comportamiento.
- Proporciona detalles de forma consistente con su nivel de abstracción, sólo se muestran los adornos que son esenciales para su comprensión.
- Ayudan a representar los procedimientos más complejos.
- Ayudan a representar las reglas de negocio o las decisiones lógicas.

Elementos del diagrama:
- Estados de acción
- Estados de actividad
- Estado inicial
- Estado final
- Transiciones
- Decisiones
- Condiciones
- Objetos
- Barra de sincronización

Estado de acción
Son estados del sistema que involucran la ejecución de una acción interna, tienen por lo menos una transición que identifica la culminación de la acción (por medio de un evento implícito). No deben tener transiciones internas ni transiciones basadas en eventos (esto se representa en un diagrama de estados). Algunos ejemplos de estados de acción podrían ser: evaluar una expresión, invocar una operación sobre un objeto, enviar una señal a un objeto, crear o destruir un objeto.

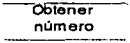
Estos estados no se pueden descomponer, son computaciones ejecutables atómicas, lo que significa que pueden ocurrir eventos, pero no se interrumpe la ejecución del estado de acción. Este tipo de estado conllevan un tiempo insignificante.

Estados de actividad:
Es un elemento compuesto, cuyo flujo de control se compone de otros estados de actividad y estados de acción. Los estados de actividad no son atómicos, ya que pueden descomponerse aún más y pueden ser interrumpidos, invierten más tiempo en completarse. Si se entra en los detalles de un estado de actividad puede encontrarse otro diagrama de actividad. Los estado de actividad pueden tener partes adicionales, como acciones relacionadas con la entrada y salida (entry/exit) del estado y especificaciones de submáquinas. Los estados de actividad importantes porque ayudan a dividir los cálculos complejos en partes.

Al entrar en un estado de acción o un estado de actividad, se ejecuta la acción o la actividad, al terminar, el control pasa a la siguiente acción o actividad.

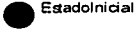
Actividad: es una ejecución no atómica de en curso dentro de una máquina de estados, producen alguna acción compuesta de computaciones atómicas ejecutables que producen un cambio estado del sistema o la devolución de un valor. Incluyen llamadas a otras operaciones, envio de señales, creación o destrucción de objetos o simples cálculos, como la evaluación de una expresión.

Notación:
Se representan por un rectángulo con bordes redondeados, dentro de la figura se puede escribir cualquier expresión.

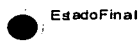


Estado inicial:

Notación:
Representado con un circulo relleno.



Estado final:
Notación:
Representado con un circulo lleno rodeado por otro circulo.



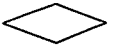
Transiciones
Es un flujo que muestra el camino de un estado (de actividad o de acción) al siguiente. Cada flujo representa transiciones con un evento implicito. Cuando se completa la acción o la actividad de un estado, el flujo de control pasa inmediatamente al siguiente estado (de acción o de actividad). Son disparadas como consecuencia de la finalización de la acción.

Pueden tener una condición en el caso de decisiones.
Estas transiciones se llaman transiciones sin disparadores o de terminación porque una vez finalizada la tarea del estado origen el control pasa inmediatamente al siguiente estado. Al completarse la acción o actividad de un estado origen, se ejecuta la acción de salida del estado e inmediatamente el control sigue por la transición y pasa al siguiente estado (de actividad o de acción). Este flujo de control continúa indefinidamente (en el caso de una actividad infinita) o hasta que encuentra un estado de parada.

Notación:
Se representa como una línea dirigida.

Decisiones
Son transiciones múltiples que sale de un estado, donde cada camino tiene destinos diferentes.

Notación:
Se representa mediante un diamante al cual llega la transición del estado inicial y del cual salen las dos o más transiciones de los estados finales etiquetadas con condiciones. Se considera como el caso OR, sucede uno u otro disparador.



Condiciones:
Se incluyen en las transiciones cuando es necesario. Ayudan a habilitar las transiciones entre una acción y otra.

Notación:
Se adjuntan a la flecha de transición como paréntesis cuadradas y dentro de ellos la expresión de la condición.

Carriles (swim lanes):
Son divisiones verticales que representan la división de trabajo o de responsabilidades entre objetos, cada división es etiquetada con el nombre del objeto, clase, departamento o sección responsable de realizar las acciones de dicha sección. Los carriles extienden la utilidad del diagrama de actividades pues ayudan a visualizar las responsabilidades representadas en el diagrama de interacción. Para diagramas complejos puede resultar complejo dibujar dichas líneas.
Se puede realizar el diagrama de actividades representando solo el comportamiento y posteriormente dibujar los carriles o bien ir asignando inmediatamente el comportamiento y responsabilidades a los objetos.

Notación:

Objetos:
Incluidos como insumos o resultados de una actividad o simplemente como afectados por ella, y las señales, que se envían y reciben en las transiciones.

Barra de sincronización:
Muestra la unificación de varias transiciones que se sincronizan para dar lugar a una nueva acción, sólo si ambas transiciones se ejecutan tiene lugar la ocurrencia de la acción. Son útiles cuando se tienen comportamientos en paralelo, los cuales necesitan sincronizarse. La barra de sincronización de salida indica que el disparo ocurre sólo cuando han sucedido todos los disparos de entrada, la barra puede considerarse como el caso AND, se realiza si suceden todos los disparadores. Las actividades que salen o entran de la barra de sincronización pueden realizarse en paralelo, en forma intercalada o simultánea.

Notación:
Representada con una línea gruesa, por lo general horizontal, dibujada en la transición de la condición. También puede dibujarse de forma vertical.

Modelado del diagrama:
El diagrama de actividades puede utilizarse de dos formas:

1. Para modelar un flujo de trabajo (Workflow):
Se modelan las actividades tal y como son vistas por los actores que colaboran con el sistema. En sistemas grandes existen flujos de trabajo que se utilizan para visualizar, especificar, construir y documentar los procesos de negocio del sistema a desarrollar. En esta forma es importante el modelado de los flujos de objetos.
Todo sistema interactúa con actores, sobre todo si son sistemas con gran cantidad de software, los cuales trabajan en el contexto de procesos de negocio que representan el flujo de trabajo y objetos a través del negocio. Los diagramas de actividad se emplean para modelar los procesos de negocio en los que colaboran sistemas automáticos y humanos.

Modelado:
Establecer un centro de interés del flujo de trabajo a modelar.

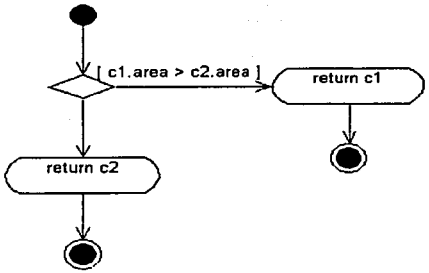
Seleccionar los objetos del negocio que tienen las responsabilidades de más alto nivel en cada parte del flujo de trabajo global.
Identificar las precondiciones del estado inicial del flujo de trabajo y las poscondiciones del estado final. Para modelar los límite del flujo de trabajo.
Comenzar con el estado inicial del flujo de trabajo, se especifican las actividades y acciones que tienen a desarrollar, se representan como estados (de acción o de actividad).
Representar las transiciones que conectan los estados, comenzando con los flujos secuenciales, después las bifurcaciones y por último las divisiones y uniones, estos sólo aparecen en sistemas más complejos.
Representar objetos importantes, en caso que los haya, sus valores y estados.

Ejemplo:

2. Para modelar una operación:
Las operaciones son los elementos a los que más se asocia un diagrama de actividades, en este caso se usan como diagramas de flujo que describe las operaciones de la acción. En esta forma se da importancia a la bifurcación, la división y la unión. Se incluyen los parámetros de la operación y objetos locales.

Modelado:
Reunir las abstracciones implicadas en la operación (parámetros de la operación, atributos de la clase a la que pertenece y ciertas clases vecinas).
Identificar las precondiciones (estado inicial de la operación) y las poscondiciones (estado final). Identificar cualquier invariante de la clase a la que pertenece y que deba mantenerse durante la ejecución de la operación.
Especificar las actividades y acciones que tienen lugar a lo largo de la ejecución, comenzando por el estado inicial de la operación y representarlos como estados de actividad o de acción.
Usar bifurcaciones cuando sea necesario especificar caminos alternativos e iteraciones.
Usar divisiones y uniones cuando sea necesario especificar flujos paralelos de control, sólo si la operación se encuentra en una clase activa.

Ejemplo: El siguiente ejemplo muestra una operación para comparar dos círculos y obtener el mayor de ellos. Primero se tiene un guarda que comprueba cuál de los círculos tiene el área mayor, si c1 o c2, si el área de c1 es mayor, return devuelve la referencia al objeto c1, que es el círculo mayor, de lo contrario regresa la referencia al objeto c2, porque será el círculo mayor.



Un solo diagrama de actividades no puede capturar toda la dinámica de un sistema, se utilizan varios diagramas de actividades para modelar la dinámica de un flujo de trabajo o una operación.

Ejemplos:

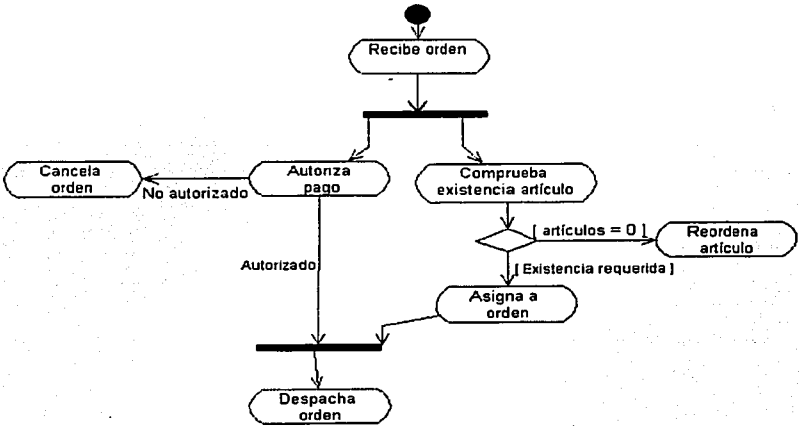


DIAGRAMA DE INTERACCIÓN SECUENCIA Y DIAGRAMA DE ITERACIÓN COLABORACIÓN

Un diagrama de interacción es una representación gráfica de interacciones entre objetos. Hay dos tipos de diagramas de interacción:

•Diagramas de secuencia
•Diagramas de colaboración

Los diagramas de secuencia y los diagramas de colaboración ambos llamados diagramas de interacción se utilizan para modelar los aspectos dinámicos del sistema. Un *diagrama de interacción colaboración* muestra cómo interactúan un conjunto de objetos en el sistema y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellos. Un *diagrama de interacción secuencia* es como un diagrama de interacción en donde se destaca la ordenación de los mensajes.

Los diagramas de interacción implican el modelado de clases, interfaces, componentes y nodos, junto con los mensajes enviados entre ellos proporcionando un contexto o escenario que ilustra un comportamiento que puede existir en un caso de uso especifico.

A través de los diagramas de interacción podemos visualizar, construir, especificar y documentar para modelar y construir sistemas ejecutables por medio de ingenieria directa e inversa al imaginarnos por un momento cómo podria visualizarse el sistema en ejecución.

Utilidad

- Los diagramas de interacción permiten modelar la parte dinámica del sistema
- Muestra la interacción conformada por el conjunto de objetos y sus relaciones
- Muestra los mensajes que se envian y reciben entre si los objetos
- Se utilizan para visualizar el contexto de comportamiento que tendrán los objetos
- Permiten documentar, construir y visualizar las interfaces del sistema

Existen dos representaciones de los diagramas de interacción, ya sea destacando la ordenación temporal de los mensajes (D. Secuencia) o destacando la relación estructural entre los objetos que interactúan (D. Colaboración), estos diagramas son prácticamente equivalentes y basta con desarrollar uno de los dos para comprender la dinámica del sistema.

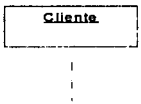
Diagrama de Secuencia

Un diagrama de secuencia muestra interacciones de objetos ordenados en secuencia de tiempo.

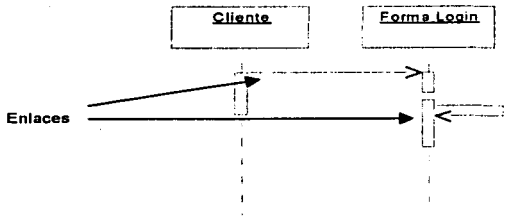
Elementos

- Objetos
- Enlaces
- Mensajes
- Notas o restricciones

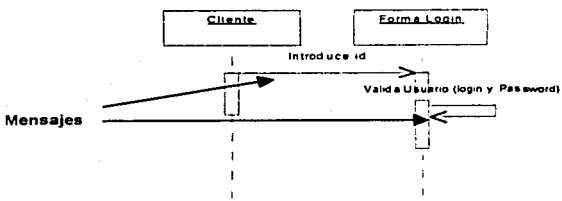
Objeto: es un concepto, abstracción o cosa con limites bien definidos y significado para una aplicación. Un objeto es algo que tiene estado, comportamiento e identidad.



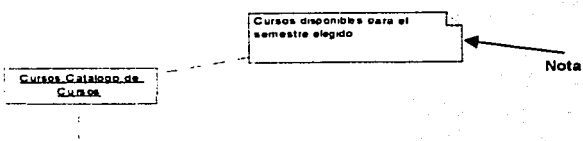
Enlaces: son lineas horizontales que representan las relaciones que existen entre los diferentes objetos, para mostrar la forma en que interactúan.



Mensajes: El mensaje enviado entre objetos es etiquetado con una descripción y puede incluir argumentos e información de control, los mensajes también pueden mostrar autodelegación que es el mensaje que un objeto se envía así mismo.



Notas: Las notas pueden usarse para agregar más información al diagrama



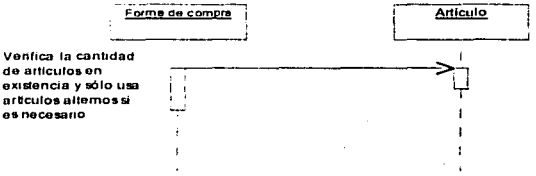
El diagrama muestra:
- Los objetos que participan en la interacción
- La secuencia de mensajes intercambiados

Un diagrama de secuencia contiene:
- Objetos con sus "líneas de vida"
- Mensajes intercambiados entre objetos en orden secuencial
- Enfoque de control (opcional)

Los objetos se dibujan como rectángulos con nombres subrayados y las "líneas de vida" se representan con líneas de guiones descendentes.

Características
- La interacción entre objetos se indica con flechas horizontales que se dirigen desde la línea vertical que representa al objeto cliente hasta la línea que representa al objeto proveedor
- Las flechas horizontales se etiquetan con mensajes
- El tiempo de orden de mensajes se indica por posición vertical, con el más cercano en la parte superior
- La numeración es opcional, ya que la orden se basa en posición vertical
- Para escenarios complejos, los diagramas de secuencia pueden mejorar mediante el uso de scripts
- Un script se escribe a la izquierda de un diagrama de secuencia con los pasos del script alineados con las interacciones del objeto
- Los scripts se pueden escribir en lenguaje natural o en pseudo-código

Ejemplo de un script



Ejemplo de un Diagrama de Secuencia

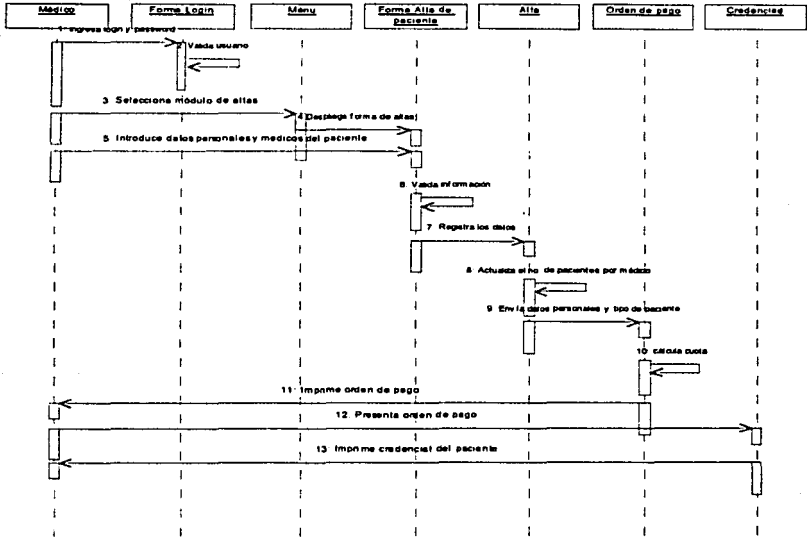


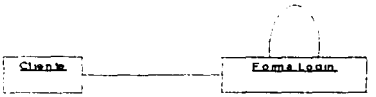
Diagrama de Colaboración

Un diagrama de colaboración es una forma alternativa de representar los mensajes intercambiados por un conjunto de objetos. El diagrama muestra interacciones de objeto organizadas alrededor de los objetos y sus ligas a cada uno.

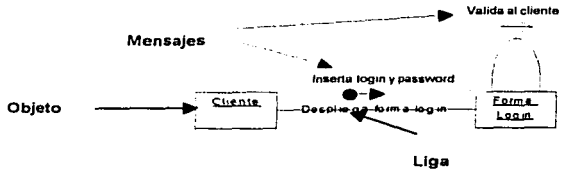
Un diagrama de colaboración contiene:
•Objetos
•Ligas entre objetos
•Mensajes intercambiados entre objetos
•Flujo de datos entre objetos, si hay alguno

Los objetos se dibujan como rectángulos con nombres subrayados

Una liga de interacción en un diagrama de colaboración se representa como una línea que conecta iconos de objetos. Una liga indica que hay una ruta para comunicación entre objetos conectados.

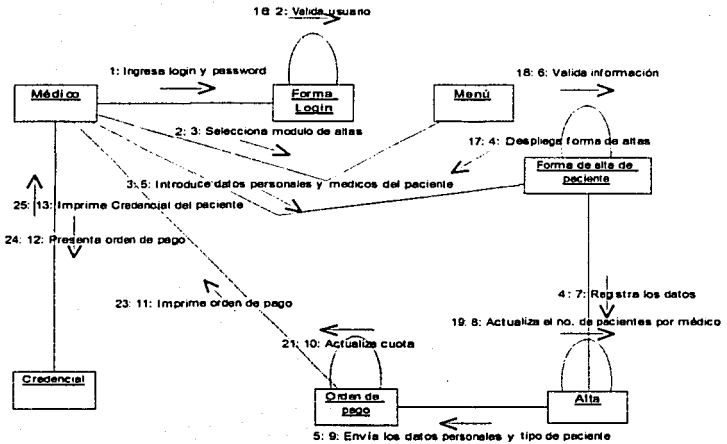


Una liga de interacción en un diagrama de colaboración se puede anotar con:
- Una flecha apuntando del objeto cliente al objeto proveedor
- El nombre del mensaje con una lista opcional de parámetros y/o un valor de dato de regreso
- Un número de secuencia opcional que muestre el orden relativo en el cual se envían los mensajes



NOTA: Se recomienda utilizar los diagramas de interacción cuando se desea ver el comportamiento de varios objetos en un caso de uso. Son buenos para mostrar la colaboración entre objetos; sin embargo, no sirven para la definición precisa del comportamiento.

Ejemplo de un Diagrama de Colaboración



RUP Proceso Unificado Racional

Proceso de ingeniería de software, ya que provee de una disciplina asignando tareas y responsabilidades a los que participan en el desarrollo de un sistema orientado a objetos, el principal objetivo de RUP es asegurar la producción de software de alta calidad que satisfaga las necesidades de los usuarios.

RUP es un proceso que está en continua actualización y mejoramiento, basado en experiencias ya que provee las mejores prácticas, es una guía que muestra cómo utilizar eficientemente el Lenguaje Unificado de Modelado (UML). RUP es soportado por diversas herramientas denominadas artefactos a través de las cuales se logra automatizar las etapas de desarrollo de software abarcando el modelado visual, la programación y las pruebas.

RUP no es rígido, es flexible, ya que es un proceso configurable y adaptable a cualquier desarrollo de software.

A través de este proceso se provee a cada miembro del equipo de guías, plantillas y herramientas necesarias para cubrir cada rol de los integrantes del equipo de desarrollo proporcionando numerosas ventajas clave a comparación de otros procesos de desarrollo de software orientado a objetos.

RUP posee tres características clave que lo distinguen de todos los demás procesos de desarrollo de software:

1. Dirigido Por Casos De Uso: La interacción de alguien o algo, ya sean humanos u otros sistemas, con el sistema que se desarrollará es representada representa un casos de uso. Un caso de uso es un requisito funcional y todos los casos de uso en conjunto construyen el modelo de casos de uso, el cual describe la funcionalidad total del sistema. Esta característica es muy importante, ya que los casos de uso, notación de UML, permiten la especificación de los requisitos funcionales del sistema a desarrollar, además son una guía para el diseño, desarrollo, implementación y pruebas, es decir, durante todo el proceso de desarrollo. Los casos de uso no sólo inician el proceso de desarrollo, sino que además proporcionan una directriz a seguir a través de una serie de flujos de trabajo

2. Centrado en la Arquitectura: Una de las características de mayor utilidad es que a través de este proceso la arquitectura del sistema a desarrollar se describe mediante diferentes vistas antes de construirlo, esta arquitectura del sistema permite percibir si es como la desea el usuario o cliente, ya que incluye la plataforma (software, hardware, sistema operativo, bases de datos y protocolos de comunicación de red) lo que proporciona un marco de trabajo para el diseño completo del sistema.

La arquitectura de software permite tomar decisiones importantes sobre:

- La organización del desarrollo del sistema
- Los elementos que compondrán al sistema y sus interfaces junto con su comportamiento
- El estilo de la arquitectura que guía la organización del sistema
- Reutilización de módulos
- Mejora la comprensión del sistema
- Permite visualizar la evolución del sistema a desarrollar

Esto permitirá observar cómo puede ser afectada la estructura del software en el comportamiento, funcionalidad, rendimiento, flexibilidad, la reutilización , facilidad de comprensión, los costos, la tecnología y la estética del mismo.

La arquitectura se desarrolla mediante iteraciones, principalmente durante la fase de elaboración, ya que en cada iteración se desarrolla un esbozo comenzando con los requisitos y siguiendo con el análisis, diseño, implantación y pruebas, pero siempre centrándose en los casos de uso desde el punto de vista de la arquitectura.

3. Iterativo e incremental: A través de esta característica, RUP divide de manera práctica el desarrollo de un sistema en partes más pequeñas o miniproyectos, cada uno de ellos es una iteración que resulta en un incremento, es decir, las iteraciones hacen referencia a los flujos de trabajo y los incrementos al crecimiento del producto. Es importante que las iteraciones sean controladas, ya que deben de ejecutarse y seleccionarse en forma planificada; si una iteración cumple con sus objetivos el desarrollo continua con la siguiente iteración y así sucesivamente hasta que todos lo miniproyectos constituyen el proyecto final.

El resultado de una iteración es un incremento ¿Pero, qué es un incremento? Es la diferencia que existe entre la versión resultante de una iteración y la versión resultante de la siguiente iteración.

Dentro de cada fase, el proceso pasa por una serie de iteraciones e incrementos que nos conducen a los siguientes criterios:

Fase de Inicio: el criterio esencial es la viabilidad del proyecto que llevamos a cabo, para lo cual se deben llevar a cabo actividades como:

- Identificación y la reducción de riesgos para la viabilidad del desarrollo de un sistema
- Creación de una arquitectura candidata, a partir de la identificación de ciertos requisitos
- Estimación de coste, esfuerzo, calendario y calidad del producto

Fase de elaboración: el criterio esencial es la capacidad de construir el sistema dentro de un marco de trabajo, para lograr esto se debe:

- Identificar y reducir los riesgos que afectan de manera significativa la construcción del sistema
- Especificación la mayoría de los casos de uso que representan la funcionalidad que ha de desarrollarse
- Preparar el plan de proyecto
- Realizar una estimación para justificar la inversión

Fase de construcción: el criterio esencial es un sistema con operatividad inicial en el entorno del usuario, para ello se debe realizar:

- Una serie de iteraciones con incrementos y entregas periódicas.

Fase de transición, el criterio esencial es un sistema que alcanza una operatividad final, que se logra a través de:

- La modificación del producto para subsanar problemas que no se identificaron en fases anteriores
- La corrección de defectos

Por qué es importante un desarrollo iterativo incremental:

- Para atenuar los riesgos criticos desde el principio
- Poner en marcha una arquitectura que guíe el desarrollo de software
- Proporcionar un marco de trabajo que permita la gestión del desarrollo de software
- Proporcionar un proceso de desarrollo a través del cual el personal pueda trabajar de manera más eficaz
- Permite gestionar mejor aquellos requisitos cambiantes
- Se logra una integración continua y funcional del sistema
- Los modelos, que en conjunto conforman el sistema, van evolucionando con las iteraciones, puesto que en cada iteración se añade algo más al modelo.

Las seis mejores prácticas sobre las que se basa RUP con:

1. Desarrollo iterativo de software: El desarrollo de software sofisticado no se desarrolla de manera secuencial, primero definiendo el problema, diseñando la solución, construyendo el software, haciendo pruebas y entregando el producto final. El objetivo de RUP es realizar un producto de software de calidad y de acuerdo a las necesidades del usuario, para lo cual propone un esquema iterativo de desarrollo que permita reducir los riesgos en cada plataforma del ciclo de vida. Este esquema iterativo ayuda a reducir los riesgos, demostrando progresos en los productos ejecutables que se generan durante todas las fases de desarrollo.

2. Administración de requerimientos: RUP describe cómo obtener, organizar y documentar los requerimientos funcionales, así como su fácil captura y comunicación en casos de uso, los cuales, junto con los escenarios son el camino para conocer los requerimientos funcionales para asegurar que esto dirija el proceso de diseño, la implementación y las pruebas de software.

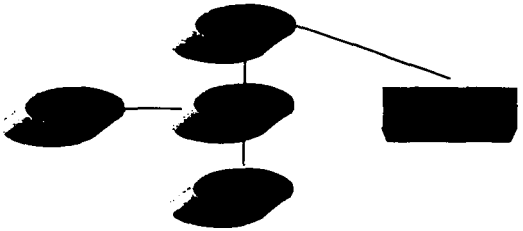
3. Arquitectura basada en componentes: El proceso se enfoca a un fácil desarrollo y una arquitectura robusta que permitirá un desarrollo escalable y flexible, ya que es adaptable a cualquier cambio y provee con eficiencia la reutilización de software. RUP provee un esquema para definir la arquitectura usando los componentes existentes.

4. Modelado de software visual: El modelo es mostrado visualmente a través de simbología unificada, la cual captura la estructura y comportamiento de la arquitectura y de todos los componentes de software, esto permite esconder los detalles y escribir el código usando bloques gráficos construidos. Las abstracciones visuales ayudan a comunicar los diferentes aspectos del sistema y cómo los diferentes elementos que lo conforman se relacionan entre sí, asegurando la consistencia de los bloques visuales construidos con el código, con lo cual se logra mantener la consistencia entre el diseño y la implementación, lo que elimina las posibles ambigüedades de la comunicación.

5. Verificación de la calidad de software: RUP verifica la calidad con respecto a los requerimientos de los usuarios basándose en la confiabilidad, funcionalidad y el desempeño del sistema. Es un proceso integral, ya que abarca y da soporte en la planeación, diseño, implementación, ejecución y evaluación del producto de software a desarrollar. La calidad es implementada durante todas las fases de construcción en todas las actividades involucrando a todos los participantes.

6. Control de cambio en el desarrollo de software: La habilidad para la administración de cambios durante el desarrollo de software es importante para verificar que todos los cambios sean aceptables, sin descuidar la calidad del producto de software. El proceso describe cómo el control y monitoreo de los cambios permiten un desarrollo iterativo exitoso.

Las cuatro "P", puntos clave en el desarrollo de un sistema



1. Personas:

Los principales autores de un proyecto de software son los desarrolladores, ingenieros de pruebas y el personal de gestión que les da soporte, además de los usuarios y clientes. El desarrollo de un sistema involucra y afecta a las personas que participan en aspectos como:

Viabilidad del proyecto: Si los proyectos no son viables, el proceso juzga que pueden detenerse tempranamente.

Gestión del proyecto: Permite la exploración de los riesgos y su minimización en las primeras fases, atenúa problemas.

Estructurar el proyecto: Las personas trabajan de manera más eficaz en grupos pequeños de seis a ocho miembros, esto motiva más a las personas y facilita la organización y exploración de las ideas de cada integrante del grupo.

Planificación del proyecto realista: Motiva mucho a las personas para trabajar, ya que cada integrante está consiente de que las metas son alcanzables. Las técnicas de las fases de inicio y elaboración del RUP permiten a los desarrolladores tener una adecuada notación para atenuar aún más el problema de "nunca acabaremos".

Comprensión del proyecto: Si se conoce bien el proyecto, la gente sabe lo que tiene que hacer y posee una comprensión global del proyecto.

Sensación de cumplimiento: Se percibe por los miembros del equipo con la retroalimentación frecuente ya que esto acelera el ritmo del trabajo.

Las personas son las que ejecutan las actividades clave en el desarrollo de software por lo cual es necesario que el proceso de desarrollo sea uniforme y que se apoye en herramientas como el UML para que el desarrollo del mismo sea más eficaz, es decir, crear un sistema en el tiempo estimado, con calidad y dentro de los costos.

Al desarrollar software complejo, se deberá contemplar un equipo de desarrollo más grande, por lo cual se requiere de un proceso que sirva como guía lo que ocasionará que los desarrolladores trabajen de manera más inteligente dirigiendo sus esfuerzos individuales a aquello que proporcione un valor al cliente:

Centrando los esfuerzos en lo que el usuario necesita a través de los casos de uso, los cuales reflejan los requerimientos reales de los usuarios, desarrollando una arquitectura que permita a los sistemas su funcionamiento a futuros cambios en el medio en que interactúan. Compra y reutilización de software cuando sea posible.

Se debe motivar a los integrantes del equipo de desarrollo a ser creativos, a encontrar nuevas oportunidades utilizando la razón y a comunicarse con los clientes y usuarios.

Cada integrante del equipo de desarrollo es responsable de un conjunto de actividades y para trabajar eficazmente requieren de información para llevar a cabo sus actividades, ya que necesitan comprender cuáles son sus roles en relación con los otros integrantes del equipo de desarrollo, RUP ayuda a lograr esto, ya que describe formalmente los puestos, cada trabajador tiene un conjunto de responsabilidades y lleva a cabo un conjunto de actividades en el desarrollo de software.

El jefe de proyecto debe identificar las aptitudes de los miembros del equipo de desarrollo y casarlas con las actitudes requeridas, es importante resaltar que un integrante del equipo puede tener muchos roles durante el desarrollo de un sistema.

2. Proyecto

El proyecto es el elemento organizativo a través del cual se gestiona el desarrollo de software. El resultado de un proyecto es una versión de un producto. El primer proyecto dentro del ciclo de vida se denomina proyecto inicial e innovador para la gestión del proyecto.

Es importante que el equipo de desarrollo esté alerta a los aspectos clave que puedan alterar el desarrollo del proyecto como:

La secuencia de cambio: En el desarrollo de una sistema se obtienen varios productos, pero el camino para obtenerlos es una serie de cambios, ya que cada fase, cada ciclo y cada iteración modifican al sistema continuamente .

Una serie de iteraciones: Dentro de cada fase de un ciclo, los desarrolladores llevan acabo las actividades de la fase a través de una serie de iteraciones, cada iteración implementa un conjunto de casos de uso relacionados. En una iteración los desarrolladores progresan a través de los flujos de trabajo (requerimientos, análisis, diseño, pruebas e implementación), y debido a que cada iteración pasa por cada uno de esos flujos de trabajo se puede pensar que cada iteración es como un miniproyecto.

3. Producto

Bajo el contexto de RUP, el producto que se obtiene es un sistema de software, es decir, al sistema entero (modelos, código fuente, ejecutables y documentación) que se entrega, y no solo el código.

¿Pero, qué es el código fuente? Es una serie de instrucciones escritas bajo ciertas reglas, las cuales pueden ser leídas e interpretadas por un compilador, sin embargo un sistema de software está compuesto de todos los artefactos que se necesitan para representarlo en forma comprensible para los hombres, las màquinas (herramientas), los desarrolladores y usuarios, no importando si poseen una formación en el desarrollo de sistemas.

¿Qué es un artefacto? Cualquier tipo de información creada cambiada o utilizada por los desarrolladores del sistema, como por ejemplo: los diagramas de UML, su documentación asociada, los bocetos de interfaces, los componentes, los planes de prueba. Existen dos tipos de artefactos de ingeniería (son aquellas descripciones y diagramas que se utilizan a lo largo de las diversas fases y flujos de trabajo durante el desarrollo del sistema) y artefactos de gestión (análisis de negocio, plan del proyecto, plan de versiones e iteraciones, plan para la asignación del equipo de desarrollo, diseño de las actividades de cada integrante del equipo).

¿Qué es un sistema? Un sistema es una colección de modelos y el Proceso de Desarrollo Unificado utiliza modelos para cada integrante del equipo de desarrollo, ya que cada uno requiere de una perspectiva diferente del sistema, por lo que la construcción del sistema es un proceso de construcción de modelos para describir así todas las perspectivas del sistema.

¿Qué es un modelo? Es una abstracción o simulación del sistema real desde un cierto punto de vista. Un sistema no es sólo una colección de modelos sino que además contiene las relaciones entre ellos.

4. Proceso

Un proceso es la definición del conjunto completo de actividades necesarias para transformar los requisitos del usuario en un producto. Un proceso de desarrollo de software es un conjunto de actividades necesarias para construir artefactos y conformar un producto de software que cumpla con los requisitos de los usuarios. ¿Qué son los requisitos? Son las necesidades y requerimientos de los usuarios del sistema a desarrollar.

Un proceso es la definición de un conjunto de actividades a realizar las cuales, relacionadas en conjunto, conforman los flujos de trabajo que se ejecutarán durante todo el desarrollo del sistema.

Un proceso común a lo largo del desarrollo de un sistema proporciona varios beneficios:

- Cada elemento del equipo de desarrollo comprende lo que tiene que hacer para desarrollar el sistema
- Los desarrolladores pueden comprender mejor lo que otros están haciendo en cualquier etapa del proyecto, y en proyectos similares.
- Los supervisores y directores, personas ajenas al desarrollo de sistemas, pueden comprender lo que los desarrolladores están haciendo a través de los esquemas.

- El desarrollo de software es repetible , puede planificarse y estimarse en coste con suficiente exactitud
- Los desarrolladores pueden cambiar de proyecto sin tener que aprender un nuevo proceso.

Herramientas de RUP

Las herramientas son esenciales en el proceso de desarrollo de software, ya que ayudan a automatizar los procesos repetitivos, a mantener las cosas estructuradas y a gestionar grandes cantidades de información, por ello es importante soportar con herramientas todos los aspectos del ciclo de vida de los sistemas con diversas herramientas:

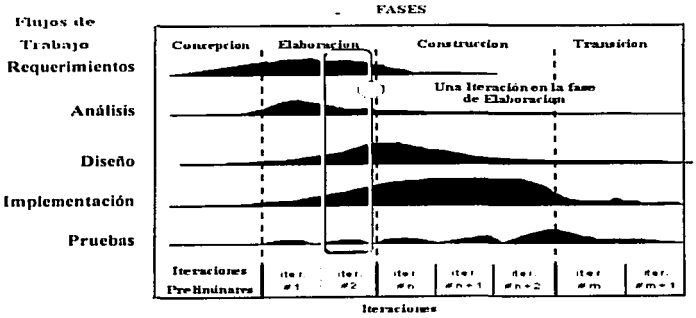
Para la gestión de requisitos: se utilizan para almacenar, examinar, revisar, hacer el seguimiento y navegar por los diferentes requisitos del proyecto de software

Para el modelado visual: Se utiliza para automatizar el uso de UML, es decir, para ensamblar y construir una aplicación visualmente, logrando que el modelo, la programación y la implementación sean consistentes.

De programación: Proporciona una gama de herramientas como editores, compiladores, depuradores, detectores de errores y analizadores de rendimiento.

Para el aseguramiento de la calidad: Se utiliza para probar aplicaciones y componentes, para registrar y ejecutar casos de prueba de las interfaces de un componente.

RUP puede ser descrito en dos dimensiones o dos ejes:



El eje horizontal representa el tiempo y muestra el aspecto *dinámico del proceso*, expresado en términos de ciclos, fases, iteraciones y milestones.

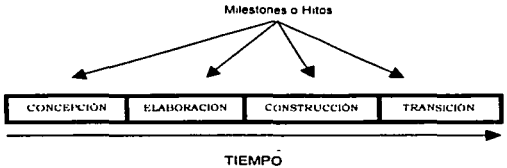
El eje vertical representa el aspecto *estático del proceso*, descrito en términos de actividades, artefactos y flujos de trabajo.

Fases e Iteraciones, Proceso Dinámico (dimensión del tiempo)

Es la organización dinámica del proceso a lo largo del tiempo. El ciclo de vida de todo el software es dividido en subciclos o iteraciones y cada ciclo trabaja en la nueva generación de un producto ejecutable que es mejorado a través de los diversos ciclos.
RUP está dividido en cuatro principales ciclos de desarrollo en cada una de las cuatro faases consecutivas, éstos a su vez poseen n cantidad de subciclos o iteraciones, esto puede ser adaptado a cada producto de software que se desee. Las fases que componen al RUP son cuatro:

- *Concepción:* objetivos del ciclo de vida y determinación de requerimientos
- *Elaboración:* arquitectura del ciclo devida
- *Construcción:* desarrollo de la funcionalidad operativa
- *Transición:* versión del producto semifinal y final

Cada fase es concluida con un milestone (hito) o punto en el tiempo, en el cual se determinan decisiones críticas que se deben realizar para construir software de calidad y de acuerdo a las necesidades de los usuarios. Cada fase tiene un propósito específico.



Los hitos tienen varios objetivos:

- Permiten que el líder y los desarrolladores tomen decisiones importantes para poder continuar con las demás fases.
- Permiten a los desarrolladores controlar el progreso de trabajo.
- Permiten obtener datos del esfuerzo y tiempo consumido en cada fase por lo que esos datos serán útiles en la estimación de tiempo y recursos humanos para otros proyectos.

RUP está basado en componentes, utilizando un nuevo estándar de modelado visual UML (Lenguaje Unificado de Modelado) para hacer que esto funcione se necesita un proceso que tenga en cuenta ciclos, fases, flujos de trabajo, gestión de riesgo, control de calidad, gestión de proyecto y control de la configuración.

Las actividades relacionadas conforman los flujos de trabajo, identificando los tipos de trabajadores que participarán en el proceso y los artefactos que se requieren crear durante todo el proceso. Por lo tanto, un proceso de desarrollo de software es una definición del conjunto completo de actividades necesarias para convertir los requisitos de usuario en un conjunto consistente de artefactos que conforman un producto de software.

El objetivo de cada hito principal es garantizar que los diferentes modelos de flujos de trabajo, evolucionen de manera equilibrada durante las fases del ciclo de vida del sistema, es decir, se deben tomar las decisiones oportunamente para influir eficientemente en costos, calidad y funcionalidad de los modelos que conformarán al sistema.

Antes de comenzar a describir las fases de RUP, es importante señalar que éstas representan un camino flexible a seguir durante el proceso de desarrollo de un sistema, no obstante no se presenta el "Cómo" sino el "Qué".

FASE DE CONCEPCIÓN:

Objetivo
Establecer el ámbito de lo que deberia hacer el sistema , la reducción de riesgos y la preparación del negocio inicial, que indiquen que resulta factible realizar el proyecto desde la perspectiva de negocio. En esta fase se tienen que identificar y mitigar los riesgos criticos ya que esto provocaria un proyecto inviable.

Antes de iniciar esta etapa se puede comenzar reuniendo la información y organizàndola para crear un plan provisional que clarifiquen los requisitos para detallarlos en los casos de uso. El plan inicial es provisional ya que conforme se recopile más información se modificará el mismo.

Los participantes en esta primera fase pueden ser el líder de proyecto, el diseñador, uno o dos desarrolladores con experiencia, una persona para pruebas y representantes del cliente o los usuarios.

Para comenzar esta fase se deberá planificar la descripción del sistema y reunir los criterios de evaluación, ya que entre un sistema y otro pueden variar, el plan detallado de la primera iteración presentará los criterios de evaluación los cuales permitirán indicar cuando la iteración ha alcanzado sus objetivos, algunos criterios son:

Decidir el ámbito del sistema: es la descripción inicial, su propósito es delimitar qué es lo que se encontrará dentro del sistema propuesto de lo que está afuera, para lo cual se identifican los actores externos con los que interactuará el sistema .
Resolver ambigüedades de los requisitos: los requisitos iniciales pueden variar por lo cual se deberá verificar que ya se han identificado y detallado con exactitud los requisitos y limitado el número de casos de uso.

Determinar una arquitectura candidata: se deberá desarrollar una arquitectura factible que satisfaga las necesidades de los usuarios y que sea funcional.

Mitigar los riesgos criticos: son aquellos riesgos que si no son mitigados pueden poner en riego el proyecto.

Juzgando el valor del caso de negocio inicial: se debe evaluar el caso de uso inicial para determinar si es lo suficientemente factible para que el proyecto siga adelante.

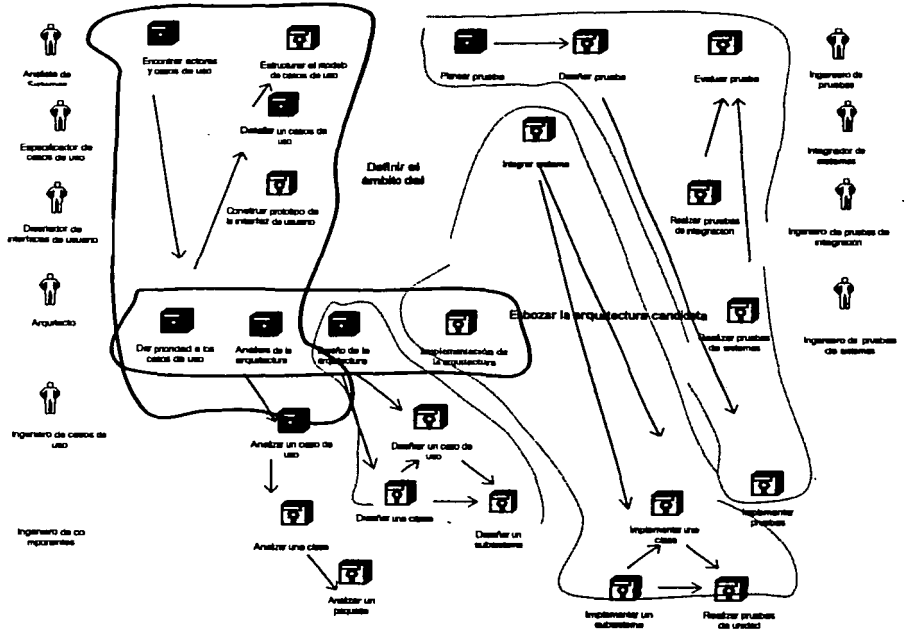
Actividades principales que se desarrollan

- Planificación de las iteraciones
- Determinación de los flujos de trabajo
- Seleccionar el entorno de trabajo adecuado para el proyecto

Los casos de uso que se describen detalladamente son aquellos que son relevantes para comprender el ámbito y la delimitación del sistema, para su arquitectura candidata y para entender los riesgos criticos que serán de utilidad para el análisis inicial de negocio, el resultado de esto es el modelo de caso de uso.

Flujos de Trabajo

Una iteración es como un pequeño ciclo en cascada, pasando a través de cinco flujos de trabajo o bien los que se determinen desde los requerimientos hasta las pruebas.



Flujo de determinación de los requerimientos: se trata de identificar y detallar los casos de uso pertenecientes a esta fase incluyendo los requisitos posibles que proporcionarán las características del sistema, comprender el contexto del sistema, descubrir los requisitos funcionales basándonos en los casos de uso y de la misma manera los requisitos no funcionales.

Flujo de trabajo de análisis: el objetivo de este flujo es enumerar los requisitos, refinarlos y plasmarlos en un modelo de objetos que sirva como base para el modelo de diseño. Se analizan los casos de usos creando un modelo inicial para la mayoría de los casos de uso y los escenarios para comprender claramente los requerimientos y las características que tendrá el sistema, el análisis de este modelo permite descubrir si hay recursos compartidos como bases de datos o hardware. Además se realiza un análisis de la arquitectura llevando a cabo la descripción de la arquitectura candidata, incluyendo bocetos de las vistas de los modelos, se necesita además examinar y clasificar todos los casos de uso y escenarios posibles para que el riesgo sea menos crítico o mitigable. El entorno de desarrollo del proyecto debe ser adecuado y planeado ya que consta de herramientas, asesoría, servicios, procesos y configuración.

Flujo de diseño: se esboza un modelo de arquitectura con el objetivo de incluirlo como descripción de arquitectura preliminar, si se requiere desarrollar un prototipo de demostración que incluya interfaces y algoritmos de usuario debe de ser un desarrollo rápido que simplemente muestre la idea del futuro sistema.

Flujo de Implementación: en este flujo se deberá decidir si debe finalizar la fase de inicio y tener así algo de certeza de que la arquitectura preliminar que se ha propuesto es la adecuada, no obstante a esta altura no se puede estar seguro de que así sea, al menos al cien por ciento, por lo tanto se debe finalizar este flujo con la descripción de la arquitectura candidata.

Flujo de pruebas: se realiza paralelamente a los flujos anteriormente citados ya que se va determinado qué pruebas se necesitarán y se desarrollan planes de pruebas iniciales.

Elaboración del Análisis inicial de Negocio
El análisis de negocio posee dos partes fundamentales una propuesta económica y la recuperación de la inversión además de las ganancias que se obtenga el cliente por su uso.

Propuesta Económica
Debido a la corta visión que se tiene del producto final de software se puede tener una visión corta y errónea de los costes del producto, hay que tener en cuenta varios factores futuros:

- Complejidad del producto, ya que entre más complejo más costoso será
- Tamaño del producto, ya que este puede repercutir en las horas –hombre
- Colaboración por parte de los clientes y usuarios
- Tipo de aplicación, no es lo mismo un sistema monousuario a un sistema multiusuario
- Una adecuada planeación del proyecto

Recuperación de la Inversión

Se debe calcular las ganancias que proporcionará el software basándose en proyectos anteriormente desarrollados si se carece de experiencia.
Si se trata de un software comercial se deben de considerar aspectos como:
El número de unidades vendidas
El precio del producto
El tiempo en el que se tendrá a la venta

Si se trata de software para uso interno se deben considerar aspectos como:
Ahorro de los departamentos por el uso de este software

Para asegurarse de que estas estimaciones sean correctas se debe de manejar un margen de error para alcanzar el margen de recuperación esperado, asi se llega en esta fase a un análisis de negocio general para determinar si es rentable el proyecto o no, hasta esta parte del proceso de desarrollo no se tendrán aún cifras exactas, sino solamente un conocimiento de que el proyecto está económicamente al alcance de la organización de desarrollo y de los clientes.

Evaluación de la Fase de Concepción
El lider de proyecto debe asignar a un grupo de personas, incluyendo a los representantes de cada una de las áreas involucradas que realicen la evaluación de esta fase, trasladando a futuras iteraciones los criterios no alcanzados por lo que se modificarán los planes y agendas. Un resultado importante de esta evaluación es la determinación de continuar o no con el proyecto.

Al final de esta fase se comenzará a planear la siguiente fase, denominada fase de Construcción, ya que cada vez se va teniendo una mayor conciencia de los costes y del tiempo estimado.

Productos
- Caracteristicas del futuro sistema
- Primera versión del modelo de negocio
- Esbozo de los modelos que representan una primera versión de los modelos de casos de uso, el modelo de análisis y el modelo de diseño
- Modelo de implementación y pruebas a nivel general
- Primera versión de los requisitos adicionales

FASE DE ELABORACIÓN

Objetivo
Obtener la base de la arquitectura sólida, capturar la mayoria de los requisitos pendientes, formulando los requisitos funcionales, reducir los siguientes peores riesgos, estableciendo la arquitectura del ciclo de vida y completar los detalles del plan del proyecto. Al final de esta fase se obtendrán los costes, fechas y la planificación de la fase de construcción con detalle.

Al comienzo de esta fase se cuenta con un plan inicial para la fase de elaboración, un modelo de casos de uso parcialmente y un modelo de la arquitectura y modelos generales de análisis y diseño, así como un prototipo que muestre el funcionamiento del sistema. En esta fase líder de proyecto debe seleccionar algunas personas para mantenerlas en la fase de construcción estableciendo quiénes dirigirán lo equipos de diseño, se continua haciendo cambios en el entorno de desarrollo, se continúa verificando los criterios de evaluación verificando los casos de uso, los riesgos, la arquitectura, las necesidades de los usuarios, así como determinar si el plan del proyecto está bien definido para definir un precio.
En esta fase los diseñadores analizan las clases y paquetes, los encargados de pruebas se centran en construir el entorno de pruebas. Mientras el equipo de desarrollo sea pequeño se tiene la oportunidad de iterar y ensayar diferentes soluciones hasta conseguir una arquitectura estable. Una iteración puede ser suficiente si el sistema es pequeño, pero puede ser la primera iteración si el sistema es grande y complicado, el número de iteraciones depende de diversos factores como riesgos, complejidad y arquitectura.
Es importante cubrir el 80% de los requisitos para encontrar los casos de uso que son significativos desde un punto de vista de la arquitectura con el propósito de concluir esta fase con una arquitectura base ejecutable.

Actividades
Se desarrollan cuatro grupos de actividades en paralelo:
- Flujos de trabajo fundamentales
- Planificar las iteraciones
- Realizar la evaluación de esta fase
- Preparación más detallada del entorno de desarrollo

Flujos De Trabajo
Los flujos de trabajo fundamentales aquí propuestos son cinco:

Flujo de Recopilación y refinación de requisitos: se establecerá la prioridad, se estructurarán los casos de uso, en este flujo el analista debe identificar más casos de uso y actores adicionales a aquellos identificados en la fase de inicio, se debe verificar en esta etapa que no se pase por alto nada que pueda tener un impacto en la arquitectura o en aspecto económico. Durante este flujo se deben identificar y desarrollar las interfaces de usuario, además se deberá determinar la prioridad de los casos de uso, y detallar los casos de uso para completar los requisitos que sean necesarios. Al estructurar los casos de uso el analista debe buscar similitudes y oportunidades para mejorar la estructura del modelo de casos de uso, esto se logra a través de los mecanismos como extensión o generalización.

Flujo de análisis: se retoma el borrador que se comenzó a realizar en la fase de inicio para trabajar con los casos de uso que son importantes desde un punto de vista de la arquitectura, así como los que se necesiten refinar para comprender mejor los detalles de la propuesta económica.

En este flujo se deberá realizar lo siguiente:

Análisis de la arquitectura: En esta fase se debe de extender el análisis de la arquitectura que se elaboró durante la fase de inicio hasta que se pueda obtener la base de la arquitectura ejecutable, para lograr esto se puede emplear una arquitectura en capas, identificando los paquetes específicos de la aplicación y los paquetes.
Análisis de un caso de uso: para esto los desarrolladores de casos de uso buscan clases importantes que utilizarán como entrada, así como las responsabilidades que se asignarán a dichas clases, por lo cual para cada caso de uso debe ser especificado en forma más detallada en función de las clases y sus responsabilidades.
Análisis de clases: apara ello se deben tomar en cuenta las responsabilidades que han sido asignadas a estas clases desde diferentes casos de uso.
Análisis de paquete: para realizar este análisis el diseñador meditará sobre los servicios que brindará el sistema y sobre el agrupamiento de clases en paquetes con funciones establecidas.

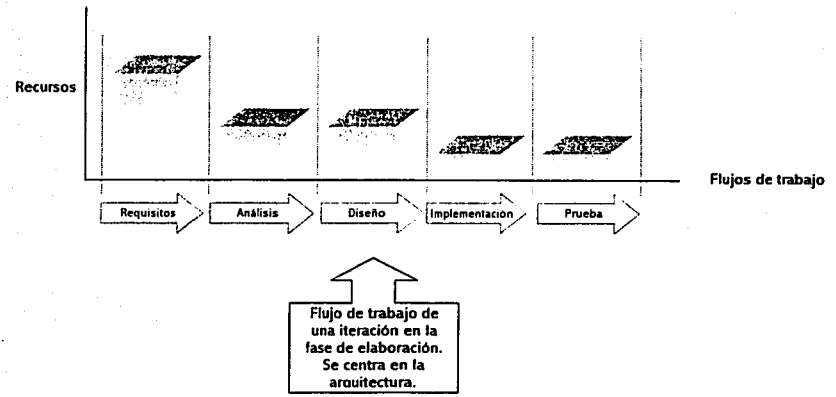
Flujo de Diseño: se diseñará e implementará por lo menos un diez por ciento de los casos de uso. En esta fase se diseñarán los aspectos arquitectónicos importantes. La vista de la arquitectura del modelo de diseño incluirá subsistemas, clases, interfases, el diseñador identifica la arquitectura en capas, los módulos y sus interfaces, las clases de diseño, así como las configuraciones de nodos. Durante esta fase, el diseñador actualizará si es necesario la vista arquitectónica del modelo de diseño.

Flujo de Implementación: se prueban e implementan los componentes arquitectónicamente importantes como:

Implementación de la arquitectura: basándose en la vista de arquitectura y del modelo de despliegue se identifican los componentes para implementar los módulos funcionales, a los componentes ejecutables se les asigna un nodo de red en el cual van a ejecutarse.
Implementación de una clase y de un módulo: consiste en crear una versión ejecutable preliminar.

Integración del sistema: El responsable del plan de integración marca las directrices a seguir para implementar las iteraciones y la integración del plan.

Flujo de Pruebas: se verifica que todos los módulos integrados funcionen, muchas de las capas de estos módulos no necesitan ser probadas, por lo que es importante probar las capas superiores primero para observar poco a poco cómo las capas superiores de módulos hacen uso de las capas inferiores.



Desarrollo del Análisis de Negocio

El objetivo del análisis de negocio es empezar la fase de construcción con plena confianza, es decir que no se salga de los límites del negocio, de la planificación, esfuerzo, calidad y coste estimados. Las estimaciones se basan en el tamaño del proyecto y en la productividad de los desarrolladores. Con lo anterior, se debe actualizar la recuperación económica estimada, por lo que ahora la organización de desarrollo dispone de una estimación de costo de las fases de construcción y transición.

Evaluación de las Iteraciones de la Fase de Elaboración

Se deben establecer criterios de evaluación en el plan de iteración preparado para esta fase, ya que se deberán evaluar los resultados de cada iteración para verificar que se esté cumpliendo con los objetivos iniciales y mitigar los riesgos, al final de la fase los integrantes del equipo de desarrollo deben de convencerse de que en la fase de elaboración se han mitigado riesgos graves.

Al terminar la fase de elaboración se deben detallar todas las iteraciones restantes de la siguiente fase, es importante recordar que el número de iteraciones depende del tamaño y la complejidad del proyecto.

Productos

- Modelo completo de negocio
- Arquitectura sólida
- Plan del proyecto para la fase de construcción y transición
- Manual de usuario preliminar
- Análisis de negocio completo con la estimación económica
- Lista de riesgo actualizada
- Descripción de la arquitectura incluyendo vistas de modelos de casos de uso, análisis, diseño, despliegue e implementación.
- Nueva versión de los modelos: casos de uso, análisis, diseño, despliegue, implementación.

FASE DE CONSTRUCCIÓN

Objetivo
Desarrollar el sistema entero y mostrar una funcionalidad operativa inicial. Durante esta fase todos los componentes y las caracteristicas de la aplicación permanecen y son desarrolladas e integradas en un producto, esta fase es el proceso de manufacturación donde se pone énfasis en la administración de recursos y el control de las operaciones para optimizar costos, esquemas y la calidad.

Se inicia esta etapa con la planeación de la fase anterior y con la autorización de los responsables financieros, para seguir adelante se puede modificar el plan de acuerdo a las circunstancias, por ejemplo, que los fondos y agenda sean menor de lo planificado. En esta fase el trabajo se extiende más allá de la arquitectura, es decir, de los elementos importantes del modelo arquitectónico, ya que la funcionalidad identificada por los diseñadores en la fase de elaboración no estaban completos, sólo se implementaron los casos de uso fundamentales y sus principales escenarios por lo cual el lider de proyecto deberá asignar más personal.

En esta fase se pueden emplear los siguientes puestos de trabajo: ingeniero de casos de uso, ingeniero de componentes, ingeniero de pruebas, responsable dela integración del sistema, encargado de pruebas de integración y encargado de pruebas del sistema a comparación con la fase de elaboración los empleados en la fase de Construcción aumentan al doble.

El volumen de trabajo en la fase de construcción por cada flujo de trabajo es un 60 % de análisis y un 90% de diseño, implementación y pruebas.

Criterios de Evaluación

Los casos de uso representan requisitos funcionales, aunque también tienen requisitos no funcionales tales como el rendimiento y la mitigación de riesgos. Los criterios de evaluación, relativos a esos casos de uso, permiten a los desarrolladores ver claramente cuándo han completado una construcción o iteración preparando además material adicional que se requiere para la evaluación:

Material de usuario: es la preparación por escrito de una primera versión de los materiales de ayuda a los usuarios finales, como guias de ayuda , textos de ayuda, notas de versión y manuales de usuario.

Material de cursos: es una primera versión de materiales para cursos que den soporte a usuarios finales, como diapositivas, tutoriales, notas y ejemplos.

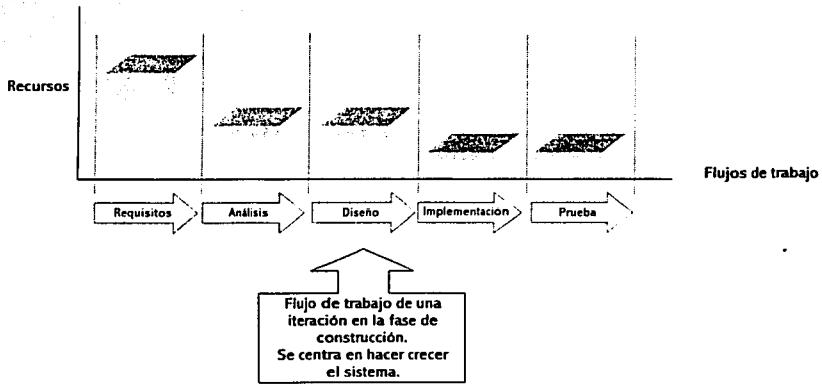
ACTIVIDADES

La iteración arquetipica en esta fase se conformada por cinco flujos de trabajo, y de nuevo se desarrollan cuatro grupos de actividades en paralelo:

- Se realizan los cinco flujos de trabajo fundamentales
- Se planifican las iteraciones
- Se realiza el análisis de negocio

- Se realiza la evaluación de esta fase

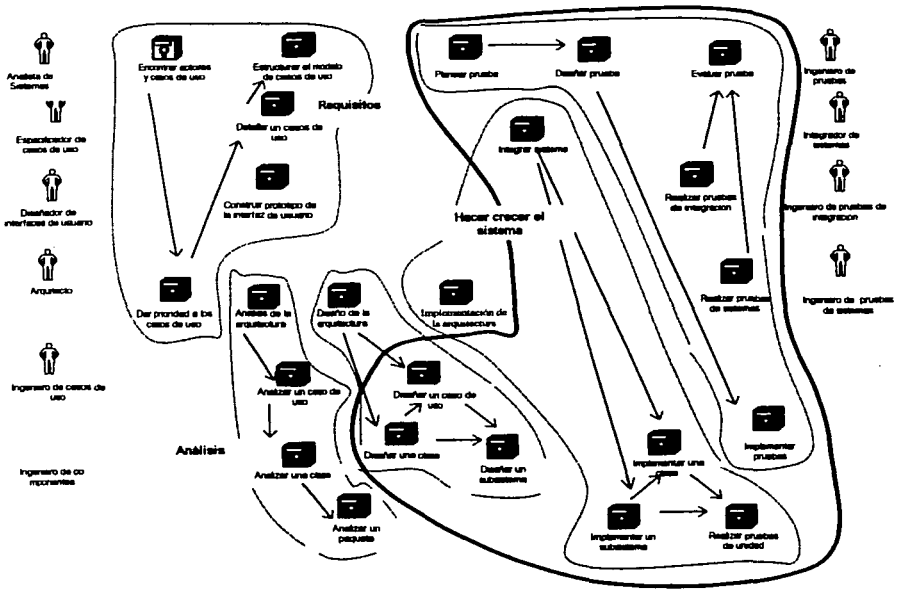
En los primeras iteraciones de la fase de construcción los flujos de trabajo iniciales reciben mayor énfasis y éste se va desplazando a lo largo de las iteraciones en la fase de construcción.



Hasta esta fase los requisitos y la arquitectura son estables, por lo cual se pone énfasis en completar la realización de casos de uso diseñando módulos y clases necesarias implementándolos como componentes y probándolos tanto de forma individual como en conjunto.

Un desarrollo iterativo, guiado por casos de uso y centrado en la arquitectura, construye el software mediante pequeños incrementos y añade cada incremento a la acumulación previa de incrementos de manera que siempre se pueda obtener un ejecutable, esto al ordenarlo de forma progresiva ocasiona que los desarrolladores nunca tengan que volver hacia atrás varios incrementos y rehacerlos.

Flujos de Trabajo



Flujo de Requisitos o requerimientos: en este flujo nos concentraremos a encontrar más casos de uso y actores construyendo prototipos de las interfases de usuario y detallando la estructura de los mismos. En esta fase sólo resta encontrar una pequeña parte de los casos de uso y actores aproximadamente como un 20%, por lo que se actualizarán los casos de uso y actores en el modelo de casos de uso.

En este flujo se deberán diseñar las interfases de usuario en prototipo, sobre todo si es muy complicada la interfaz, la construcción de este prototipo es parte del flujo de requisitos y no del de diseño, es necesario antes de continuar con los flujos siguientes.

En esta fase, en la medida que identificamos casos de uso los clasificamos por prioridad, de esta manera los encargados de especificar los casos de uso los detallarán completamente, es importante destacar que de preferencia no se podrán hacer cambios en la estructura del modelo de casos de uso debido a que en esta fase el sistema ya tiene una estructura estable y cualquier cambio debe referirse a casos de uso que no han sido desarrollados.

Flujo de Análisis: se utiliza el modelo de análisis que teníamos al final de la fase de elaboración, es decir la vista de la arquitectura, la cual nos servirá para construir el modelo de análisis completo para que al final de la construcción se obtenga un modelo de análisis completo y la vista de la arquitectura sólo será una pequeña parte de él. Para obtener el modelo de análisis completo se requerirá los siguientes análisis:

De la arquitectura: como este análisis ya fue realizado en la fase de elaboración, en este análisis se tendrán pocas actividades, como por ejemplo, las actualizaciones necesarias de los cambios que puedan afectar la arquitectura.
De los casos de uso: para lograr un completo análisis del modelo de casos de uso se requerirá que en cada iteración de esta fase se amplíe el modelo de análisis .
De las clases: al realizar este análisis se continua el trabajo realizado, que se comenzó en la fase de elaboración.
De los paquetes: en esta parte se refinarán los paquetes que fueron encontrados durante la fase de elaboración para acomodarlos a los nuevos casos de uso.

Flujo de Diseño: se diseña e implementa el 90% restante de los caso de uso, así como los que no fueron utilizados para desarrollar la base de la arquitectura.
Para el *diseño de la arquitectura*, en esta fase no se agregarán más módulos, el diseñador puede añadir módulos si son similares o bien alternativos a los que ya están en la arquitectura. El comportamiento de un submódulo funcional puede derivarse de partes de un único caso de uso o de casos de uso relacionados y puede ser adecuado autorizar el desarrollo completo de la funcionalidad en este momento, incluso si otras partes del subsistema provienen de otros casos de uso no dominante, ya que trabajar con casos de uso de menor prioridad en este momento permite al ingeniero de componentes desarrollar el sistema de una sola vez.

Flujo de Implementación: se implementa y se llevan a cabo pruebas de unidad de todos los componentes, trabajando a partir del modelo de diseño, el resultado después de varias iteraciones, integracion y pruebas es la versión operativa inicial que representa el 100% de los caso de uso. Lo que se pretende implementar la arquitectura, clases y submódulos realizando pruebas de unidad e integrales de todo el sistema. Para lograrlo se realiza lo siguiente:

- Implementación de una arquitectura
- Implementar una clase y un submódulo
- Realizar pruebas de unidad
- Integrar el sistema

Flujo de Pruebas: los encargados de pruebas de integración y pruebas de sistemas son los responsables de la comprobación de las construcciones y de la construcción final que constituye la versión completa del sistema. Se realizarán las siguientes pruebas en esta fase:

Planificación de pruebas: se seleccionarán las pruebas adecuadas que cumplan con los objetivos de las siguientes construcciones y al sistema en su totalidad.
Diseño de pruebas: se determinará cómo probar los requisitos en conjunto preparando casos y procedimientos de prueba con ese propósito.
Pruebas de integración: Se ejecutarán casos de prueba donde se pruebe la funcionalidad del sistema en forma integrada, si se detectan fallas estas se anotarán y se le notificará al jefe de proyecto.
Pruebas del sistema: Se realizarán pruebas a la versión parcial del sistema, siguiendo los procedimientos de prueba del sistema, si en estas comprobaciones se detectan fallos el encargado de pruebas lo notificará al líder de proyecto o a la persona que éste designe para su solución.
Evaluar las pruebas: A medida que se realizan las diversas pruebas se revisarán los resultados al final de cada construcción comparándolos con los objetivos anteriormente fijados.

Control del Análisis de Riesgos

La propuesta de negocio , desarrollada al final de la fase de construcción, debe servir de guía al líder de proyecto para ejecutar la fase de construcción, el cual comparará el progreso real al final de cada iteración con el tiempo, costes y esfuerzo planificados, para detectar posibles discrepancias que afecten al proyecto y determinar las medidas necesarias.

Evaluación de las Iteraciones y la Fase de Construcción

Basados en la revisión de los resultados de pruebas y otros criterios de evaluación el jefe de proyecto y grupo de evaluación realizarán lo siguiente:

- Revisar lo logrado en una iteración comparándolo con lo que había sido planificado
- Planificación de las iteraciones siguientes en las que se deberá llevar a cabo el trabajo no completado
- Determinar si la construcción está lista para entrar a la siguiente etapa
- Actualizar el estado de la lista de riesgos
- Realizar el plan de la iteración siguiente
- Actualizar el plan de proyecto
- Al fina de esta fase se determinará si el producto ha superado las pruebas del sistema y ha alcanzado la operatividad inicial requerida

Se debe planificar la fase de transición pero no a detalle como en las fases anteriores sin embargo posiblemente si se pueda planificar a detalle como se deben probar las versiones beta, reproducir el codigo, prepara las instrucciones de prueba.

Productos

- Plan de proyecto para la fase de transición
- Sistema de software ejecutable con capacidad operativa inicial
- Todos los artefactos incluyendo los modelos del sistema
- Descripción de la arquitectura, minimamente modificada y actualizada
- Una versión preliminar del manual de usuario
- Análisis de negocio, que refleja la situación al final de la fase

FASE DE TRANSICIÓN

Objetivo
Garantizar que se tiene un producto preparado para entregar a los usuarios y enseñarles a utilizar el software. En esta fase el líder de proyecto considera que el sistema es confiable para operar en el entorno del usuario, sin embargo, algunos problemas, riesgos y defectos pueden surgir en esta fase.

En esta fase, el usuario puede descubrir nuevas necesidades, no obstante, los cambios deben ser lo suficientemente pequeños como para que puedan ser introducidos sin afectar seriamente el plan del proyecto. Los objetivos de esta fase son:

- Cumplir con todos los requisitos establecidos durante las fases anteriores, hasta la satisfacción de los usuarios.
- Administrar todos los aspectos relativos a la operación del sistema en el entorno del usuario, incluyendo la corrección de los defectos remitidos

La fase se centra en implantar el producto en su entorno de operación, para lo cual se requiere información de los usuarios para:

- Determinar si el sistema hace lo que requieren los usuarios
- Descubrir riesgos inesperados
- Anotar problemas no resueltos
- Encontrar fallas
- Eliminar ambigüedades y lagunas en la documentación de usuario
- Dedicarse a las áreas en las que el sistema muestre deficiencias

Contando con la información anterior, el equipo de desarrollo modifica al sistema y los artefactos relacionados, es importante señalar que no se busca, en esta fase, reformar el producto, el equipo sólo buscará pequeñas deficiencias que pasaron desapercibidas durante la fase de construcción y que pueden ser corregidas dentro de la arquitectura ya establecida. Esta fase finaliza con el lanzamiento del producto.

La planificación de la fase de transición no se puede realizar en forma detallada al final de la fase de construcción, ya que el líder de proyecto sólo iniciará esta fase a partir de las versión beta desarrollada en la fase de construcción para que la prueben los usuario , habrá una cantidad de trabajo desconocida , en función de los resultados de las pruebas beta.

Debido a que el proceso de desarrollo se realiza de forma iterativa, permite a los desarrolladores experimentar durante las iteraciones iniciales, descubrir los errores, en esta fase la reelaboración que se lleve a cabo debe ser menor al 5%, puesto que grandes modificaciones afectarían al sistema.

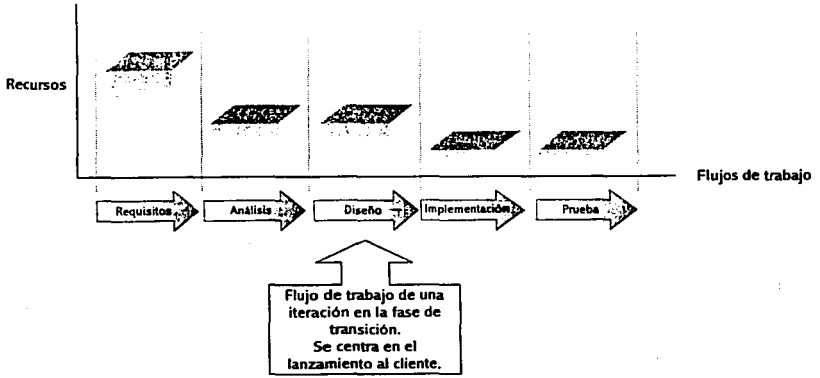
El análisis del resultado de las pruebas beta o de aceptación requiere de personas que estén más orientadas al servicio que al desarrollo.

Criterios de Evaluación

Al final de la fase de construcción, después de las pruebas del sistema, se consideró que el producto alcanzaba su capacidad operativa inicial, por lo que ahora es necesario evaluar lo siguiente:

- Ha superado el producto las pruebas de aceptación realizadas por el cliente
- Han cubierto los usuarios beta las funciones clave
- Tienen los manuales una calidad aceptable
- Está listo el material de los cursos
- Parecen los usuarios y clientes satisfechos con el producto

En esta fase, la actividad de los flujos de trabajo es baja, ya que casi todo el trabajo se realiza en la fase de construcción, las actividades de diseño disminuyen durante la fase de transición, ya que consisten en pequeñas mejoras de diseño para corregir problemas o defectos o para realizar mejoras de última hora, sin embargo, se debe verificar que estas modificaciones no introduzcan nuevos defectos.



Actividades

- En esta fase se realizan fases en paralelo:
- Realizar los cinco flujos de trabajo
- Planificación de las iteraciones
- Análisis de negocio más profundo
- Evaluación de la fase
- Completar los artefactos
- Determinar cuándo se acaba el proyecto

Los detalles de esta secuencia de actividades variarán dependiendo de si el proyecto consiste en la producción de software para el mercado o para un cliente concreto o si es una mejora de un producto ya existente.

La mayor parte del conjunto inicial de usuarios para las pruebas beta, serán usuarios experimentados y trabajarán a partir de documentación preliminar, de la misma forma se les proporciona instrucciones especificas de cómo informar de hallazgos y observaciones.

Al instalar la versión beta se realizan dos tipos de pruebas :
Pruebas Beta: son aquellas en las que el personal de transición no está presente por lo que deben darse instrucciones especificas sobre cómo instalar el software de prueba, cómo operar el sistema y cómo informar de los fallos y problemas encontrados.

Pruebas de aceptación: por lo general está presentes los miembros del equipo de desarrollo, habrá un documento formal de pruebas de aceptación y éste será completado mediante comunicaciones informales. Se recopilan y analizan los resultados de las pruebas con el objeto de llevar a cabo las acciones correctivas y los resultados entrarán en dos categorias: *fallos de codificación menores*, que simplemente tienen que ser localizados y corregidos o bien *problemas más amplios,* que tienen ramificaciones más extensas e incluso la posibilidad de un cambio en la arquitectura.

Esta etapa implica realizar *la adaptación del producto a entornos de usuario* variados, si el producto de software es para usuarios de mercado, los productos son instalados por el propio usuario, si el producto de software es para un cliente individual, implica observar las pruebas finales del sistema, ayudar a instalar el sistema en la sede del cliente, si el producto de software es la mejora de un sistema existente, se tendrá la necesidad de migración de datos o de conversión de bases de datos.

La fase de transición no termina hasta que el proyecto haya completado todos los artefactos, incluyendo los modelos, la descripción de la arquitectura y hasta que el cliente queda completamente satisfecho.

Control del Progreso

El líder del proyecto comparará el progreso real en la fase contra la agenda, esfuerzo y coste planificado para la fase, para ver si el proyecto ha alcanzado los objetivos deseados, añadir métricas para el proyecto para su uso futuro en otros proyectos. Con la revisión del plan de negocio, se prevé si el proyecto tendrá éxito económicamente; el éxito se mide de acuerdo a si el producto alcanzará los objetivos planteados en el margen de los beneficios obtenidos sobre el capital invertido en el desarrollo.

Evaluación de la Fase de Transición
El jefe de proyecto reunirá a un pequeño grupo para evaluar la fase y al ciclo de desarrollo en conjunto, si la organización del proyecto ha llevado a cabo las tres primeras fases de forma efectiva, la fase de transición debería desarrollarse sin sobresaltos y completarse de acuerdo con la agenda y el presupuesto asignado.
Por otro lado, si la organización del proyecto fracasa al identificar todos los riesgos importantes o al desarrollar una arquitectura que no cumpla con los requisitos, o si se fracasa al implementar un diseño que proporcione el sistema requerido, este tipo de deficiencias se mostrarán evidentemente en esta fase .

Es importante analizar lo el equipo del proyecto ha hecho bien o mal, con el objetivo de proporcionar un registro que permitirá organizar futuros proyectos de forma más efectiva y llevar a cabo un proceso de desarrollo con mayor éxito.

Productos

- Software ejecutable, incluyendo el software de instalación
- Documentos legales (licencias, garantias)
- La versión completa y corregida del producto incluyendo todos los modelos del sistema
- La descripción completa y actualizada de la arquitectura
- Manuales y material de formación para el usuario final
- Referencias en páginas web para la ayuda al cliente

3.6 TENDENCIAS FUTURAS

Los métodos orientados a objetos para el análisis y diseño están evolucionando cada vez más, se considera que éstos superarán a las metodologías del análisis y diseño estructurado en los próximos años, lo que traerá nuevos métodos y herramientas CASE que se adoptarán en el desarrollo orientado a objetos por las numerosas ventajas que poseen:

- Abstracción de datos y ocultamiento de la información, los cuales aumentan la confiabilidad y ayudan a separar la especificación de la implantación.
- El encadenamiento dinámico incrementa la flexibilidad.
- La herencia, junto con el encadenamiento tardío permite la reusabilidad, aumentando así la productividad.

La tendencia de los métodos estructurados hacia la orientación a objetos está siendo visible en el desarrollo de herramientas CASE que prestan su apoyo a los métodos orientados a objetos.

UML y RUP ofrecen incrementos potenciales en la productividad y calidad en el desarrollo de sistemas. La adopción, utilización y administración con éxito de estás técnicas, pueden dar lugar a beneficios importantes para todas las personas involucradas en el desarrollo de sistemas, lo que indicará eficiencia competitiva.

Para incrementar los beneficios de UML y RUP se debe contemplar la formación de un equipo de desarrollo con las mejores personas y herramientas que permitan cumplir los objetivos establecidos para concluir con éxito el desarrollo de un sistema orientado a objetos.

Existirán además tendencias relacionadas a los lenguajes de programación orientados a objetos en convergencia con los estructurales, para determinar la posibilidad de entornos de programación con lenguajes mixtos para pequeñas interfaces, generando así, nuevos productos y lenguajes de programación que se irán mejorando, tomando lo mejor de cada uno.

Las bases de datos orientadas a objetos sustituirán a las relacionales, tanto para pequeños sistemas y sobre todo para los que están implicados en tipos de datos complejos. Los RDBMS existentes seguirán siendo bases de datos para entornos comerciales pero irán incluyendo, cada vez más, características orientadas a objetos.

LOS TRES ELEMENTOS CLAVE EN EL DESARROLLO DE SISTEMAS ORIENTADOS A OBJETOS: UML, RUP Y PERSONAS CAPACITADAS

El estudio que hemos realizado del Lenguaje Unificado de Modelado (UML) y del Proceso Unificado Racional (RUP), nos ha permitido asimilar todas las ventajas que brindan dentro del desarrollo de sistemas.

Hasta ahora se han explicado los diagramas de UML, sus características, elementos, notación y su utilidad por un lado, y por otro se hizo el estudio de RUP, su utilidad, las fases que lo componen y las actividades que se desarrollan dentro de cada una de ellas.

Hemos mencionado que UML puede ser utilizado de manera independiente en el modelado de sistemas y que no se casa con ningún proceso de desarrollo en específico, caso similar el de RUP, el cual no utiliza una notación en específico. Sin embargo, consideramos que la conjugación de una notación explícita, fácil de aprender, de utilizar, con una semántica bien definida y detallada como UML, junto con un proceso flexible, configurable y adaptable como RUP, aunados estos dos elementos a un equipo de trabajo capacitado, son los elementos clave para el desarrollo de sistemas de calidad, que satisfagan las necesidades de usuarios y clientes, con los costos y tiempos estimados.

En nuestro estudio se ha encontrado que la combinación de estos elementos permitirá analizar y diseñar sistemas, haciendo énfasis en la construcción de modelos unificados y apegados a la realidad, mejorando así la comunicación entre los usuarios, clientes, desarrolladores, diseñadores y cualquier persona que participe en el desarrollo del mismo.

El Proceso Unificado Racional y el Lenguaje Unificado de Modelado únicamente proponen las directrices a seguir, es decir, son flexibles, ya que pueden ser adaptados y mejorados para cualquier entorno de desarrollo.

Un aspecto muy importante por el cual se propone el uso del RUP, es debido a que no se descuida, durante todo el desarrollo del sistema, la calidad de cada fase, logrando así productos de software exitosos. Por otro lado, proponemos el uso de UML porque cuanta con una notación fácil de entender por todos los participantes en el desarrollo del sistema, desde los analistas hasta los usuarios finales, porque sus diagramas facilitan la etapa de programación debido a su nivel de detalle, sin perder su facilidad de entendimiento. No obstante, un sistema de calidad no se obtendría si no se cuenta con el personal motivado y capacitado en el uso de tecnologías actuales.

CAPÍTULO IV

4. MARCO METODOLÓGICO

4. 1 VARIABLES

Debido al trabajo realizado en el marco problemático y el marco conceptual de la presente investigación, llegamos a la conclusión de que las variables y la hipótesis subsisten de la siguiente manera.

Variables Independientes

- El desarrollar Sistemas Orientados a Objetos utilizando el Lenguaje Unificado de Modelado (UML), el Proceso Unificado Racional (RUP) y un equipo de desarrollo capacitado y motivado.

Variables Dependientes

- Permitirá obtener Sistemas Orientados a Objetos que cumplan con la funcionalidad que el usuario necesita, dentro del tiempo y presupuestos estimados.

4. 2 HIPÓTESIS DEFINITIVA

"Al desarrollar Sistemas Orientados a Objetos utilizando el Lenguaje Unificado de Modelado (UML), el Proceso Unificado Racional (RUP) y contando con un equipo de desarrollo capacitado y motivado, se logrará obtener Sistemas Orientados a Objetos de calidad que cumplan con la funcionalidad que el usuario necesita, dentro del tiempo y presupuestos estimados."

A continuación se muestra una lista valores hipotéticos derivados de la relación anterior:

- El uso del Proceso Unificado Racional (RUP) permitirá controlar las responsabilidades de los integrantes del equipo de desarrollo para realizar liberaciones tempranas que respondan a las necesidades de información de los usuarios.
- El uso del Lenguaje Unificado de Modelado (UML) permitirá mejorar la comunicación entre el equipo de desarrollo y los usuarios al modelar la estructura y comportamiento del sistema con modelos completos y cercanos a la realidad.
- Al contar con un equipo de desarrollo capacitado y motivado, se incrementa la productividad y minimiza el tiempo de desarrollo.

4.3 DEFINICIÓN DEL UNIVERSO

El universo considerado para esta investigación, que tiene como propósito aprobar la relación hipotética planteada, está integrado por empresas e instituciones que se dedican a desarrollar sistemas de información orientados a objetos, que utilicen el UML y/o RUP.

4.4 DETERMINACIÓN DE LA MUESTRA

Debido a que el propósito de esta tesis es dar a conocer la utilidad de la notación del Lenguaje Unificado de Modelado (UML), del Proceso Unificado Racional (RUP), así como destacar la importancia que tiene el contar con un equipo de desarrollo capacitado en el desarrollo de sistemas orientados a objetos, se ha determinado tomar una muestra no probabilística conocida como *muestra por juicio*, ya que la opinión de las personas que entrevistaremos, estará basada en sus conocimientos y experiencia en UML y/o RUP, lo que será suficiente para probar nuestra relación hipotética anteriormente mencionada y proponer estas dos herramientas, junto con un equipo de desarrollo capacitado, como elementos clave de una metodología que permita desarrollar exitosos sistemas orientados a objetos, que cumplan con la funcionalidad que requiere el usuario, dentro del tiempo y presupuestos estimados.

Como la muestra será por juicio en este caso la opinión de un experto es crucial para avalar la utilización de los resultados obtenidos, la muestra por juicio es aquella que resulta de una opinión, tomando en cuenta la concurrencia de efectos y cómo estos afectan al objeto de estudio.[1]

4.5 DEFINICIÓN DEL MÉTODO DE LA INVESTIGACIÓN

Para esta investigación se utilizará el cuestionario con el propósito de conocer el punto de vista de personas que pertenecen a las empresas e instituciones que desarrollen sistemas orientados a objetos, adicionalmente procuraremos realizar entrevistas con dichas personas para profundizar más en el tema y compartir sus experiencias y puntos de opinión, para aprobar la relación hipotética planteada.

[1] Mark L- Berenson, David M. Levine Estadística Básica en Administración, Conceptos y Aplicaciones, Prentice Hall, Hispanoamericana 4ª Edición, México 1992 ISBN 58259.

4.6 COSTO DE LA INVESTIGACIÓN.

Desarrollo de la Tesis Profesional "RUP y UML como elementos clave en el desarrollo de Sistemas Orientados a Objetos"

Costo	Cantidad	Costo por unidad	Mensual	Anual
Sueldo de personas:				
Investigadores	2	8,000.00	16,000.00	160,000.00
Equipo de Cómputo: (computadora, impresora, escáner,)	2	19,00. 8,000 depreciación del 30%		6,750.00
Papelería y artículos diversos de oficina:				
Hojas	8 millares	150.00		1,200.00
Cartuchos	6	350.00		2,100.00
Discos 3 ½	12	72.00		72.00
CD	2	13.00		13.00
Plumas	4	2.00		8.00
Lápices	8	2.00		16.00
Carpetas	6	50.00		300.00
Cuadernos	2	10.00		20.00
Mobiliario y Equipo:				
Escritorio	2	500.00		124.00
Sillas	2	100.00 depreciación del 25 %		
Servicios:				
Luz			100.00	1,000.00
Agua			58.00	580.00
Teléfono			150.00	1,500.00
Gastos diversos:				
Copias	4280	.20		856
Transporte			80.00	800.00
Renta oficina			1,500.00	15,000.00
Renta Internet			420.00	4,200.00
Total:				**194,539.00**

4.7 CONSTRUCCIÓN DEL CUESTIONARIO

Pregunta		
1. ¿Utiliza algún proceso específico para el desarrollo de sistemas?	Conocer el proceso de desarrollo que más utilizan.	RUP o TSP
2. ¿Qué notación utiliza para el Análisis y Diseño?	Conocer la notación más utilizada al desarrollar sistemas orientados a objetos.	UML
3. ¿La notación utilizada le ha permitido una comunicación eficiente con los usuarios?	Saber qué tan útil es UML para realizar modelos que expresen la funcionalidad del sistema en términos del lenguaje del usuario	Si (si utiliza UML).
4. ¿La notación utilizada le ha permitido identificar fácilmente todos los requerimientos de los usuarios?	Conocer si en la práctica se cumple el objetivo que persigue la notación de UML.	SI (si utiliza UML), ya que provee simbología y diagramas que son útiles para la comprensión de los requerimientos, procesos y estructura del sistema a desarrollar.
5. ¿La notación que utiliza le ha permitido diseñar fácilmente sistemas que cumplan satisfactoriamente con los requerimientos de los usuarios?	Comprobar qué tan fácil resulta la aplicación de este lenguaje.	SI (si utiliza UML), ya que posee una simbología muy cercana a la realidad.
6. ¿El proceso de desarrollo y la notación utilizada le ha permitido incrementar la calidad del software desarrollado y la satisfacción de los usuarios?	Saber si el proceso de desarrollo que emplean reduce los riesgos y errores del producto desde las etapas tempranas	Si (Si utiliza RUP) porque en cada etapa de desarrollo se realiza una mejora continua del producto semi-final.
7. ¿Cree que el proceso de desarrollo que utiliza es el adecuado para verificar el cumplimiento de la calidad y los requerimientos de los usuarios?	Conocer los beneficios que han obtenido al aplicar este proceso de desarrollo.	*Si utiliza RUP:* Menores errores. Menor tiempo de desarrollo. Simplifica procesos y facilita el desarrollo. Permite la reutilización de modelos. Ayuda a comprender sistemas complejos. Permite realizar estimaciones razonables. Ayuda a comunicar ideas a otras personas.
8. ¿La notación que utiliza le permite generar documentación que refleje real y eficientemente la estructura.	Conocer la opinión de los entrevistados en base a su experiencia acerca de la utilidad del	SI (si ocupa UML).

comportamiento y el funcionamiento del sistema?	UML.	
9. ¿La documentación que realiza durante el análisis y diseño, le proporciona una guía eficientemente de programación del sistema?	Verificar si se cumple el propósito de UML al aplicarlo.	SI (si ocupa UML).
10. ¿El proceso utilizado le ha ayudado a realizar liberaciones continuas que le permitan verificar la calidad del sistema que se está desarrollando?	Conocer la opinión de los entrevistadores, en base a su experiencia, acerca de la importancia que tiene el equipo de trabajo en el desarrollo de un sistema	SI, ya que la buena capacitación y motivación influye en el tiempo de desarrollo.
11. ¿El proceso de desarrollo que utiliza ha influido en la culminación de los sistemas en el tiempo y presupuestos estimados?	Verificar si se cumple el propósito de RUP, al aplicarlo.	SI (si ocupa RUP), porque fomenta la organización y planeación.
12. ¿Considera que el proceso de desarrollo utilizado le permite controlar las responsabilidades de cada uno de los integrantes del equipo de trabajo (analistas, diseñadores y desarrolladores)?	Conocer si el proceso de desarrollo que utilizan fomenta el establecimiento de actividades y responsabilidades para cada uno de los integrantes del equipo de desarrollo de manera eficiente.	Si (si utiliza RUP).
13 ¿Cuáles piensa que sean las causas por las que no se lleva a cabo una adecuada documentación en las etapas de análisis y diseño?	Conocer la problemática actual que se presenta al desarrollar sistemas.	Respuesta abierta.
14. ¿Conoce el término de UML (Unified Modeling Language, Lenguaje Unificado de Modelado)?	Si no utiliza UML, saber si al menos conoce su concepto	No (si no utiliza UML).
15.¿Conoce el término de RUP (Rational Unified Process, Proceso Unificado Racional)?	Si no utiliza RUP, saber si al menos conoce su concepto	No (si no utiliza RUP).
16.¿Ha utilizado UML?	Saber si ha experimentado la utilización de este lenguaje (notación)	No (si no conoce el término UML).
17.¿Qué beneficios ha experimentado al utilizar UML en el desarrollo de sistemas?	Conocer las ventajas que se obtienen al utilizar UML	Respuesta abierta.
18.¿Ha utilizado RUP?	Saber si ha experimentado la utilización de este método de desarrollo	No (si no conoce el término RUP).

Pregunta	Razón de ser	Respuesta esperada
19.¿Qué beneficios ha experimentado al utilizar RUP en el desarrollo de sistemas?	Conocer las ventajas que se obtienen al utilizar RUP.	Respuesta abierta.

4.8 FORMATO DEL CUESTIONARIO

UNAM

FCA

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

OBJETIVO: *Obtener una opinión empírica o profesional acerca de las ventajas que se obtienen con el uso de la notación UML (Lenguaje Unificado de Modelado) y/o RUP (Proceso Unificado Racional) para el desarrollo de sistemas orientados a objetos.*

Cuestionario aplicado por las alumnas: **Silvia Jiménez Luna** *con número de cuenta* **9411958-9** *y* **Claudia Elvira Soriano Monzalvo** *con número de cuenta* **9425813-4**, *de la FCA-UNAM, para fines de la aprobación o 7disprobación de la relación hipotética preliminar planteada en su tesis profesional titulada "UML y RUP como elementos clave en el desarrollo de sistemas orientados a objetos".*

CUESTIONARIO APLICADO A:

Fecha: ____/____/____

Nombre: _____

INSTRUCCIONES: *Conteste las siguientes preguntas marcando Sí o No según considere. En caso de ser necesario escribir en la parte reversa de la hoja o bien anexar hojas.*

1. ¿Utiliza algún proceso específico para el desarrollo de sistemas?

( ) Cascada *( ) RUP (Proceso Unificado Racional)*
( ) Espiral *( ) TSP (Team Software Process)*
( ) Evolutiva *( ) Ninguno*

2. ¿Qué notación utiliza para el Análisis y Diseño?

( ) Yourdon *( ) UML (Lenguaje Unificado de Modelado)*
( ) Booch *( ) Ninguna*

3. ¿La notación utilizada le ha permitido una comunicación eficiente con los usuarios?

SÍ____ NO____

¿Por qué?

_____

_____

4. *¿La notación utilizada le ha permitido identificar fácilmente todos los requerimientos de los usuarios?*

SÍ_____ NO_____

¿Por qué?

_____

_____

5. *¿La notación que utiliza le ha permitido diseñar fácilmente sistemas que cumplan satisfactoriamente con los requerimientos de los usuarios?*

NO_____

SÍ_____

¿Por qué?

_____

_____

6. *¿El proceso de desarrollo y la notación utilizada le ha permitido incrementar la calidad del software desarrollado y la satisfacción de los usuarios?*

NO_____

SÍ_____

¿Por qué?

_____

_____

7. *¿Cree que el proceso de desarrollo que utiliza es el adecuado para verificar el cumplimiento de la calidad y los requerimientos de los usuarios?*

SÍ_____ NO_____

¿Por qué?

_____

_____

8. *¿La notación que utiliza le permite generar documentación que refleje real y eficientemente la estructura, comportamiento y el funcionamiento del sistema?*

SÍ_____ NO_____

¿Por qué?

_____

_____

9. *¿La documentación que realiza durante el análisis y diseño, le proporciona una guía eficientemente de programación del sistema?*

SÍ_____ NO_____

¿Por qué?

_____

_____

10. *¿El proceso utilizado le ha ayudado a realizar liberaciones continuas que le permitan verificar la calidad del sistema que se está desarrollando?*

SÍ_____ NO_____

¿Por qué?

_____

_____

11. *¿El proceso de desarrollo que utiliza ha influido en la culminación de los sistemas en el tiempo y presupuestos estimados?*

SÍ_____ NO_____

¿Por qué?

_____

_____

12. *¿Considera que el proceso de desarrollo utilizado le permite controlar las responsabilidades de cada uno de los integrantes del equipo de trabajo (analistas, diseñadores y desarrolladores)?*

SÍ_____ NO_____

¿Por qué?

_____

_____

13 *¿Cuáles piensa que sean las causas por las que no se lleva a cabo una adecuada documentación en las etapas de análisis y diseño?*

_____

_____

14. *¿Conoce el término de UML (Unified Modeling Language, Lenguaje Unificado de Modelado)?*

SÍ_____ NO_____

15.*¿Conoce el término de RUP (Rational Unified Process, Proceso Unificado Racional)?*

SÍ_____ NO_____

16.*¿Ha utilizado UML?*

SÍ_____ NO_____

***17.** En caso de haber respondido afirmativamente la pregunta 16:*
¿Qué beneficios ha experimentado al utilizar UML en el desarrollo de sistemas?

***18.**¿Ha utilizado RUP?*

SÍ_____ NO_____

***19.** En caso de haber respondido afirmativamente la pregunta 18:*
¿Qué beneficios ha experimentado al utilizar RUP en el desarrollo de sistemas?

4.9 PERSONAS A ENTREVISTAR:

Nombre: Lic. Jorge Luis Olguin Ochoa
Puesto: Líder de Proyecto
Empresa: IFE (Instituto Federal Electoral)

Nombre: Ing. Alejandro Rodriguez Hernández
Puesto: Desarrollo de Sistemas Consultor Especialista Senior
Empresa: Telcel Ericcson

Nombre: Act. Alejandro Talavera Rosales
Puesto: Técnico Académico Laboratorio de Multimedia
Empresa: DGSCA

Nombre: Lic. Ricardo Peña Vera Cervantes
Puesto: Desarrollador
Empresa: GNP

Nombre: Ing. Héctor Valdivia Rosas
Puesto: Desarrollador
Empresa: Dual Hi-Tech

Nombre: Lic. Luis Castillo
Puesto: Desarrollador
Empresa: ATM Consultores

Nombre: Ing. José Jaime Valencia Pérez
Puesto: Sub-Gerente de Sistemas de Seguridad en T.I.
Empresa: Valper Computer Art. S.A. de C.V. / TELMEX, S.A. de C.V.

Nombre: Lic. Guadalupe Ibargüengoitia G.
Puesto: Profesora de Ingeniería de Software
Empresa: Facultad de Ciencias, UNAM

4.10 MATRIZ DE RESULTADOS DE LOS CUESTIONARIOS PROFESIONALES Y EMPÍRICOS APLICADOS

PREGUNTA	1	2	3		4		5		6		7		8		9	
Talavera	TSP	UML	SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		SÍ	
Olguín	TSP	UML	SÍ		SÍ		SÍ			NO	SÍ		SÍ		SÍ	
Valdivia	RUP	UML	SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		SÍ	
Castillo	Espiral TSP	UML IDEFO, Idef 1x	SÍ			NO	SÍ		SÍ		SÍ			NO	SÍ	
Peña	Cascada	Yourdon	SÍ		SÍ		SÍ			NO		NO	SÍ		SÍ	
Valencia	Evolutivo	OOSA		NO		NO	SÍ			NO		NO		NO		NO
Rodríguez	Cascada	Ninguna		NO	SÍ		SÍ		SÍ		SÍ		SÍ		SÍ	
Ibargüengoitia	RUP TSP	UML	SÍ		SÍ		SÍ		SÍ		SÍ		SÍ		SÍ	
PREGUNTA	1	2	3		4		5		6		7		8		9	
TOTAL	RUP =2 TSP =4 Cascada = 2 Evolutivo =1 Espiral=2	UML =5 Idef 1x=1 Yourdon=1 OOSA=1 Ninguna=1	SÍ 6	NO 2	SÍ 6	NO 2	SÍ 8	NO 0	SÍ 5	NO 3	SÍ 6	NO 2	SÍ 6	NO 2	SÍ 7	NO 1

PREGUNTA	10		11		12		13	14		15		16		17	18		19
Talavera	SÍ		SÍ		SÍ		Falta de estándares de producción y buena integración de los equipos de desarrollo.	SÍ		SÍ		SÍ		Fácil de aprender y con ello se genera la documentación del mismo.	SÍ		Define etapas de desarrollo concretas y tareas generales en el desarrollo de software.
Olguín		NO	SÍ		SÍ		Porque no se cuenta con una herramienta adecuada (software), para llevar a cabo esta documentación.	SÍ			NO	SÍ		Hace más entendible el sistema para *todas* las áreas que están involucradas con él (Análisis y diseño, desarrollo, usuario que solicita el sistema, capacitación, etc).		NO	
Valdivia	SÍ		SÍ		SÍ			SÍ		SÍ		SÍ		Las afirmaciones de las preguntas de sí y no.	SÍ		Las afirmaciones de las preguntas de sí y no.
Castillo	SÍ			NO	SÍ			SÍ		SÍ		SÍ		Mejor especificación una vez establecidos los requerimientos del usuario.		NO	
a		NO		NO	SÍ		Porque no existe un forma eficiente de mantenerlos actualizados y que participen como entrada (física/real) para la generación de otros entregables a través de su propia evolución.	SÍ		SÍ		SÍ		Es la columna vertebral de las disciplinas para el desarrollo de sistemas, ya que como lenguaje permite a nivel organización hablar en términos desde los usuarios cuando modelan sus procesos de negocio y casos de uso, hasta los programadores cuando traducen las clases, operaciones, relaciones y atributos en componentes funcionales de la aplicación e inclusive en pruebas.	SÍ		Está en proceso de hacerlo.

PREGUNTA	10		11		12		13	14		15		16		17	18		19
Valencia		NO		NO	SÍ		No hay un sistema que lo lleve de manera automatizada. En la mayoría de los proyectos, hay otros que sí lo permiten como SAP.	SÍ			NO		NO			NO	
Rodríguez	SÍ			NO	SÍ		El planteamiento del sistema no es el adecuado. Programación al vapor (bomberazos).		NO	SÍ			NO		SÍ		Desarrollo más rápido y sofisticado, detección de problemas inmediatos, mejor administración del proyecto.
Ibargüengoitia	SÍ		SÍ		SÍ		Falta de herramientas de apoyo al proceso que integren todas las etapas y mantenga la congruencia de los modelos.	SÍ		SÍ		SÍ		Es una notación que permite que se use durante todo el proceso.	SÍ		Estamos en el desarrollo de un proceso con RUP, pero ayuda al tener definido lo que se debe de hacer y las plantillas.

238

PRENGUNTA	10		11		12		13	14		15		16		17	18		19
TOTAL	SÍ 5	NO 3	SÍ 4	NO 4	SÍ 8	NO 0		SÍ 7	NO 1	SÍ 6	NO 2	SÍ 6	NO 2		SÍ 5	NO 3	

4.11 CONCLUSIONES POR PREGUNTA

1. ¿Utiliza algún proceso específico para el desarrollo de sistemas?	
TALAVERA	TSP
OLGUIN	TSP
VALDIVIA	RUP
CASTILLO	Espiral Y TSP
PEÑA	Cascada
VALENCIA	Evolutivo
RODRIGUEZ	Cascada
IBARGÜENGOITIA	RUP y TSP

Conclusión de la Pregunta 1. En esta pregunta, cuatro de las ocho personas entrevistadas utilizan TSP, dos mas utilizan RUP, otros dos utilizan el proceso en cascada y una el evolutivo. Cabe mencionar que dos de las personas hacen uso de dos procesos, tal es el caso de Castillo y de Ibargüengoitia. Con los resultados obtenidos podemos darnos cuenta de que TSP es uno de los procesos más utilizados y después se encuentra RUP. Debido a que TSP es un proceso de desarrollo para sistemas orientados a objetos que, junto con RUP, está teniendo gran auge dentro del ámbito informático, nosotras esperábamos que los entrevistados contestaran RUP o TSP, ya que son los procesos más adecuados y eficientes que existen para el desarrollo de sistemas de información debido a los beneficios que ofrecen.

2. ¿Qué notación utiliza para el Análisis y Diseño?	
TALAVERA	UML
OLGUIN	UML
VALDIVIA	UML
CASTILLO	UML y IDEFO, Idef Ix
PEÑA	Yourdon
VALENCIA	OOSA
RODRIGUEZ	Ninguna
IBARGÜENGOITIA	UML

Conclusión de la Pregunta 2. Cinco de las ocho personas entrevistadas utilizan UML, por lo que podemos concluir que la notación de UML es la más utilizada dentro de las etapas de análisis y diseño, lo cual concuerda con los resultados esperados por nosotras. Es importante destacar que ésta notación está siendo cada vez más utilizada por la comunidad informática.

3. ¿La notación utilizada le ha permitido una comunicación eficiente con los usuarios?	
TALAVERA	SI Presenta elementos gráficos de fácil entendimiento.
OLGUIN	SI El modelo permite entender el sistema sin necesidad de complicarse con códigos, se ve el funcionamiento del sistema en un modelo visual.
VALDIVIA	SI Amplia especificación de la funcionalidad del negocio a automatizar.
CASTILLO	SI
PEÑA	SI Es fácilmente asimilada por los usuarios la notación utilizada.
VALENCIA	NO Los usuarios no saben del sistema ni de desarrollos.
RODRIGUEZ	NO Porque el contacto con el usuario lo tiene la empresa adquiriente.
IBARGÜENGOITIA	SI Es clara y entendible, no ambigua.

Conclusión de la Pregunta 3. Todos los entrevistados que utilizan UML contestaron que SÍ a esta pregunta, la mayoria de ellas coincidieron en que el uso de esta notación les ha permitido tener una comunicación eficiente con los usuarios, debido a que es fácil de entender por su claridad. Las personas que contestaron NO a esta pregunta son las que utilizan otra notación, en general opinan que los usuarios no saben del desarrollo de sistemas ni de la notación.

4. ¿La notación utilizada le ha permitido identificar fácilmente todos los requerimientos de los usuarios?	
TALAVERA	SI Al ser un estándar se pueden integrar mesas de discusión y acuerdos.
OLGUIN	SI En la etapa de análisis y diseño se especifican todos estos requerimientos y se representan en este modelo visual, el cual es más fácil de entender.
VALDIVIA	SI
CASTILLO	NO
PEÑA	SI En esencia la identificación de procesos de negocio y su descomposición fundamental en la mayoria de las técnicas de análisis y diseño parten de los mismos principios.
VALENCIA	NO Los usuarios saben adónde llegar pero no saben cómo y las personas de sistemas saben cómo pero no sabe qué desean los usuarios.
RODRIGUEZ	SI

IBARGÜENGOITIA	SI Los casos de uso facilitan la identificación de los requerimientos funcionales

Conclusión de la Pregunta 4. Cuatro de las cinco personas que utilizan UML contestaron afirmativamente a esta pregunta, comentan que la notación UML les ayuda a crear modelos fáciles de entender, a través de los cuales los mismos usuarios pueden agregar o modificar la funcionalidad del sistema. Se menciona que los casos de uso ayudan a identificar fácilmente los requerimientos de los usuarios. Otra de las ventajas mencionadas de UML es que es un estándar o lenguaje único entre usuarios y equipo de desarrollo que facilitan llegar a acuerdos. Con dichos resultados concluimos que UML si ayuda a identificar los requerimientos de los usuarios.

Como podemos observar, uno de los entrevistados (Valencia) respondió NO a esta pregunta, dicha persona utiliza la notación OOSA.

5. ¿La notación que utiliza le ha permitido diseñar fácilmente sistemas que cumplan satisfactoriamente con los requerimientos de los usuarios?

TALAVERA	SI Sintetiza en mucho los acuerdos logrados por el equipo de desarrollo.
OLGUIN	SI Con el modelo visual se genera un prototipo (no se programa inmediatamente) que el usuario verificará hasta quedar convencido, momento en el cual se comenzará a desarrollar.
VALDIVIA	SI
CASTILLO	SI
PEÑA	SI Teniendo clara la técnica o metodología de Análisis y Diseño y aplicando herramientas complementarias para cubrir los vacíos de especificación a cierto nivel de detalle, es muy viable diseñar sistemas que cumplan con los requerimientos.
VALENCIA	SI A veces
RODRIGUEZ	SI
IBARGÜENGOITIA	SI La notación apoya, pero lo que realmente permite identificar los requerimientos es el proceso.

Conclusión de la Pregunta 5. Todas las personas entrevistadas que utilizan UML respondieron que SÍ en esta pregunta, lo que nos permite afirmar que UML permite diseñar fácilmente sistemas que cumplan satisfactoriamente con los requerimientos de los usuarios debido a que a través de los modelos visuales de UML se pueden crear prototipos que el usuario puede verificar y en determinado momento la notación es flexible a los posibles cambios. Las personas que utilizan otras metodologías opinan que a veces, ya que es importante que se entienda claramente la metodología a aplicar.

6. ¿El proceso de desarrollo y la notación utilizada le ha permitido incrementar la calidad del software desarrollado y la satisfacción de los usuarios?	
TALAVERA	SI
OLGUIN	NO El software es el mismo, el desarrollo del sistema si es más comprensible.
VALDIVIA	SI
CASTILLO	SI Particularmente el proceso.
PEÑA	NO Porque no unifica el trabajo, ni tampoco permite aprovechar los artefactos que resultan de cada una de las disciplinas (modelado de negocio, requerimientos, análisis y diseño construcción y pruebas) como entrada para las actividades de la disciplina que le preceden.
VALENCIA	NO Es incipiente todavia en su implementación.
RODRIGUEZ	SI
IBARGÜENGOITIA	SI Se complementan para hacer un buen trabajo en la construcción de software.

Conclusión de la Pregunta 6. En esta pregunta, las cinco personas que utilizan UML y las dos que utilizan RUP respondieron que SÍ; lo cual reafirma nuestra idea de que el proceso de desarrollo RUP y la notación de UML permite incrementar la calidad del software y la satisfacción de los usuarios.
Es importante mencionar que la persona (Peña) que utiliza el proceso en cascada y la notación de Yourdon contestó que NO a esta pregunta y menciona que la notación no le permite aprovechar los artefactos obtenidos en cada etapa del desarrollo, problema que está resuelto en UML.
Otro caso es el de Valencia, quien utiliza la notación de OOSA, quien respondió que NO a la pregunta por considerarlos incipientes (que está en sus inicios).

7. ¿Cree que el proceso de desarrollo que utiliza es el adecuado para verificar el cumplimiento de la calidad y los requerimientos de los usuarios?	
TALAVERA	SI Incluye evaluaciones intermedias con los usuarios
OLGUIN	SI En el modelo visual se especifican las reglas y validaciones que deben cumplir el sistema, así cuando el usuario acepte el prototipo, también acepta estas reglas y validaciones
VALDIVIA	SI
CASTILLO	SI
PEÑA	NO Porque no son claras las métricas de calidad, los indicadores de calidad, los procesos de verificación y validación de calidad

VALENCIA	NO No se cumplen las especificaciones de los usuarios
RODRIGUEZ	SI
IBARGÜENGOITIA	SI Se apoya en buenas prácticas de desarrollo

Conclusión de la Pregunta 7. En esta pregunta todos las personas que utiliza TSP y/o RUP contestaron afirmativamente a esta pregunta, con lo cual concluimos que ambos procesos de desarrollo ayudan a verificar la calidad y los requerimientos de los usuarios continuamente, cabe mencionar que ambos procesos de desarrollo tienen la caracteristica de hacer evaluaciones en todo el proceso de desarrollo (desde el análisis hasta la implantación) en conjunto con el usuario para que este verifique que el sistema que se está desarrollando cumple con sus requerimientos.
Las dos personas que contestaron que NO (Peña y Valencia) utilizan el proceso en cascada y el evolutivo, respectivamente, y mencionan que los proceso no tienen métricas claras, indicadores que les permitan verificar y validar la calidad de los sistemas.

8. ¿La notación que utiliza le permite generar documentación que refleje real y eficientemente la estructura, comportamiento y el funcionamiento del sistema?	
TALAVERA	SI Son el núcleo del desarrollo de sistemas
OLGUIN	SI Porque esta documentación se refleja tanto requerimientos del usuario como reglas y validaciones creadas por el usuario que solicitó el sistema y el equipo de desarrollo
VALDIVIA	SI
CASTILLO	NO
PEÑA	SI El problema es cuando en una fase posterior del ciclo de desarrollo del sistema, si existe un cambio en la estructura de la arquitectura del sistema, no se refleja de forma automática por no estar ligados los artefactos generados por las distintas disciplinas, según lo comentado en la respuesta 6
VALENCIA	NO No se tiene ningún sistema documental. Es a la voluntad del desarrollador
RODRIGUEZ	SI
IBARGÜENGOITIA	SI Pero es necesaria una herramienta que mantenga la integridad de los modelos

Conclusión de la Pregunta 8. Cuatro de las cinco personas que utilizan UML respondieron que SÍ en esta pregunta, por lo que se deduce que UML si permite generar documentación que muestre real y eficientemente la estructura, comportamiento y funcionalidad del sistema.

9. ¿La documentación que realiza durante el análisis y diseño, le proporciona una guía eficientemente de programación del sistema?	
TALAVERA	SI Contiene información clave de los sistemas a utilizar
OLGUIN	SI Porque esta documentación se reflejan tanto requerimientos del usuario, como reglas y validaciones creadas por el usuario que solicitó el sistema y el equipo de desarrollo
VALDIVIA	SI
CASTILLO	SI
PEÑA	SI Aunque no se puede aprovechar como entrada para estructurar los subsistemas y objetos de los mismos
VALENCIA	NO Son especificaciones muy generales
RODRIGUEZ	SI
IBARGÜENGOITIA	SI En el análisis y diseño se toman las decisiones importantes que faciliten y se codifica

Conclusión de la Pregunta 9. Todas las personas entrevistadas que utilizan UML respondieron SI en esta pregunta, comentan que la notación proporciona desde la etapa de análisis y diseño información clave (como reglas, validaciones) que será de gran utilidad en la fase de desarrollo del sistema (codificación). Con este resultado podemos concluir que el uso de UML no sólo beneficia en las etapas de análisis y diseño, sino en todo el proceso de desarrollo debido a que la documentación generada con UML en el análisis y diseño será de gran utilidad en las etapas posteriores, por ejemplo, facilitará las tareas de los programadores disminuyendo el tiempo requerido para esta etapa, la documentación también servirá para realizar las pruebas del sistema.

10. ¿El proceso utilizado le ha ayudado a realizar liberaciones continuas que le permitan verificar la calidad del sistema que se está desarrollando?	
TALAVERA	SI Porque incluye evaluaciones intermedias con los usuarios
OLGUIN	NO Nuestra liberación final es el prototipo
VALDIVIA	SI
CASTILLO	SI Usamos Project Manager de Aimware
PEÑA	NO Por el esquema en cascada que se utiliza y no un proceso iterativo
VALENCIA	NO Está en implementación un proceso de control de cambios y de versiones

RODRIGUEZ	SI
IBARGÜENGOITIA	SI Se tiene bien definida la documentación y se pueden realizar cambios

Conclusión de la Pregunta 10. Las personas que utilizan RUP y/o TSP contestaron afirmativamente a esta pregunta, por lo que concluimos que ambos procesos permiten realizar liberaciones continuas, lo cual permite ir verificando la calidad del sistema durante todo el proceso de desarrollo.
Las tres personas que contestaron NO a esta pregunta, coincide con las personas que usan el proceso en cascada y el evolutivo, también está incluida una persona que utiliza TSP, ya que comenta que sólo se hace una liberación final. Esto nos confirma que sólo un proceso iterativo, en el cual se hagan de verdad liberaciones continuas al usuario nos permitirá verificar la calidad de los sistemas que se desarrollen.

11. ¿El proceso de desarrollo que utiliza ha influido en la culminación de los sistemas en el tiempo y presupuestos estimados?	
TALAVERA	SI Permite una correcta planeación y división de tareas
OLGUIN	SI Con el modelo visual es más fácil comprender los sistemas y desarrollarlos
VALDIVIA	SI
CASTILLO	SI
PEÑA	NO Se tiene la falsa idea de que con una sola vez que se haga un buen análisis y diseño es posible tener todos los requerimientos, las implicaciones y las estrategias necesarias para resolverlos y la realidad ha demostrado que los requerimientos están siempre presentes en el ciclo de vida
VALENCIA	NO Normalmente no en tiempo
RODRIGUEZ	NO
IBARGÜENGOITIA	SI Esta bien definidas las actividades a realizar, quién las debe hacer y cuándo

Conclusión de la Pregunta 11. Todas las personas que utilizan RUP y/o TSP contestaron afirmativamente a esta pregunta, lo cual reafirma la idea que tenemos de que el uso de RUP influye en la culminación de los sistemas en el tiempo y con el presupuesto estimados desde el principio, esto se debe a que el proceso permite definir las actividades, planear tiempos y personas asignadas a cada tarea.
El resto de las personas que contestaron negativamente a esta pregunta son las que utilizan el proceso en cascada y el evolutivo.

12. ¿Considera que el proceso de desarrollo utilizado le permite controlar las responsabilidades de cada uno de los integrantes del equipo de trabajo (analistas, diseñadores y desarrolladores)?	
TALAVERA	SI Define roles y actividades para un grupo de trabajo
OLGUIN	SI Un buen análisis y diseño se refleja en un menor tiempo de desarrollo y con el modelo visual cada desarrollador se enfoca en el módulo que le haya sido asignado
VALDIVIA	SI
CASTILLO	SI Con el proceso de Administración de Proyectos, no de ingenieria de Software, y un buen de control de cambios se logra
PEÑA	SI Es clara su actividad, el problema es el enfoque que se le da al proceso de ingenieria de software, ya que no es clara la intensidad de las tareas de cada uno de los participantes en función de la etapa y/o iteración por la que está pasando el proyecto
VALENCIA	SI
RODRIGUEZ	SI
IBARGÜENGOITIA	SI Están definidos sus roles y sus responsabilidades

Conclusión de la Pregunta 12. Según los resultados obtenidos en esta pregunta, todos contestaron afirmativamente. Las dos personas que utilizan RUP contestaron afirmativamente, comentan que el proceso define los roles y las responsabilidades para el equipo de trabajo.
Es importante mencionar que todos los entrevistados consideran importante que cada persona o desarrollador se enfoque en las actividades que le corresponden y que se tenga un buen control de las actividades y personas que lleven a cabo dichas actividades, sin embargo con el uso de RUP esta tarea se facilita.

13 ¿Cuáles piensa que sean las causas por las que no se lleva a cabo una adecuada documentación en las etapas de análisis y diseño?	
TALAVERA	Falta de estándares de producción y buena integración de los equipos de desarrollo
OLGUIN	Porque no se cuenta con una herramienta adecuada de software para llevar a cabo esta documentación
VALDIVIA	No opinó
CASTILLO	No opinó
PEÑA	Porque no existe una forma eficiente de mantenerlos actualizados y que participen como entrada (fisica / real) para la generación de otros entregables a través de su propia evolución
VALENCIA	No hay un sistema que lo lleve de manera automatizada en la mayoria de los

	proyectos, hay otros que si lo permiten como SAP
RODRIGUEZ	Porque el planteamiento del sistema no es el adecuado, otra la programación al vapor (bomberazos)
IBARGÜENGOITIA	Falta de herramientas de apoyo al proceso que integren todas las etapas y mantengan la congruencia de los modelos

Conclusión de la Pregunta 13. De acuerdo a la experiencia de los entrevistados las principales causas por las que no se lleva a cabo una adecuada documentación son las siguientes:

- Falta de estándares al desarrollar software.
- Falta de herramientas que apoyen este proceso y permitan hacerlo de manera automatizada.
- No existe una forma eficiente de mantener actualizada la documentación que sirva de entrada (física/ real) para la generación de otros entregables.
- Desarrollos y programación al vapor por fechas de entrega apresuradas.
- Falta de conciencia de la importancia de estas actividades (análisis y diseño).
- Buena integración de los equipos de desarrollo.

Como podemos observar muchas de las causas mencionadas por los entrevistadas son resueltas por la notación UML, ya que esta notación es un lenguaje estándar, existen herramientas automatizadas para crear y mantener actualizada la documentación.

14. ¿Conoce el término de UML (Unified Modeling Language, Lenguaje Unificado de Modelado)?	
TALAVERA	SI
OLGUIN	SI
VALDIVIA	SI
CASTILLO	SI
PEÑA	SI
VALENCIA	SI
RODRIGUEZ	NO
IBARGÜENGOITIA	SI

Conclusión de la Pregunta 14. La mayoría de los entrevistados conocen el término de UML, solamente una persona de las ocho entrevistados desconoce el término y por consecuencia nunca lo ha utilizado en el desarrollo de sistemas. Por lo que podemos concluir que cada vez está siendo más conocido el término de UML, sin embargo, todavía falta difundir más las bondades de esta notación.

15.¿Conoce el término de RUP (Rational Unified Process, Proceso Unificado Racional)?	
TALAVERA	SI
OLGUIN	NO
VALDIVIA	SI
CASTILLO	SI

PEÑA	SI
VALENCIA	NO
RODRIGUEZ	NO
IBARGÜENGOITIA	SI

Conclusión de la Pregunta 15. La mayoría de las personas entrevistadas conocen el término de RUP, ya que tres de los ocho entrevistados no lo conocen. Consideramos que es importante difundir más acerca de los beneficios que ofrece RUP para mejorar la calidad de la cultura informática.

16.¿Ha utilizado UML?	
TALAVERA	SI
OLGUIN	SI
VALDIVIA	SI
CASTILLO	SI
PEÑA	SI
VALENCIA	NO
RODRÍGUEZ	NO
IBARGÜENGOITIA	SI

Conclusión de la Pregunta 16. Es importante resaltar que de siete personas entrevistadas que conocen el término de UML, solamente seis personas entrevistadas lo han utilizado, al menos en una ocasión. Cabe mencionar que UML es un concepto nuevo y por lo tanto son pocas las personas que tienen experiencia en el uso de esta notación, pero poco a poco se está introduciendo dentro de la comunidad informática y gracias a los beneficios que se están obteniendo.

17. En caso de haber respondido afirmativamente la pregunta 16: ¿Qué beneficios ha experimentado al utilizar UML en el desarrollo de sistemas?	
TALAVERA	Al ser un lenguaje gráfico es fácil de aprender y con ello se genera la documentación del mismo
OLGUIN	Un modelo visual hace más entendible el sistema para todas las áreas que están involucradas con él (análisis y diseño, desarrollo, usuario que solicita el sistema capacitación, etc.)
VALDIVIA	Permite una comunicación eficiente con los usuarios, permite identificar fácilmente los requerimientos de los usuarios, permite generar documentación que refleje el comportamiento del sistema proporcionando así una guía para la documentación.
CASTILLO	Mejor especificación una vez establecidos los requisitos de los usuarios
PEÑA	Es la columna vertebral de las disciplinas para el desarrollo de sistemas, ya que como lenguaje permite a nivel organización hablar en términos de los usuarios cuando modelan sus procesos de negocio y casos de uso, hasta los programadores cuando traducen las clases, operaciones, relaciones y atributos en componentes

	funcionales de la aplicación e inclusive en las pruebas
VALENCIA	
RODRIGUEZ	
IBARGÜENGOITIA	Es una notación que permite que se use en todo el proceso

Conclusión de la Pregunta 17. De acuerdo a la experiencia de los entrevistados las ventajas que proporciona UML son las siguientes:

- Fácil de aprender,
- Proporciona modelos visuales más entendibles para todos las personas involucradas en el desarrollo de sistemas (análisis y diseño, desarrollo, usuario),
- Permite una comunicación eficiente con los usuarios,
- Permite identificar fácilmente los requerimientos de los usuarios,
- Permite generar documentación que refleje el comportamiento del sistema, proporcionando así una guía para la documentación,
- Estandarización de la notación que permite que se use en todo el proceso.

Como podemos ver todos los beneficios que han obtenido las personas entrevistadas han sido mencionados dentro de esta tesis, por lo que la experiencia de los entrevistados nos ayuda a confirmar la existencia de las ventajas de UML en el desarrollo de sistemas.

18.¿Ha utilizado RUP?	
TALAVERA	SI
OLGUIN	NO
VALDIVIA	SI
CASTILLO	NO
PEÑA	NO
VALENCIA	NO
RODRÍGUEZ	SI
IBARGÜENGOITIA	SI

Conclusión de la Pregunta 18. Es importante resaltar que de los cinco entrevistados que conocen el término de RUP; sólo tres personas entrevistadas lo han utilizado al menos en una ocasión. Como podemos ver son pocas las personas que han optado por experimentar las bondades de RUP, ya que de las ocho personas entrevistadas sólo dos lo han utilizado.

19. En caso de haber respondido afirmativamente la pregunta 18: ¿Qué beneficios ha experimentado al utilizar RUP en el desarrollo de sistemas?	
TALAVERA	Define etapas de desarrollo concretas y tareas generales en el desarrollo de software
OLGUIN	

VALDIVIA	Permite incrementar y verificar la calidad del software y los requerimientos de los usuarios, permite realizar liberaciones tempranas e influye en la culminación de los sistemas en tiempo y en el presupuesto estimado, además de controlar las responsabilidades de cada integrante del equipo de desarrollo
CASTILLO	
PEÑA	
VALENCIA	
RODRIGUEZ	El desarrollo es un poco más rápido, aunque más sofisticado, permite la detección de problemas y mejora la administración del proyecto
IBARGÜENGOITIA	Ayuda mucho el tener definido qué se debe de hacer y las plantillas

Conclusión de la Pregunta 19. De acuerdo a la experiencia de los entrevistados en la utilización de RUP las ventajas que proporciona RUP son las siguientes:

- Define las tareas generales durante el desarrollo de software,
- Permite incrementar y verificar la calidad del software y los requerimientos de los usuarios, permite realizar liberaciones tempranas e influye en la culminación de los sistemas en tiempo y en el presupuesto estimado,
- Controlar las responsabilidades de cada integrante del equipo de desarrollo,
- Desarrollo un poco más rápido,
- Permite detectar problemas a tiempo,
- Mejora la administración del proyecto.

Es importante confirmar, gracias a la experiencia de los entrevistados, las ventajas que ofrece RUP en el desarrollo de sistemas, ya en términos prácticos.

4.12 CONCLUSIÓN POR PERSONA

PERSONA ENTREVISTADA	CONCLUSIÓN
TALAVERA	Utilizado UML y TSP en el desarrollo de sistemas, pero también conoce RUP y lo ha utilizado. Su opinión acerca de *TSP* es que le permite incrementar y verificar la calidad del software, identificar los requerimientos de los usuarios, controlar las responsabilidades de cada integrante del equipo de desarrollo, realizar liberaciones tempranas y terminar los sistemas en tiempo y en el presupuesto estimado. Es importante destacar que TSP es un proceso con una filosofía similar a la de RUP, por lo que también declara los mismos beneficios para RUP. Su opinión acerca de *UML* es que es una notación que le ha permitido una comunicación eficiente con los usuarios, identificar fácilmente los requerimientos de los usuarios, generar documentación que refleje real y eficientemente la estructura y comportamiento del sistema, lo que le proporciona una guía eficiente para la programación del sistema. *Los causas por las que considera que no se lleva una adecuada documentación en el desarrollo de sistemas* es debido a la falta de estándares y a la inadecuada integración de los equipos de desarrollo.

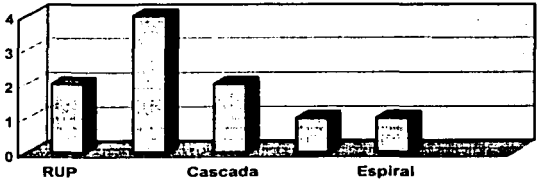
OLGUIN	Ha utilizado UML y TSP en el desarrollo de sistemas, no conoce ni ha utilizado RUP. Su opinión acerca de *TSP es* que le permite verificar la calidad del software y los requerimientos de los usuarios, influye en la culminación de los sistemas en tiempo y en el presupuesto estimado, además de controlar las responsabilidades de cada integrante del equipo de desarrollo. Su opinión acerca de *UML* es que es una notación que le ha permitido una comunicación eficiente con los usuarios, le ayuda a identificar fácilmente los requerimientos de los usuarios y a generar documentación que refleje real y eficientemente la estructura y comportamiento del sistema, proporcionándole una guía eficiente para la programación del sistema. *Los causas por las que no se lleva una adecuada documentación* en el desarrollo de sistemas es debido a la falta de una herramienta adecuada (software).
VALDIVIA	Ha utilizado RUP y UML en el desarrollo de sistemas. Su opinión acerca de *RUP* es que le permite incrementar y verificar la calidad del software y los requerimientos de los usuarios, realizar liberaciones tempranas, lo cual ha influido en la culminación de los sistemas en tiempo y en el presupuesto estimado, además de que le ayuda a controlar las responsabilidades de cada integrante del equipo de desarrollo. Su opinión acerca de *UML* es que es una notación que le ha permitido una comunicación eficiente con los usuarios, le ayuda a identificar fácilmente los requerimientos, a generar documentación sobre la estructura y comportamiento del sistema, esto le proporciona una guía eficiente para la programación del sistema.
CASTILLO	Ha utilizado UML y TSP en el desarrollo de sistemas y NO ha utilizado RUP. Su opinión acerca de *TSP* es que le permite incrementar y verificar la calidad del software y los requerimientos de los usuarios, le ayuda a realizar liberaciones continuas y NO influye en la culminación de los sistemas en tiempo y en el presupuesto estimado, le permite controlar las responsabilidades de cada integrante del equipo de desarrollo. Su opinión acerca de *UML* es que es una notación que le ha permitido una comunicación eficiente con los usuarios, puede diseñar sistemas que cumplan satisfactoriamente con los requerimientos de los usuarios, genera documentación que refleja real y eficientemente la estructura y comportamiento del sistema, lo que le proporciona una guía eficiente para la programación del sistema.
PEÑA	Ha utilizado Cascada y Yourdon en el desarrollo de sistemas. Si ha utilizado RUP y UML. Su opinión acerca de *cascada* es que NO permite verificar la calidad del software y los requerimientos de los usuarios, NO permite realizar liberaciones tempranas y NO influye en la culminación de los sistemas en tiempo y en el presupuesto estimado, además de controlar las responsabilidades de cada integrante del equipo de desarrollo. Su opinión acerca de *Yourdon* es que es una notación que le ha permitido una comunicación eficiente con los usuarios, le ayuda a identificar fácilmente los requerimientos de los usuarios, a generar documentación que refleje real y eficientemente la estructura y comportamiento del sistema, lo que proporciona una guía eficiente para la programación del sistema.

	Los causas por las que considera que no se lleva una adecuada documentación en el desarrollo de sistemas es debido a que no existe una forma eficiente de mantener actualizados los sistemas.
VALENCIA	Ha utilizado el proceso evolutivo y OOSA en el desarrollo de sistemas y no conoce ni ha utilizado RUP, conoce el término de UML pero no lo ha utilizado. Su opinión acerca del proceso *evolutivo es* que le permite incrementar y verificar la calidad del software y los requerimientos de los usuarios, NO le permite realizar liberaciones tempranas y NO influye en la culminación de los sistemas en tiempo y en el presupuesto estimado, pero si puede controlar las responsabilidades de cada integrante del equipo de desarrollo. Su opinión acerca de *OOSA* es que es una notación que NO permite una comunicación eficiente con los usuarios, NO permite identificar fácilmente los requerimientos de los usuarios, NO permite generar documentación que refleje real y eficientemente la estructura y comportamiento del sistema, lo cual NO le proporciona una guia eficiente para la programación del sistema. *Los causas por las que no se lleva una adecuada documentación* en el desarrollo de sistemas es debido a la falta de automatización para realizar la documentación.
RODRIGUEZ	Ha utilizado el proceso en cascada y no utiliza ninguna notación en el desarrollo de sistemas. No conoce y no ha utilizado UML Su opinión acerca de *cascada* es que le permite incrementar y verificar la calidad del software y los requerimientos de los usuarios, le ayuda a realizar liberaciones tempranas, pero NO influye en la culminación de los sistemas en tiempo y en el presupuesto estimado. Sin embargo puede controlar las responsabilidades de cada integrante del equipo de desarrollo. *Los causas por las que no se lleva una adecuada documentación* en el desarrollo de sistemas es debido a la falta de estándares y la inadecuada integración de los equipos.
IBARGÜENGOITIA	Utiliza RUP y UML en el desarrollo de sistemas. Su opinión acerca de *RUP* es que le permite incrementar y verificar la calidad del software y los requerimientos de los usuarios, realizar liberaciones tempranas, le permite la entrega delos sistemas en tiempo y en el presupuesto estimado, además de que le ayuda controlar las responsabilidades de cada integrante del equipo de desarrollo. Su opinión acerca de *UML* es que es una notación que le ha permitido una comunicación eficiente con los usuarios, le ayuda a identificar fácilmente los requerimientos de los usuarios, le permite generar documentación que refleje real y eficientemente la estructura y comportamiento del sistema, lo que proporciona una guia eficiente para la programación del sistema. *Los causas por las que no se lleva una adecuada documentación* es por la falta de herramientas de apoyo al proceso que integren todas las etapas y mantengan la congruencia de los modelos

4.13 PROYECCIÓN DE LAS RESPUESTAS POR CADA PREGUNTA

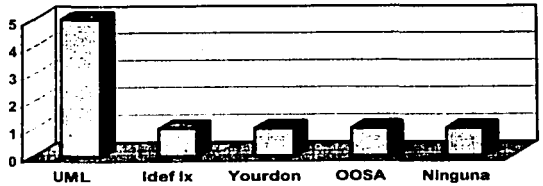
PREGUNTA # 1

¿Utiliza algún proceso específico para el desarrollo de sistemas?



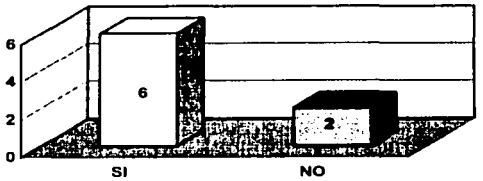
PREGUNTA # 2

¿Qué notación utiliza para el Análisis y Diseño?



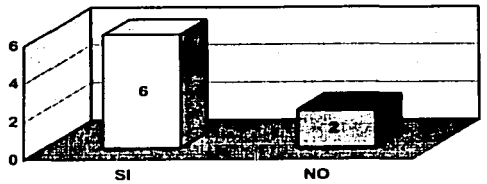
PREGUNTA # 3

¿La notación utilizada le ha permitido una comunicación eficiente con los usuarios?



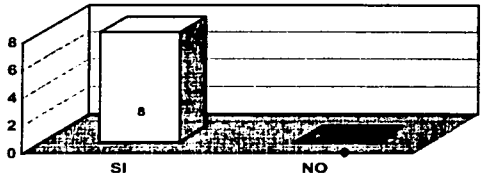
PREGUNTA # 4

¿La notación utilizada le ha permitido identificar fácilmente todos los requerimientos de los usuarios?



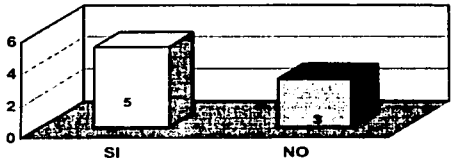
PREGUNTA # 5

¿La notación que utiliza le ha permitido diseñar fácilmente sistemas que cumplan satisfactoriamente con los requeriemientos de los usuarios?



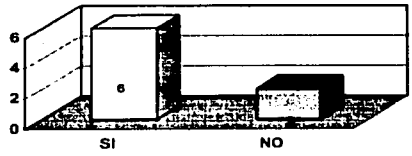
PREGUNTA # 6

¿El proceso de desarrollo y la notación utilizada le ha permitido incrementar la calidad del software desrrollado y la satisfacción de los usuarios?



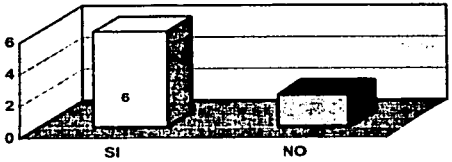
PREGUNTA # 7

¿Cree que el proceso de desarrollo que utiliza es el adecuado para verificar el cumplimiento de la calidad y los requerimientos de los usuarios?



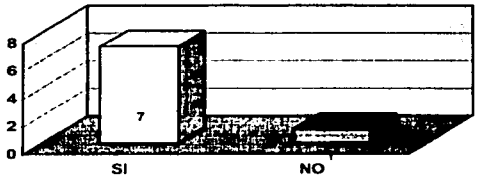
PREGUNTA # 8

¿La notación que utiliza le permite generar documentación que refleje real y eficientemente la estructura, comportamiento y el funcionamiento del sistema?



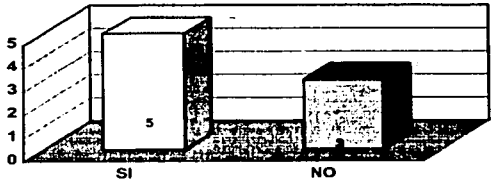
PREGUNTA # 9

¿La documentación que realiza durante el análisis y diseño, le proporciona una guía eficientemente de programación del sistema?



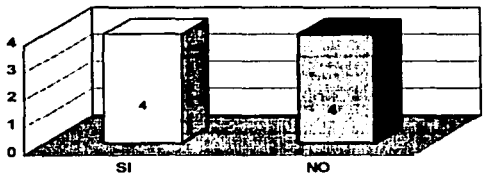
PREGUNTA # 10

¿El proceso utilizado le ha ayudado a realizar liberaciones continuas que le permitan verificar la calidad del sistema que se está desarrollando?



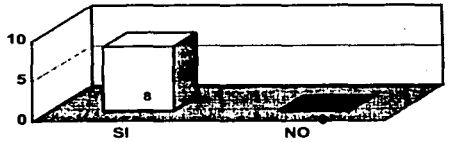
PREGUNTA # 11

¿El proceso de desarrollo que utiliza ha influido en la culminación de los sistemas en el tiempo y presupuestos estimados?



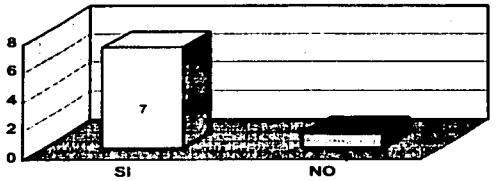
PREGUNTA # 12

¿Considera que el proceso de desarrollo utilizado le permite controlar las responsabilidades de cada uno de los integrantes del equipo de trabajo (analistas, diseñadores y desarrolladores)?



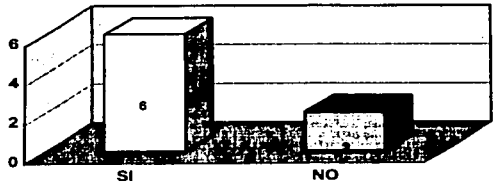
PREGUNTA # 14

¿Conoce el término de UML (Unified Modeling Language, Lenguaje de Modelado Unificado)?



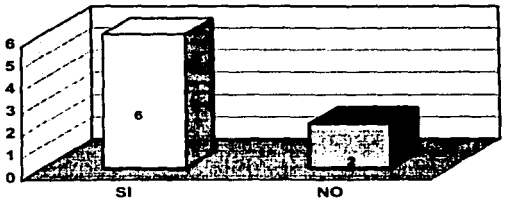
PREGUNTA # 15

¿Conoce el término de RUP (Rational Unified Process, Proceso Unificado Racional)?



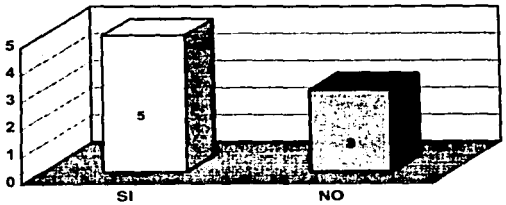
PREGUNTA # 16

¿Ha utilizado UML?



PREGUNTA # 18

¿Ha utilizado RUP?



4.14 CONCLUSION GENERAL SOBRE LA APLICACIÓN DEL CUESTIONARIO

Considerando la opinión y experiencia en el desarrollo de Sistemas Orientados a Objetos de las ocho personas entrevistadas, la notación más utilizada, por los beneficios que se obtienen es UML, principalmente porque no es una notación compleja se considera que es fácil de aprender, ya que proporciona modelos visuales y entendibles para todas las personas involucradas en el desarrollo de sistemas ya sean analistas, usuarios, líderes de proyecto, programadores, clientes o usuarios. UML es considerado como un estándar o notación única que se puede aplicar durante todo el proceso de desarrollo, generando asi documentación que refleja el funcionamiento real que tendrá el sistema a desarrollar. Por estos beneficios UML permite diseñar fácilmente Sistemas que cumplan satisfactoriamente los requerimientos de los usuarios, ya que los usuarios participan activamente en el diseño del sistema.

Un problema latente por el cual fracasan los sistemas hoy en dia, es debido a la falta de una adecuada documentación, ocasionada por la falta herramientas y estándares durante todo el proceso de software, por programaciones al vapor y entregas apresuradas, falta de conciencia sobre la importancia de esta actividad y sobre todo por la falta de una adecuada integración del equipo de desarrollo.

Es importante resaltar que sólo el 62% de las personas entrevistadas conocen el término de RUP, y por consecuencia una minoría de estas personas lo han utilizado, consideramos que hace falta más difusión acerca de los beneficios obtenidos al utilizar este proceso de desarrollo flexible, que cuida durante toda la fase de desarrollo la calidad del sistema. La opinión de las personas entrevistadas que utilizan RUP es principalmente que RUP permite incrementar la calidad del software desarrollado y la satisfacción de los usuarios y además este proceso les permite realizar liberaciones continuas a diferencia de las personas que no utilizan RUP, por ello podemos afirmar que RUP, UML y el equipo de desarrollo capacitado son factores de suma importancia que influyen en la mejora del tiempo y presupuesto estimado al desarrollar un sistema O.O. Entre los beneficios que han experimentado las personas entrevistadas que utilizan RUP son: una mejor administración del proyecto, permite detectar problemas a tiempo y corregirlos, un desarrollo más rápido, permite verificar e incrementar la calidad y los requerimientos de usuarios, además se logra definir tareas generales y controlar las responsabilidades de cada integrante del equipo de desarrollo.

4.15 APROBACIÓN DE LA HIPOTESIS

La hipótesis propuesta en está tesis es aprobada, ya que consideramos que por las respuestas obtenidas en el cuestionario aplicado, un 80% de los criterios son acordes y el 20% restante conlleva a la opinión de personas que no han experimentado realmente los beneficios de RUP y UML, y siguen posiblemente usando otros procesos de desarrollo y notaciones propias de la empresa en la cual laboran, consideramos además, que falta difusión sobre el tema.

La hipótesis propuesta es:

> *"Al desarrollar Sistemas Orientados a Objetos utilizando el Lenguaje Unificado de Modelado (UML), el Proceso Unificado Racional (RUP) y contando con un equipo de desarrollo capacitado y motivado,* se logrará obtener Sistemas Orientados a Objetos de calidad que cumplan con la funcionalidad que el usuario necesita, dentro del tiempo y presupuestos estimados."

Aunque conceptualmente es válida la idea hipotética, consideramos que su descripción queda mejor en los siguientes términos:

> ***"Al desarrollar Sistemas Orientados a Objetos utilizando el Lenguaje Unificado de Modelado (UML), el Proceso Unificado Racional (RUP) y contando con un equipo de desarrollo capacitado y motivado, se logrará obtener Sistemas Orientados a Objetos de calidad que cumplan con la funcionalidad que el usuario necesita, influyendo notablemente en la mejora del tiempo y en el presupuesto estimado."***

Es importante mencionar que el Lenguaje Unificado de Modelado y el Proceso de Unificado Racional son elementos flexibles y por tanto pueden ser utilizados y adaptados en el desarrollo de sistemas no Orientados a Objetos.

CAPÍTULO V

5. MARCO INSTRUMENTAL

A continuación se listan las actividades que realizamos con el objetivo de dar a conocer los resultados obtenidos en la investigación.

1. Caso práctico

Desarrollamos un ejercicio práctico usando UML y RUP, dicho ejercicio puede ser tomado como ejemplo para que se comprenda de una mejor manera el uso de esta notación y del proceso en el análisis y diseño de sistemas orientados a objetos. Este ejercicio puede ser observado en la sección de anexos.

2. Inclusión temática en el programa de estudios

Debido a que el desarrollo de sistemas de información orientados a objetos es un tema de actualidad, proponemos un temario (ver anexos) que podría ser integrado en el temario general de la materia de Análisis y Diseño de Sistemas impartido a los alumnos que estudian la Licenciatura en Informática en la Facultad de Contaduría y Administración. Dicha propuesta ha sido presentada a la coordinadora de la carrera de informática, la Ing. Mtra. Graciela Bribiesca Correa.

3. Publicación de artículos

Elaboramos un artículo (ver anexos) para ser publicado en alguna revista. Dicha propuesta ha sido entregada al Licenciado Gustavo Almaguer Pérez, Jefe de la Gaceta de la Facultad de Contaduría y Administración.

4. Página en Internet

Desarrollamos una página en Internet con la información necesaria para dar a conocer las generalidades de los puntos clave que deben tenerse en cuenta al desarrollar sistemas de información orientados a objetos. Un ejemplo de la página realizada puede ser visto en la sección de anexos de esta tesis.

5. Participación ante grupos (conferencias, seminarios, congresos)

Preparamos una serie de diapositivas para realizar pláticas y conferencias en diversas organizaciones y/o instituciones educativas para dar a conocer los beneficios de RUP y UML. Un ejemplo de este material puede ser observado en los anexos.

CONCLUSIÓN

CONCUCLUSIÓN

Los procesos y la tecnología de herramientas utilizadas para desarrollar sistemas orientados a objetos son un instrumento estratégico que permitirá a las organizaciones tener una ventaja competitiva, ya que podrán minimizar riesgos y aprovechar oportunidades, incrementando así la calidad de los sistemas desarrollados.

Herramientas como RUP y UML pueden ser adaptadas a cualquier tipo de sistema, ya que son muy flexibles y a través de ellas se aprovechan los beneficios que brinda el paradigma orientado a objetos y el principio de la ingeniería fundamental, es decir, la no reinvención, sino la adaptación de elementos existentes de calidad probada y la construcción de nuevos elementos sólo en caso de ser necesario, de acuerdo a los requerimientos y objetivos que se deseen cumplir.

Hoy en día, dentro de la comunidad informática de nuestro país, falta difusión sobre UML y RUP, además de modificar nuestra manera de pensar acerca de los diseños estructurales tradicionales al desarrollar sistemas, no obstante es la experiencia, la adopción gradual y la evolución del estándar lo que dará a conocer su verdadero potencial y permitirá a las organizaciones darse cuenta de sus beneficios.

Para abatir la crisis mundial del software, es imprescindible que las organizaciones e instituciones se aseguren de que los sistemas desarrollados cumplen con los requerimientos de desempeño y funcionalidad, así como con los estándares de documentación, facilidad de mantenimiento y sobre todo la calidad.

Al contar con RUP, UML y un equipo de desarrollo motivado y organizado se tendrán herramientas clave con los cuales una organización creará modelos completos que representen el dominio del problema y el sistema de software, evitando la complejidad de los mismos.

A través de nuestra experiencia nos hemos percatado de que un proceso de desarrollo debe evolucionar y ser más dinámico, no se puede realizar una análisis y una vez "terminado" seguir con el diseño y luego con el desarrollo, y así hasta llegar a las pruebas, implantación y mantenimiento, ya que todas las etapas terminan interactuando entre sí, y en ocasiones no importa cuánto análisis se realice, hay algunas cosas sobre el sistema que no se revelarán hasta la etapa de diseño e incluso algunas cosas no serán reveladas hasta escribir el código, o hasta que el programa está efectivamente ejecutándose, esto es uno de los aspectos que el Proceso Unificado Racional cubre perfectamente.

Por otra parte nos hemos dado cuenta de que cada vez más la notación de UML está siendo cada día más adoptada por las organizaciones debido a las grandes ventajas que ofrece, entre las que podemos mencionar:

- Permite mejorar la comunicación entre el equipo de desarrollo (analistas, diseñadores, desarrolladores, usuarios, expertos en el dominio), ya que se utiliza una terminología común, con un bajo grado de complejidad.
- Ayuda a pasar de una manera sencilla los modelos del sistema a la programación.
- Permite identificar los problemas de una manera sencilla, así como realizar los cambios y mejoras necesarios de los modelos.
- Se modela el sistema lo más apegado a la realidad.
- Permite identificar las partes reutilizables del sistema.
- Modela de una manera clara y sencilla la estructura y el comportamiento del sistema.

Por último es importante tener en cuenta los siguientes puntos clave que influirán en el tiempo y presupuesto estimados en el desarrollo de sistemas orientados a objetos exitosos:

- No se debe considerar la etapa de documentación como un proceso posterior a la entrega y funcionamiento del sistema.
- Mejorar la integración y coordinación de los equipos de desarrollo.
- Evitar los desarrollos y programación al vapor, con fechas de entrega apresuradas.
- No olvidar la importancia de las etapas de análisis y diseño.
- No descuidar la organización e integración del equipo de desarrollo.
- No limitar el tiempo para documentar, dedicar a ciertos integrantes del equipo de desarrollo exclusivamente para esta actividad.
- Planear y programar todas las actividades a realizar.
- Difundir una cultura de Ingeniería de Software entre los integrantes del equipo de desarrollo.
- Contar con herramientas CASE adecuadas para agilizar los procesos de desarrollo.

ANEXOS

CASO PRÁCTICO

CASO PRÁCTICO

VIDEOCENTRO

El propietario de un videocentro, que tiene más de 3000 videocasetes y 2000 DVD's, necesita llevar control de todas las transacciones que se llevan a cabo en dicho videocentro, por lo que ha pedido que se le desarrolle un sistema que registre automáticamente todas sus operaciones, es decir, que le permita conocer en cualquier momento el número de cintas disponibles, cuáles y cuántas películas tiene cada socio del club, las fechas de préstamo y de devolución, determinar si se les aplicará alguna multa, el motivo y monto de ésta, entre otra información, la cual se detallará más adelante. También se describirán algunas políticas de negocio.

Cada uno de las cintas tiene un número que los identifica de manera única y pueden estar en formato de videocasete o DVD. Para cada película se necesita conocer su título, su categoría (por ejemplo: comedia, suspenso, drama, acción, etc). Se tienen muchas copias de la mayoría de las películas, a cada película se le da un identificador específico y así se puede saber qué película tiene cada una de las cintas. Siempre se tienen por lo menos una cinta de cada película.
Las cintas no son muy largas, así que no se tienen películas que requieran de múltiples cintas.
Frecuentemente los socios preguntan por los protagonistas de determinada película, así que se necesita llevar el registro de los actores que aparecen en cada película. No todas las películas tienen actores, a los socios y clientes les gustaría conocer el nombre real del actor y la fecha de la película.

El videocentro solamente renta películas a la gente que es socia del videocentro, para que una persona pueda pertenecer al videocentro deben tener buen crédito, por cada socio del club se debe registrar su nombre, apellidos, su número de teléfono, su dirección; a cada socio se le asigna un número de membresía. El videocentro tiene dos tipos de clientes: habituales y esporádicos.
Un socio puede alquilar varios videos en un mismo punto de tiempo, se necesita llevar el registro de qué videos ha rentado cada socio, por lo cual se debe registrar el número de membresía del socio, las películas rentadas, la fecha en que se realiza el préstamo y la fecha de devolución, así como el número de días que tendrá la película.
En el videocentro también se venden películas a cualquier persona, no es necesario que sea socio, sin embargo, también se requiere registrar sus datos personales.

El videocentro es atendido por tres empleados, quienes deben poder hacer las siguientes operaciones:

- Obtener un listado de las cintas disponibles de acuerdo a su tipo.
- Preguntar por el costo de las rentas, de acuerdo al tipo de película rentada.
- Preguntar por el descuento ofrecido a los socios habituales.
- Preguntar por el monto total que pagará el socio de acuerdo al número de películas rentadas y a su tipo.
- Registrar la renta del socio especificando las cintas rentadas y el número de membresía del socio.
- Registra la devolución de películas.
- Cancelar las rentas especificando el número de membresía del socio.
- Registrar la venta de películas a los clientes.

El administrador del videocentro puede usar el sistema para:

- Cambiar el costo de la renta de las películas de acuerdo a su tipo.
- Cambiar el valor del descuento ofrecido a los socios habituales

CASO DE USO: ***Obtener datos de película***

No. CASO DE USO: 1

ACTORES: Empleado

PRECONDICIONES: Que esté dado de alta el catálogo de Película y de Cinta.
Que el empleado esté dado de alta como usuario para que pueda ingresar al sistema.

BREVE DESCRIPCIÓN: Este caso inicia cuando el empleado solicita al sistema datos de una película.

FLUJO PRINCIPAL: El empleado ingresa al sistema para consultar los datos de una o varias películas. Actividades a realizar:

SUB-FLUJO	ACTIVIDADES	
A-1	BUSCAR	Se buscan los datos de películas para su renta, venta o reservación.
A-2	MOSTRAR	Se muestran los resultados de la búsqueda en pantalla.
A-3	IMPRIMIR	Se imprime el listado con los datos de películas.

FLUJOS ALTERNOS:

A-1.BUSCAR: El sistema solicita al empleado que ingrese algún criterio de búsqueda que le permita encontrar los datos de una película. Los criterios pueden ser:
Por código de barras: El empleado ingresa el código de barras de la película, dicho código puede ser obtenido de las cajas de películas que se encuentran en exhibición.
Por categoría: El sistema permite elegir de un listado la categoría de la película.
Por título: El sistema permite que el empleado introduzca el título de la película. En caso de que se desconozca el título completo, se puede introducir alguna palabra o texto que coincida con el título, el sistema buscará todas las películas que tengan dicha palabra o texto.
Por fecha: El sistema solicita un rango de fechas, se debe introducir una fecha de inicio y una fecha de fin, o bien se podrá proporcionar una fecha en específico (E1).

A-2.MOSTRAR: Una vez elegido el criterio, el sistema realiza la búsqueda a partir de los datos proporcionados y muestra los resultados de la búsqueda en pantalla:
Por clave de película: El sistema muestra la siguiente información: Número de cinta, título de la película, formato (videocasete o DVD), tipo de servicio (venta, renta o reservación), status (Alta, Baja, Disponible, Agotada).
Por categoría: En caso de que el empleado haya seleccionado esta opción, el sistema debe mostrar un listado de todas las películas de la categoría seleccionada con los siguientes datos: Número de cinta, título de la película, formato (videocasete o DVD), tipo de servicio (venta, renta o reservación), status (Alta, Baja, Disponible, Agotada).
Por título: Si el empleado eligió esta opción, el sistema debe mostrar un listado con: Número de cinta, título de la película, formato (videocasete o DVD), tipo de servicio (venta, renta o reservación), status (Alta, Baja, Disponible, Agotada); de todas las cintas encontradas con el título introducido. Si se tecleó una palabra o

texto de coincidencia, el sistema mostrará todos los títulos de películas que hayan coincidido con éstos (E2).
Por fecha: Con esta opción, el sistema debe mostrar un listado de todas las películas que tengan la fecha o entren en el rango especificado. El listado contendrá: Número de cinta, título de la película, fecha, formato (videocasete o DVD), tipo de servicio (venta, renta o reservación), status (Alta, Baja, Disponible, Agotada)

A-3.IMPRIMIR: El sistema permite imprimir el listado de películas que se obtuvo.

EXCEPCIONES:

E1.- En caso de que se ingrese un dato diferente al de una fecha, el sistema mostrará un mensaje de error que especifique que el formato de la fecha no es correcto, por lo que el usuario deberá ingresar una fecha en el formato válido o el caso de uso termina.

E2.- En caso de que el sistema no haya encontrado ningún título de película que coincida con el título, palabra o texto introducido, mostrará un mensaje que indique la no existencia de películas con ese título. El empleado podrá intentar nuevamente su búsqueda.

POST-CONDICIONES:

El empleado puede determinar si disponen o no de cierta película para llevar a cabo una renta, venta o reservación.
El empleado conoce la ubicación de las cintas a partir del número de cinta.

CASO DE USO: ***Rentar película***

No. CASO DE USO: 2

ACTORES: Empleado

PRECONDICIONES: Que esté dado de alta el catálogo de Película y de Cinta. Que esté dado de alta el socio. Que esté dado de alta el empleado como usuario para que pueda ingresar al sistema.

BREVE DESCRIPCIÓN: Este caso de uso inicia cuando el empleado ingresa al sistema para registrar la renta de una o varias películas.

FLUJO PRINCIPAL: El empleado ingresa al sistema para registrar la renta de una o varias películas. Actividades que se pueden realizar:

SUB-FLUJO	ACTIVIDADES	
A-1	CONSULTAR	Se consultan las películas disponibles para renta.
A-2	REGISTRAR	Se registra la renta de un socio.
A-3	MODIFICAR	Se modifican los datos del registro de la renta.
A-4	ELIMINAR	Se elimina una renta.
A-5	IMPRIMIR	Se imprime el comprobante de la renta.

FLUJOS ALTERNOS:

A-1. El empleado realiza una consulta de películas para obtener la ubicación de la cinta que desea rentar el socio (ver caso de uso "Obtener datos de película") y para determinar si existen cintas disponibles para su renta (E1). Posteriormente el empleado busca en los anaqueles, por número de cinta, la cinta que se rentarán al socio.

A-2. El empleado introduce el número de membresía del socio al sistema (E2). Una vez ingresado dicho número, el sistema debe verificar que el socio no tenga adeudos de multas, ya sea por daño, no rebobinado o extravío de películas (E3). El sistema verifica que el socio no se encuentre en el límite de sus préstamos (E4).
En caso de que el sistema no haya mostrado ningún mensaje advirtiendo algún problema con el registro de la renta, el empleado ingresará uno el número de cinta, dicho número puede leerse directamente del estuche de cada cinta. Para cada cinta el sistema verifica su status (E5).
Para cada cinta prestada el sistema deberá calcular la fecha de devolución a partir de la fecha de inicio del préstamo, ésta última fecha será la del sistema. El sistema también calculará el número de días de préstamo. En caso de que el socio sea habitual, el sistema calculará el descuento correspondiente que se le hará (ver caso de uso "Calcular descuento de socios habituales"). El empleado puede ingresar hacer algunas observaciones de la renta de la película. Si el socio renta más de una película, el sistema debe calcular el monto total que deberá pagar el socio, es decir, se debe sumar el costo de renta de cada una de las cintas rentadas.
Una vez que se tiene el registro de todas las películas rentadas y monto total, el sistema registrará la renta de las películas.

A-3. El sistema sólo permitirá modificar el número de socio que rente las películas (E6) y las observaciones que se anotan para la renta.
En caso de que el socio ya no desee rentar las películas, el sistema permitirá eliminar el registro correspondiente de la renta.

A-4.-El sistema permite imprimir el comprobante de la renta de la película, el cual es entregado al socio.

EXCEPCIONES:

E1.- En caso de que no existan cintas disponibles el socio o cliente tendrá que elegir otra película o el caso de uso termina.

E2.- En caso de que el número de membresía sea incorrecto el sistema mostrará un mensaje de error, se deberá ingresar un número de membresía correcto o el caso de uso termina.

E3.- En caso de que el socio tenga una o varias multas sin pagar, deberá liquidarlas en ese momento para que se le puedan prestar más películas o el caso de uso termina.

E4.- En caso de que el socio se encuentre en el límite de sus préstamos deberá devolver primero las películas que tiene prestadas para poder hacerle más préstamos o el caso de uso termina.

E5.- Si el sistema detecta un status de "Dañada", "Rentada" o "Baja" para alguna de las cintas, ésta no podrá ser prestada al socio, deberá buscarse otra cinta de la misma película con status de "Disponible" y que no esté dañada.

E6.- El número de socio sólo podrá modificarse por el de otro socio del videocentro que no tenga adeudos de multas y que no tenga excedido su límite de préstamos.

POST-CONDICIONES:

La renta del socio queda registrada.

CASO DE USO: ***Registrar devolución de película***

No. CASO DE USO: 3

ACTORES: Empleado

PRECONDICIONES: Que el empleado esté dado de alta como usuario para que pueda ingresar al sistema.

BREVE DESCRIPCIÓN: El caso de uso inicia cuando el empleado entra al sistema para registrar la devolución de una o varias películas.

FLUJO PRINCIPAL: El empleado ingresa al sistema para registrar la devolución de películas del socio. Actividades que se pueden realizar:

SUB-FLUJO	ACTIVIDADES	
A-1	VERIFICAR FECHA DE DEVOLUCIÓN	Se verifica que no se haya excedido la fecha de devolución de las películas.
A-2	VERIFICAR ESTADO FISICO DE PELÍCULAS	El empleado verifica el estado físico de las películas devueltas.
A-2	REGISTRAR DEVOLUCIÓN	Se registra la devolución de las películas.

FLUJOS ALTERNOS:

A-1.- El empleado introduce el número de membresía del socio al sistema (E1). Posteriormente el sistema verifica que la fecha de entrega de la película (fecha actual) no sea mayor que la fecha de devolución (E2).

A-2.- El empleado debe verificar el estado físico en el que el socio entrega las películas (E3).

A-3.- Una vez verificadas las condiciones de entrega de la película y asignado el monto de las multas, en caso de que existan, el empleado ingresa el código de barras de las cintas. A cada cinta el sistema le asignará un status de "Devuelta", quedando así descargada la película al socio.

EXCEPCIONES:

E1.- En caso de que el número de membresía sea incorrecto el sistema mostrará un mensaje de error, se deberá ingresar un número de membresía correcto o el caso de uso termina.

E2.- En caso de que el sistema detecte que la fecha actual es mayor a la fecha de devolución, el sistema asignará una multa. (ver caso de uso "Asignar multa").

E3.- En caso de que el empleado detecte alguna anomalía se asignarán las multas correspondientes (ver caso de uso "Asignar multa") al socio.

POST-CONDICIONES:

La devolución del socio queda registrada.

CASO DE USO: ***Vender película***

No. CASO DE USO: 4

ACTORES: Empleado

PRECONDICIONES: Que esté dado de alta el catálogo de Película y de Cinta. Que el empleado esté dado de alta como usuario para que pueda ingresar al sistema.

BREVE DESCRIPCIÓN: Este caso de uso inicia cuando el empleado ingresa al sistema para registrar la venta de una o varias películas.

FLUJO PRINCIPAL: El empleado ingresa al sistema para registrar la venta de una o varias películas. Actividades que se pueden realizar:

SUB-FLUJO	ACTIVIDADES	
A-1	CONSULTAR	Se consultan las películas disponibles para venta.
A-2	REGISTRAR	Se registra la venta de una o varias películas al cliente.
A-3	MODIFICAR	Permite modificar los datos del registro de la venta.
A-4	ELIMINAR	Se elimina una venta.
A-5	IMPRIMIR	Se imprime el comprobante de la venta.

FLUJOS ALTERNOS:

A-1.- El empleado realiza una consulta de películas para obtener la ubicación de la(s) cinta(s) que desea comprar el socio (ver caso de uso "Obtener datos de película") o para determinar si existen cintas disponibles para su venta (E1). El sistema el deberá verificar el status de la cinta a vender (E2). Posteriormente el empleado busca, con el número de cinta, la o las películas que se venderán al cliente.

A-2.- El empleado ingresa el código de barras para registrar la venta. El sistema calcula el pago total que debe hacer el cliente, según el número de películas vendidas y su precio.
El sistema debe almacenar los datos del cliente como: nombre, apellido paterno, apellido materno, su RFC, su número telefónico y su dirección. En caso de que el cliente sea un socio ya no se registran estos datos, se ingresará su número de socio.
Una vez que se tienen todos los datos necesarios, el sistema registrará la venta de la película.

A-3.- El sistema sólo permitirá modificar los datos personales del cliente.

A-4.-En caso de que el cliente ya no desee comprar las películas, el sistema permitirá eliminar el registro correspondiente que se había ingresado al sistema (E3).

A-5. El sistema permite imprimir el comprobante de la venta de la película, el cual es entregado al cliente.

EXCEPCIONES:

E1.- En caso de que no existan cintas disponibles el cliente tendrá que elegir otra película o el caso de uso termina.

E2.- Si el sistema detecta un status de "Dañada" o "Baja" para alguna de las cintas, éstas no se podrán vender a los clientes. Deberá buscarse otra cinta de la misma película con status de "Alta" que no esté dañada.

E3.- Una vez depositado el dinero a la caja ya no se podrá cancelar la venta.

POST-CONDICIONES:

La venta de la o las películas quedan registradas.

CASO DE USO: ***Asignar multa***

No. CASO DE USO: 5

ACTORES:

PRECONDICIONES: Que el empleado esté dado de alta como usuario para que pueda ingresar al sistema.

BREVE DESCRIPCIÓN: El caso de uso inicia cuando el empleado registra las multas asignadas al socio.

FLUJO PRINCIPAL: El empleado registra las multas que deberán asignarse al socio. Actividades que se pueden realizar:

SUB-FLUJO	ACTIVIDADES	
A-1	ASIGNAR MULTA	El empleado registra el estado físico en el que el socio entrega las películas y el monto de la multa, correspondiente por cada concepto.
A-2	CALCULAR MONTO MULTA	El sistema calcula el monto total de las multas.

FLUJOS ALTERNOS:

A-1.- El empleado asigna las multas por alguno de los siguientes conceptos, en caso que el socio :
Exceder la fecha de devolución:
No rebobinado de la película: El empleado ingresará la cantidad que será cobrada al socio.
Película dañada: El empleado consultará con el administrador el pago que debe realizar el socio de acuerdo al grado de daño en el que se entregue la película, una vez determinada dicha cantidad, el empleado le ingresará en el sistema.
Una vez ingresados los montos de las multas por daño o no rebobinado, el sistema calculará el monto total de la multa que debe pagar el socio. No siempre el socio tendrá multas, por lo que éstos campos pueden quedar vacíos , lo cual quiere decir que no existen multas para dicho socio.

A-2 Una vez ingresadas las multas del socio, el sistema realiza el cálculo la suma de todas las multas asignadas y registra el monto total a pagar por concepto de multas.

EXCEPCIONES:

POST-CONDICIONES:

La multas del socio quedan registradas.

CASO DE USO: ***Cambiar precio de películas o costo de rentas***

No. CASO DE USO: 6

ACTORES:

PRECONDICIONES: Que esté dado de alta el catálogo de Película.

BREVE DESCRIPCIÓN: Este caso de uso inicia cuando el administrador ingresa al sistema para cambiar el precio de las películas que se encuentran en venta la venta o bien el cambio del costo de las rentas de películas.

FLUJO PRINCIPAL: El administrador ingresa al sistema para realizar alguna modificación, ya sea en el precio de algunas películas o el costo de las rentas. Actividades que se pueden realizar:

SUB-FLUJO	ACTIVIDADES	
A-1	MODIFICAR	Se consultan las películas disponibles para su renta o compra.
A-2	IMPRIMIR	Se imprime el listado de películas disponibles.

FLUJOS ALTERNOS:

A-1. El empleado ingresa el número de cinta para registrar la venta, el sistema el deberá verificar el status de la cinta a vender (E1). El sistema calcula el pago total que debe hacer el cliente, según el número de películas vendidas y su precio.
El sistema debe almacenar los datos del cliente como: nombre, apellido paterno, apellido materno, su RFC, su número telefónico y su dirección.
Una vez que se tienen todos los datos necesarios, el sistema registrará la venta de la película.

A-2. El sistema permite imprimir el comprobante de la venta de la película, el cual es entregado al cliente.

EXCEPCIONES:

E1.- Si el sistema detecta un status de "Dañada" o "Baja" para alguna de las cintas, éstas no se podrán vender a los clientes. Deberá buscarse otra cinta de la misma película con status de "Alta" que no esté dañada.

POST-CONDICIONES:

La venta de la o las películas quedan registradas.

CASO DE USO: ***Administrar Usuarios***

No. CASO DE USO: 7

ACTORES: Administrador

PRECONDICIONES: Ninguna

BREVE DESCRIPCIÓN: Este caso de uso inicia cuando el administrador ingresa al sistema para dar de alta a un usuario del sistema.

FLUJO PRINCIPAL: El administrador ingresa al sistema para dar de alta a un empleado del videocentro como usuario del sistema. Actividades que se pueden realizar:

SUB-FLUJO	ACTIVIDADES	
A-1	REGISTRAR	Se registra a un empleado como usuario.
A-2	CONSULTAR	Se consultan los datos de los usuarios.
A-3	MODIFICAR	Se actualizan los datos de los usuarios.
A-4	ELIMINAR	Se da de baja a un usuario.

FLUJOS ALTERNOS:

A-1. El administrador introduce el nombre completo del usuario y el tipo de usuario, una vez ingresado estos datos, el sistema le asigna una clave consecutiva de usuario, el empleado podrá ingresar su login y su password personalmente (E1) para que pueda acceder al sistema. También se almacenan otros datos del empleado como nombre, dirección, número de teléfono y sus privilegios según el tipo de usuario.

A-2. El administrador puede realizar consultas acerca de los usuarios del sistema, realizando búsquedas por el nombre de usuario(E2) o bien por el tipo de usuario (E3).

A-3. El administrador ingresa el número consecutivo de usuario(E4) y puede cambiar los datos del usuario seleccionado, excepto la clave consecutiva que le es asignada por el sistema. El sistema almacena los cambios que el administrador realice en los datos del usuario.

A-4. Para dar de baja a un usuario del sistema el administrador ingresa la clave de usuario (E3) o bien el nombre completo del usuario(E2). El sistema almacena la baja de usuario del sistema.

EXCEPCIONES:

E1.- En caso de que el login o el password ya existan en el sistema, éste mostrará un mensaje de error que advierta esta situación. El empleado deberá ingresar otro login o pasword, según sea el caso.

E2.- Si el sistema no encuentra al usuario, el sistema desplegará un mensaje indicando que el usuario no existe, y preguntará si se desea continuar con otra búsqueda o no.

E3.- Si no existen usuarios por el tipo de usuario seleccionado el sistema desplegará un mensaje indicándolo y preguntará si se desea realizar otra búsqueda.

E3.- En caso de que no exista esa clave de usuario el sistema lo indicará a través de un mensaje, el sistema debe permitir ingresar nuevamente la clave.

POST-CONDICIONES:

La persona queda registrada como usuario.

CASO DE USO: ***Administrar Catálogos***

No. CASO DE USO: 8

ACTORES: Administrador

PRECONDICIONES: Ninguna

BREVE DESCRIPCIÓN: Este caso de uso inicia cuando el administrador ingresa al sistema para dar de alta los catálogos del sistema o actualizarlos.

FLUJO PRINCIPAL: El administrador ingresa al sistema para agregar registros a los catálogos o bien para actualizarlos. Actividades que se pueden realizar:

SUB-FLUJO	ACTIVIDADES	
A-1	REGISTRAR	Se registran los datos en los catálogos.
A-2	ACTUALIZAR	Se pueden modificar los datos de los catálogos.

FLUJOS ALTERNOS:

A-1. El administrador ingresa al sistema para registrar los datos en los siguientes catálogos: Protagonista, Película, Cinta, Descuento, Socios, Usuario.

A-2. El administrador ingresa al sistema para modificar los datos de los catálogos. Con esta opción el administrador podrá modificar el costo de las rentas, el precio de las películas, los descuentos otorgados a los socios.

EXCEPCIONES:

POST-CONDICIONES:

Se registran y modifican los datos de los catálogos.

CASO DE USO: ***Calcular descuento de socios habituales***

No. CASO DE USO: 9

ACTORES: Empleado

PRECONDICIONES: Que este dado de alta el catálogo de descuentos.

BREVE DESCRIPCIÓN: Esté caso de uso inicia cuando el sistema verifica el número de rentas del mes anterior al de la renta que se está registrando para calcular el descuento del cliente.

FLUJO PRINCIPAL: El sistema consulta cuántas rentas ha realizado el socio el mes anterior al de la renta que se está registrando, de acuerdo a la cantidad de rentas busca el descuento que se aplicará para el socio.

SUB-FLUJO	ACTIVIDADES	
A-1	CALCULAR	Permite determinar el descuento a aplicar.

FLUJOS ALTERNOS:

A-1. El sistema identifica que el socio tiene por lo menos más de una renta(E1) el mes anterior al de la fecha de la renta que se está registrando, el sistema calcula el número de rentas y de acuerdo a éste número calculará el descuento correspondiente. El sistema tendrá un tabulador con número de rentas y el descuento que corresponde a dicho número de rentas realizadas.

EXCEPCIONES:

E1.- Si el sistema verifica que el socio no tiene ninguna renta realizada el mes pasado se cancela el cálculo del descuento y el sistema lo indicara a través de un mensaje.

POST-CONDICIONES:

El descuento por socio habitual queda registrado para que posteriormente se calcule el monto final de su renta.

CASO DE USO: ***Reservación de películas***

No. CASO DE USO: 10

ACTORES: Empleado

PRECONDICIONES: Que este dado de alta el catálogo de descuentos.

BREVE DESCRIPCIÓN: Este caso de uso inicia cuando el empleado reserva de una a tres películas para un socio a través de una llamada telefónica.

FLUJO PRINCIPAL: El empleado recibe una llamada del socio, quien desea reservar películas. El empleado entra al sistema como usuario para registrar una o tres reservaciones de películas para un socio. Dicha reservación es valida hasta cuatro horas antes de su préstamo, el cual se realizará personalmente en la tienda.

Actividades que se pueden realizar:

SUB-FLUJO	ACTIVIDADES	
A-1	RESERVAR	Se reservan las rentas de una o hasta tres películas.
A-2	CANCELAR	Se eliminan las reservaciones de películas de los socios dentro de las tres horas, después de haber realizado la(s) reservación(es).
A-3	ELIMINAR	Se dan de baja las reservaciones.

FLUJOS ALTERNOS:

A-1. El empleado ingresa el número de membresía del socio (E1). El sistema debe verificar que el socio no tenga adeudos de multas (E2). El sistema verifica que el socio no se encuentre en el límite de sus préstamos (E3).

Una vez que se verifica que el socio no tiene adeudos y que no excede el número de sus préstamos permitidos, el socio puede proporcionar el título de la película que desea, el empleado obtiene los datos de la película (ver caso de uso "Obtener datos de película") para conocer el status de las cintas a reservar (E4). Para hacer la reservación de las cintas (E4), el empleado cambiará el status de la(s) cinta(s) a "Reservada".

Para cada película reservada, máximo 3, el sistema registrará la fecha y la hora en que se realizó la reservación.

A-2. Cada 15 minutos el sistema verifica todas las reservaciones de películas, y cancela aquellas reservaciones que no se hallan convertido en prestamos, si ya han transcurrido 3 hrs. desde su reservación.

A-3. El empleado da de baja aquellas reservaciones de los socios que las soliciten ya sea personal ó vía telefónica. Para realizar dicha baja la recepcionista realiza una búsqueda por clave de socio(E4), de esta manera quedan las cintas disponibles para otras reservaciones o prestamos a otros socios.

EXCEPCIONES:

E1.- En caso de que el número de membresía sea incorrecto el sistema mostrará un mensaje de error, se deberá ingresar un número de membresía correcto o el caso de uso termina.

E2.- En caso de que el socio tenga una o varias multas sin pagar, deberá liquidarlas en ese momento para que se le puedan prestar más películas o el caso de uso termina.

E3.- En caso de que el socio se encuentre en el límite de sus préstamos deberá devolver primero las películas que tiene prestadas para poder hacerle más préstamos o el caso de uso termina.

E4.- Si el sistema detecta un status de "Dañada", "Rentada" o "Baja" en alguna de las cintas, ésta no podrá ser reservada por el socio, deberá buscarse otra cinta de la misma película que no tenga ningún status de los antes mencionados.

E5.- Si se reservan más de 3 películas el sistema desplegará un mensaje indicando que el máximo de películas para reservar son 3"

E4.- Si no se encuentra la clave del socio, el sistema desplegará un mensaje indicando que no existen reservaciones para la clave del cliente que se solicito.

POST-CONDICIONES:

La reservación de la(s) película(s) queda(n) registrada(s).

DIAGRAMA DE CASOS DE USO

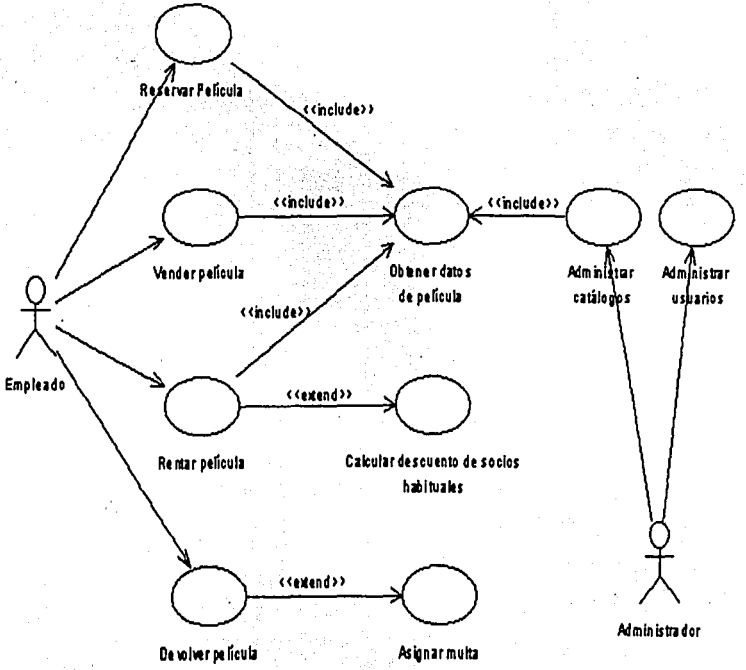


DIAGRAMA DE CLASES (CONCEPTUAL)

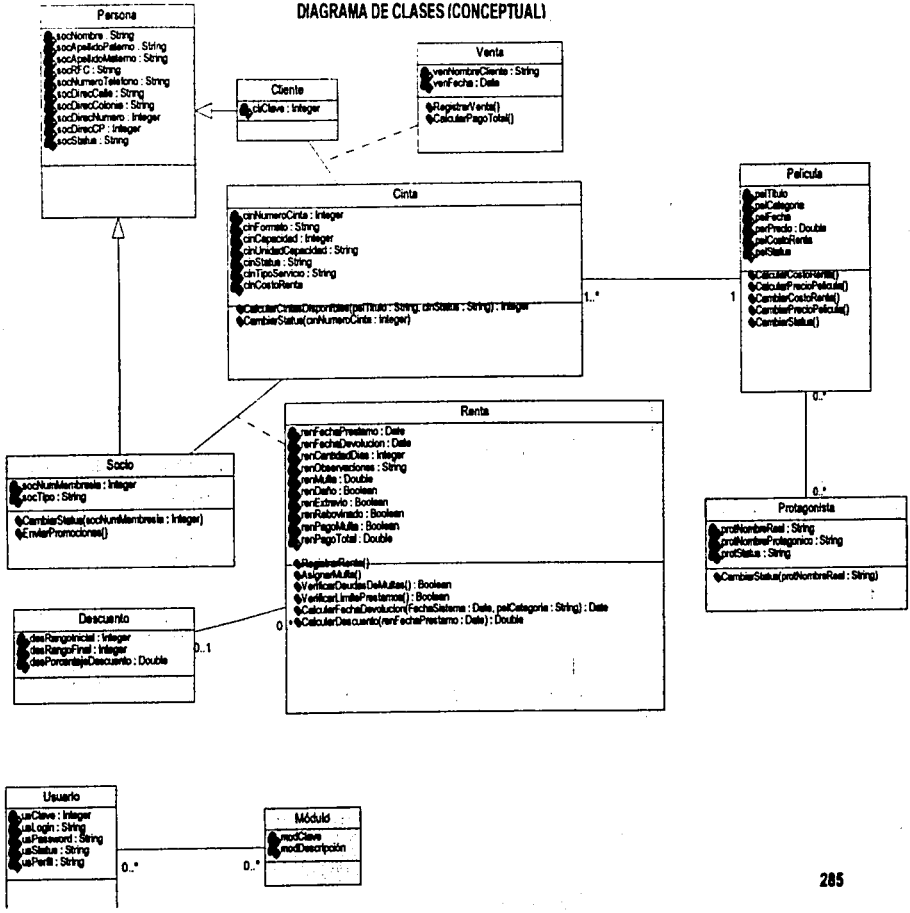


DIAGRAMA DE CLASES (ESPECÍFICO)

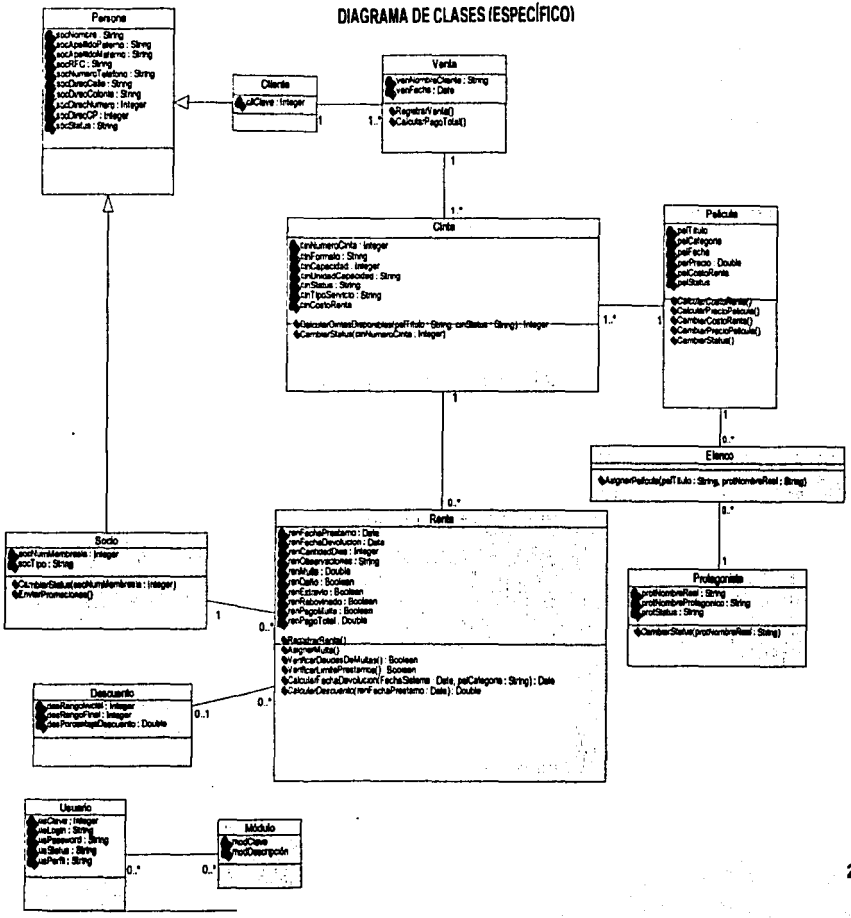


DIAGRAMA DE ACTIVIDAD

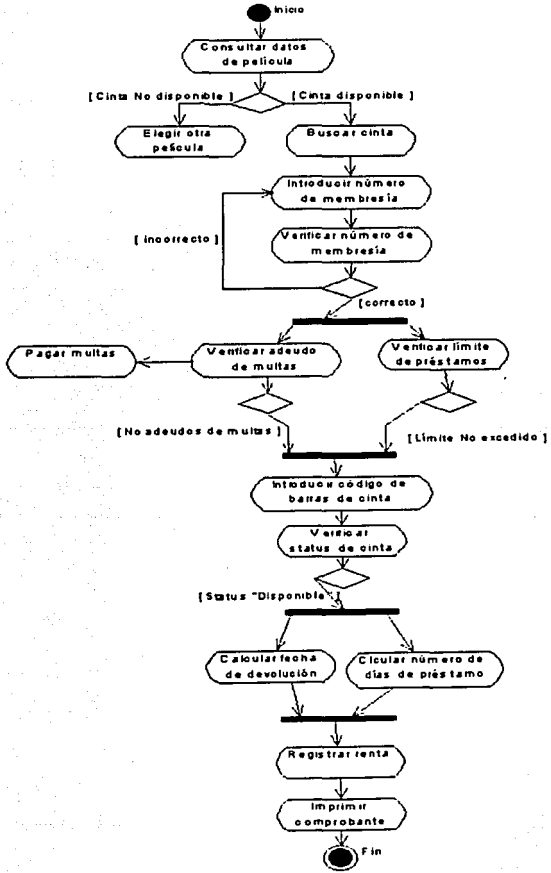


DIAGRAMA DE SECUENCIA

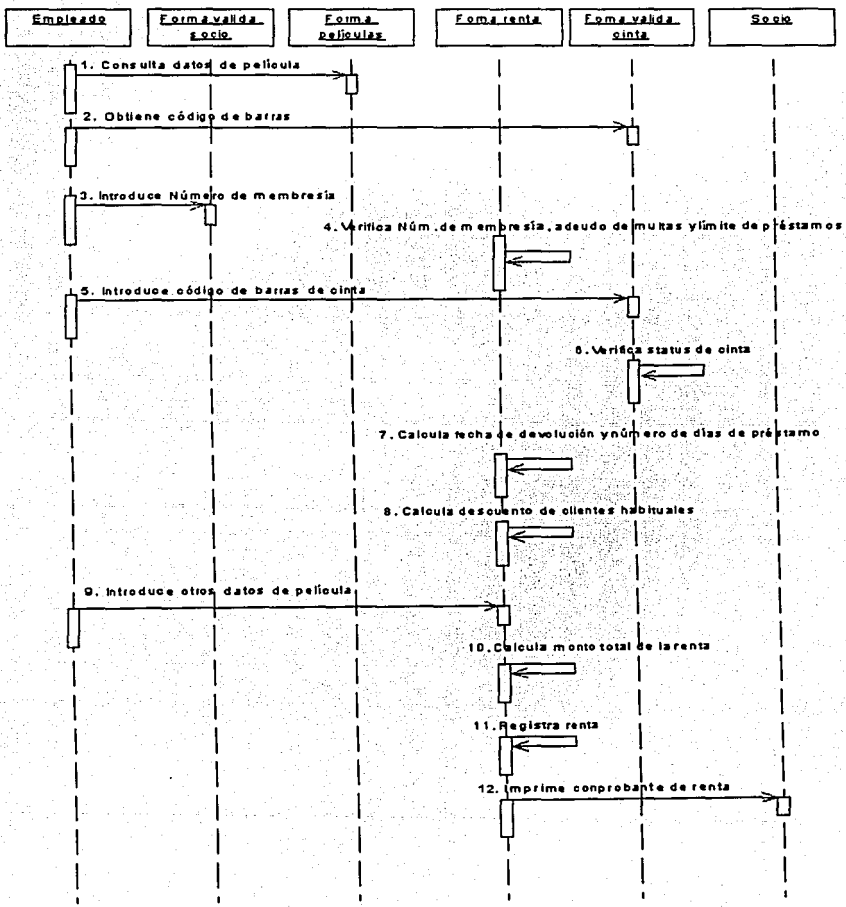


DIAGRAMA DE COLABORACIÓN

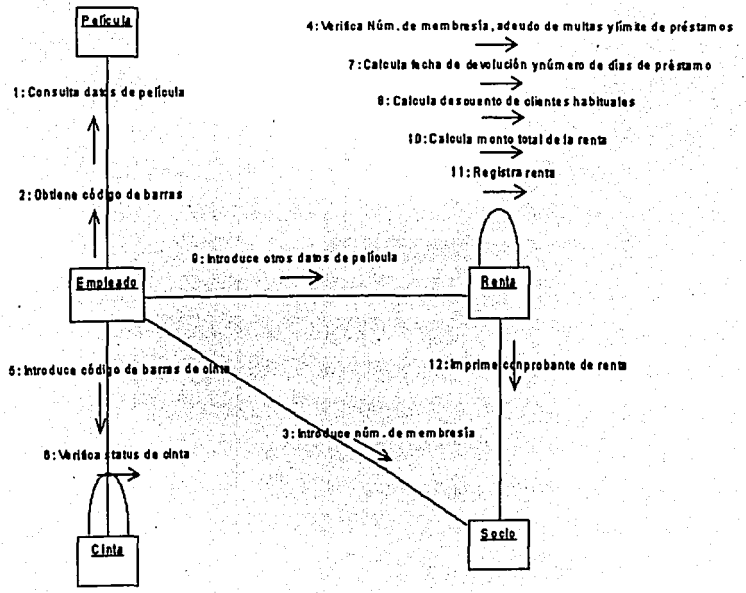


DIAGRAMA DE ESTADOS

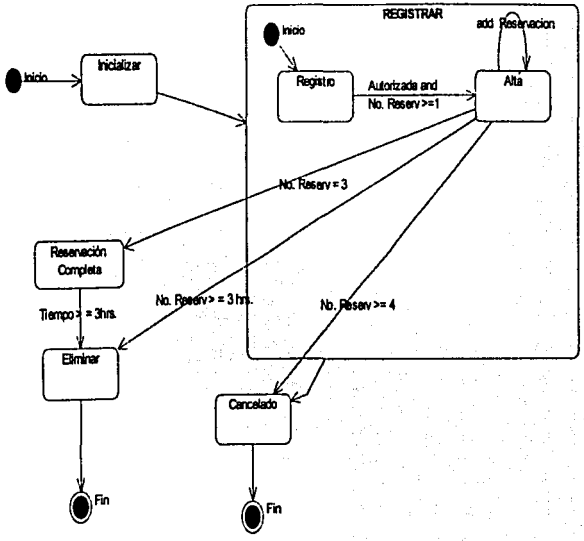


DIAGRAMA DE DESPLIEGUE

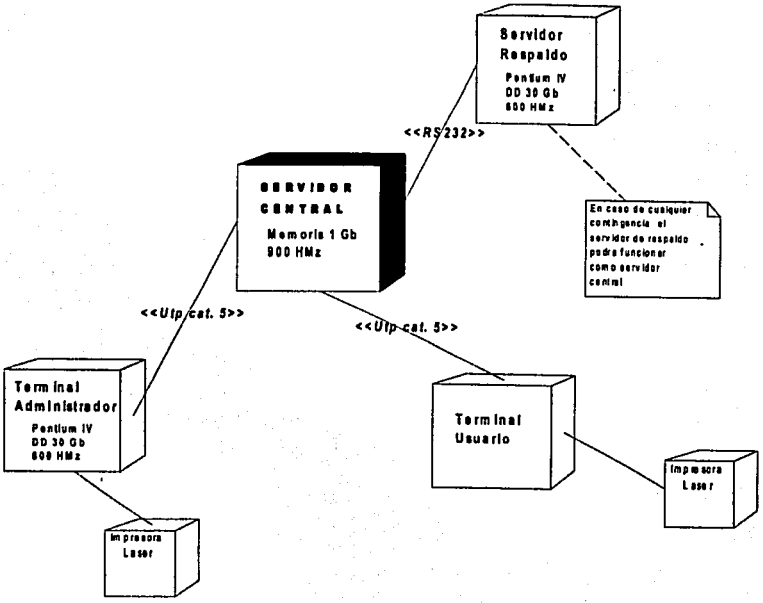
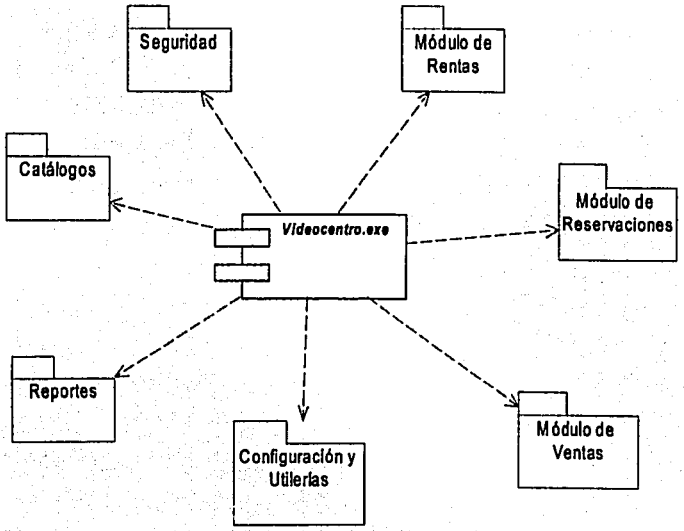
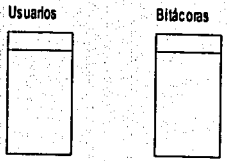


DIAGRAMA DE COMPONENTES

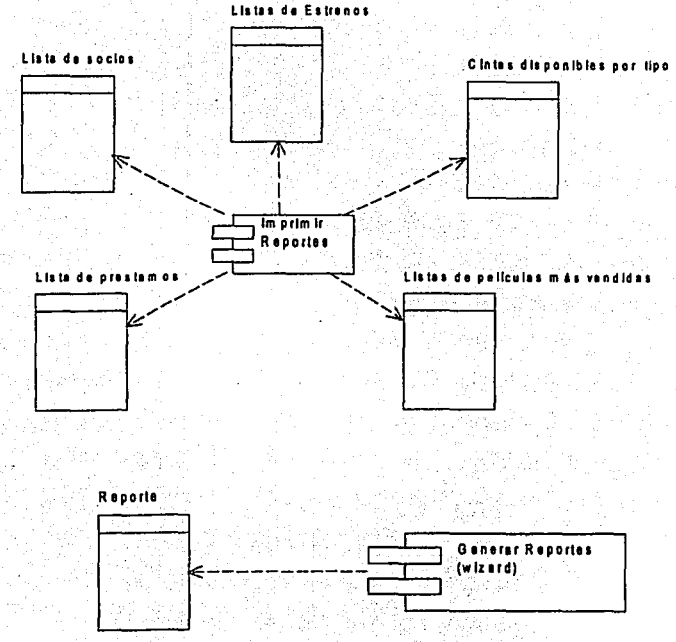


292

Seguridad



Reportes


Listas de Estrenos
Lista de socios
Cintas disponibles por tipo
Imprimir Reportes
Lista de prestamos
Listas de peliculas más vendidas
Reporte
Generar Reportes (wizard)

TEMARIO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE COPNTADURÍA Y ADMINISTRACIÓN

MTRA. GRACIELA BRIBIESCA
Jefa de la División de Informática Avanzada
PRESENTE

Debido al incremento del conocimiento tecnológico que se va dando día con día, nos permitimos hacer una propuesta para agregar al temario de la materia de análisis y diseño de sistemas un tema de actualidad e interés para los futuros profesionales de la materia, dicha propuesta es dada como resultado de la tesis profesional denominada "RUP y UML como elementos clave en el desarrollo de sistemas Orientados a Objetos".

Anexamos el temario propuesto para que sea evaluado por usted. Esperando que dicha propuesta sea de utilidad, estamos a sus órdenes para cualquier aclaración; y si lo desea, para impartir alguna plática referente a este tema. Todo lo anterior lo hacemos por sugerencia y recomendación de nuestro asesor, el Dr. Ricardo rivera Soler.

Aprovechamos la ocasión para enviarle nuestro más cordial saludo.

ATENTAMENTE

Claudia Elvira Soriano Monzalvo
Tels. 2617 07 51 particular
52 37 90 00 Ext. 1277 oficina

Silvia Jiménez Luna
Tels. 57 87 68 29 particular
56 22 36 04 oficina

INFORMÁTICA III
(ANÁLISIS Y DISEÑO DE SISTEMAS)

CLAVE:
PLAN:
LICENCIATURA (SEMESTRE): INFORMÁTICA (III)
DEPTO. ACADÉMICO INFORMÁTICA AVANZADA
ÁREA: ANÁLISIS Y DISEÑO DE SISTEMAS
REQUISITOS: INFORMÁTICA II

CRÉDITOS: 8
HORAS POR CLASE: 2
CLASES POR SEMANA: 2
HORAS POR SEMESTRE: 80

PRESENTACIÓN: La metodología orientada a objetos es un nuevo paradigma para el desarrollar Sistemas de Información, ya que ha demostrado ser de gran utilidad para la solución de problemas que se presentan al desarrollar sistemas, brindando entre otros beneficios: reutilización de código, reducción de costos y tiempos de desarrollo menores , así como sistemas de alta calidad, fáciles de modificar y mejorar.

OBJETIVO GENERAL: El alumno describirá y analizará los alcances, conceptos y beneficios del análisis y diseño orientado a objetos como una alternativa viable para el desarrollo de sistemas, además evaluará las metodologías y su importancia de uso durante la construcción de sistemas de información.

HORAS	TEMATICA	OBJETIVOS EDUCACIONALES	SUGERENCIAS DIDÁCTICAS	REFERENCIAS BIBLIOGRAFICAS
5 HRS.	1. ANTECEDENTES 1.1 La problemática del software 1.2 Requerimientos y características de los desarrollos actuales 1.3 Impacto institucional en los costos, tiempo y calidad de los desarrollos 1.4 Paradigma orientado a objetos 1.4.1 Historia 1.4.2 Filosofía general.	1.PARTICULAR DE LA UNIDAD Al término de esta unidad el alumno conocerá los antecedentes y la problemática actual del desarrollo de software en México 2. ESPECÍFICOS El Alumno será capaz de : Explicar y entender los requerimientos y características de los desarrollos actuales	Exposición oral y/o audiovisual por parte del profesor. Lluvia de ideas.	1, 3, 4

HORAS	TEMATICA	OBJETIVOS EDUCACIONALES	SUGERENCIAS DIDÁCTICAS	REFERENCIAS BIBLIOGRAFICAS
5 HRS.	2. CONCEPTOS 2.1 Software 2.2 Sistemas de Información 2.3 Modelos 2.4 Objetos y clases 2.5 Atributos y operaciones 2.6 Encapsulamiento, Abstracción y Clasificación 2.7 Mensajes 2.8 Métodos 2.9 Herencia y Poliformismo	1.PARTICULAR DE LA UNIDAD Al término de esta unidad el alumno conocerá conceptos básicos del paradigma orientado a objetos 2. ESPECÍFICOS El Alumno será capaz de : Explicar y entender los conceptos O.O.	Exposición oral y/o audiovisual por parte del profesor.	1, 2
10 HRS.	3. PROCESOS DE DESARROLLO Y METODOLOGÍAS PARA EL ANÁLISIS Y DISEÑO DE SISTEMAS 3.1 Cascada 3.2 Espiral 3.3 Evolutiva 3.4 TPS 3.5 RUP 3.6 OMT 3.7 BOOCH 3,8 Ivar Jacobson (Uses Cases) 3.9 UML	1.PARTICULAR DE LA UNIDAD Al término de esta unidad el alumno conocerá las metodologías de análisis y diseño O.O. y los procesos de desarrollo más utilizados. 2. ESPECÍFICOS El Alumno será capaz de : Explicar y entender las ventajas de las metodologías y procesos de desarrollo de sistemas de información O.O	Exposición oral y/o audiovisual por parte del profesor.	1,2,5
25 HRS.	4. PROCESO UNIFICADO RACIONAL 4.1 Fase de Inicio o Concepción 4.2 Fase de Elaboración 4.3 Fase de Construcción 4.4 Fase de Transición	1.PARTICULAR DE LA UNIDAD Al término de esta unidad el alumno conocerá las etapas para desarrollar un Sistema de Información O.O 2. ESPECÍFICOS El Alumno será capaz de Explicar y aplicar el proceso de desarrollo de un Sistema de Información O.O	Exposición oral y/o audiovisual por parte del profesor. Casos prácticos para ser desarrollados en equipos.	4

HORAS	TEMATICA	OBJETIVOS EDUCACIONALES	SUGERENCIAS DIDÁCTICAS	REFERENCIAS BIBLIOGRAFICAS
25 HRS.	5. LENGUAJE DE MODELADO UNIFICADO 5.1 Modelado Inicial del sistema 5.2 Modelado Estático 5.3 Modelado Dinámico 5.4 Arquitectura del sistema	1.PARTICULAR DE LA UNIDAD Al término de esta unidad el alumno conocerá el lenguaje de desarrollo para realizar análisis y diseño O.O. 2. ESPECÍFICOS El Alumno será capaz de explicar y desarrollar el análisis y diseño de un Sistema de Información O.O	Exposición oral y/o audiovisual por parte del profesor. Casos prácticos para ser desarrollados en equipos.	1,2,5
10 HRS.	6. HERRAMIENTAS PARA EL DESARROLLO DE S.O.O 6.1 Lenguajes OO 6.2 Bases de Datos OO 6.3 Herrameintas CASE OO	1.PARTICULAR DE LA UNIDAD Al término de esta unidad el alumno conocerá las herramientas para desarrollar Sistemas de Información O.O. 2. ESPECÍFICOS El alumno será capaz de : Explicar la utilidad de las herramientas de desarrollo Aplicables a los sistemas de Información O.O.	Exposición oral y/o audiovisual por parte del profesor.	3,4,5

BIBLIOGRAFÍA:

1.Boccch, Grady, Análisis y Diseño Orientado a Objetos 2ª edición, México Addison-Wesley Iberoamericana 1996
2. Fowler, Martin, UML Gota a Gota México, Addison-Wesley 1999
3. Jacobson, Ivar. Object Oriented Software Engineering EE. UU, Addison-Wesley 1994
4. Jacobson, Ivar y Rumbaugh. El proceso unificado de desarrollo de software
5. Rumbaugh, James y otros. Object oriented modeling and design EE. UU. Prentice-Hall 1991

294

REFERENCIAS EN INTERNET:
*http://www.rational.com (Rational Software Corporation) *http://www.well.com/user/rtchie/oo.html *http://www.cetus-links.org
PERFIL DEL PARTICIPANTE:
Dirigido a todo profesional dedicado al análisis y diseño de sistemas de información, como líderes de proyectos, analistas y programadores que deseen conocer herramientas esenciales para el desarrollo de sistemas de información orientados a objetos. El alumno deberá tener conocimientos de Técnicas de Programación, Introducción a Bases de Datos y Análisis y Diseño Estructurado de Sistemas de Información
PERFIL PROFESIOGRÁFICOS DEL DOCENTE:
Los requisitos que debe reunir el docente para impartir esta asignatura son los siguientes: ACADÉMICOS: Ser Lic. En Informática PROFESIONALES: Tener experiencia y /o conocimiento del Análisis, Diseño y Programación Orientada a Objetos DOCENTES: Acreditar dos cursos de didáctica de 30 horas en la Coordinación de Calidad Académica de la FCA. Conocer la asignatura de Informática II

ARTÍCULO DE REVISTA

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE COPNTADURÍA Y ADMINISTRACIÓN

L.A. Gustavo Almaguer Pérez
Jefe de la gaceta de la FCA
PRESENTE

Por recomendación de nuestro asesor de tesis, el Dr. Ricardo Rivera Soler, nos permitimos distraer su atención haciéndole llegar un artículo de revista, que ha sido uno de los resultados obtenidos de la tesis profesional denominada "RUP y UML como elementos clave en el desarrollo de sistemas Orientados a Objetos", la cual sus servidoras hemos sido autoras, para que sea tomado en cuenta para su posible publicación en la gaceta de la facultad.

Anexamos a la presente carta, el artículo, para ser tomado a su consideración. Agradecemos de antemano su atención y esperando que dicha propuesta sea tomada en cuenta, estamos a sus órdenes para cualquier aclaración.

Aprovechamos la ocasión para enviarle nuestro más cordial saludo.

Recibí
17-10-02
HILDA TELLEZ-GIRON

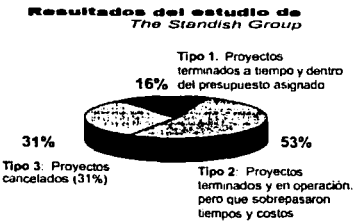
ATENTAMENTE

Claudia Elvira Soriano Monzalvo
Tels. 2617 07 51 particular
52 37 90 00 Ext. 1277 oficina

Silvia Jiménez Luna
Tels. 57 87 68 29 particular
56 22 36 04 oficina

"EXPERTAS INDUSTRIAS DECLARAN LA CRISIS DEL SOFTWARE..... 1964"

HAN MEJORADO LAS COSAS DESDE ENTONCES?



CONOZCA LOS FACTORES QUE EVITARAN QUE SU PROYECTO DE SOFTWARE SEA UN FRACASO

Existen dos tipos de fallas: aquellas que se previnieron pero no ocurrieron, y aquellas que ocurrieron pero no se previnieron.

Al desarrollar un sistema te habrás preguntado muchas veces ¿Cómo lograr un sistema de calidad? ¿Cómo lograr desarrollar un sistema en el tiempo y presupuesto estimado? ¿Al cumplir con los requerimientos de los usurarios se logra la calidad del sistema?

Hoy en dia la necesidad de desarrollar software de alta calidad ha venido creciendo dentro de la industria mundial del software, la preocupación de las empresas desarrolladoras de software por mejorar sus procesos de desarrollo es latente. Por otra parte, encontramos empresas con necesidades de información veraz, oportuna que requieren sistemas seguros, confiables, adaptables en el menor tiempo posible.

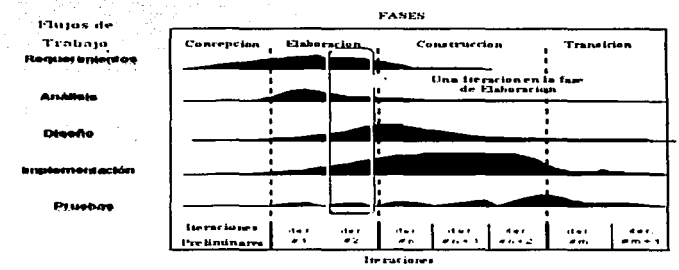
Un factor que ha dado origen al problema de no desarrollar software de calidad que cumpla al 100% con los requerimientos de los usuarios es la serie de dificultades que se tienen dentro de las etapas de Análisis y Diseño de los sistemas, esto se da por no emplear las herramientas adecuadas.

El paradigma orientado al objeto es una nueva y diferente forma de pensar, actualmente no solo se aplica a los lenguajes de programación, sino que también se ha propagado a los métodos de análisis y diseño y a otras áreas, tales como las bases de datos.

UML y RUP ofrecen incrementos potenciales en la productividad y calidad en el desarrollo de sistemas. La adopción, utilización y administración con éxito de estas técnicas, pueden dar lugar a beneficios importantes para todas las personas involucradas en el desarrollo de sistemas, lo que indicará eficiencia competitiva.

RUP (Proceso Unificado Racional) es un conjunto de actividades y responsabilidades necesarias para transformar los requisitos de un usuario en un sistema de software, siendo este un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes organizaciones y diferentes tamaños de proyectos, cuyo propósito principal es la producción de software de alta calidad y de acuerdo a las necesidades de los usuarios.

Fase y Flujos de Trabajo del RUP



LAS 6 MEJORES PRÁCTICAS SOBRE LAS QUE SE BASA EL RUP SON:

- Desarrollo Interativo de software: El desarrollo de software sofisticado no se desarrolla de manera secuencial primero definiendo el problema, diseñando la solución, construyendo el software, haciendo pruebas y entregando el producto final.
- Administración de requerimientos: RUP describe cómo obtener, organizar y documentar los requerimientos funcionales, así como su fácil captura y comunicación de los requerimientos de casos de uso.
- Arquitectura basada en componentes: El proceso se enfoca a un fácil desarrollo y una arquitectura robusta que permitirá un desarrollo escalable y flexible, ya que es adaptable a cualquier cambio y provee con eficiencia la reutilización de software.
- Modelado de software visual: El modelo es mostrado visualmente a través de simbología unificada capturando la estructura y comportamiento de la arquitectura y de todos los componentes de software, esto permite esconder los detalles y escribir el código usando bloques gráficos construidos.
- Verificación de la calidad de software: El proceso de desarrollo unificado verifica la calidad con respecto a los requerimientos de los usuarios basándose en la confiabilidad, funcionalidad y el desempeño del sistema.
- Control de cambio en el desarrollo de software: La habilidad para la administración de cambios durante el desarrollo de software es importante para verificar que todos los cambios sean aceptables, sin descuidar la calidad del producto de software.

LAS FASES QUE COMPONEN AL RUP:

- Fase de Concepción: objetivos del ciclo de vida y determinación de requerimientos
- Fase de Elaboración: arquitectura del ciclo de vida
- Fase de Construcción: desarrollo de la funcionalidad operativa
- Fase de Transición: versión del producto semifinal y final

FLUJOS DE TRABAJO QUE COMPONEN AL RUP:

- Determinación de requerimientos
- Análisis, ámbito del sistema y determinación de la arquitectura
- Diseño
- Implementación
- Prueba

UML (Lenguaje Unificado de Modelado) es un lenguaje de modelado y no un método, que permite mostrar el diseño y los requerimientos de un sistema, creando modelos precisos de desarrollo, lo que permite mejorar la comunicación con los usuarios y el equipo de trabajo, ya que es una herramienta útil para generar documentación detallada de la arquitectura del sistema a desarrollar.

Al utilizar UML se pueden crear los siguientes diagramas que permiten diseñar al sistema desde diferentes puntos de vista:

Diagrama de clases y objetos: Diagrama que muestra las clase y objetos encontrados en el sistema, lo cual permitirá definir la base de datos.

Diagrama de casos de uso: Está compuesto por un conjunto de actores, casos de uso y relaciones entre ellos. Describe una secuencia de transacciones desarrolladas por un sistema en respuesta a un evento iniciado por un actor sobre el sistema. Los actores representan los diferentes roles desempeñados por los usuarios del sistema

Diagrama de secuencia: Un diagrama de secuencia muestra interacciones de objetos ordenados en secuencia de tiempo.

Diagrama de colaboración: Un diagrama de colaboración es una forma alternativa de representar los mensajes intercambiados por un conjunto de objetos. El diagrama muestra interacciones de objeto organizadas alrededor de los objetos y sus ligas a cada uno.

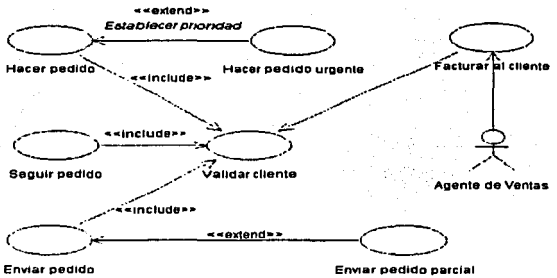
Diagrama de interacción colaboración: Muestra cómo interactúa el conjunto de objetos en el sistema y sus relaciones incluyendo los mensajes que se pueden enviar entre ellos.

Diagrama de interacción secuencia: Es un diagrama de interacción en donde se destaca la ordenación de los mensajes.

Diagrama de estados: Estos diagrama utilizado para modelar objetos reactivos, especialmente instancias de clases (que tenga sus estados identificables claramente o un comportamiento complejo), casos de uso y el sistema global.

Diagrama de actividades: Estos describen el flujo de control de una secuencia de acciones, ya sea un algoritmo de la operación de una clase, la interacción de un grupo de objetos, la especificación de un caso de uso, las actividades de un procedimiento, etc.

Diagrama de componentes: Los componentes son la parte intangible, es decir, parte del sistema que se desarrolla y se despliega sobre un hardware (Nodo).
Diagrama de despliegue: Permite representar el diseño lógico y físico, la parte no tangible, se pueden diseñar clases, interfaces, colaboraciones, iteraciones y estados.



D. Casos de Uso

BENEFICIOS QUE OFRECE UML

- **UML ayuda a mantener una buena comunicación con los usuarios (expertos en el dominio).**
- **UML ayuda al desarrollo en equipo.**
- **Ayudan a comprender el dominio del problema, aún de sistemas complejos.**
- **UML es una técnica que ayuda a mostrar los requerimientos y el diseño.**
- **Permite crear modelos precisos, no ambiguos y completos.**
- **UML permite aplicar a ingeniería directa, es decir, generar código en algún lenguaje de programación a partir de un modelo UML. Los modelos de UML pueden conectarse directamente a una gran variedad de lenguajes de programación, a pesar de que UML no es un lenguaje de programación visual. Se puede establecer correspondencias desde un modelo UML a un lenguaje de programación o base de datos orientada a objetos o incluso a tablas en una base de datos relacional. También se puede aplicar la ingeniería inversa, construir un modelo UML a partir de alguna serie de líneas de código.**
- **Permite la ejecución directa de modelos, la simulación de sistemas y la instrumentación de sistemas en ejecución.**
- **Es una herramienta muy útil para generar la documentación detallada de la arquitectura de un sistema.**
- **Es un lenguaje útil para modelar las actividades de planificación de proyectos y estión de versiones.**
- **Su vocabulario y reglas indican corno crear y leer modelos bien formados**

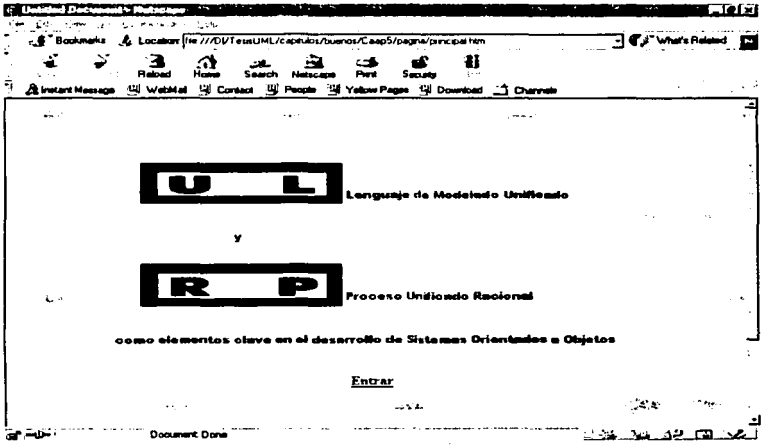
Los creadores de UML son Grady Booch, Jim Rumbaugh, Ivar Jacobson, Martin Fowler, Buschmann, Robert Martín,Cook and Daniels, Goldberg & Rubin, McConnell, y Graham.

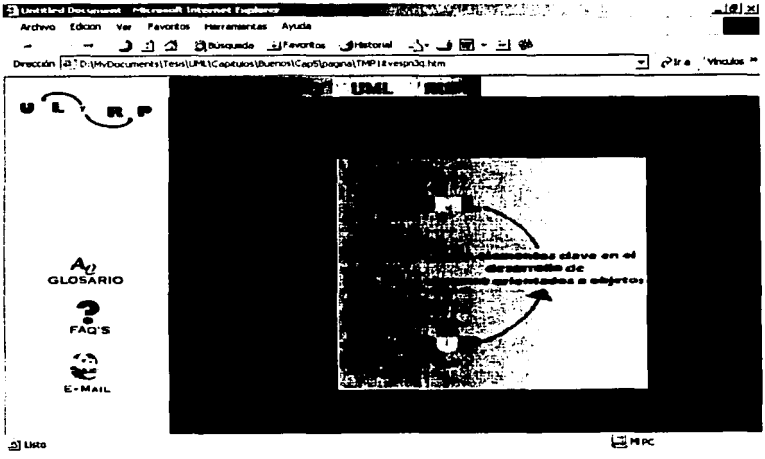
Cuando se emplea el lenguaje UML y el proceso de desarrollo RUP es necesario complementarlos, utilizando a las mejores personas (capacitadas y motivadas) y herramientas que permitan cumplir los objetivos establecidos para concluir con éxito el desarrollo de un sistema orientado a objetos.

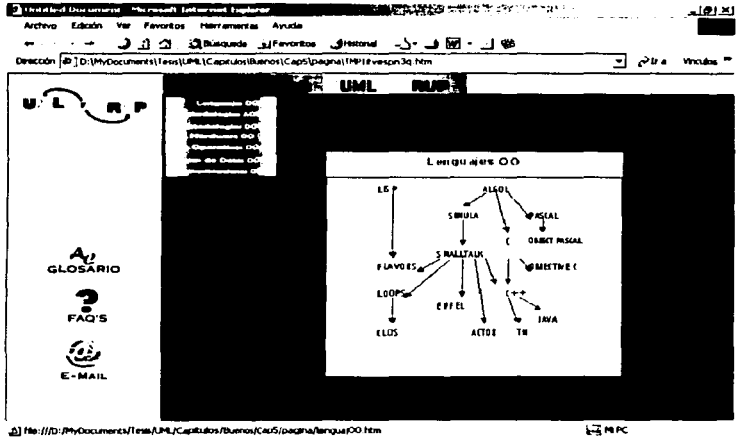
A continuación proporcionamos unas direcciones de Internet en donde podrás encontrar más información sobre UML y RUP

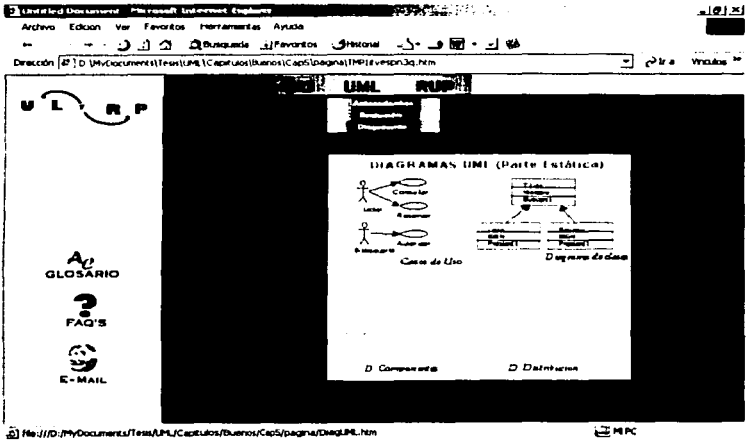
REFERENCIAS EN INTERNET:
*http://www.rational.com (Rational Software Corporation) *http://www.well.com/user/rtchie/oo.html *http://www.cetus-links.org

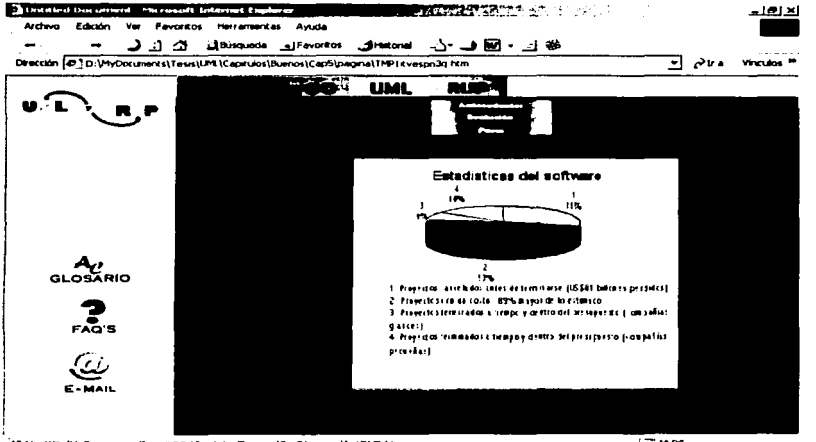
PÁGINA WEB

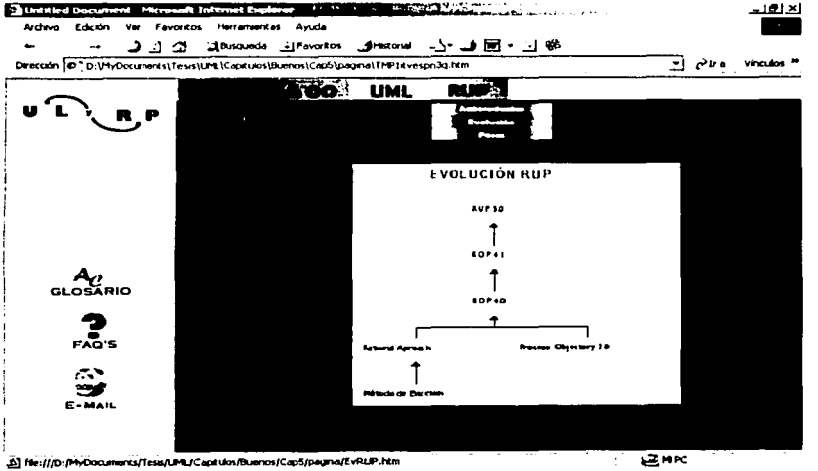

Bookmarks
Location: file:///D|/Tesis/UML/Capitulos/Buenos/Cap5/pagina/principal.htm
What's Related
Reload
Home
Search
Netscape
Print
Security
Instant Message
WebMail
Contact
People
Yellow Pages
Download
Channels
U L
Lenguaje de Modelado Unificado
y
R P
Proceso Unificado Racional
como elementos clave en el desarrollo de Sistemas Orientados a Objetos
Entrar
Document Done

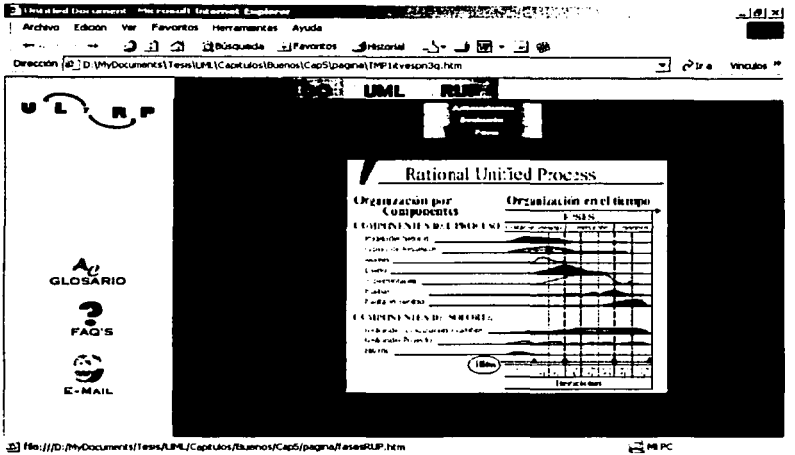

Archivo
Edición
Ver
Favoritos
Herramientas
Ayuda
Búsqueda
Favoritos
Historial
Dirección
D:\MyDocuments\Tesis\UML\Capitulos\Buenos\Cap5\pagina\TMP18vespn3q.htm
Ir a
Vínculos
U L
R P
UML
GLOSARIO
FAQ'S
E-MAIL
Listo
Mi PC

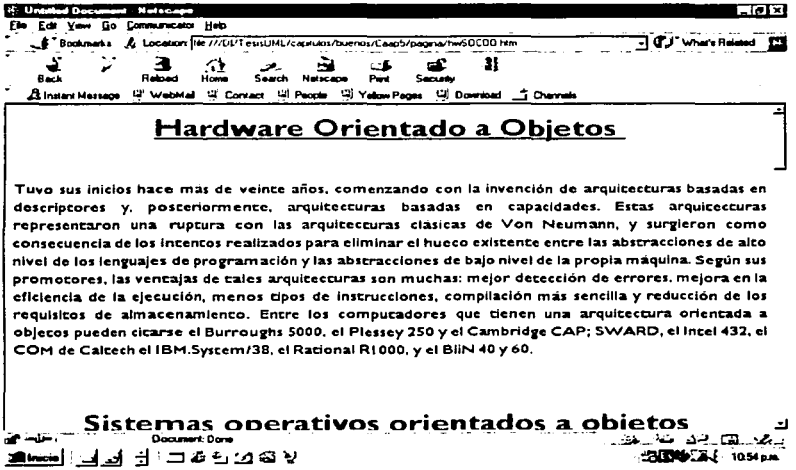

Archivo Edición Ver Favoritos Herramientas Ayuda
Dirección D:\MyDocuments\Tesis\UML\Capitulos\Buenos\Cap5\pagina\TMP1\vespn3q.htm
UML RUP
GLOSARIO
FAQ'S
E-MAIL
Lenguajes OO
LISP
ALGOL
SIMULA
PASCAL
C
OBJECT PASCAL
SMALLTALK
FLAVORS
OBJECTIVE C
LOOPS
EIFFEL
C++
JAVA
CLOS
ACTOR
TM
file:///D:/MyDocuments/Tesis/UML/Capitulos/Buenos/Cap5/pagina/lenguajeOO.htm
MI PC

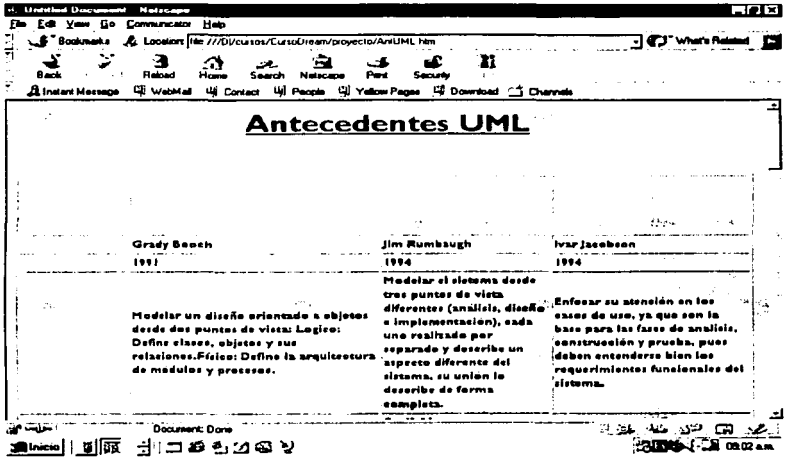

Archivo Edicion Ver Favoritos Herramientas Ayuda
Dirección D:\MyDocuments\Tesis\UML\Capitulos\Buenos\Cap5\pagina\TMP1\vespn3q.htm
UML RUP
GLOSARIO
FAQ'S
E-MAIL
DIAGRAMAS UML (Parte Estática)
Casos de Uso
Diagrama de clases
Componentes
Distribución
file:///D:/MyDocuments/Tesis/UML/Capitulos/Buenos/Cap5/pagina/DiagUML.htm
MI PC

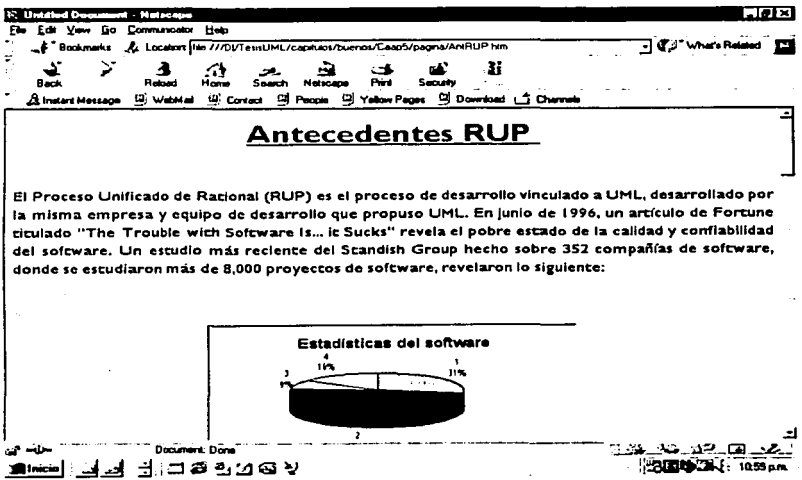

Archivo Edición Ver Favoritos Herramientas Ayuda
Dirección D:\MyDocuments\Tesis\UML\Capitulos\Buenos\Cap5\pagina\TMP1kvespn3q.htm
UML
RUP
Estadisticas del software
GLOSARIO
FAQ'S
E-MAIL
file:///D:/MyDocuments/Tesis/UML/Capitulos/Buenos/Cap5/pagina/AntRUP.htm
Mi PC


Archivo Edición Ver Favoritos Herramientas Ayuda
Dirección D:\MyDocuments\Tesis\UML\Capitulos\Buenos\Cap5\pagina\TMP14vespn3q.htm
UML
RUP
EVOLUCIÓN RUP
AVP 90
ROP 41
ROP 40
GLOSARIO
FAQ'S
E-MAIL
file:///D:/MyDocuments/Tesis/UML/Capitulos/Buenos/Cap5/pagina/EvRUP.htm
Mi PC


Archivo Edición Ver Favoritos Herramientas Ayuda
Dirección D:\MyDocuments\Tesis\UML\Capitulos\Buenos\Cap5\pagina\TMP1kvespn3q.htm
U L R P
UML
RUP
Rational Unified Process
Organización por Componentes
Organización en el tiempo
GLOSARIO
FAQ'S
E-MAIL
file:///D:/MyDocuments/Tesis/UML/Capitulos/Buenos/Cap5/pagina/TesisRUP.htm
Mi PC


File Edit View Go Communicator Help
Bookmarks Location: file:///D|/Tesis/UML/capitulos/buenos/Cap5/pagina/hw50C00.htm
What's Related
Back Reload Home Search Netscape Print Security
Instant Message WebMail Contact People Yellow Pages Download Channels
Hardware Orientado a Objetos
Tuvo sus inicios hace mas de veinte años, comenzando con la invención de arquitecturas basadas en descriptores y, posteriormente, arquitecturas basadas en capacidades. Estas arquitecturas representaron una ruptura con las arquitecturas clasicas de Von Neumann, y surgieron como consecuencia de los intentos realizados para eliminar el hueco existente entre las abstracciones de alto nivel de los lenguajes de programación y las abstracciones de bajo nivel de la propia máquina. Según sus promotores, las ventajas de tales arquitecturas son muchas: mejor detección de errores, mejora en la eficiencia de la ejecución, menos tipos de instrucciones, compilación más sencilla y reducción de los requisitos de almacenamiento. Entre los computadores que tienen una arquitectura orientada a objecos pueden citarse el Burroughs 5000, el Plessey 250 y el Cambridge CAP; SWARD, el Intel 432, el COM de Caltech el IBM.System/38, el Rational R1000, y el BliN 40 y 60.
Sistemas operativos orientados a objetos
Document: Done
Inicio
10:54 p.m.


Antecedentes UML
Grady Booch
1991
Modelar un diseño orientado a objetos desde dos puntos de vista: Lógico: Define clases, objetos y sus relaciones.Físico: Define la arquitectura de modulos y procesos.
Jim Rumbaugh
1994
Modelar el sistema desde tres puntos de vista diferentes (análisis, diseño e implementacion), cada uno realizado por separado y describe un aspecto diferente del sistema, su union lo describe de forma completa.
Ivar Jacobson
1994
Enfocar su atención en los casos de uso, ya que son la base para las fases de análisis, construcción y prueba, pues deben entenderse bien los requerimientos funcionales del sistema.

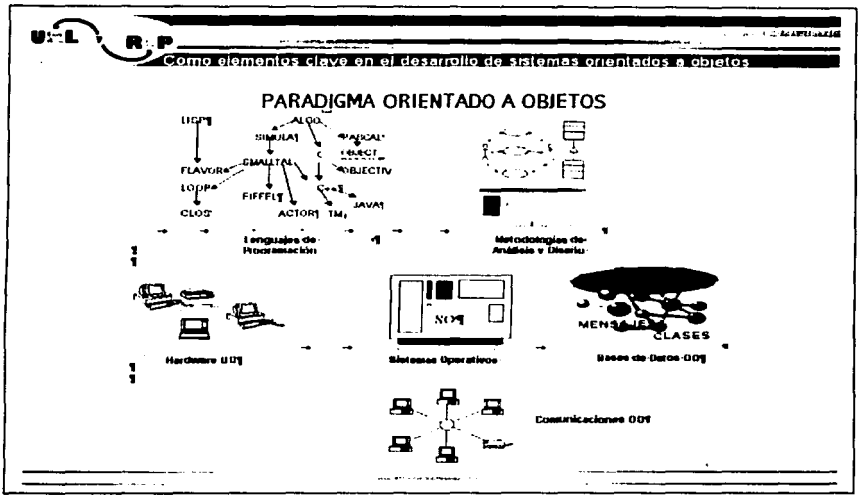

Antecedentes RUP
El Proceso Unificado de Racional (RUP) es el proceso de desarrollo vinculado a UML, desarrollado por la misma empresa y equipo de desarrollo que propuso UML. En junio de 1996, un artículo de Fortune ticulado "The Trouble with Software is... it Sucks" revela el pobre estado de la calidad y confiabilidad del software. Un estudio más reciente del Standish Group hecho sobre 352 compañías de software, donde se estudiaron más de 8,000 proyectos de software, revelaron lo siguiente:
Estadísticas del software

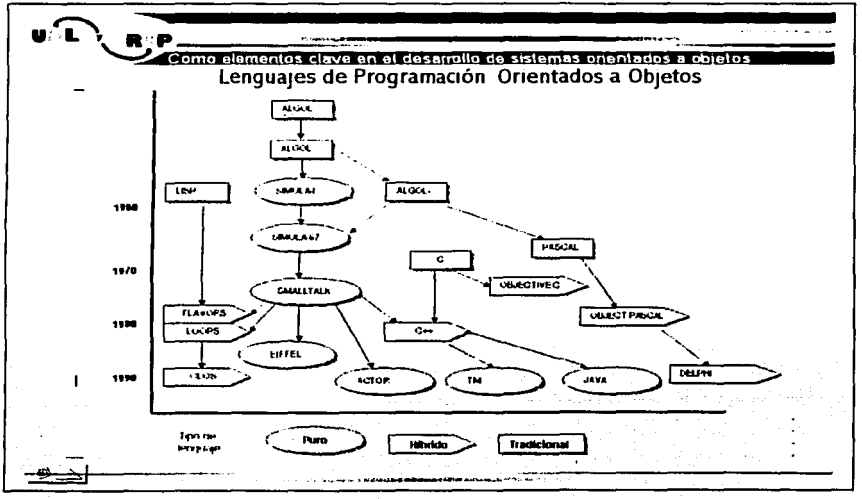
DIAPOSITIVAS PARA CONFERENCIAS

" UML y RUP como elementos clave en el desarrollo de sistemas orientados a objetos "

METODOLOGÍAS DE PROGRAMACIÓN

- **Programación no Estructurada**
- **Programación Estructurada o Modular**
- **Programación Procedimental**
- **Programación Orientada a Eventos**
- **Programación Orientada a Objetos**


Como elementos clave en el desarrollo de sistemas orientados a objetos
PARADIGMA ORIENTADO A OBJETOS
ALGOL
SIMULA
SMALLTALK
FLAVORS
LOOPS
CLOS
EIFFEL
ACTOR
C++
JAVA
OBJECTIV
Lenguajes de Programación
Metodologías de Análisis y Diseño
Hardware OO
Sistemas Operativos
MENSAJES
CLASES
Bases de Datos OO
Comunicaciones OO


Como elementos clave en el desarrollo de sistemas orientados a objetos
Lenguajes de Programación Orientados a Objetos
ALGOL
ALGOL
LISP
SIMULA
ALGOL
1960
SIMULA67
1970
C
PASCAL
OBJECTIVEC
SMALLTALK
1980
FLAVORS
LOOPS
C++
OBJECT PASCAL
EIFFEL
1990
CLOS
ACTOR
TM
JAVA
DELPHI
Tipos de lenguajes
Puro
Híbrido
Tradicional

METODOLOGÍAS PARA EL ANÁLISIS ORIENTADO A OBJETOS

- OOSA de Shlaer/Mellor
- Coad/Yourdon
- Rumbaugh - OMT
- Martin/Odell - Ptech
- Objectory CASE
- DESFRAY Método de relaciones de clases
- OORASS
- OSA
- Systems Enginecring 00
- TEXEL
- BON - Nerson
- Fusión - Coleman
- OBA

METODOLOGÍAS PARA EL DISEÑO ORIENTADO A OBJETOS

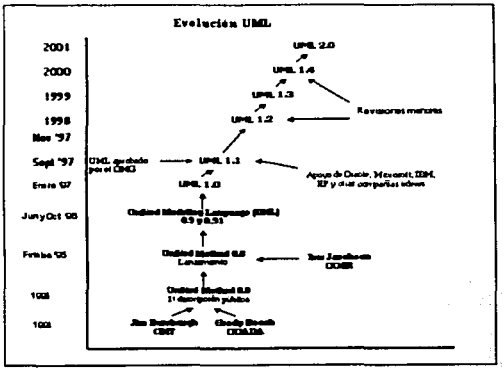
- GOOD
- HOOD
- OOSD
- OODLE
- JSD y OOJSD
- Booch (1991)
- CRC y RDD

ANTECEDENTES UML

- Metodología OOADA (Object Oriented Analysis and Design with Aplications)
- Metodología OMT (Object Modeling Technique)
- Metodología (OOSE) (Object Oriented Software Enginnering (Ingeniería de Software Orientada a Objetos)

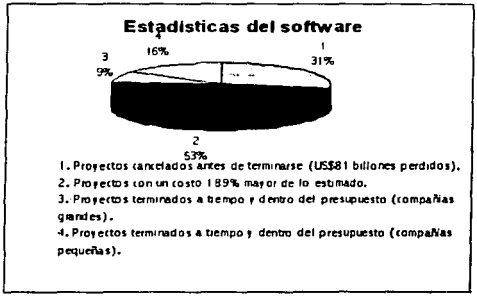
Como elemento clave en el desarrollo de sistemas orientados a objetos

EVOLUCIÓN UML



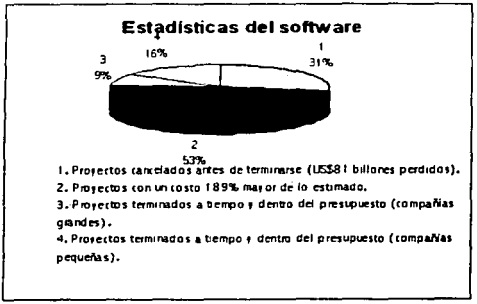
UML y RUP
como elementos clave en el desarrollo de sistemas orientados a objetos

ANTECEDENTES RUP



UML y RUP
como elementos clave en el desarrollo de sistemas orientados a objetos

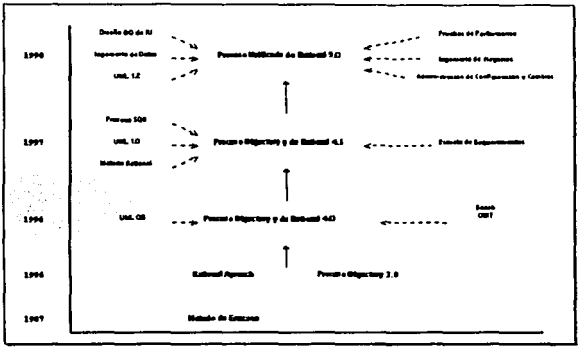
ANTECEDENTES RUP



U L R P

Como elementos clave en el desarrollo de sistemas orientados a objetos

EVOLUCIÓN DE RUP

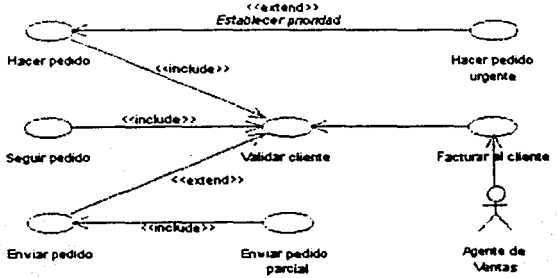


U L R P

Como elementos clave en el desarrollo de sistemas orientados a objetos

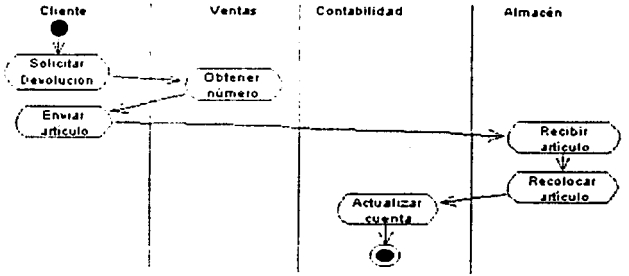
DIAGRAMAS UML

Diagrama de Casos de Uso



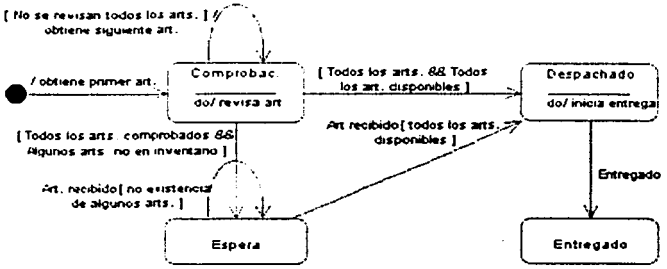
DIAGRAMAS UML

Diagrama de Actividad



DIAGRAMAS UML

Diagrama de Estados



GLOSARIO

Abstracción: Agrupar el comportamiento común de las clases y o futuras subclases y generar una clase padre abstracta que especifica a sus hijos.

Acción: Operación ejecutable que constituye la abstracción de un procedimiento computacional. Una acción da lugar a un cambio de estado y se lleva a cabo enviando un mensaje a un objeto o modificando un valor en un atributo.

Actividad: Conjunto de operaciones o tareas propias de una persona o entidad para conseguir sus objetivos.

Actor: Entidad externa al sistema que se modela y que puede interactuar con él; un ejemplo de actor podría ser un usuario o cualquier otro sistema.

Ada: Lenguaje de programación de alto nivel, utiliza la programación modular, su programación se hace vía hardware, con lo que se consigue una mayor velocidad de proceso. Es muy utilizado en sistemas que requieren un cálculo en tiempo real.

Administrar: Ordenar, organizar, en especial los bienes. Racionar algo.

Agregación: Unión o adición de una parte a un todo.

Algoritmo: Conjunto de instrucciones que especifican la secuencia de operaciones a realizar, en orden, para resolver un sistema específico o clase de problema. Suelen expresarse a través de letras, cifras y símbolos, que forman un algoritmo determinado.

Análisis: Énfasis en el qué, dominio del problema.

Análisis Orientado a Objetos: Es un método que examina los requerimientos desde la perspectiva de clases y objetos que se localizan en la definición de un problema.

Aplicación: Cada uno de los programas que, una vez ejecutados, permiten trabajar con una computadora. Son aplicaciones los procesadores de textos, hojas de cálculo, bases de datos, programas de dibujo, paquetes estadísticos, etc.

Applets: Pequeñas aplicaciones escritas en Java, se difunden a través de la red para ejecutarse en el visualizador cliente.

Archivo: Conjunto de bytes estructurados almacenados como una entidad individual. Todos los datos en disco se almacenan como un archivo con un nombre único dentro del directorio en que reside. Los datos almacenados pueden recuperarse fácilmente y usarse en una aplicación determinada, los archivos no contienen elementos de la aplicación que los crea, sólo los datos o información con los que trabaja el usuario.

Arquitectura: Integra la selección de componentes de los cuales está compuesta la arquitectura, su interacción, composición entre ellos, y en general de los patrones que guían esta composición. No solo se refiere a la estructura del sistema, pero si a la funcionalidad, el rendimiento, diseño, selección entre
Arreglos: Conjunto ordenado de elementos de datos. Un vector es un arreglo unidimensional, una matriz es un arreglo bidimensional. La mayoría de los lenguajes de programación tienen la capacidad para almacenar y manipular arreglos de una o más dimensiones. Los arreglos multidimensionales son ampliamente utilizados en simulación científica y procesamiento matemático; sin embargo, un arreglo puede ser tan simple como una tabla de precios que está en memoria para ser accedida instantáneamente por un programa de ingreso de pedidos.

Artefacto: Artificio, máquina, aparato.

Asociación binaria: Una asociación entre dos clases.

Asociación n-aria: Una asociación entre n clases. Cuando n es i-ual a de),

Atómico: El átomo o relativo a él. Partícula más pequeña e indivisible.

Atributo: Es una característica inherente, discreta, una propiedad de cantidad o calidad de un objeto o clase. Son utilizados para identificar, describir, proporcionar el estado de objetos y clases.

Base de Datos: Conjunto de datos relacionados que se almacenan de forma que se pueda acceder a ellos de manera sencilla, con la posibilidad de relacionarlos, ordenarlos en base a diferentes criterios, etc. Entre las más conocidas pueden citarse dBase, Paradox, Access y Aproach, para entornos PC, y Oracle, ADABAS, DB/2, Informix o Ingres, para sistemas medios y grandes.

Benchmark: Aalizan el rendimiento de las computadoras en diferentes facetas.

Biblioteca: Colección o conjunto de programas desarrollados por un mismo fabricante que suelen ser compatibles e interoperables entre sí.

Bit: Unidad de información más pequeña. Puede tener sólo dos valores o estados: 0 o 1, encendido o apagado. La combinación de estos valores es la base de la informática, ya que los circuitos internos de la computadora sólo son capaces de detectar si la corriente llega o no llega (0 o 1). Su nombre proviene de la contracción de las palabras «binary» y «digit» (dígito binario).

Bucle: Conjunto de instrucciones contenidas en un programa o rutina que se repite un número determinado de veces.

Byte: Unidad de comunicación de datos compuesta de ocho bits (unos y ceros) y que representa un carácter.

C: El lenguaje C es una herramienta de programación de tipo general, utilizada para el desarrollo del sistema operativo Unix. Fue realizado a principios de la década de los setenta por Dennis Ritchie, como evolución del lenguaje B que creara Ken Thompson.

Calidad: Propiedades inherentes a una persona o cosa que permiten apreciarla con respecto a las restantes de su especie. Superioridad o excelencia.

Cardinalidad: El número de elementos en un conjunto.
Case: (Computer Aided Software Engineering). Bajo el término de Ingeniería de Software Asistida por Computadora se incluyen una serie de herramientas, lenguajes y técnicas de programación que permiten la generación de aplicaciones de manera semiautomática. Las herramientas CASE liberan al programador de parte de su trabajo y aumentan la calidad del programa a la vez que disminuyen sus posibles errores.

Caso de Uso: Secuencia de transacciones desarrolladas por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema.

Ciclo de Vida: Algunas veces llamado "modelo en cascada", el paradigma del ciclo de vida exige un enfoque sistemático y secuencial del desarrollo del software que comienza en el nivel del sistema y progresa a través del análisis, diseño, codificación, prueba y mantenimiento.

Clase: Descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y semántica. Una clase es una implementación de un tipo (Class). En programación orientada a objetos, un tipo de datos definido por el usuario que especifica un conjunto de objetos que comparten las mismas características. Un miembro de la clase (objeto) es un "ejemplo" o caso de la clase. Las clases concretas están diseñadas para citar como ejemplos, mientras que las clases abstractas, para pasar las características por herencia.

Clase asociación: Un elemento de modelado que tiene a la vez propiedades de asociación y de clase. Una clase asociación puede verse como una asociación que tiene además propiedades de clase, o una clase que tiene además propiedades de asociación. clase concreta: Una clase que puede ser instanciada directamente.

Clase Control: Tipo de clase, se usa para modelar el comportamiento de un conjunto de casos de uso. Crea, inicializa o borra objetos controlados. Controla la secuencia de ejecución de los objetos controlados. Se controla la concurrencia de las clases involucradas.

Clase Entity: Tipo de clase que se utiliza para modelar la información y el comportamiento de una clase.

Cliente: Cualquier elemento de un sistema de información que requiere un servicio mediante el envío de solicitudes al servidor. Cuando dos programas se comunican por una red, el cliente es el que inicia la comunicación, mientras que el programa que espera ser contactado es el servidor. Cualquier programa puede actuar como servidor para un servicio y como cliente para otro.

Codificación: En los antiguos lenguajes de programación, la codificación era la fase más penosa del trabajo de programación, al tener que escribirse líneas de código entendibles por el ordenador.
En la actualidad, el trabajo de codificación suele ser automático, y su revisión también, mediante las herramientas de programación adecuadas.

Código Binario: Código basado en dos valores (0 y 1) que es «entendido» por los ordenadores.

Colaboración: Una sociedad de clases, interfaces y otros elementos que trabajan juntos para proporcionar algún comportamiento cooperativo que es mayor que la suma de todos los elementos; la especificación de cómo un elemento, como un caso de uso o una operación. es llevada a cabo por un conjunto de clasificadores y asociaciones desempeñando roles específicos utilizados de una forma específica. comentario Una anotación que se adjunta a un elemento o colección de elementos.

Comando: Orden dada al ordenador para que realice una acción determinada.

Commit: Rollback. Esto ocurre cuando hay algún tipo de fallo en una transacción utilizando Bases de Datos Oracle, todos los datos que no hayan sido guardados (dar commit), regresan a su estado original, evitando así la perdida de datos cuando un cliente falla.

Compilador: Programa que traduce los programas escritos en lenguajes de alto nivel al lenguaje de la máquina.

Complejidad: Calidad de complejo.

Componentes: Conjunto de elementos que forman un todo.

Comportamiento: Conducta (proceder). Conducta, manera de portarse o actuar

Computadora: Ordenador, máquina electrónica que trata automáticamente la información y que ejecuta procesos lógicos a gran velocidad.

Comunicación: Unión que se establece o conducto que existe entre ciertas cosas o lugares. Medios gracias a los cuales las personas se comunican o relacionan, como el correo, teléfono o las carreteras.

Concurrencia: Acción o efecto de reunir. Reunión de personas.

Conocimiento: Entendimiento, inteligencia. Acción y resultado de conocer. Ciencia, conjunto de nociones e ideas que se tiene sobre una materia.

Consultoría: Entidad que se dedica profesionalmente a aconsejar sobre temas técnicos.

Controlar: Ejercer una persona el control sobre algo o alguien. Dominar o ejercer autoridad sobre una o varias personas.

Datos: Información amplia o concreta que permite una deducción o conocimiento exacto.

Decisión: Resolución o determinación acerca de algo dudoso.

Depurar: Limpiar, purificar, perfeccionar.

Desarrolladores: Puede aplicarse para denominar a toda persona involucrada en la realización de un producto de software, en ocasiones se relaciona más el concepto con los responsables de la etapa de programación.

Diagrama de Actividades: Es un caso especial de diagrama de estados en el que todos, o la mayoría, son estados activos y en el que todas, o la mayoría, de las transiciones son disparadas por la finalización de las

Diagrama de Caso de Uso: Sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y/o otros sistemas. Muestra la relación entre los actores y los casos de uso en un sistema. Ayudan a organizar y visualizar los requisitos del sistema
Diagrama de Clase: Muestra una colección de elementos del modelo declarativo (estático), como clases y tipos, y sus contenidos y relaciones.

Diagrama de Colaboración: Forma alternativa de representar los mensajes intercambiados por un conjunto de objetos. El diagrama muestra interacciones de objeto organizadas alrededor de los objetos y sus ligas a cada uno.

Diagrama de Componentes: Representa los elementos físicos de un sistema, es decir, los componentes software, que puede ser código fuente, binario o ejecutable, unidos por medio de relaciones de dependencia (generalmente de compilación).

Diagrama de Contexto: Diagrama de la notación de Yourdon, consiste en flujos de datos y flujos de control, almacenes de datos y un solo proceso que representa a todo el sistema.

Diagrama de Despliegue: Diagrama que modela los elementos tangibles como: computadoras personales, servidores, impresoras, procesadores, entre otros dispositivos; que pertenecen a los componentes físicos o hardware del sistema.

Diagrama de Estado: Representa todos los estados que puede tomar un objeto particular durante su existencia y cómo va cambiando el estado del objeto, ya sea por el paso del tiempo, los mensajes recibidos, errores encontrados o condiciones verdaderas, también representa los eventos que producen dichos cambios de estado de los objetos de una clase.

Diagrama de Flujo de Datos (DFD): Es una técnica gráfica que representa el flujo de información y las transformaciones que se aplican a los datos al moverse desde la entrada hasta la salida.

Diagrama de Interacción: Es una representación gráfica de interacciones entre objetos. Hay dos tipos de diagramas de interacción: Diagramas de secuencia y Diagramas de colaboración.

Diagrama de Objetos: Contiene a los objetos y sus relaciones en un momento dado del tiempo. Puede ser considerado un caso especial de un diagrama de clases o de un diagrama de colaboraciones.

Diagrama De Secuencias: Muestra interacciones entre objetos organizadas en secuencia temporal. En particular muestra los objetos participantes de la interacción y la secuencia de mensajes intercambiados. A diferencia del diagrama de colaboraciones, un diagrama de secuencias incluye la secuencia temporal pero no incluye las relaciones entre los objetos.

Diagrama de Tiempo: Muestra la secuencia de operaciones de los objetos

Diagrama de Transición de Estados: Es un modelo de comportamiento que se basa en la definición de un conjunto de estados del sistema.

Diagramas: Representación gráfica en la que se muestran las relaciones entre las diferentes partes de un conjunto o sistema o los cambios de un determinado fenómeno.

Diagramas de Flujo: Representa gráficamente la lógica de procedimiento de un programa de computadora.

Dinámica: De la dinámica o relativo a esta parte de la mecánica.

Disciplina: Conjunto de normas que rigen una actividad o una organización.

Diseño: Actividad creativa y técnica encaminada a idear objetos útiles y estéticos que puedan llegar a producirse en serie. Explicación breve, descripción somera de alguna cosa.

Diseño Dinámico: Es el modelado del comportamiento del sistema, donde se diseñan los diagramas de actividad, de colaboración, el diagrama de estados y el de interacción.

Diseño Estático: Se refiere al diseño de los diagramas de casos de uso, diagrama de clases, el diagrama de componentes y el de distribución.

Diseño Orientado a Objetos: Produce un diseño que interconecta objetos de datos (elementos de datos) y operaciones de procesamiento en una forma que modulariza la información y el procesamiento, en lugar de dejar aparte el procesamiento.

Disparar: Ejecutar una transición de estado.

Documentación: Conocimiento que se tiene de un asunto por la información que se ha recibido de él. Documento o conjunto de documentos, generalmente de carácter oficial, que sirven para la identificación personal o para acreditar alguna condición.

Dominios: Conocimiento profundo de alguna materia, ciencia, técnica o arte.

Eficiente: Que consigue un propósito empleando los medios idóneos.

Eiffel: Lenguaje de programación orientado a objetos avanzado que enfatiza el diseño y construcción de alta calidad y la reusabilidad del software.

Emergentes: Brotar o surgir algo.

Encapsulamiento: Ocultar las propiedades y comportamiento de un objeto.

Enlaces Dinámicos: Es un enlace que se realiza después de la petición de la salida.

Entidad: Cualquier cosa con existencia individual y definitiva dentro o fuera de la mente, cualquier cosa real en si misma.

Entorno: Mecanismo abstracto de ordenación de mecanismo abstracto de ordenación de clases que permite factorizar los elementos comunes de varias clases en una clase más general

Escenario: Conjunto de circunstancias que se consideran el entorno de una persona o suceso.

Especialización: Adiestramiento, preparación, estudio o ensayo en una determinada habilidad, arte o rama del conocimiento.

Especificación: Determinación, explicación o detalle de las características o cualidades de una cosa.

Esquema: Representación gráfica y simbólica de una cosa. Resumen de ideas y actos que se van a tratar.

Estado: Es cualquier modo de comportamiento observable.

Estándares: Es el estar unificado respecto a un modelo o norma común.

Estereotipo: Nuevo tipo de elemento a modelar, nuevo tipo de clase.

Estructura: Distribución y orden de las partes importantes que componen un todo.

Evento: Acontecimiento, suceso.

Evolución: Transformación, cambio progresivo. Es la serie de transformaciones sucesivas.

Experiencia: Enseñanza que se adquiere con la práctica. Acontecimiento que se vive y del que se aprende algo.

Fase: Cada uno de los estados sucesivos de una cosa que cambia o se desarrolla.

Fase de Construcción: Fase del proceso RUP en la que se codifica y programa el sistema.

Fase de Elaboración: Fase de RUP en la que se planea el proyecto.

Fase de Inicio: Fase del proceso de desarrollo RUP en la que se define el alcance del sistema a desarrollar.

Fase de Transición: Fase del proceso RUP en la que se libera el sistema desarrollado.

Filosofía: Ciencia que trata de la esencia, propiedades, causas y efectos de las cosas naturales.

Flexibilidad: Facilidad para acomodarse a distintas situaciones o a las propuestas de otros.

Función: Actividad propia de alguien o algo.

Funcionalidad: Conjunto de características que hacen que algo sea práctico y utilitario.

Generalización: Denota un tipo de relación entre una clase más general (superclase o padre) y una más especializada (subclase o hija).

Gestión: Dirección, administración de una empresa, negocio, etc.

Glosario: Repertorio de palabras difíciles o dudosas con su explicación.

Guía: Lo que dirige o encamina.

Hardware: Conjunto de elementos materiales que constituyen el soporte físico de un ordenador.
Herencia: Es una relación transitiva entre Clases, que permite a un Objeto de una Clase utilizar como propios los Datos y Métodos definidos en un Objeto de otra Clase.

Herencia Múltiple: Cuando una clase hija o subclase tiene varias clases padres (superclases).

Herramienta: Objeto que se utiliza para trabajar en diversos oficios o realizar un trabajo manual.

Híbrido: Se aplica en general a lo que está formado por elementos de distinta naturaleza:

Hipertexto: Sistema que permite que un texto contenga enlaces con otras secciones del documento o con otros documentos.

Hipótesis: Suposición sin pruebas que se toma como base de un razonamiento. Hipótesis de trabajo, la que se formula provisionalmente para guiar una investigación científica que debe demostrarla o negarla.

Hito principal: Punto en el que han de tomarse decisiones importantes de negocio, Cada fase acaba en un hito principal en el cual los gestores han de tomar decisiones cruciales de continuar o no con el proyecto y decidir sobre la planificación, presupuesto y requisitos del mismo. Son puntos de sincronización en los que coinciden una serie de objetivos bien definidos.

Hito secundarios: Hito intermedio entre dos hitos principales.

Homogénea: Iguales.

Identidad: Conjunto de rasgos o informaciones que individualizan o distinguen algo y confirman que es realmente lo que se dice que es.

Implantación: Establecimiento de doctrinas, instituciones o costumbres para su funcionamiento.

Indexación; Se usa en las aplicaciones de bases de datos para indicar la operación de ordenar los registros contenidos en ella de manera especial, en función de unos parámetros definidos previamente.

Información: Acción y resultado de informar o informarse. Conjunto de datos sobre una materia determinada.

Informática: Conjunto de conocimientos científicos y técnicos que hacen posible el tratamiento automático de la información por medio de computadoras.

Ingeniería: Conjunto de técnicas que permiten aplicar el saber científico a la utilización de la materia y de las fuentes de energía, mediante invenciones o construcciones útiles para el hombre.

Ingeniería directa: Transformación de un modelo en código escrito en un determinado lenguaje de programación.

Ingeniería de Software: Es la combinación de métodos para las fases del desarrollo del software, herramientas para automatizar estos métodos, y mejores técnicas para la garantía de calidad del software y una.
Ingeniería Inversa: Transformación de software, transformación de código en un modelo a través de traducción desde un determinado lenguaje de programación.

Instancia: Es un objeto de una clase. De acuerdo con el paradigma Orientación a Objetos

Integridad: Se refiere a que la información de un base de datos sea correcta, ya que si se modifica la información en una relación esto debe afectar a toda la información relacionada con esta para que sea íntegra.

Interfaz (ces): Zona de comunicación o acción de un sistema sobre otro.

Internet: Conjunto de redes de ordenadores creada a partir de redes de menos tamaño, cuyo origen reside en la cooperación de dos universidades estadounidenses Es la red global compuesta de limes de redes de área local (LAN) y de redes de área extensa (WAN) que utiliza TCP/IP para proporcionar comunicaciones de ámbito mundial a hogares, negocios, escuelas y gobiernos.

Investigación: Estudio profundo de alguna materia.

Iteración: Conjunto de actividades llevadas a cabo de acuerdo a un plan de iteración y a unos criterios de evaluación, que lleva a producir una versión interna o externa.

Iterativo: Proceso que implica la gestión de una serie de versiones ejecutables.

Java: Es un lenguaje de programación orientado, principalmente, a la programación en Internet o intranets, es un lenguaje de programación para la creación de programas seguros, robustos, multitarea y lo más importante de todo, orientado a objetos. Permite desarrollar aplicaciones muy rápidamente y obtener resultados satisfactorios con muy poco esfuerzo.

Jerarquía: Clasificación de las funciones de acuerdo a una relación de subordinación. Organización de personas o cosas por categorías.

Liberación: Acción de poner en libertad.

Librería: Conjunto de programas de un ordenador electrónico.

Marco Conceptual: Marco de la metodología de investigación en el cual se desarrolla una temática.

Marco Metodológico: Marco de la metodología de investigación en el cual se precisan todas las actividades que se van a realizar con motivo del resultado de la investigación.

Marco Problemático: Marco de la metodología de investigación en el cual se plantea una problemática que se piensa que hay en algún universo.

Marco Teórico: Marco de la metodología de investigación en el cual se profundiza en el conocimiento de algún tema.

Memoria: Elemento esencial de almacenamiento de información. Facultad de recordar.

Middleware: Capa intermedia, capa que ofrece bloques de construcción reutilizables (paquetes o subsistemas) a marcos de trabajo y servicios independientes de la plataforma, para computación con objetos distribuidos, o interoperabilidad en ambientes heterogéneos.
Conjunto de señales, signos o símbolos que son objeto de una comunicación.

Metodología: Conjunto de métodos utilizados en la investigación científica.

Modelado: Técnica que consiste en modelar figuras en una materia.

Modelo: Arquetipo digno de ser imitado que se toma como pauta a seguir.

Modelo Objeto: Es la colección de principios que forman el corazón del diseño orientado a objetos; el paradigma de ingeniería de software enfatiza los principios de abstracción, encapsulamiento, modularidad, jerarquía, tipos, concurrencia y persistencia.

Módulo: Pieza o conjunto unitario de piezas que se repiten o encajan en una construcción de cualquier tipo.

Muestra por Juicio: conjunto de clases empaquetadas

Multimedia: Integración de soportes o procedimientos que emplean sonido, imágenes o textos para difundir información, especialmente si es de forma iterativa.

Notación: Una notación de diseño debe conducir a una representación procedimental que sea fácil de comprender y revisar. Además, la notación debe facilitar la "codificación", de forma que el código se obtenga de hecho como un producto natural del diseño.

Objeto: Es una entidad delimitada precisamente y con identidad, que encapsula al estado y comportamiento. El estado es representado por sus operaciones y métodos. Un objeto es una instancia de una clase. [OMA]

Operación: Realización de algo. Mat. Conjunto de reglas que permiten obtener otras cantidades o expresiones.

Opinión: Idea, juicio o concepto que se tiene sobre alguien o algo.

Ordenador: Sinónimo de computadora.

Organización: Acción y resultado de organizar u organizarse. Formación social o grupo institucionalmente independiente.

Orientado: Situación, colocación, disposición, informe, instrucción, consejo, adiestramiento, encauzamiento, encarrilamiento, guía, enderezamiento, encaminamiento.

Paquete: Conjunto de programas o de datos: ha comprado un paquete de software nuevo para la oficina.

Paradigma: Ejemplo o ejemplar.

Persistencia: Mantenerse firme y constante en alguna cosa.

Plataforma: Base, elemento de apoyo.

Poliformismo: Se refiere a un mismo método con diferentes nombre.

Portabilidad: Capacidad de los productos de software para ser fácilmente transportados de un ordenador a otro: la portabilidad de este juego se anula al instalarlo. Propiedad de un programa o aplicación que le permite funcionar bajo distintos sistemas.

Presupuesto: Cálculo o cómputo anticipado de los ingresos y gastos de un negocio o actividad pública.

Problema: Cuestión o punto discutible que se intenta resolver.

Procedimental: Del procedimiento o relativo a él: desarrolla las habilidades procedimentales de los alumnos.

Proceso: Conjunto de operaciones lógicas y aritméticas ordenadas, cuyo fin es la obtención de unos resultados determinados: han incrementado la velocidad del proceso.

Productividad: Capacidad de producir, ser útil o provechoso: esa vía de investigación carece de productividad.

Programa: Secuencias de instrucciones detalladas y codificadas que sirven para dirigir la actuación y realización de operaciones de un computador electrónico: le han vendido un buen programa de cálculo.

Programación
Codificación de las órdenes y datos que permiten la creación de un programa o aplicación: domina varios lenguajes de programación.

Protocolos: Conjunto de normas establecidas para realizar ciertas cosas.

Prototipo: Modelo o primer ejemplar que representa una virtud o una cualidad.

Proyecto: Plan y disposición detallados que se forman para la ejecución de una cosa.

Puerto: Componente por donde se realiza la entrada y salida de datos de un ordenador: tienes que configurar bien el puerto de la impresora para poder imprimir.

Recursos: Procedimiento o medio del que se dispone para satisfacer una necesidad, llevar a cabo una tarea o conseguir algo.

Red: Conexión simultánea de distintos equipos informáticos a un sistema principal.

Refinamiento: Esmero, cuidado

Reglas: Lo que se debe obedecer o seguir por estar así establecido

Relación: Las relaciones son utilizadas para modelar las conexiones entre entre los elementos del modelo de un sistema.

Requerimientos: Necesidad o solicitud.

Responsabilidades: Cumplimiento de las obligaciones o cuidado al hacer o decidir algo.

Restricción: Reducción, limitación de algo. Limitación impuesta en el suministro de productos de consumo, generalmente por escasez de estos.

Reutilización: Utilización de algo de nuevo.

Robustez: Fuerza, vigor, resistencia, salud.

Rol(Es): Papel que desempeña una persona o grupo en cualquier actividad.

Semántica: Parte de la lingüística que estudia el significado de las palabras.

Servidor: Ordenador que es compartido, en una red informática, por múltiples usuarios.

Simbología: Estudio de los símbolos. Conjunto o sistema de símbolos.

Simula 67: Simula (Simple universal language, Lenguaje universal simple), es un lenguaje general orientado a objetos, soporta en el sistema de clases (sistema de clases Simset) y el proceso discreto orientado a la simulación (sistema de clases Simulation).

Simulación: Fingimiento, presentación de algo como real.

Sistema: Conjunto de elementos que interaccionan entre sí y que realizan una función específica para lograr un objetivo común.

Sistema Operativo: Conjunto de programas para el funcionamiento y explotación de un ordenador, encargado de controlar la unidad central, la memoria y los dispositivos de entrada y salida.

Smalltalk: Es un lenguaje de programación orientado a objetos integrado con un entorno de desarrollo multiventana.

Software: Término genérico que se aplica a los componentes no físicos de un sistema informático, por ejemplo, los programas, sistemas operativos, etc., que permiten a este ejecutar sus tareas.

Soluciones: Hecho de resolver una duda o dificultad.

Subclase: En una relación de generalización la clase que es la especialización de la otra.

Tabla: Se refiere a cada uno de los objetos que resguardan los datos dentro de una base de datos. Representan los objetos (personas, cosas) almacenados que juegan un papel muy importante dentro del sistema. Pueden ser identificados de manera única y ser descritos a través de atributos.
Tarea: Cualquier obra o trabajo. Trabajo que debe hacerse en tiempo limitado.

Tarjetas CRC: Herramienta importante que ayuda a identificar las responsabilidades de las clases y sus relaciones (colaboraciones)con otras clases. Son tarjetas de papel de 4 x 6 pulgadas, en las cuales se coloca: el nombre de la clase, la relación de las responsabilidades y las colaboraciones con otras clases.

Técnica: De la técnica o relativo a ella. Que conoce muy bien los procedimientos de una ciencia, un arte o un oficio y los lleva a la práctica con especial habilidad.

Tecnología: Estudio de los medios, de las técnicas y de los procesos empleados en diferentes ramas de la industria.

Tipo: Es una descripción de un conjunto de instancias que comparten las mismas operaciones, atributos abstractos, relaciones abstractas, y semántica. Un tipo puede definir la especificación de una operación (como su signatura) pero no su implementación (método). Nota sobre el uso: A veces tipo e interface se usan como sinónimos pero no son términos equivalentes.

Transacción: Acuerdo comercial entre personas o empresas.

Triggers: Es un tipo especial de procedimientos almacenados que están ligados a una tabla y son ejecutados automáticamente, se utilizan principalmente para asegurar la integridad referencial de la base de datos.

Unificado: Igualar. Reunir varias cosas o personas para formar un todo homogéneo.

Universo: Conjunto de elementos que configuran la realidad de un individuo

Usuarios: Que habitualmente utiliza algo.

Variable: Que varía o puede varia.

Virtual: Que tiene existencia aparente y no real.

Virus: Programa que se incorpora a un ordenador a través de disquetes u otros sistemas de comunicación, y que se ejecuta automáticamente en determinados momentos modificando o destruyendo los datos contenidos en el ordenador.

Vista de Despliegue: Mapea componentes a nodos de procesamiento, utiliza los diagramas de despliegue para mostrar los diferentes nodos (procesos y dispositivos) en el sistema

Vista de Proceso: Se enfoca en la descomposición de procesos, muestra la distribución de componentes a procesos y direcciona la disponibilidad, confiabilidad, desempeño, administración y sincronización del sistema.

Visualizar: Hacer visible lo que no puede verse a simple vista.

SIGLAS

ADOO:	Análisis y Diseño Orientado a Objetos.
BD:	Base de Datos
BDE:	Ingeniería de Base de datos Borland
CAD:	Diseño Asistido por Computadora
CGI:	Common Gateway Interface
CRC:	Tarjetas de Clase y Colaboración
CU:	Ciudad Universitaria
DF:	Distrito Federal
DOO:	Diseño Orientado a Objetos
ENEP:	Escuela Nacional de Estudios Profesionales
ER:	Entidad Relación
ERA:	Análisis Relacional Extendido
GEDYS:	Generation Dynamique de Schemas de cablage-Dynamic generation of wiring diagrams La generación dinámica de cablear diagramas)
GOOD:	Diseño General Orientado a Objetos
ISE:	Ingeniería de Software Iterativa
JVM:	Máquina Virtual de Java
MOF:	Meta Object Facility
OBA:	Análisis de Componentes de Objetos
OCL:	Object, Constraint Language
OORASS:	Análisis, Síntesis y Estructuración de Papeles Orientado a Objetos
OOSA::	Análisis de Sistemas Orientado a Objetos
OOSD:	Diseño de Sistemas Orientado a Objetos
OOSE:	Ingeniería de Sistemas Orientada a Objetos
OTM:	Técnica de Modelado de Objetos
OWL:	Librerías de Objetos en Windows
POO:	Programación Orientada a Objetos
RDD:	Diseño Controlado por Responsabilidades
RFP:	Propuesta de Requisitos
ROPES:	Rapid Object-Oriented Process for Embedded Systems
RUP:	Rational Unified Process (Proceso Unificado Racional)
SO:	Sistemas Operativos
SQL:	Lenguaje de Consulta Estructurado
SS:	Subdirección de Sistemas
TOO:	Tecnología Orientada a Objetos
UML:	Unified Modeling Lenguage (Lenguaje Unificado de Modelado)
UNAM:	Universidad Nacional Autónoma de México
VLSI:	Componentes de Librerías Visuales
WAE:	Aplicaciones Web Extensibles (WAE-Web Application Extension (Extensión de Aplicación Web

BIBLIOGRAFÍA

Libros:
"El lenguaje unificado de modelado", Grady Booch, James Rumbaugh, Ivar Jacobson; Edit. Addison-Wesley, c1999, ISBN: 84-7829-028-1.

"El proceso unificado de desarrollo de software ", Ivar Jacobson, Grady Booch, James Rumbaugh; Edit. Addison-Wesley; Edición: 2000; ISBN: 0-201-57169-2.

Developing software with UML Object-Oriented Analysis and Design in Practice (Desarrollo de software con UML , Análisis y Diseño en Práctica); Oestereich, Bernard; Edit. Addison Wesley; Edición: 1999; ISBN: 0-201-36826-5.

Fundamental of Object-Oriented Design in UML (Diseño Fundamental Orientado a Objetos en UML); Page-Jones, Meiler; Edit. :Addison-Wesley; Edición: 2000; ISBN: 0-201-69946-X.

Real-Time UML: Developing Efficent Objects from Embedded Systems (UML en Tiempo Real: Desarrollo Eficiente de Objetos para Sistemas Acoplados); Douglas, Bruce Powel; Edit.Addison-Wesley; Edición: 2000; ISBN: 0-201-65784-8.

Patterns in Java (Patrones en Java); Grand, Mark; Edit.John Wiley; Edición: 1999; ISBN: 0-471-25841-5.

Applying UML and Patterns, an introduction to Object Oriented Analysis and Design (Aplicando UML y Patrones, una introducción al Análisis y Diseño Orientado a Objetos); Craig Larman, Edit.Prentice Hall PTR; Edición: 1998; ISBN: 0-13-748880-7.

Using UML, software engineering with Objects and Components (Usando UML, ingeniería de software con Objetos y Componentes); Stevens, Perdita, Pooley, Rob; Edit.Addison Wesley; Edición: 1999; ISBN: 0-21-64860-1.

UML y Patrones: Introducción al Análisis y Diseño Orientado a Objetos; Larman, Craig; Edición: 1999; ISBN: 970-17-0261-1.

"Developing applications with Visual Basic and UML" (Desarrollando aplicaciones con Visual Basic y UML); Autor: Paul R. Reed, Jr.; Edit.:Addison-Wesley; Edición: 2000; ISBN: 2-201-61579-7.

Object oriented systems development (Desarrollo de sistemas orientados a objetos); Ali Bahrami; Edit. Mc Graw – Hill; Edición: 1999; ISBN: 0-256-25348-X.

Object oriented methods (Métodos orientados a objetos); Ian Graham; Edit.Addison – Wesley; Edición: 1996; ISBN: 0-201-56521-8.

Análisis y diseño orientado a objetos con aplicaciones; Grandy Booch; Edit. Addison – Wesley; Edición: 1996; ISBN: 0-201-60122-2.

Business Modeling with UML Business Patterns at work (Modelando negocios con UML, patrones de negocio en el trabajo); Hans - Erik Eriksson Magnus Penker OMG Press; Edit.Willey Computer Publishing John Wiley & Sons, Inc.; Edición: 2000; ISBN: 0-471-29551-5.

Objects, components, and Frameworks with UML, (Objetos, Componentes y Frameworks con UML); Desemond Francis D' Souza, Alan Cameron Wills; Edit. Addison – Wesley; Edición: 1999; ISBN: 0-201-31012-0.

Java modeling color with UML (Modelado de Java a color con UML); Coad, Peter; Lefebvre, Eric; De Luca, Jeff; Edit. Prentice hall; Edición: 1999; ISBN: 0-13-011510-X.

Object – oriented systems análisis and design (Análisis y diseño de sistemas orientados a objetos); Ronald J. Norman; Edit.Prentice may; Edición: 1996; ISBN: 013122946X.

Enterprise Modeling with UML Designing Successful Software Through Business Analysis (Modelado de negocios con UML diseñando exitosamente software a través del análisis de negocio); Chris Marshall; Edit.Adison Wesley; Edición: 2000; ISBN: 0-201-43313-3.

The object constraint language : precise modeling with UML (Restricciones del Lenguaje de Objetos: Modelado exacto con UML); Warmer, Jos B. , Anneke G. Kleppe; Edit.Addison Wesley Longman, c1999 ; Edición: 1999; ISBN: 0201379406.

Análisis y diseño orientado a objetos; James Martín y J. Odell; Edit.Prentice may; Edición: 1994
ISBN: 968-880-362-6.

"Diccionario de la Lengua Española Esencial"; Edit.LAORUSSE; Edición: 1995; ISBN: 970-607-425-2.

"Diccionario Práctico de Sinónimos y Antónimos"; Fernando Corripio; Edit. LAORUSSE; Edición: 1995; ISBN .: 970-607-127-X.

"Ensayo de un diccionario Español de Sinónimos y Antónimos"; Sainz de Robles y Federico Carlos; Editorial : Aguilar; Edición: 1971.

"Diccionario Etimologico Español e Hispanico Etimologico Español Dela Real Académia Española"; Vicente García de Diego; Editorial : S.A.E.T.A.; Edición: 1990.

"Enciclopedia abreviada de ordenadores"; Philip B. Jordain; Editorial : Aguilar; Edición: 1976; ISBN: 84-03-19058-1.

"Diccionario de Informática"; Anthony Candor; Editorial : Alianza Editorial; Edición: 1989
ISBN : 84-206-5235-0.
"Real Académia Española, Diccionario de la Lengua Española Tomo 1 y 2"; Edit,: 21°; Edición: Madrid 1992; ISBN : 84-239-9200-4.

"Diccionario General de Etimología de la Lengua Española Tomo 1 y 2"; D. Roque Barcia; Edición: Madrid 1881

Tesis:
"Documentación del análisis y diseño de sistemas orientados a objetos con UML"; Jiménez Herrada, Jenny; Fecha: 2001; Carrera: Lic. en Informática; Facultad: Facultad de Contaduría y Administración.

"Aplicación de la tecnología orientada a objetos: el lenguaje de modelado unificado (UML) y el proceso unificado de desarrollo de software en un caso de estudio"; Basto Aguilar, Eddie David; Fecha: 2001; Carrera: Maestría en Ciencias de la Computación;
Facultad: Facultad de Ingeniería.

"Una guía práctica para la evaluación de plataformas de desarrollo de software orientado a objetos"; Francisco Alejo Ríos Gascon; Fecha: 1995; Carrera: Maestría en Ciencias de la Computación; Facultad: Colegio de Ciencias y Humanidades – UNAM.

"La metodología de orientación a objetos como una nueva herramienta para el análisis y diseño";Adriana Santos Rosas; Fecha: 1994; Carrera: Ingeniería en Computación; Facultad: Facultad de Ingeniería UNAM.

"Propuesta de políticas normas y procedimientos para la elaboración de sistemas de información con orientación a objetos bajo la norma ISO9000"; Gustavo Adolfo de la Cruz Garces y Angel César Govantes Saldivar; Fecha: 1997; Carrera: Ingeniería en Computación; Facultad: Facultad de Ingeniería UNAM.

Revistas
"Computación y Sistemas Revista Iberoamericana de Computación Publicación Trimestral".
"The journal object oriented programming, JOOP" (El periódico de programación orientada a objetos).
"e – Week" (e – Semanal).
"Sdmagazine", Domicilio: http://www.sdmagazine.com/

Ligas
http://www.e-market.cl/directorio/ingsw/uml/
http://www.omg.org/technology/uml/
http://www.dsic.upv.es/~uml/
http://home.earthlink.net/~salhir/elreusoyuml.htm
http://www-lifia.info.unlp.edu.ar/~cpons/paperscpons.html
http://www2.netexplora.com/artebinario/programacion/curso/capitulo_3.htm
http://www.andrew.cmu.edu/user/conzalez/Teaching/ISW2/ooapproach.html
http://www.monografias.com/trabajos5/insof/insof.shtml
http://cannes.rhon.itam.mx/Alfredo/Espaniol/Publicaciones/IngSW.htm ()archivo objetos.pdf
http://www.ratio.co.uk/white.html
http://www.baufest.com/uml/glosario.htm#Contenidohttp://www.baufest.com/uml/glosario.htm#Contenido
http://gidis.ing.unlpam.edu.ar/personas/glafuente/uml/uml.html
http://www.ratio.co.uk/white.html
http://www.rational.com/uml/index.jsp
http://www.uml-zone.com/articles/Smith01/Smith01-1.asp
http://agamenon.uniandes.edu.co/~pfiguero/soo/uml/
http://www.cetus-links.org/oo_uml.html
http://www.therationaledge.com/admin/archives.html
http://www.uml.org/

http://www.celigent.com/omg/umlrtf/
http://www.dcc.uchile.cl/~cc61j/index.html
http://dcx.ucauca.edu.co/pregrado/proy-inv.html
http://trevinca.ei.uvigo.es/~jcmoreno/is/proceso.html
http://wwwis.cs.utwente.nl:8080/dmrg/OODOC/oodoc/oo.html
http://dec.bournemouth.ac.uk/student/resources/Methodologies.htm;
http://dec.bournemouth.ac.uk/student/resources/Methodologies.htm
http://www.baeuml e.com/info/uml.html
http://www.jeckle.de/uml_pub.htm
http://www.dcc.uchile.cl/~psalinas/uml/introduccion.html
http://www.mazalan.com/cgi-bin/Diccionarios/index.cgi?acc=multiSearch
http://asde.scouts-es.net/gs284/diccionario/descarga.htm
http://www1.ceit.es/enlaces/diccionarios.htm
http://diccionarios.elmundo.es/diccionarios/cgi/lee_diccionario.html
http://www.diccionarios.com/
http://www.lawebdelprogramador.com/diccionario/buscar.php?letra=&cadena=ordenador
http://www.wordzone.com/espanol.htm

Notas de Cursos
Curso: Análisis y Diseño Orientado a Objetos, Dependencia: Dirección de Sistemas; Instructor: José de Jesús; Fecha: 16-27 de octubre de 2000.