7

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Contaduría y Administración

Documentación del Análisis y Diseño de Sistemas Orientados a Objetos con UML

TESIS PROFESIONAL
QUE PARA OBTENER EL TITULO DE:
LICENCIADO EN INFORMATICA
PRESENTA:

Jenny Jiménez Herrada

ASESOR: DR. RICARDO RIVERA SOLER

MEXICO, D.F. **200**

AGRADECIMIENTOS

PROFUNDAMENTE A LA UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO, ME SIENTO ORGULLOSA DE PERTENECER A ELLA.

CON MUCHO CARIÑO Y RESPETO AL DR. RICARDO RIVERA SOLER, POR SU PACIENCIA Y DEDICACIÓN.

A JORGE CON AMOR, QUE SIEMPRE ESTÁ CONMIGO Y ME APOYA EN TODO MOMENTO.

A CARLOS CON CARIÑO SIMPLEMENTE POR ESTAR AHÍ.

Indice

Introducción

Las nuevas tecnologías avanzan a una velocidad vertiginosa. En un período muy corto las computadoras y los sistemas nos han facilitado la vida, pero también nos la han complicado. Para que un sistema funcione de una forma correcta es necesario primero; que nosotros entendamos al sistema y seamos capaces de plantearlo claramente, y después programarlo con instrucciones precisas que den como resultado un sistema que optimice procesos y simplifique tareas.

El trabajo de tesis que presento a continuación es un esfuerzo por resaltar la importancia que tiene la documentación en la realización de un sistema, específicamente orientado a objetos, y la propuesta del uso de una herramienta. La documentación, por muy engorrosa y tediosa que sea, resulta de gran ayuda para el desarrollo de sistemas; que es lo que queda establecido en el primer capítulo de mi tesis, donde planteo la necesidad de realizar una documentación clara y actualizada para el desarrollo de sistemas orientados a objetos. Para lo anterior me base en la opinión de personas que tienen experiencia en la realización de sistemas.

En la realización de cualquier investigación, es necesario como se puede ver en el segundo capítulo, contar con toda la información relacionada con el tema que sea posible. La información con la que disponemos hoy en día es mucha, quizá demasiada, de tal forma que resulta imposible conocerla toda; es por esto que traté de acumular la información que me pareció más relevante. El conocimiento adquirido lo obtuve de libros, revistas, tesis e Internet. Este último es ya una fuente obligada de información, donde afortunadamente siempre es posible encontrar información valiosa.

La orientación a objetos como paradigma de desarrollo de sistemas no siempre ha existido, antes hay otros paradigmas que han ido evolucionando a la par con la evolución del hardware y software. En el tercer capítulo hago una breve reseña de la evolución de los paradigmas de desarrollo hasta llegar a la orientación a objetos y al UML, objeto de estudio de mi tesis, del cual menciono a sus principales creadores y múltiples aportadores.

En el cuarto capítulo, apruebo la hipótesis que plantee en el primer capítulo en base a mis investigaciones y a entrevistas realizadas a profesionistas involucrados con el tema que además, hacen uso del UML en sus actividades de desarrollo de sistemas.

Un trabajo de tesis presupone, o debería presuponer, un aprovechamiento de los conocimientos adquiridos, por lo tanto en el capítulo quinto enumero una serie de actividades que he realizado como resultado del trabajo de tesis.

Finalmente las conclusiones se mencionan al final.

El propósito general de este trabajo de tesis es plantear la importancia que tiene la documentación de los sistemas, en particular los orientados a objetos; y proponer el uso de una herramienta que facilite la creación de la documentación.

No olvidemos, sin embargo, que para que una computadora arroje los resultados deseados, es necesario que nosotros seamos capaces de alimentarla con los datos e instrucciones adecuados.

CAPITULO 1
Marco Problemático

Marco Problemático

1.1 Antecedentes

El tema a cerca del análisis y diseño orientado a objetos me interesa, porque es la base de un paradigma relativamente nuevo de programación, del cual mucha gente conoce y habla, pero que no se lleva acabo de manera correcta.

Conocemos la programación estructurada, que es en la que están basados la mayoría de los sistemas empleados, sin embargo, pocas veces se lleva a cabo un análisis y diseño previo a la programación, que realmente permita realizar el sistema de manera adecuada y ordenada. Por otro lado, la documentación casi es inexistente o la que existe es muy pobre y poco ilustrativa.

En cuanto a la programación orientada a objetos sucede algo similar, comúnmente el interés recae más en la programación que en el análisis y diseño, que son las bases del éxito en la realización de un sistema.

Mi interés en el tema se debe, como mencione con anterioridad, a que es una tecnología nueva y considero importante entender de que manera puede realizarse el análisis y el diseño antes de entrar de lleno en la programación.

El tema se llama propuesta de documentación para realizar análisis y diseño orientado a objetos en base a la metodología de Grady Booch, Jim Rambaugh e Ivar Jacobson, que ha dado como resultado el lenguaje de análisis y diseño **UML** (Unified Modeling Languaje), Lenguaje de Modelado Unificado.

He participado en la elaboración de apuntes para un Diplomado acerca del Análisis y Diseño Orientado a Objetos, mi participación se centró en los conceptos generales, en la importancia del análisis y diseño y en un acercamiento a las metodologías existentes.

1.2 Identificación del Problema

Dentro de las etapas del desarrollo de un sistema, las primeras y quizá las más importantes son el análisis y el diseño, ya sea estructurado u orientado a objetos. Las más importantes, porque si estas dos etapas son realizadas correctamente lo más probable es que el sistema funcione correctamente, y si no, también es posible que las correcciones o modificaciones necesarias puedan llevarse a cabo con facilidad. Sin embargo he observado que el análisis y el diseño rara vez se realizan de forma adecuada, por diversas circunstancias, pero una de ellas es la falta de organización.

Por lo tanto, ya sea el desarrollo estructurado u orientado a objetos, los sistemas casi no cuentan con una documentación que los haga entendibles y accesibles.

Pretendo, pues, proponer una manera de hacer la documentación para el análisis y diseño orientado a objetos de acuerdo a una metodología ya existente.

Las consecuencias son un sistema que no cumple en su totalidad con las expectativas debido a la mala elaboración, en el mejor de los casos, o a la nula elaboración del análisis y diseño previos a las etapas posteriores de programación, pruebas e implementación.

Otra consecuencia es que cuando los desarrolladores del sistema se van sin dejar ningún tipo de documentación a sus sucesores, ellos no pueden realizar modificaciones y mejoras al sistema porque no saben como se llevaron a cabo el análisis y el diseño.

1.3 Demarcación del fenómeno

Mi marco de referencia es el Distrito Federal, el cual por ser una entidad centralizada, cuenta con un ámbito apropiado. Por otra parte debido a que habito en esta entidad y por razones económicas y de tiempo, el Distrito Federal es donde voy a realizar la investigación.

1.4 Conocimiento empírico en el medio

1.4.1 Personas empíricas a entrevistar

- *Alejandro Benitez*

Empleado en Comupuserve Network Services Mexico, ubicada en Gutenberg 143, col. Anzures. Sus actividades son: recopilación de los requerimientos de usuarios para hacer el análisis y diseño de sistemas con lenguajes estructurados y orientados a objetos.

- *Jorge Garcia González*

Estudiante de ingeniería en computación en la U.N.A.M., C.U., empleado de la Embajada de Francia en México, ubicada en Reforma esquina con Laplace. Sus actividades son: actualizar la página de internet de la embajada en sus secciones de cancillería y prensa.

- *Daniel González Ramírez*

Estudiante de ingeniería en computación en la E.N.E.P. Aragón, becario de la Dirección de Cómputo para la Administración Académica. Sus actividades son: brindar cursos, realizar programas con lenguajes estructurados y orientados a objetos con finalidades didácticas.

- *José Antonio Toscano*

Empleado en C.I.F.R.A., en el Departamento de Sistemas, ubicada en Blvd. Manuel Avila Camacho 647, del. Miguel Hidalgo. Sus actividades son: mejorar los sistemas que se usan en la empresa y brindar servicio a usuarios.

- *Jesús Ordaz*

Estudiante de Administración en la facultad de Contaduría y Administración, empleado en la empresa aseguradora Swiss Re, ubicada en Av. de los Constituyentes 956, col. Lomas Altas. Sus actividades son administrativas, actualmente revisa el funcionamiento de aplicaciones realizadas con lenguajes estructurados y orientados a objetos.

1.4.2 Definir preguntas

	Pregunta	Razón	Respuesta esperada
1.	¿Cree usted que la documentación de los sistemas que usa es importante?	Averiguar si la persona considera que la documentación de los sistemas en general es importante	Sí
2.	¿Cree usted que la realización de la documentación de un sistema es necesaria?	Averiguar si la persona no sólo considera a la documentación importante sino también necesaria	Sí
3.	¿Opina que la documentación del análisis y diseño del sistema puede ayudarlo a entender mejor el funcionamiento del sistema?	Averiguar si la persona piensa que específicamente la documentación del análisis y diseño le es útil para entender al sistema	Sí
4.	¿Opina que la documentación del análisis y diseño del sistema le facilita su trabajo?	Averiguar si la persona realiza con mayor facilidad su actividad laboral cuando cuenta con la documentación del análisis y diseño del sistema	Sí
5.	¿Piensa que podría realizar un sistema de manera correcta si no lleva a cabo la documentación del análisis y el diseño?	Averiguar si la persona cree realmente necesario que él mismo realice la documentación del análisis y diseño cuando desarrollar un sistema	No
6.	¿Considera que la documentación existente del análisis y diseño es homogénea?	Averiguar si la persona cuenta con una documentación homogénea, es decir si las personas realizan la documentación de una manera uniforme	No
7.	¿Cree que sería mejor si contara con una guía para hacer la documentación del análisis y diseño del sistema?	Averiguar si la persona está interesada en tener una guía base para realizar la documentación del análisis y diseño	Sí

1.4.3 Recopilación

Alejandro Benítez

1.	¿Cree usted que la documentación de los sistemas que usa es importante?	Sí
2.	¿Cree usted que la realización de la documentación de un sistema es necesaria?	Sí
3.	¿Opina que la documentación del análisis y diseño del sistema puede ayudarlo a entender mejor el funcionamiento del sistema?	Sí
4.	¿Opina que la documentación del análisis y diseño del sistema le facilita su trabajo?	Sí
5.	¿Piensa que podría realizar un sistema de manera correcta si no lleva a cabo la documentación del análisis y el diseño?	No
6.	¿Considera que la documentación existente del análisis y diseño es homogénea?	No
7.	¿Cree que sería mejor si contara con una guía para hacer la documentación del análisis y diseño del sistema?	Sí
8.	¿Por qué no se lleva a cabo la documentación del análisis y diseño de sistemas?	

Yo pienso que es porque las personas no le damos la importancia real, todos sabemos que debe hacerse, pero es un trabajo pesado, y nadie quiere hacerlo.

Jorge García González

1. ¿Cree usted que la documentación de los sistemas que usa es importante?	Si
2. ¿Cree usted que la realización de la documentación de un sistema es necesaria?	Si
3. ¿Opina que la documentación del análisis y diseño del sistema puede ayudarlo a entender mejor el funcionamiento del sistema?	Si
4. ¿Opina que la documentación del análisis y diseño del sistema le facilita su trabajo?	Si
5. ¿Piensa que podría realizar un sistema de manera correcta si no lleva a cabo la documentación del análisis y el diseño?	No
6. ¿Considera que la documentación existente del análisis y diseño es homogénea?	No
7. ¿Cree que sería mejor si contara con una guía para hacer la documentación del análisis y diseño del sistema?	Si
8. ¿Por qué no se lleva a cabo la documentación del análisis y diseño de sistemas? Porque no hacemos las cosas bien, y creemos que si hay una buena programación hay un buen sistema, cuando lo más importante de un sistema es el diseño.	

Daniel González Ramírez

1. ¿Cree usted que la documentación de los sistemas que usa es importante?	Si
2. ¿Cree usted que la realización de la documentación de un sistema es necesaria?	Si
3. ¿Opina que la documentación del análisis y diseño del sistema puede ayudarlo a entender mejor el funcionamiento del sistema?	Si
4. ¿Opina que la documentación del análisis y diseño del sistema le facilita su trabajo?	Si
5. ¿Piensa que podría realizar un sistema de manera correcta si no lleva a cabo la documentación del análisis y el diseño?	No
6. ¿Considera que la documentación existente del análisis y diseño es homogénea?	No

7. ¿Cree que sería mejor si contara con una guía para hacer la documentación del análisis y diseño del sistema?	Sí
8. ¿Por qué no se lleva a cabo la documentación del análisis y diseño de sistemas? Yo pienso que por las siguientes razones: 1. Si son gente de computo no saben expresar sus ideas, sería expresado en términos muy técnicos y sólo ellos se entenderían 2. No existe el hábito 3. No se delega el trabajo, se quiere hacer todo 4. Porque no llevamos una metodología de trabajo	

José Antonio Toscano

1. ¿Cree usted que la documentación de los sistemas que usa es importante?	Sí
2. ¿Cree usted que la realización de la documentación de un sistema es necesaria?	Sí
3. ¿Opina que la documentación del análisis y diseño del sistema puede ayudarlo a entender mejor el funcionamiento del sistema?	Sí
4. ¿Opina que la documentación del análisis y diseño del sistema le facilita su trabajo?	Sí
5. ¿Piensa que podría realizar un sistema de manera correcta si no lleva a cabo la documentación del análisis y el diseño?	No
6. ¿Considera que la documentación existente del análisis y diseño es homogénea?	No
7. ¿Cree que sería mejor si contara con una guía para hacer la documentación del análisis y diseño del sistema?	Sí
8. ¿Por qué no se lleva a cabo la documentación del análisis y diseño de sistemas? En mi caso personal porque no hay suficiente tiempo, yo pienso que es por falta de organización y las personas no tenemos bien definidas nuestras funciones, entonces, todos hacemos trabajo de todos.	

Jesús Ordaz

1. ¿Cree usted que la documentación de los sistemas que usa es importante?	Sí
2. ¿Cree usted que la realización de la documentación de un sistema es necesaria?	Sí
3. ¿Opina que la documentación del análisis y diseño del sistema puede ayudarlo a entender mejor el funcionamiento del sistema?	Sí
4. ¿Opina que la documentación del análisis y diseño del sistema le facilita su trabajo?	Sí

5. ¿Piensa que podría realizar un sistema de manera correcta si no lleva a cabo la documentación del análisis y el diseño?	No
6. ¿Considera que la documentación existente del análisis y diseño es homogénea?	No
7. ¿Cree que seria mejor si contara con una guía para hacer la documentación del análisis y diseño del sistema?	Sí
8. ¿Por qué no se lleva a cabo la documentación del análisis y diseño de sistemas? No se lleva a cabo porque no sabemos como hacerlo y cada quien lo hace como piensa que debe ser. Lo cual crea desorganización y confusión.	

1.4.4 Conclusiones

1. ¿Cree usted que la documentación de los sistemas que usa es importante?
Conclusión: Las personas entrevistadas consideran que es importante que exista la documentación de los sistemas.
2. ¿Cree usted que la realización de la documentación de un sistema es necesaria?
Conclusión: Las personas piensan que no sólo es importante que exista la documentación de sistemas, sino también necesaria.
3. ¿Opina que la documentación del análisis y diseño del sistema puede ayudarlo a entender mejor el funcionamiento del sistema?
Conclusión: Todos los entrevistados coinciden en que la documentación de sistemas los ayuda a entender mejor el funcionamiento del mismo. Con lo cual se reafirma la importancia de la existencia de la documentación.
4. ¿Opina que la documentación del análisis y diseño del sistema le facilita su trabajo?
Conclusión: Todos los entrevistados manejan sistemas en su trabajo, por lo tanto saben que la documentación les facilita el uso de dichos sistemas. Entonces la documentación les facilita su desempeño laboral.
5. ¿Piensa que podría realizar un sistema de manera correcta si no lleva acabo la documentación del análisis y diseño?
Conclusión: Algunos de los entrevistados definitivamente dijeron que no en base a su experiencia, dos más dijeron que si pero con algunas dificultades; por ejemplo que sólo ellos entenderían al sistema, aunque podría olvidárseles algunas cosas. Esto demuestra que la documentación es importante tanto para la o las personas que realizan el sistema como para los que no lo realizaron pero desean realizar alguna modificación.
6. ¿Considera que la documentación existente del análisis y diseño es homogénea?

Conclusión: Los entrevistados no consideran que exista una documentación homogénea. Entonces cada quien realiza su documentación (si la realiza) como se acomode o entienda.
7. ¿Cree que sería mejor si contara con un guía para hacer la documentación del análisis y diseño del sistema?
Conclusión: Todos lo entrevistados piensan que una guía les sería muy útil. Una guía práctica y bien definida podría ser de gran utilidad para las personas que diseñan sistemas.
8. ¿Por qué no se lleva acabo la documentación del análisis y diseño de sistemas?
Conclusión: Las respuestas toman diversos factores como falta de organización de trabajo en general, ignorancia, no darle la importancia debida, no saber expresar ideas. Por lo tanto es importante que haya una guía para realizar la documentación del análisis y diseño de sistemas orientados a objetos.

1.4.5 Conclusión General

Las personas entrevistadas anteriormente de alguna manera se relacionan con el uso, rediseño y modificación de sistemas sin ser profesionales, es decir su conocimiento y experiencia se ha dado en la practica. Estas personas saben que la documentación es importante, pero no se realiza porque en general, o no se tiene el conocimiento para hacerla, o no hay suficiente organización y además hay falta de definición de funciones. Lo anterior me indica que aunque las personas entienden que la documentación es importante y necesaria, en pocas ocasiones es realizada de forma adecuada. Hace falta una guía que ayude a definir la estructura de la documentación, aunque no se haga exactamente igual, pero que sí sea una guía útil, sencilla y con conceptos que podrían no conocerse o no estar muy claros.

1.5 Opiniones profesionales

1.5.1 Personas profesionales a entrevistar

- *Ing. Alexei Dezoti*

Ingeniero en Computación. Encargado del departamento de Estructuración de la Dirección de Cómputo para la Administración Académica. Sus actividades son: coordinar el análisis y el diseño de los proyectos internos y externos a la UNAM que son solicitados.

- *Ing. Gabriel Chavarría*

Ingeniero en Telecomunicaciones. Empleado en la empresa Nortel Networks ubicada en Insurgentes sur 1605, piso 10, col. San José Insurgentes. Sus actividades son el análisis y diseño de redes, así como modificación y mejoramiento de las mismas. Lo anterior con el uso de metodologías estructuradas y orientadas a objetos.

- *Ing. Ahmed Concha Dimas*

Ingeniero en Computación. Ha realizado varios programas usando la metodología orientada a objetos. Fue becario del Insituto de Ingeniería. Actualmente gestiona por una beca en el extranjero.

- *Lic. Gustavo Castañeda*

Licenciado en Informática. Empleado en el banco Mexicano. Sus actividades son: programador auxiliar en la conversión de los sistemas para el año 2000. Algunos de los sistemas están programados con lenguajes orientados a objetos.

- *Lic. Humberto Fresnedo*

Licenciado en Economía. Asesor técnico del director de PEMEX. Una de sus funciones es trabajar coordinando y supervisando algunos proyectos de informática.

1.5.2 Definir preguntas

Pregunta	Razón	Respuesta esperada
1. ¿Cree que siempre es necesario realizar la documentación del análisis y diseño de un sistema?	Averiguar si la persona considera que siempre es necesario realizar la documentación, o si sólo en algunas ocasiones	Sí
2. ¿Considera que la realización del análisis y diseño y su documentación respectiva, son más importante que la programación e implementación de un sistema?	Averiguar si la persona considera que lo más importante en el desarrollo de un sistema es el análisis y el diseño y su documentación respectiva	Sí
3. ¿Al revisar una documentación del análisis y diseño de un sistema, ésta siempre es clara y obtiene la información que necesita para entender el funcionamiento del sistema?	Averiguar si la persona siempre cuenta con documentación clara y en la que encuentre la información suficiente para entender cómo funciona el sistema	No
4. ¿Piensa que la documentación del análisis y diseño que se realiza para los diferentes sistemas es homogénea?	Averiguar si la persona piensa que existe un estándar en cuanto a la realización de la documentación del análisis y diseño	No
5. ¿Cree que sería conveniente tener un modelo a seguir para realizar la documentación y el análisis de los sistemas?	Averiguar si la persona está interesada en que exista un modelo a seguir para realizar la documentación del análisis y diseño	Sí

1.5.3 Recopilación

Ing. Alexei Dezoti

1. ¿Cree que siempre es necesario realizar la documentación del análisis y diseño de un sistema?	Sí
2. ¿Considera que la realización del análisis y diseño y su documentación respectiva, son más importante que la programación e implementación de un sistema?	Claro

3. ¿Al revisar una documentación del análisis y diseño de un sistema, ésta siempre es clara y obtiene la información que necesita para entender el funcionamiento del sistema? No, desgraciadamente son pocas las documentaciones que dan una idea real del funcionamiento del sistema, esto sucede o porque no se documento de manera correcta, o porque no se actualiza de acuerdo a los cambios que se van dando.	
4. ¿Piensa que la documentación del análisis y diseño que se realiza para los diferentes sistemas es homogénea?	No
5. ¿Cree que sería conveniente tener un modelo a seguir para realizar la documentación y el análisis de los sistemas?	Sí, creo que sería muy útil

Ing. Gabriel Chavarría

1. ¿Cree que siempre es necesario realizar la documentación del análisis y diseño de un sistema?	Sí
2. ¿Considera que la realización del análisis y diseño y su documentación respectiva, son más importante que la programación e implementación de un sistema?	Claro
3. ¿Al revisar una documentación del análisis y diseño de un sistema, ésta siempre es clara y obtiene la información que necesita para entender el funcionamiento del sistema? Bueno para empezar es difícil que exista la documentación de un sistema, generalmente nadie sabe dónde está o si alguien la hizo o de plano nadie la hizo. Por otro lado son pocas las personas que saben hacer y hacen una documentación completa y útil.	
4. ¿Piensa que la documentación del análisis y diseño que se realiza para los diferentes sistemas es homogénea?	No
5. ¿Cree que sería conveniente tener un modelo a seguir para realizar la documentación y el análisis de los sistemas? Yo creo que sí, pero lo más importante es que la gente supiera hacerla y entendiera su importancia.	

Ing. Ahmen Concha Dimas

1. ¿Cree que siempre es necesario realizar la documentación del análisis y diseño de un sistema?	Sí

2. ¿Considera que la realización del análisis y diseño y su documentación respectiva, son más importante que la programación e implementación de un sistema?	Sí
3. ¿Al revisar una documentación del análisis y diseño de un sistema, ésta siempre es clara y obtiene la información que necesita para entender el funcionamiento del sistema?	No, casi nunca
4. ¿Piensa que la documentación del análisis y diseño que se realiza para los diferentes sistemas es homogénea?	No
5. ¿Cree que sería conveniente tener un modelo a seguir para realizar la documentación y el análisis de los sistemas? Sí, pero creo que sería muy difícil a grandes niveles, aunque a niveles de organización, sería muy útil.	

Lic. Gustavo Castañeda

1. ¿Cree que siempre es necesario realizar la documentación del análisis y diseño de un sistema?	Sí
2. ¿Considera que la realización del análisis y diseño y su documentación respectiva, son más importante que la programación e implementación de un sistema?	Sí
3. ¿Al revisar una documentación del análisis y diseño de un sistema, ésta siempre es clara y obtiene la información que necesita para entender el funcionamiento del sistema?	No, muy pocas veces
4. ¿Piensa que la documentación del análisis y diseño que se realiza para los diferentes sistemas es homogénea?	No
5. ¿Cree que sería conveniente tener un modelo a seguir para realizar la documentación y el análisis de los sistemas? Sí, sería muy útil para lograr una mejor organización, aunque sería necesario contar también con una buena coordinación en los equipos de trabajo grandes.	

Lic. Humberto Fresnedo

1. ¿Cree que siempre es necesario realizar la documentación del análisis y diseño de un sistema?	Sí
2. ¿Considera que la realización del análisis y diseño y su documentación	Sí

respectiva, son más importante que la programación e implementación de un sistema?	
3. ¿Al revisar una documentación del análisis y diseño de un sistema, ésta siempre es clara y obtiene la información que necesita para entender el funcionamiento del sistema? No siempre es clara, si es que existe y está actualizada	
4. ¿Piensa que la documentación del análisis y diseño que se realiza para los diferentes sistemas es homogénea? No, cada equipo de trabajo lleva a cabo la documentación según sus propios estatutos, pero no siempre tienen una forma organizada para llevarla a cabo.	
5. ¿Cree que sería conveniente tener un modelo a seguir para realizar la documentación y el análisis de los sistemas? Sí, yo creo que sería de gran ayuda tanto para las personas que realizan la documentación como para las personas que necesitan estudiar la documentación o revisarla o buscar algo.	

1.5.4 Conclusiones

1. ¿Cree que siempre es necesario realizar la documentación del análisis y diseño de un sistema?
Conclusión de la pregunta 1. Las personas entrevistadas consideran que siempre es necesario realizar la documentación del análisis y diseño. Por lo tanto es algo muy importante.
2. ¿Considera que la realización del análisis y diseño y su documentación respectiva, son más importantes que la programación e implementación de un sistema?
Conclusión: El análisis y diseño son muy importantes y deben realizarse antes de seguir con la programación e implementación, ya que son la base.
3. ¿Al revisar una documentación del análisis y diseño de un sistema, ésta siempre es clara y obtiene la información que necesita para entender el funcionamiento del sistema?
Conclusión: La información contenida en la documentación, que existe, del análisis y diseño, no siempre es clara y a veces no está actualizada o está incompleta. Es necesaria una documentación consistente.
4. ¿Piensa que la documentación del análisis y diseño que se realiza para los diferentes sistemas es homogénea?

Conclusión: La documentación nunca es homogénea, quizá no debe ser toda exactamente igual, pero si se siguen ciertos parámetros será más fácil entenderla porque la documentación será parecida. Y si alguien quiere hacer un cambio, lo hará de acuerdo a una guía de acción.
5. ¿Cree que sería conveniente tener un modelo a seguir para realizar la documentación y el análisis de los sistemas?
Conclusión: Es importante contar con un modelo a seguir para realizar la documentación, pero no hay que olvidar que también es importante entender su importancia y contar con los conocimientos suficientes para llevarla acabo.

1.5.5 Conclusión General

Es importante aclarar que las personas entrevistadas que se dedican al diseño, programación e implementación de sistemas, también cuentan con un título universitario, es decir son profesionales. Estas personas coinciden con mi planteamiento y concluyo que el análisis y diseño de sistemas es muy importante, más aún que la programación e implantación, ya que es la base de los dos últimos. Asi también la documentación del análisis y del diseño es muy importante ya que en base a ésta se pueden realizar cambios y actualizaciones con facilidad. Pero por desgracia la documentación no siempre es clara y nunca es homogénea. Sin embargo una guía sería muy útil porque ella facilitaría la realización de la documentación y aclararía algunos conceptos que muchos usamos, pero que no están del todo claros.

1.6 Hipótesis preliminar

1.6.1 Variable Independiente

Si una persona o grupo de personas que deseen realizar un sistema con el paradigma de orientación a objetos, tienen el conocimiento de cómo llevar a cabo el análisis y diseño de una manera clara y con guía que los oriente y que les proponga una manera de hacer el análisis y diseño orientado a objetos.

1.6.2 Variable Dependiente

- Contarán con un documento que valide los requerimientos que el usuario haya solicitado.
- Podrán modificar con mayor facilidad los requerimientos del usuario en la etapa de análisis.
- Realizarán un documento que paso por paso indique el diseño total y por partes del sistema.
- Contarán con documentación al dia, es decir actualizada de acuerdo a los cambios que se le hagan al sistema.

1.6.3 Hipótesis

Si una persona o grupo de personas, que van a realizar un sistema usando el paradigma de orientación a objetos, saben como realizar el análisis y diseño, y además cuentan con una guía para hacer la documentación correspondiente, las probabilidades de que tengan éxito en la realización, modificación, corrección, mejora o migración del sistema serán muy grandes.

1.7 Objetivo (Marco justificatorio)

Personales:

- Graduarme

Reglamento General de Exámenes. Capítulo IV. Exámenes Profesionales y de Grado. Artículos 18, 19, 20 y 21.

- Dominar el tema

  Ser experta en el análisis y diseño orientado a objetos usando el lenguaje **UML**, para trabajar en algo relacionado con ello y seguir investigando nuevas metodologías.

Generales:

- Una guía para realizar el análisis y diseño orientado a objetos usando **UML**
- Un método de capacitación para las personas que estén interesadas con este paradigma o que están ya trabajando en algo relacionado a ello
- Expresar mi criterio en cuanto a la importancia del análisis y diseño orientado a objetos.
- Probar una hipótesis.

Particulares:

- Aportar una temática para un curso de análisis y diseño orientado a objetos, para la carrera de informática y otras instituciones interesadas.

CAPITULO 2
Marco Teórico

Marco Teórico

2.1 Acopio de libros

2.1.1 Libros de estudio

Datos del libro:

Título:	**Análisis y Diseño Orientado a Objetos con Aplicaciones**
Autor:	Grady Booch
Editorial:	Addison Wesley/Dias de Santos
Edición:	1996
Edición:	Segunda
ISBN:	0-201-60122-2

Resumen

Capítulo 1. Complejidad

Este capitulo trata sobre la complejidad del software y la capacidad de abstracción con la que cuentan los humanos para entender sistemas complejos. A partir de lo anterior es expresada una serie de beneficios al hacer uso del diseño orientado a objetos para dominar la complejidad asociada al desarrollo de sistemas de software. Finalmente, el autor da una breve explicación del enfoque del diseño orientado a objetos.

Capítulo2. El modelo de objetos

Mención y explicación de los diferentes paradigmas de programación que existen: orientados a procedimientos, orientados a objetos, orientados a lógica, orientados a reglas y orientados a restricciones. Se definen varios conceptos fundamentales del paradigma orientado a objetos, que son abstracción, encapsulamiento, modularidad, jerarquía, tipos, concurrencia, persistencia.

Breve explicación de lo que es la programación, diseño y análisis orientados a objetos y mención del beneficio del modelo de objetos.

Capítulo 3. Clases y objetos

Este capítulo diferencia lo que puede ser y no puede ser un objeto y una clase.

Una vez identificados y definidos los objetos, estos pueden tener un estado, un comportamiento, operaciones, roles y responsabilidades e identidad. Entre ellos existen dos tipos de relaciones: los enlaces y la agregación.

Una vez identificados y definidas las clases, éstas son elementos importantes en la interfaz e implementación. Tienen un ciclo de vida y relaciones entre ellas. Estas relaciones pueden ser: asociación, herencia, agregación, uso, instanciación y metaclase.

Entre las clases y objetos existe Interacción y relaciones. Se explica como deben construirse clases y objetos de calidad y hasta cuando debe realizarse la abstracción de objetos. Ya que calidad de una abstracción puede medirse por su acoplamiento, cohesión, suficiencia, completud y por el grado hasta el cual es primitiva.

Capítulo 4. Clasificación

Menciona lo que es la clasificación, cual es su importancia y la dificultad que existe y ha existido siempre que existe la clasificación.

Explicación más profunda de la identificación de clases y objetos en relación con las aproximaciones a la clasificación, que son: categorización clásica, agrupamiento conceptual y teoría de prototipos.

Diferencia entre el análisis y el diseño orientado a objetos, ya que mientras el primero pretende modelar el mundo descubriendo clases y objetos que forman el vocabulario del dominio del problema, en el segundo las abstracciones y mecanismos que proporcionan el comportamiento que este modelo requiere son inventadas.

Explicación de cómo se realiza el análisis de dominio y el de casos de uso mediante el uso de fichas CRC y descripción formal en español del análisis del dominio.

Menciona cómo buscar abstracciones clave, es decir cuales son las clases u objetos que forman parte del vocabulario del dominio del problema. Finalmente explica como identificar mecanismos que denotan decisiones estratégicas de diseño respecto a la actividad de colaboración entre muchos tipos diferentes de objetos.

Capítulo 5. El método

Explicación del método que debe seguirse para realizar el diseño, donde lo importante no es únicamente dibujar un diagrama, también es importante la estructura física - lógica y la semántica estática - dinámica.

Notación para el desarrollo orientado a objetos, en base a los estudios del autor del libro, que incluye cuatro diagramas básicos: de clases, de objetos, de módulos y de procesos, así como de transición de estados y de interacción. El diagrama de clases muestra las clases y sus relaciones; el de objetos muestra los objetos y sus relaciones; el de módulos muestra la asignación de clases y objetos a módulos de diseño físico del sistema; el de procesos muestra la asignación de procesos a procesadores en el diseño físico de un sistema; el de transición muestra el espacio de estados de una instancia de una clase dada, los eventos que causan una transición de un estado a otro y las acciones que se derivan de un cambio de estado; y el de interacción sirve para seguir la pista de la ejecución de un escenario en el mismo contexto que un diagrama de objetos.

Capítulo 6. El proceso

Explicación del proceso que debe seguirse para el diseño, donde debemos tomar en cuenta el ciclo de vida iterativo e incremental. El proceso general es identificar clases y objetos a nivel de abstracción, luego identificar la semántica de estas clases y objetos, después identificar sus relaciones, una vez identificado lo anterior implantar estas clases y objetos. Otra etapa es la conceptualización, crear una arquitectura para la implantación y establecer políticas comunes, después refinar sucesivamente para llegar finalmente al sistema en producción; la etapa final es el mantenimiento, es decir la gestión de la evolución post-entrega.

Capítulo7. Aspectos pragmáticos

Menciona algunos aspectos importantes que deben tomarse en cuenta al diseñar un sistema. Por ejemplo; una vez generado el código es importante realizar recorridos de inspección, debe institucionalizarse la reutilización para tener éxito. También es importante poner atención en la tasa de descubrimiento de defectos y su densidad. El desarrollo orientado a objetos requiere herramientas sutilmente diferentes que los sistemas no orientados a objetos. La transición de una organización hacia el uso de modelo de objetos requiere un cambio de mentalidad. Existen ventajas y riesgos al implantar técnicas orientadas a objetos, el autor considera que las ventajas sobrepasan los riesgos.

Capítulos 8, 9, 10, 11 y 12

Estos capítulos presentan varios ejemplos en casos reales de diferente índole desde una estación de monitoreo del clima hasta la gestión del tráfico de trenes, pasando por el control de una biblioteca de clases, el mantenimiento de catálogos de clientes y criptoanálisis. Todos con ejemplos programados en C++.

Apéndice

Contiene los principales lenguajes orientados a objetos según el autor: Smalltalk, Object Pascal, C++, Common Lisp Object System, Ada e Eiffel.

Datos del libro:

Título:	**Object-Oriented Software Engineering (Ingeniería del Software orientado a objetos)**
Autor:	Ivar Jacobson, Magnus Christerson, Patrik Jonsson y Gunnar Övergaard
Editorial:	Addison Wesley
Edición:	1995
Edición:	Séptima
ISBN:	0-201-54435-0

Resumen

Parte I Introduction (Introducción)

Capítulo 1. System development as an industrial process (Desarrollo de sistemas como un proceso industrial)

En este capítulo se plantea una analogía entre los procesos industriales bien establecidos en la industria de la construcción y la industria del software. El objetivo de esta analogía es comparar y entender aspectos relacionados, como: arquitectura, método, proceso y herramientas; los cuales son comunes a ambas ramas y funcionan de manera parecida.

También se mencionan las características más importantes del desarrollo del software, y se introduce el objetivo de la metodología propuesta que se llama OOSE (Object –Oriented Software Engineering) Ingeniería del Software Orientado a Objetos.

Capítulo2. The system life cycle (El ciclo de vida del sistema)

Estudio del desarrollo de sistemas, el autor considera que el desarrollo de sistemas es un proceso de cambio constante ya que la creación y mantenimiento de sistemas evoluciona con las nuevas tecnologías y tendencias.

En este capítulo se menciona el hecho de que los sistemas grandes tienen cambios durante su ciclo de vida, lo cual debe considerarse en un proceso industrial. El desarrollo de sistemas debe darse con reglas y estándares establecidos.

Capítulo 3. What is object-orientation? (¿Qué es orientación a objetos?)

Este capítulo da una breve introducción de lo que es Orientado a Objetos, que es una técnica para desarrollar modelos. Por lo tanto se supone que sean fáciles de entender. Explica lo que es un objeto, o sea, una representación de una entidad con comportamiento e información. Solo las

operaciones que pueden desempeñarse en el objeto se ven exteriormente. Los objetos están relacionados y el sistema funciona cuando los objetos empiezan a enviarse estímulos. Una clase describe las similitudes entre objetos. Por lo que todo objeto es una instancia de una clase. Se usa el polimorfismo para hacer más flexible la descripción de las clases. Explicación de la herencia, herramienta muy poderosa cuando es bien empleada, porque cuando se emplea incorrectamente da resultados confusos y modelos difíciles de mantener. Explicación de lo que es y como se usa la herencia múltiple.

Capítulo 4. Object-oriented system development (Desarrollo de sistemas orientados a objetos)

En este capítulo hay una propuesta de cómo deben desarrollarse sistemas de software mediante un método de desarrollo. Cada método se enfoca a propiedades diferentes. Además se menciona que uno de los modelos más comunes es el de cascada, aunque la mayoría son de naturaleza iterativa.

Los métodos tradicionales tratan a las funciones y a los datos por separado, lo cual crea algunos problemas, mientras que los métodos orientados a objetos consideran a las funciones y a la información como un todo. El objetivo del análisis orientado a objetos es entender el sistema para después desarrollar y construir un modelo lógico del sistema. Este modelo debe estar basado en objetos naturales del dominio del problema.

En la implementación el objetivo es mantener la estructura lógica del modelo de análisis en el sistema final. Y se debe poner especial atención a la realización continua de pruebas al sistema.

Capítulo 5. Object-oriented programming (Programación orientada a objetos)

La programación orientada a objetos es un estilo de programación donde se usan y aplican los conceptos de orientación a objetos. Lo que se programa son las clases. En cada clase es necesario definir las variables y operaciones asociadas con la clase e instancias de la clase.

La herencia en este tipo de programación proporciona un mecanismo para crear clases nuevas y modificar las existentes. Sin embargo la herencia no es un pre-requisito del re-uso. También puede implementarse el polimorfismo usando ligas dinámicas.

Parte II Concepts (Conceptos)

Capítulo 6. Architecture (Arquitectura)

Introducción a la arquitectura OOSE. Esta arquitectura consiste en cinco modelos diferentes, las transiciones de estos modelos se repiten continuamente, es decir, diferentes desarrolladores repiten el modelo. Las transiciones entre estos modelos se manejan por medio de procesos, se usan cuatro procesos: análisis, construcción, prueba y componentes. Se manejan objetos en todos los modelos.

El primer modelo es el modelo de requerimientos, que consiste en actores y casos de uso soportados por un modelo de dominio de objeto y descripciones de interfaz. El modelo de análisis se desarrolla a partir del modelo de requerimientos, su objetivo es construir una estructura lógica y robusta fácil de mantener durante el ciclo de vida del sistema. El modelo de diseño refina el modelo de análisis y considera el ambiente de implementación actual. El modelo de implementación consiste principalmente del código fuente para implementar bloques. Y el modelo de prueba se desarrolla al probar el sistema.

Capítulo 7. Analysis (Análisis)

Explicación del modelo de análisis. El proceso de análisis tiene como objetivo definir y especificar el sistema que va a construirse. Se desarrollan dos modelos, el modelo de requerimientos y

el de diseño. Ambos modelos son lógicos en el sentido de que no incorporan requerimientos del ambiente de implementación actual.

El modelo de requerimientos usa actores y casos de uso para describir con detalle cada forma de usar el sistema desde el punto de vista del usuario. Los actores modelan factores externos, como gente o máquinas que interactúan con el sistema. Los casos de uso son flujos que estos actores desempeñan en el sistema.

El modelo de análisis pretende formar una estructura lógica y fácil de mantener en el sistema. Se enfoca esencialmente en la funcionalidad del sistema.

El desarrollo de estos modelos forman un proceso iterativo, antes de que se convierta en un modelo estable, debido a las correcciones y modificaciones que deben hacerse con el o los usuarios.

Capítulo 8. Construction (Construcción)

En la etapa de construcción el diseño y la implementación del sistema son realizados. En el capítulo anterior se describe el análisis,que consiste en la descripción del sistema bajo condiciones ideales, su propósito es lograr una estructura robusta y fácil de mantener durante el ciclo de vida entero del sistema. En esta capítulo, referido a la construcción, la tarea es adaptar este modelo ideal a las condiciones existentes. En la construcción se deben tomar en cuenta todos los elementos con los que se cuenta para la realización del sistema, lo cual bien puede hacerse de forma paralela con el análisis.

El modelo de diseño es refinado usando los casos de uso especificados en el análisis, se dibujan diagramas de iteración, se definen estímulos enviados entre objetos, después se estructuran e implementan los objetos usando uno o varios módulos de objetos. Lo siguiente es considerar el lenguaje de programación que va a usarse, si el lenguaje no es orientado a objetos, aunque puedan simularse algunas características, se deberán hacer algunos cambios.

Finalmente hay que considerar métodos de abstracción a varios niveles; como los subsistemas, los cuales dividen el modelo de diseño para facilitar la definición de interfaces explícitas.

Capítulo 9. Real-time specialization (Especialización en tiempo real)

Este capítulo trata el tema de la especialización en tiempo real, pasando por todas las fases descritas anteriormente, que son: análisis, construcción prueba y verificación. Primero hay una explicación de lo que es un sistema de tiempo real y cuáles deben ser sus características, algunas de éstas son: procesos, sincronización y comunicación. Resalta la importancia de crear un OOSE especializado para cada implementación del ambiente específico.

Durante cada etapa son especificados algunos elementos importantes aparte de los ya explicados en capítulos anteriores para adaptarlos a sistemas de tiempo real.

Capítulo 10. Database specialization (Especialización de Bases de Datos)

Explicación de la función de las bases de datos, que es almacenar objetos de forma persistente. Se menciona que el uso de DBMS Sistemas de Manejo de Bases de Datos (Data Base Management System) proporcionan soporte para la persistencia que se necesita, aunque también dice que el nuevo enfoque de las bases de datos es relacional, que almacena datos en tablas. Finalmente menciona la importancia de que los vendedores de sistemas para bases de datos se adapten a los nuevos paradigmas de programación para ir a la par con éstos. La base de datos que para los autores facilita su trabajo en este enfoque es ODBMS Sistema de Manejo de Bases de Datos de Objetos (Object Data Base Management System).

Capítulo 11. Components (Componentes)

Este capítulo se enfoca al re-uso del software, el cual es poco usado, sin embargo con el enfoque orientado a objetos se le ha dado una mayor importancia y difusión. El re-uso de componentes de software se basa principalmente en el re-uso de la programación. Se le da tal importancia a los componentes que incluso se menciona que un modelo orientado a objetos en el cual no se da el re-uso de componentes no es totalmente correcto.

Ahora bien, también la administración de los componentes es importante, ya que sin ella, el re-uso podría resultar complicado y confuso.

Capítulo 12. Testing (Pruebas)

La prueba es una actividad que se lleva a cabo para comprobar que el sistema está siendo construido de manera correcta. La prueba debe darse desde las primeras etapas del desarrollo de un sistema, para poder corregir errores desde el principio y no cuando el sistema está ya terminado o casi terminado. Existen varios tipos y técnicas de pruebas, pero cada desarrollador debe elegir la o las que más le convengan para el tipo de sistema que este realizando.

Entonces la prueba debe llevarse a cabo desde el inicio del desarrollo de un sistema de manera continua y al final debe realizarse una prueba integral del sistema.

Parte III Aplications (Aplicaciones)

Capítulo 13. Case study: warehouse management system (Estudio de Caso: sistema de administración *warehouse*)

Este capítulo da un ejemplo aplicando la metodología descrita en este libro, que es la construcción de un sistema de administración de warehousing, pasando por todas las etapas explicadas con anterioridad.

Capítulo 14. Case study: telecom (Estudio de Caso: telecom)

Otro ejemplo más de la aplicación de la metodología, pero ahora con sistemas de conexión de telecomunicaciones.

Capítulo 15. Managing objetc-oriented software engineering (Administración de ingeniería de software orientada a objetos)

Este capítulo menciona el cuidado que debe tenerse cuando se implementa una nueva metodología para el desarrollo de sistemas, en este caso la orientada a objetos, específicamente la OOSE. Estos cambios deben considerarse en un plan a largo plazo en el desarrollo de la organización. Por lo tanto debe planearse un programa de entrenamiento y educación para integrar a los desarrolladores con la nueva metodología. Además cada organización debe tomar las medidas necesarias y particulares para lograr un cambio exitoso. Las técnicas más conocidas son : revisión, prueba y métrica, la sugerida es la última, ya que es la que asegura un mayor logro de calidad.

Capítulo 16. Other object-oriented methods (Otros métodos orientados a objetos)

Este capítulo da una breve explicación de los diferentes métodos orientados a objetos que existen y toma en cuenta los más sobresalientes, para hacer incluso una tabla comparativa. Los que se consideran son: OOSE Ingeniería de Sistemas Orientados a Objetos (Object Oriented System Engineering) de los autores de éste libro, OOA Análisis Orientado a Objetos(Object Oriented Analysis) de Peter Coad y De Yourdon), OOD Diseño Orientado a Objetos (Object Oriented Design) de Grady Booch, HOOD Diseño Orientado a Objetos Jerárquicamente (Hierarchical Object Oriented Design) de la European Space Agency, OMT Técnica del Modelado de Objetos (Object Modeling

Technique) de Rumbaugh et al. y Diseño Dirigido a Responsabilidades (Responsability-Driven Design) de WirfsBrock.

Apéndice A On the development of Objectory (En el desarrollo de Objectory)

En este apéndice se explica el funcionamiento del software Objectory como herramienta para ayudar al análisis y diseño de sistemas usando la metodología descrita en este libro. Menciona ventajas y desventajas del uso de este software y el desarrollo de la herramienta desde sus inicios hasta la última versión.

Datos del libro:

Título:	**Métodos Orientados a Objetos: Conceptos Fundamentales**
Autor:	James Martin y James J. Odell
Editorial:	Prentice Hall
Edición:	1997
Edición:	Primera
ISBN:	970-17-0045-7

Resumen

Primera Parte La Base de la OO: La Estructura Objeto

Capítulo 2. Conceptos

Este capítulo explica sin mencionar de lleno a los objetos, como es que los humanos percibimos el mundo externo y como manejamos la información que recibimos del exterior. En resumen se define lo que es concepto en el mundo real y como se usa para la percepción de los objetos.

Capítulo 3. Objetos

Explicación de la relación que existe entre los conceptos y los objetos, y como no puede existir el uno sin el otro. Define un conjunto que es una colección de objetos a los que se le aplica un concepto. Finalmente se menciona a la clasificación múltiple que es cuando un objeto puede ser miembro de varios conjuntos en cualquier momento.

Capítulo 4. Tipos de Objetos (Conceptos)

Se explica que un tipo de objeto es un concepto, es decir, una noción o idea que se aplica a ciertos objetos de nuestro entorno.

Capítulo 5. Asociación de Objetos

Se explica que la asociación define la forma en la que objetos de diferentes tipos se pueden ligar o conectar. Las relaciones y los mapeos son dos formas de expresar dichas ligas. Entonces los mapeos existen donde hay ligas.

Capítulo 6. Mapeos

Explicación de los mapeos, los cuales sirven para formar relaciones conceptuales. Las restricciones de cardinalidad limitan el número de objetos con los que se mapea un determinado objeto. Los mapeos que permanecen fijos por definición son los mapeos básicos. Los mapeos derivados calculan o infieren valores del mapeo mediante otros básicos o derivados. Los mapeos

híbridos pueden tener instancias básicas o derivadas. Los mapeos tipo permiten definir asociaciones para un tipo de objeto especifico como un objeto único.

Capítulo 7. Relaciones

Las relaciones resaltan las diferencias entre objetos. Una relación es un tipo de objeto cuyos pasos particulares son composiciones invariantes de objetos.

Capítulo 8. Manejo de la complejidad de los objetos

Es este capítulo son enunciados y definidos algunos mecanismos para el manejo de la complejidad de los objetos, estos mecanismos son: la clasificación, la generalización, la especialización y la composición.

Capítulo 9. Subtipos y supertipos: Primera Parte

Explicación detallada de lo que es y como se usa la generalización y la especialización. Se introducen los términos de subtipo y supertipo y la importancia que tiene la herencia con los términos anteriores.

Capítulo 10. Subtipos y Supertipos: Segunda Parte

En este capítulo se estudia todavía con un poco más de detalle la generalización y su importancia.

Capítulo 11. Estados

El estado de un objeto es una colección de asociaciones que el objeto tiene con otros objetos y tipos de objetos. Un estado completo de un objeto consta de todas las asociaciones que existen durante todo el ciclo de vida. Los estado puntuales son estados en un instante particular.

Segunda Parte Fundamentos de la OO: Comportamiento de Objetos

Capítulo 12. Cambios de Estado

En este capítulo se proponen diagramas para expresar los estados y las formas posibles en las que pueden cambiar. Lo más importante de este capítulo es que el autor resalta el hecho de que un estado puede cambiar.

Capítulo 13. Eventos

Un evento es un cambio de estado importante. Definición de lo que es un estado previo y posterior a un evento. Finalmente un evento interno y un evento externo son también definidos.

Capítulo 14. Operaciones

Una operación es un proceso que puede ser solicitado como una unidad y puede controlarse como un objeto. Cada función puede conceptualizarse como una función: dada una entrada, la operación produce un resultado.

Capítulo 15. Métodos

Un método es la especificación del proceso de una operación. La técnica sugerida para representar los métodos es el diagrama de eventos.

Capítulo 16. Elementos activadores

Un elemento activador es un vínculo entre un evento y la operación llamada. Es posible definir ciertas reglas, reglas de activación, para que rijan los tipos de evento ante los cuales reaccionamos y como enfrentarlos. Existen los activadores múltiples que sirve para llamar a una operación varias veces.

Capítulo 17. Condiciones de control

Una condición de control es aquella que permite que su operación asociada inicie sólo cuando se cumplan sus restricciones. Su finalidad principal es sincronizar una serie de operaciones.

Tercera Parte Nivel Ampliado de los Fundamentos de la OO

Capítulo 18. Composición

La composición, conocida por otros autores como agregación, es un mecanismo para formar un objeto visto como un todo, utilizando otros objetos como sus partes. Explicación de seis maneras de formar asociaciones entre el todo y las partes.

Capítulo 19. Restricciones

Una restricción es una propiedad que siempre debe cumplirse. Existen diferentes tipos de restricciones algunas de las cuales se mencionan, sin embargo el analista y el usuario pueden definir sus propias restricciones de acuerdo a sus necesidades.

Capítulo 20. Reglas

Las reglas sirven para especificar políticas y condiciones, que deben ser comprendidas por los usuarios finales, deben además ser rigurosas y formales, y poder aplicarse en toda circunstancia y todo momento.

Capítulo 21. Uso de reglas con diagramas

Las reglas representadas en los diagramas ayudan a comprenderlas mejor. Cada desarrollador puede elegir la forma de representarlas para facilitar su trabajo.

Capítulo 22. Metamodelo

El metamodelo se refiere al modelado del sistema dentro de un marco referencial único, cuyo mantenimiento se realice de forma integrada, que abarque todos los objetos y casos particulares y que le sea posible tener cualquier cantidad de metaniveles.

Capítulo 23. Tipos de Potencia

Un tipo de potencia es definido como un tipo de objeto cuyos casos son subtipos de otro tipo de objeto y expresan una situación del mundo real.

Cuarta Parte Representación de Construcciones de AOO

Capítulo 24. Representación de la estructura de un objeto

En este capítulo se recomienda el uso de los modelos de relación binaria y de entidad-relación-atributo (ERA) para representar la estructura de un objeto. Resalta la facilidad de uso de estos modelos y explica en qué consisten.

Capítulo 25. Enfoques del modelado de la estructura de un objeto

Este capítulo presenta un panorama de seis enfoques del análisis OO comparando definiciones y símbolos usados por cada uno de ellos. La comparación es de manera general.

Capítulo 26. Representación del comportamiento de un objeto

Son dos las formas que se proponen para la especificación del comportamiento de un objeto: la relacionada y la no relacionada con el estado. Explicación de en qué consiste cada forma y recomendación de escoger la que mejor se adecue a las necesidades del sistema.

Capítulo 27. Enfoques del modelado mediante máquinas de estados finitos

Explicación del uso de las máquinas de estados finitos para representar el comportamiento OO, los cuales son fáciles de mapear con lenguajes de programación OO.

Capítulo 28. Enfoques del modelado basado en escenarios

Aquí se explica lo que son los escenarios y de qué forma facilitan el modelado orientado a objetos, enumerando algunas ventajas.

Capítulo 29. Otros enfoques de modelado

En este capítulo se explican los enfoques de contexto y funcional para lograr la descomposición de agrupaciones de alto nivel en términos de tipos de objeto.

Quinta parte Diseño e implementación

Capítulo 30. Consideraciones para el Diseño y la implementación

Breve explicación de la forma en la que se puede implementar el modelo conceptual.

Datos del libro:

Título:	**Object-Oriented Modeling and Design (Modelado y Diseño Orientado a Objetos)**
Autor:	James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy y William Lorensen
Editorial:	Prentice Hall
Edición:	1991
Edición:	Primera
ISBN:	0-13-629841-9

Resumen

Capítulo 1. Introduction (Introducción)

Breve explicación del concepto orientado a objetos, que es una nueva forma de pensar en los problemas usando modelos organizados en conceptos del mundo real. De manera breve y clara se definen algunos conceptos importantes como la identidad, clasificación, polimorfismo, herencia. Explicación de los cuatro pasos que conforman la metodología orientada a objetos: 1. Análisis, 2. Diseño del sistema, 3. Diseño de objetos y 4. Implementación.

Parte 1: Modeling Concepts (Conceptos de modelado)

Capítulo2. Modeling as a Design Technique (El modelado como una técnica de diseño)

Este capítulo menciona la importancia de los modelos, que son abstracciones que sirven para entender un problema antes de implementar una solución. También se menciona que el OMT (Object Modeling Technique) Técnica de Modelado de Objetos consiste en tres tipos de modelos: el modelo de objetos, el modelo dinámico y el modelo funcional; cada uno con una breve explicación.

Capítulo 3. Object Modeling (El modelado de objetos)

Elmodelado de objetos describe la estructura de datos estática de objetos, clases y sus relaciones entre ellas. El contenido de un modelo de objeto depende del juicio del desarrollador y de su relevancia en la aplicación. Un objeto es un concepto, una abstracción o cosa que tiene límites y significado para una aplicación. Después de definir lo que es un objeto se definen varios conceptos llamados constructores: ligas, asociaciones, ligas a atributos, rol, calificador, agregación, generalización, módulo y hoja. Los cuales al trabajar juntos describen sistemas complejos.

Capítulo 4. Advanced Object Modeling (El modelado avanzado de objetos)

Este capítulo explica varios conceptos del modelado de objetos. Estos conceptos son importantes para aplicaciones complejas. Resalta la importancia de que el contenido de un modelado de objetos debe dirigirse de acuerdo a la relevancia de su aplicación.
Algunos de estos conceptos son: la agregación, que es una forma especial de asociación transitiva. La agregación recursiva, permite que también un componente sea agregado y no sólo agregue. Enfasis en la diferencia entre la agregación y la generalización, ya que se puede prestar a confusiones.
Abstracto y concreto son dos términos útiles de la herencia en una jerarquía. Se explica con ejemplos la diferencia e importancia de herencia simple y herencia múltiple. También se explican las definiciones de metadato y llave candidata (que no tiene nada que ver con las bases de datos), así como las restricciones.

Capítulo 5. Dynamic Modeling (Modelado Dinámico)

El modelado dinámico representa control de la información: las secuencias de eventos, estados y operaciones que ocurren en un sistema de objetos. Como el modelo de objetos, el modelo dinámico representa un compromiso entre simplicidad y extensibilidad; existen algunas restricciones que deben presentarse en lenguaje normal.

Un eventos es una señal de que algo ha ocurrido, un estado representa el intervalo entre eventos y especifica el contexto en el que se interpretan los eventos, una transición entre estados representa la respuesta a un evento, una condición es una función booleana que controla si se le permite ocurrir a una transición, un diagrama de estados es una gráfica de estados y transiciones nivelados por eventos. El capítulo explica, con ejemplos, cómo es que los conceptos anteriores funcionan juntos y dan resultados.

Capítulo 6. Functional Modeling (Modelado Funcional)

El modelado funcional muestra cómo, cuándo y dónde se calculan los valores del sistema., controla las operaciones que deben llevarse a cabo y el orden de las mismas, define la estructura de valores en el que operan las operaciones. Para lo cual se usan diagramas de flujo de datos. En este capítulo se explica como deben hacerse los diagramas de flujo de datos incluyendo la información necesaria para entenderlos. Finalmente explica cómo los modelos de objeto, dinámico y funcional funcionan juntos para entender completamente el problema.

Parte 2: Design Methodology (Metodología de diseño)

Capítulo7. Methodology Preview (Revisión de la metodología)

Este capítulo explica de manera muy breve en qué consiste la metodología OMT Técnica de Modelado de Objetos (Object Modeling Technique), el cual consiste en los tres modelos mencionados en capítulos anteriores. Menciona que la metodología OMT produce sistemas que son más estables con respecto a cambios de requerimientos.

Capítulo 8. Analysis (Análisis)

Este capítulo explica con algunos ejemplos la forma en la que según la metodología que se maneja se debe llevar a acabo el análisis. Dice que el propósito del análisis es establecer y entender el problema y el dominio de la aplicación. Un buen análisis captura los elementos esenciales del problema sin introducir elementos de implementación los cuales son considerados hasta el diseño.

El primer paso es escribir el problema inicial con palabras simples pero claras, para esto es necesario hablar con las personas que requieren el sistema, usuarios y expertos del dominio. Los requerimientos deben describir lo que debe hacerse y no como debe implementarse. Después hay una explicación de la forma en la deben usarse los modelos de objetos, dinámico y funcional en esta fase que es el análisis. Es importante resaltar que en éste capítulo se aclara que la metodología puede modificarse un poco para adecuarse a las necesidades del problema.

Capítulo 9. System Design (Diseño del sistema)

Una vez concluido el análisis y antes de pasar de lleno al diseño deben tomarse algunas decisiones básicas como el nivel de estructura del sistema, esto es cómo se va a dividir en subsistemas, su concurrencia heredada, posición de subsistemas al hardware y software, administración de información, coordinación de fuentes globales de información, implementación de control de software, condiciones boundary(límite), y prioridades del sistema. Este capítulo explica cómo deben tomarse las decisiones antes mencionadas, da algunas recomendaciones y define claramente de qué se trata cada decisión que se tome y de qué forma afecta al sistema.

Capítulo10. Object Design (Diseño de objeto)

Este capítulo explica la fase de diseño de objetos, la cual agrega detalles de implementación; como reestructuración de clases, estructura interna de datos y algoritmos, implementación de control, implementación de asociaciones, y empaquetamiento de módulos físicos. En esta fase lo que se hace es extender el modelo de análisis con modificaciones necesarias al sistema.

Capítulo 11. Methodology Summary

Este capítulo resume en algunas líneas las fases de análisis, diseño del sistema y diseño de objetos. Menciona que la metodología OMT está basada en notación orientada a objetos para describir clases y relaciones a través del ciclo de vida del sistema.

Capítulo 12. Comparison of Methodologies

Breve explicación de algunas metodologías: SA/SD Análisis Estructurado/Diseño Estructurado (Structured Analysis/Structured Design), que tiene un enfoque estructurado y por lo tanto es muy diferente a la metodología tratada en este libro. Después menciona la metodología JSD de Michael Jackson la cual está enfocada a acciones del mundo real, para después ir creando un pseudocódigo del sistema. Finalmente se comparan estas metodologías, que son en esencia muy diferentes entre sí, y cada una aporta algo importante al diseño de sistemas.

Parte 3: Implementation (Implementación)

Capítulo13. From Design to Implementation (Del diseño a la implementación)

Una vez realizado el análisis y diseño lo siguiente es la implementación que puede hacerse usando un lenguaje de programación, un sistema de bases de datos o fuera de una computadora.

Capítulo 14. Programming Style (Estilo de programación)

Uno de los elementos más importantes en el desarrollo de un sistema orientado a objetos es el estilo de programación orientada a objetos que se use para implementar el diseño realizado, y asi obtener los resultado deseados. Menciona algunas características y aspectos en los que se debe poner mayor atención para lograr una implementación orientada a objetos eficiente y "fácil" de modificar.

Capítulo 15. Object-Oriented Languages (Lenguajes orientados a objetos)

Este capítulo menciona algunos lenguajes orientados a objetos, algunos ejemplos de cómo programar y finalmente un cuadro comparativo de lo que puede y no puede hacer cada uno de estos lenguajes. Los lenguajes que compara son : C++, Smalltalk, CLOS, Eiffel y Objective C. Aunque también menciona otros lenguajes como LISP y también menciona la importancia y la gran ayuda que dan los sistemas de bases de datos orientados a objetos.

Capítulo16. Non-Object Oriented Languages (Lenguajes no orientados a objetos)

Este capítulo explica como se pueden usar lenguajes no orientados a objetos para implementar diseños orientados a objetos, para lo cual se requiere mucha disciplina en la programación. Menciona algunas recomendaciones para implementar clases y algunas otras características propias de los lenguajes orientados a objetos usando C, Pascal y Ada. Al final se reconoce que lo más apropiado para implementar un diseño orientado a objetos es usar un lenguaje orientado a objetos.

Capítulo 17. Relational Databases (Bases de datos relacionales)

Descripción de cómo son y de qué forma ayudan los RDBMS Sistemas de Manejo de Bases de Datos Relacionales (Relational Data Base Management System).

Parte 4: Applications (Aplicaciones)

Capítulo 18. Object Diagram Compiler (Compilador del diagrama de objetos)

Aplicación de la metodología sugerida programando un compilador de diagrama de objetos, el cual además de transformar las salidas detecta errores de compilación. Las herramientas que los autores eligieron fueron DSM por su extensa librería de clases que simplifica la programación y UNIX que tiene un excelente desempeño.

Capítulo 19. Computer Animation (Animación por computadora)

Aplicación de la metodología sugerida esta vez con animaciones, para lo cual se escogió el sistema de animación a tres dimensiones OSCAR, el cual es extensible y capaz de integrar aplicaciones nuevas y avances en la tecnología de graficación.

Capítulo 20. Electrical Distribution Design System (Sistema de diseño de distribución eléctrica)

Una vez más se aplica la metodología creando un sistema de diseño de distribución eléctrica, para lo cual se usó OLIE que es una herramienta de diseño asistido por computadora (CAD,

Computer Aided Design). Se eligió esta herramienta porque es de dos dimensiones y muestra conexiones entre elementos eléctricos.

Apéndice A OMT Graphical Notation (Una notación gráfica del OMT)

Finalmente en esta apéndice se hace mención de la notación gráfica usada en el libro la cual se encuentra resumida en las cubiertas del libro. Sin embargo se hace la aclaración que sólo una persona que tenga conocimientos sobre la metodología puede entender la notación que se propone

Datos del libro:

Título:	**UML Distilled Applying the Standard Object Modeling Language (Aplicación destilada del lenguaje de modelado de objetos estándar UML)**
Autor:	Martin Fowler, Kendall Scott
Editorial:	Addison-Wesley Object Technology Series
Edición:	1997
Edición:	Quinta
ISBN:	0-201-32563-2

Resumen

Capítulo 1. Introduction (Introducción)

Explica como surge UML (Unified Modeling Language) Lenuaje de Modelado Unificado, cuándo y quienes son los principales aportadores. Explicación de porqué no solamente hay que limitarse a la programación, sino también el análisis y el diseño son muy importantes. Sugerencias para tener una idea más clara de lo que es la orientación a objetos.

Capítulo 2: An Outline Development Process (Proceso de Desarrollo Externo)

Revisión del proceso de desarrollo del software, incluye: incepción, elaboración, construcción y transición, con una breve explicación de cada etapa. Explicación general de cómo usar UML.

Capítulo 3. Use Cases (Casos de Uso)

Explicación de un caso de uso, cómo se realizan los diaramas de caso de uso, las partes en las que consiste, qué es cada parte y cuando usarlos.

Capítulo 4. Class Diagrams: The Essentials (Diagramas de Clase: Lo esencial)

Explicación de lo que es un diagrama de clase, para que sirve y en que consiste. Se dan algunos conceptos esenciales muy usados en dichos diagramas, los conceptos son: asociaciones, atributos, operaciones, generalización y reglas de constantes. Sugerencias de cuando usar este tipo de diagrama.

Capítulo 5. Class Diagrams: Advanced Concepts (Diagramas de Clase: Conceptos Avanzados)

Una vez definido el diagrama de clases en el capítulo anterior, así como conceptos generales, en este capítulo se introducen conceptos avanzados, que raramente son usados en el diseño con UML, ya sea por que se desconoce su uso correcto o porque no son necesarios; los conceptos son: estereotipos, clasificación dinámica y múltiple, agregación y composición, asociaciones dirigidas y atributos; interfaces y clases abstractas, objetos de referencia y de valor, colecciones para roles

multi-valuados, inmutabilidad, asociaciones calificadas, clases de asociaciones, clases parametizadas y visibilidad.

Capítulo 6. Interaction Diagramas (Diagramas de Interacción)

Definición y explicación de lo que son los diagramas de interacción. Explica los dos tipos de diagramas, que son: diagramas de secuencia y diagramas de colaboración. Al final se hace una comparación entre los dos tipos de diagramas y se dan sugerencias de cuando usar cada uno de ellos.

Capítulo 7. Package Diagramas (Diagramas de Paquetes)

En este capítulo se explica como, según el UML, dividir un sistema muy grande en pequeños sistemas, el mecanismo que se sugiere se llama paquete. Se explica como y cuando debe usarse esta técnica.

Capítulo 8. State Diagrams (Diagramas de Estado)

En este capítulo se explica lo que son los diagramas de estado, anteriormente ya existían pero aquí se hacen algunas modificaciones para su mejor funcionamiento. Explicación de los diagramas de estado concurrente, que es una variante del anterior. Finalmente mención de los casos en los que deben usarse estos diagramas.

Capítulo 9. Activity Diagrams (Diagramas de Actividad)

Explicación de los diagramas de actividad y la forma de usarse para los casos de uso, que es donde realmente son útiles mayor frecuencia. Los diagramas de actividad representan el todo del sistema, pero también se puede realizar una descomposición de cada actividad, detallando más una parte del diagrama original.

Capítulo10. Deployment Diagrams (Diagramas de Expansión)

Explicación de lo que es un diagrama de expansión y cuando usarlo.

Capítulo 11. UML and Programming (UML y Programación)

Se explica como facilitan el trabajo de la programación los diagramas descritos en capítulos anteriores, se dan ejemplos de cada diagrama con su respectivo código.

Capítulo 12. Techniques and their uses (Técnicas y sus usos)

En un cuadro sinóptico se mencionan las técnicas existentes y su uso para el modelado de objetos. Estas técnicas no pertenecen de forma directa al UML, pero pueden ser muy útiles si se usan en combinación con la técnica descrita durante el libro.

2.1.2 Libros de lectura ligera

Datos del libro:

Titulo:	**Object-Oriented Systems Analysis A Model-Drive Approach (Análisis de Sistemas Orientados a Objetos un Enfoque Orientado a Modelos)**
Autor:	David W. Embley, Barry D. Kurtz y Scott N. Woodfield
Editorial:	Yourdon Press Prentice
Edición:	1992
Edición:	Primera
ISBN:	0-13-629973-3

Resumen

Capítulo 1

1.3 Approaches to Systems Analysis (Enfoques al Análisis de Sistemas)

Esta parte brinda una introducción a la forma en la que se utiliza el lenguaje natural para la comprensión de los sistemas, y cómo han evolucionado las diferentes formas para entender a los sistemas hasta llegar al enfoque orientado a objetos.

Se introduce también el concepto OSA Análisis de Sistemas Orientados a Objetos (Object-oriented Systems Analysis), que es la técnica que se presenta en este libro.

1.4 Model-Driven Analysis (Análisis orientado a modelos)

Se explica que la técnica OSA no sólo es orientada a métodos sino también orientada a modelos, la primera consiste en una secuencia estructurada a seguir, la segunda consiste en capturar las características específicas del sistema.

1.5 Representing Reality (Representación de la realidad)

Expresa la importancia de representar la realidad en los modelos orientados a objetos.

1.7 An OSA Overview (Un resumen OSA)

El fundamento de los modelos de sistema de OSA son las clases y los objetos. Es importante describir el comportamiento de cada objeto, los más importantes son: los estados que cada objeto tiene durante su existencia, las condiciones que causan que los objetos pasen de un estado a otro y las acciones que llevan a cabo los objetos o que otros objetos llevan a cabo sobre ellos. Para facilitar el entendimiento de sistemas grandes OSA propone vistas que permiten abstraer modelos de relaciones y de comportamiento de objetos complejos en modelos que solo presentan los elementos más importantes. Para lo cual se usan componentes de alto nivel, que incluyen clases, conjuntos de relaciones, estados y transiciones.

Capítulo 6

6.1 Basic Interaction Diagrams (Disgramas de interacción básicos)

Muestra y explicación de algunos dibujos que usa la OSA para representar interacción entre objetos.

6.2 Interaction Descriptions (Escripciones de la interacción)

De acuerdo a los diagramas mostrados en la parte anterior se explica como deben usarse.

6.3 Synchronous Object Interaction (Interacción de objetos scíncronos)

Explicación de la necesidad e importancia de sincronizar objetos.

6.4 Asynchronous Object Interaction (Interacción de objetos ascíncronos)

Se recalca que la sincronización de objetos no siempre es necesaria y que en algunas ocasiones es mejor usar interacciones ascíncronas.

6.5 Specifying Particular Interacting Object (Especificación particular de la interacción de objetos)

En esta parte se muestran diagramas con iteracción a un objeto de destino específico y con interacción con objetos origen y destino específicos.

6.6 Interacting with multiple Objects (Interacción con múltiples objetos)

Una vez más se muestran diagramas que representan la interacción de varios objetos.

6.7 Bidirectional Interaction (Interacción bidireccional)

En algunas ocasiones debido a la naturaleza misma de las interacciones éstas pueden darse en ambos sentidos y de forma idéntica, por lo cual se muestran ejemplos del diagrama propuesto para esta situación.

6.8 Special Interaction Activities (Actividades de Interacción Especial)

Actividades de interacción: acceso, modificar, borrar, destruir, agregar y crear. Y se explican cada una de ellas proponiendo una forma de representación gráfica.

6.9 Bulleting-Board Communication (Comunicación boletín-pizarra)

Explicación de este tipo de comunicación entre objetos, que consiste en un objetos receptor de todos los mensajes y que luego los reparte a sus receptores correctos.

6.10 Model Boundary-Crossing Interactions (Modelo de interacciones límites-cruzadas)

Explica de que forma pueden cruzarse las interacciones sin llegar a confundirse.

6.11 Continuing Interaction (Interacción continua)

Representación gráfica de la interacción continua.

6.12 Time-Constrained Interactions (Interacciones con restricciones de tiempo)

En este caso se toma en cuenta el tiempo para expresar las interacciones, con comentarios entre paréntesis.

6.13 General Interaction Constraints (Restricciones de interacción general)

Expresa el uso de restricciones generales de varias formas para direccionar varios elementos de interacción.

6.14 Interaction Within an Object Class (Interacciones dentro de las clases)

Se muestran ejemplos de interacciones entre objetos en la misma clase, e incluso algunos objetos comunicándose entre ellos mismos.

2.1.3 Libros de lectura rápida

Datos del libro:

Título:	**Object Oriented Methods (Métodos Orientados a Objetos)**
Autor:	Ian Graham
Editorial:	Addison-Wesley Publishing Company
Edición:	1992
Edición:	Segunda
ISBN:	0-201-56521-8

Resumen

Capítulo 1. Basic Concepts (Conceptos Básicos)

Breve revisión histórica del nacimiento primero de la programación orientada a objetos y después del análisis y diseño orientado a objetos. También se introducen algunos conceptos fundamentales como son: la abstracción, la encapsulación, el polimorfismo y la herencia.

Capítulo 7. Object-oriented design and analysis (Análisis y Diseño Orientado a Objetos)

En este capítulo primero se hace un recorrido por algunos de los métodos de diseño orientados a objetos más comunes, el GOOD Diseño Orientado a Objetos General (General Object-Oriented Design) desarrollado por la NASA, el HOOD Diseño Orientado a Objetos Jerárquico (Hierarchical Object-Oriented Design) desarrollado por la European Space Agency, el OOSD Diseño Estructurado Orientado a Objetos (Object-Oriented Structured Design) desarrollado por Wasserman, Pircher y Muller, la JSD Diseño Estructurado de Jackson (Jackson Structured Design) desarrollada por Jackson, el de Booch y el de Ptech de Associative Design Technology.

Explicación de como debe realizarse el análisis y después el diseño en base a lo que proponen Coad y Yourdon, pero también tomando algunos elementos de otros autores. Lo siguiente es cómo identificar objetos para lo cual el autor propone una forma de hacerlo.
Menciona, también algunas herramientas CASE pero en versiones anteriores. Finalmente hay un estudio de caso muy pequeño para graficación.

Capítulo 8. Managing object-oriented methods (Administración de métodos orientados a objetos)

Este capítulo se basa principalmente en la importancia de hacer prototipos para asegurar el buen desarrollo del sistema, además estos prototipos son de gran ayuda para administrar el análisis y diseño de una manera sencilla y controlada.

2.2 Tesis

Datos de la tesis:

Título:	**Una guía práctica para la evaluación de plataformas de desarrollo de software orientado a objetos**
Autor:	Francisco Alejo Rios Gascon
Fecha:	1995
Carrera:	Maestría en Ciencias de la Computación
Universidad-Facultad:	UNAM-Colegio de Ciencias y Humanidades, Unidad Académica de los Ciclos Profesionales
Ubicación:	Biblioteca Central 001-03063-R1-1995

La tesis inicia mencionando las características generales del software y sus ciclos de vida. Hace una comparación entre el enfoque de desarrollo de descomposición funcional y orientado a objetos.

Habla de las metodologías CASE, su definición, ventajas y desventajas, y una breve explicación de su funcionamiento. También habla de las herramientas CASE a diferentes niveles; bajo, medio y alto. El nivel que el autor toma en cuenta para el desarrollo es el bajo.

No podía faltar por supuesto una breve explicación del desarrollo orientado a objetos definiendo conceptos y enunciando las características indispensables de un ambiente de desarrollo con este paradigma.

Finalmente le da una aplicación práctica al modelo de objetos en redes, usando un sistema operativo orientado a objetos, que se llama NEXTSTEP. El desarrollo es breve pero claro.

Esta tesis se basa principalmente es los parámetros de evaluación que deben seguirse para evaluar una plataforma de desarrollo de software orientado a objetos, y no al análisis y diseño, aunque también se menciona. Sin embargo hay algunos elementos importantes para mi tema de estudio, ya que le da a la orientación a objetos una perspectiva de evaluación, que me permite tomar en cuenta para el análisis y diseño.

Datos de la tesis:

Título:	**Diseño orientado a objetos fundamentos y aplicaciones**
Autor:	Olga Hernández Sánchez
Fecha:	1995
Carrera:	Ingeniero en Computación
Universidad-Facultad:	UNAM-Aragón
Ubicación:	Biblioteca Nacional 001-41132-H4-1995

Esta tesis se centra en definición de conceptos totalmente basada en la metodología de Booch, así como en la explicación del análisis y el diseño también basados en Booch.

La autora hace mención de varios lenguajes orientados a objetos y una breve explicación, los lenguajes son: SIMULA, SMALLTAK, OBJECT PASCAL, EIFFEL, C++. Menciona elementos importantes que se deben tomar en cuenta para elegir un lenguaje orientado a objetos.

La aplicación que esta autora resalta es la de un juego de vídeo programado en C++. Esta tesis aporta algunos elementos a mi tema de estudio debido a que esta basada en su totalidad en la metodología de Booch.

Datos de la tesis:

Título:	**El software orientado a objetos: panorama general**
Autor:	José Guillermo Preciado López
Fecha:	1996
Carrera:	Licenciatura en Actuaría
Universidad-Facultad:	UNAM-Facultad de Ciencias
Ubicación:	Biblioteca Nacional 001-0321-P4-1996

Esta tesis una vez más explica los conceptos propios de la orientación a objetos, enunciando ventajas y desventajas de este paradigma.

De manera breve y basado totalmente el Booch, el autor define y desarrolla los procesos a seguir del análisis y diseño orientado a obejtos.

Este trabajo se centra en las herramientas de software existentes orientadas a objetos. Hace mención de algunas herramientas para programar, de bases de datos, de desarrollo, para ambientes personlaes de productividad.

Finalmente enuncia las iniciativas y alianzas de algunos fabricantes de software en cuanto a lenguajes y bases de datos, así como interfaces de usuarios con sistemas operativos orientados a objetos.

Datos de la tesis:

Título:	**Una aplicación de la metodología OMT para el diseño orientado a objetos**
Autor:	Isabel Laura Oliver Suárez
Fecha:	1998
Carrera:	Licenciatura en Actuaría
Universidad-Facultad:	UNAM-Facultad de Ciencias
Ubicación:	Biblioteca Central 01-0321-O2-1998

Esta tesis explica conceptos funamentales de la orientación a objetos. Esta vez se basa en OMT (Object Modeling Technique) que fue creado por Rumbaugh et al, por lo tanto el análisis, el diseño y la implementación difieren de las tesis anteriores por contar con fuentes de investigación diferentes.

El caso práctico de esta tesis consiste en un sistema para evaluar el rendimiento de las diferentes plantas de un grupo petroquímico desde el punto de vista financiero de PEMEX.

Datos de la tesis:

Título:	**Modelo reutilizable orientado a objetos de empresas**
Autor:	Ari Schoenfeld Liberman
Fecha:	1998
Carrera:	Maestría en Ciencias de la Computación
Universidad-Facultad:	UNAM-Colegio de Ciencias y Humanidades, Unidad Académica de los Ciclos Profesionales

Ubicación: Biblioteca Central 001-03063-S2-1998

Esta tesis habla principalmente de patrones, que son un elemento importante de reciente surgimiento en el paradigma orientado a objetos, mediante el cual algunos elementos pueden ser reutilizables a lo largo del sistema que se este creando o incluso en otro donde puedan usarse. Define al patrón como una regla en tres partes que expresa la relación entre un contexto, un problema y una solución.

El ejemplo, basado en una empresa, consta de tres patrones : personas y roles, documentos y roles de personas y valores convertibles.

La notación que utiliza está totalmente basada en Booch.

Al final de la tesis está incluido un artículo, escrito por el mismo autor de la tesis, que fue su trabajo preliminar a su trabajo de tesis: "Three Domain Specific Design Patterns"

Datos de la tesis:

Título:	**La metodología de orientación de objetos como nueva herramienta para el análisis y diseño de software**
Autor:	Adriana Santos Rosas
Fecha:	1994
Carrera:	Ingeniería en Computación
Universidad-Facultad:	UNAM-Facultad de Ingeniería
Ubicación:	Biblioteca Central 01-41132-S2-1994

El fundamento de la tecnología orientada a objetos se encuentra totalmente basada en Booch, una vez más.

Menciona métodos de programación orientados a objetos, lenguajes orientados a objetos y bases de datos orientados a objetos.

El caso práctico es el desarrollo de un sistema de control de facturas y proyectos para PEMEX, el cual incluye programas fuentes hechos en Paradox.

Menciona las ventajas, desventajas y costos del paradigma orientado a objetos sobre todo cuando se desea realizar la migración del paradigma estructurado al orientado a objetos.

Datos de la tesis:

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 Angel César Govantes Saldivar
Fecha:	1997
Carrera:	Ingeniería en Computación
Universidad-Facultad:	UNAM-Facultad de Ingeniería
Ubicación:	Biblioteca Central 001-01132-C8-1997

Esta tesis es muy interesante ya que empieza por definir lo que es ISO9000, para que sirve, un panorama general e importancia.

Después da una breve explicación de lo que son los sistemas de información, en base al ISO9000.

Los principios de la metodología orientada a objetos, diferencias entre ésta y la metodología tradicional, el modelado del análisis y diseño orientado a objetos y sus características y beneficios están basados en Booch, Rumbaugh y Jmartin_J.Odell.

Las políticas, normas y manual de procedimientos para el desarrollo de sistemas orientados a objetos, incluyen las etapas de análisis, diseño, programación, pruebas, implantación y capacitación y mantenimiento, todo basado en ISO9000. Algunos formatos para la documentación de sistemas que los autores proponen son para : estudio de factibilidad, calendario de trabajo, relación de documentos, diagrama de flujo de datos, diccionario de términos, diagrama de clases, diagrama de seguimiento de sucesos, diagrama de estados, acta de acuerdos, selección de identificación de software y hardware, diagrama de entidad-relación, estructura de tabla, registro de llaves secundarias, entre otros. Todos estos formatos se centran en el desarrollo completo del sistema sindetallar ninguna fase.

Finalmente en el apéndice se mencionan todas las normas ISO9000 en las que se baso la tesis.

2.3 Revistas

Datos de la revista

Título:	**BYTE de México**
Director de la revista:	Lic. Quina Monroy Baker
Domicilio:	Balboa 813, col. Portales 03300, México, D.F.
Periodo revisado:	Enero 1998 – Febrero 1999

Artículos localizados y leídos

Título:	***Escribiendo en Rhapsody***
Autor:	Tom Thompson
Año:	10
No.:	122
Fecha :	Marzo 1998
Ubicación	**Biblioteca Central F.112733**

Este artículo habla sobre la creación del sistema operativo orientado a objetos Rhapsody, que es el resultado de la unión de los sistemas operativos Next y OpenStep en 196. Se mencionan varias de sus características y funciones orientadas a objetos. Así como algunas diferencias con los sistemas operativos que no son orientados a objetos. Resalta la facultad del compilador mejorado GNN 2.7.2 que traduce programas C, C++ y Objective-C a código ejecutable. Finalmente aporta algunas soluciones para el uso efectivo de este sistema operativo, todo con un desarrollo orientado a objetos.

Título:	*Jasmine reta a las bases de datos tradicionales*
Autor:	Barry Nance
Año:	10
No.:	127
Fecha :	Agosto 1998
Ubicación	Biblioteca Central F.112733

Se presentan las características de Jasmine, que es un software para bases de datos orientados a objetos. Este sobre puede ejecutarse en NT y diferentes tipos de Unix (HP, Solaris y AIX, mientras que los clientes pueden ser NT o Windows 95.

El autor realiza algunas pruebas a fin de evaluar el software y concluye que es muy bueno para los que desarrollen con orientado a objetos; aunque resulte lento, en comparación con el software para bases de datos tradicionales.

Datos de la revista

Título:	**JOOP The Journal of Object-Oriented Programming**
Director de la revista:	Dr. Richard Wiener
Domicilio:	Publications Inc., 71 West 23rd Street, 3rd floor, New York, N.Y. 10010
Período revisado:	Enero 1998 – Enero 1999

Artículos localizados y leídos

Título:	*Experience Report SSADM-Designed System to Object-Oriented System (Reporte de experiencia del sistema diseñado con SSADM a un sistema orientado a objetos)*
Autor:	S. Y. Liao, Y. P. Shao, W. H. Tsang
Volumen:	10
No.:	9
Fecha :	February 1998
Ubicación	Biblioteca DGSCA 3J/1998/10/9

Este artículo explica cómo se realizó la re-ingeniería de un sistema que estaba diseñado con SSAOM (Structured Analysis and Design Method) programado con Clipper a análisis y diseño orientado a objetos siguiendo la metodología de Booch.

De manera breve y general se presentan los requerimientos , diagramas de flujo de datos, bases de datos relacionales, diagramas entidad-relación del diseño original y a continuación la reingeniería con el método Booch ; primero se identifican las clases principales, se revisan los DFD, se validan y especifican las clases, se crea un diccionario de datos, relaciones entre clases, operaciones y atributos, herencia y diagrama de transición. A continuación se ejemplifica un poco la programación y finalmente se muestra una comparación entre ambos diseños y aunque ambos presentan ventajas, la opción de orientación a objetos resultó ser más favorable, aunque más compleja.

Título:	*Our Cases with Use Cases (Nuestros casos con Casos de Uso)*
Autor:	Ari Jaaksi
Volumen:	10
No.:	9
Fecha :	February 1998
Ubicación	Biblioteca DGSCA 3J/1998/10/9

Explicación del funcionamiento de la herramienta OMT++ basada en la metodología de Ivar Jacobson, presenta características generales y cómo usar la herramienta desde el análisis, pasando por el diseño y programación hasta llegar al soporte.

La conclusión de dicho artículo es que las herramientas de casos de uso orientadas a objetos son muy útiles para los desarrolladores de sistemas OO ; sin embargo hay que entender la metodología en la que están basadas y saber usarlas, de lo contrario los resultados no serán satisfactorios.

Título:	*Improving OO Analysis Methods (Mejora de métodos de análisis orientados a objetos)*
Autor:	Graham Berrisford
Volumen:	11
No.:	1
Fecha :	March-April 1998
Ubicación	Biblioteca DGSCA 3J/1998/11/1

Los autores presentan las características generales del desarrollo del OO en base a Booch y Jacobson. Concluye que los métodos OO deben enfocarse a : presentar el análisis y el código como dos cosas diferentes, presentar subtipos y herencia, diferencias entre procesos de negocios, prácticas de trabajo y transacciones, hacer énfasis en el *rol* de los procesos de transacción como reglas de negocios y alentar el análisis como un preámbulo fundamental de la programación.

Título:	*A More Complete Model of Relations and their Implementation: Roles (Modelo más completo de relaciones y su implementación: roles)*
Autor:	Conrad Bock y James Odell
Volumen:	11
No.:	2
Fecha :	May 1998
Ubicación	Biblioteca DGSCA 3J/1998/11/2

En este artículo se presentan modelos de análisis y diseño que clarifican y formalizan la noción de *rol*. Cuando un objeto ejecuta un *rol* quiere decir que el objeto participa en una relación de forma particular. Lo fundamental es formalizar el concepto de *rol* como tipo que identifique a los objetos en un momento específico realizando una acción específica y resalta la importancia del uso del mapeo para definir el análisis de *roles*.

Título:	*Interface Specification, Refinement and Design with UML/Catalysis (Especificación de interfaz, refinamiento y diseño con UML/Catalysis)*
Autor:	Desmond D'Souza

Volumen:	11
No.:	3
Fecha :	June 1998
Ubicación	Biblioteca DGSCA 3J/1998/11/3

A partir de los tres conceptos del modelado del UML : colaboraciones, tipos y refinamiento ; el autor explica como pueden definirse de forma más precisa los aspectos del diseño. Se muestran algunos ejemplos con diagramas para crear un marco de trabajo que sea suficientemente completo para incluir tipos, colaboraciones, patrones y refinamientos ; y así ir directo a la programación. Los ejemplos del autor son con Java.

Título:	***Testing Models : The Requirements Model (Modelos de prueba: modelo de requerimiento)***
Autor:	John D. McGregor
Volumen:	11
No.:	3
Fecha :	June 1998
Ubicación	Biblioteca DGSCA 3J/1998/11/

Se expresa la importancia de los modelos de prueba de requerimientos, los cuales deben revisarse durante todas y cada una de las etapas del sistema. Al inicio es importante dedicarle el tiempo suficiente a la comprensión de requerimientos para que éstos queden muy claros y no existan problemas posteriores importantes. Se sugieren algunos pasos para realizar pruebas y también para realizar el modelo de requerimientos de manera efectiva.

Título:	***The Fifty-Foot Look at Analysis and Design Models (Mirada de cincuenta y cinco pies a los modelos de análisis y diseño)***
Autor:	John D. McGregor
Volumen:	11
No.:	4
Fecha :	July/August 1998
Ubicación	Biblioteca DGSCA 3J/1998/11/4

La labor de probar modelos de análisis y diseño es muy intensa, pero efectiva. La técnica OO sólo puede funcionar correctamente si desde el análisis y el diseño se realiza correctamente, para lo cual es necesario hacer varias pruebas. Una forma que propone el autor d}es detallar los casos de uso para examinarlos con mayor precisión. Las pruebas deben ser entonces un proceso continuo y no una actividad que se realice cuando el sistema este terminado.

Título:	***A More Complete Model of Relations and their Implementation : Aggregation (Un modelo más completo de relaciones y su implementación: agregación)***
Autor:	Conrad Bock, James Odell
Volumen:	11
No.:	5
Fecha :	September 1998

Ubicación Biblioteca DGSCA 3J/1998/11/5

Es importante considerar que la agregación se ha visto un tanto limitada en los lenguajes de modelado. Los autores sugieren usos más amplios de la agregación : una de ellos es que los objetos pueden ser partes o agregados, y la otra es que se pueden agregar objetos y relaciones en un sup0ertipo común, creando un continuo orden de objetos y relaciones agregadas, lo que permite un fácil reuso de dichos elementos.

Título: ***Entity/Skill/Reliance : Applying Functional Decomposition Safely of Object-Oriented Design (Entidad/habilidad/confiabilidad: aplicación segura de la descomposición funcional del diseño orientado a objetos)***
Autor: Avner Ben
Volumen: 11
No.: 6
Fecha : October 1998
Ubicación Biblioteca DGSCA 3J/1998/11/6

Durante el diseño es importante usar una lista de habilidades del sistema para estudiar las dependencias entre subsistemas. De tal forma que puedan expresarse subsistemas como habilidades del sistema. Durante el análisis las lista de habilidades debe usarse para capturar requerimientos. Esta forma de modelar tiene algunos problemas y sólo es aplicable dependiendo del tipo de sistema que se trate.

Título: ***Writing Effective Use Cases and Introducing Collaboration Cases (Escribir casos de uso efectivos e introducir casos de colaboración)***
Autor: Ben Thompson
Volumen: 11
No.: 6
Fecha : October 1998
Ubicación Biblioteca DGSCA 3J/1998/11/6

Los casos de uso son una técnica muy útil para capturar requerimientos. Es importante que sean simples y comprensibles. Sin embargo la documentación de los casos de uso puede ser muy grande y difícil de mantener, por lo tanto los casos de colaboración son de gran ayuda. Los casos de colaboración separan información (primaria, captada durante el análisis que tenga que ver con interacciones internas del sistema de en los casos de uso. Por lo tanto tener casos de uso con guías de casos de colaboración proporciona una documentación cohesiva que puede ser reusable para requerimientos, pruebas y entrenamiento de usuarios finales.

Título: ***Object-Oriented Analysis through a Knowledge-Based System (Análisis orientado a objetos a través de sistemas basados en conocimiento)***
Autor: Boumediene Belkhouche y Alejandro Mendoza Gamiño
Volumen: 11
No.: 7
Fecha : November-December 1998
Ubicación Biblioteca DGSCA 3J/1998/11/7

En este artículo se propone una base para el análisis OO independiente del problema y sistemático. Se describe la estrategia OORA (Object Oriented Requirements Analysis), la cual está basada en un esquema de representativo que proporciona una serie de constructores para organizar los requerimientos de una forma estructurada. Esta técnica permite entender los requerimientos y facilitar el mapeo de dominio del problema al software. Para demostrar la viabilidad de este enfoque se implementó un prototipo basado en el conocimiento, el cual es capaz de realizar análisis diferentes en el modelo inicial de requerimientos : 1)patrones comunes entre clases, 2)conexión de mensajes y 3grupos de clases cohesivas.

Título:	***Behavioral Specification and Analysis of Object-Oriented Designs (Especificación y análisis de comportamiento de diseños orientados a objetos)***
Autor:	Boumediene Belkhouche y Joel Wu
Volumen:	11
No.:	8
Fecha :	January 1999
Ubicación	Biblioteca DGSCA 3J/1999/11/8

Se propone una base para soportar la especificación del análisis y el diseño OO. Incluye : especificación formalizada del comportamiento arquitectónico y estructural, definición de las interacciones de objetos y su composición en sistemas grandes, descripción de implementación del procesador del lenguaje. La tarea más importante del procesador es el análisis sintáctico y semántico.

Datos de la revista

Título:	**Rose Architect visual modeling with UML (Modelado visual de la arquitectura Rose con UML)**
Director de la revista:	Terry Quatrani
Domicilio:	http://www.researchitect.com/mag/
Período revisado:	Octubre 1998 – Febrero 1999

Artículos localizados y leídos

Título:	***The Visual Modeling of Software Architecture for the EnterpriseEscribiendo en Rhapsody (El modelado visual de arquitectura de software***
Autor:	Grady Booch
No.:	1
Fecha :	Octubre-Diciembre 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo habla sobre la importancia de realizar un modelado antes de iniciar un proyecto. Menciona algunas ventajas de llevar a cabo el modelado en todos los niveles del ciclo de vida de un sistema. Menciona de que forma el UML ayuda al modelado y facilita la realización de proyectos, ya que no solamente ayuda a crear la documentación del sistema, también ayuda a crear un modelo que

se puede reusar en un futuro para sistemas similares. Otra de las cosas importantes que menciona es el manejo de riesgos que también puede hacerse con el uso del UML y con la participación de las personas involucradas en la realización del sistema.

Título:	***Making theEarth Move: The Paradigm Shift to Object-Oriented***
Autor:	Christian Buckley & Darren Pulsipher
No.:	1
Fecha :	Octubre-Diciembre 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo trata sobre la importancia del cambio de paradigma de estructurado a orientado a objetos. Según el autor lo primero que debe hacerse es crear una relación de confianza con el grupo de trabajo, después entablar una buena comunicación para intercambiar conocimientos. El paso siguiente es hacer una planeación previa tomando en cuanto a elementos como: el tiempo de terminación del proyecto, prioridades de desarrollo y mantener al equipo unido y productivo hasta el final del proyecto. Finalmente establecer un lenguaje común, en este caso el UML, para llevar a cabo el desarrollo orientado a objetos.

Título:	***Deciphering the UML***
Autor:	Lisa Dornell
No.:	2
Fecha :	Enero-Marzo 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo es la transcripción de una conversación entre los *tres amigos*: Grady Booch, Jim Rumbaugh e Ivar Jacobson; dirigida por Lisa Dornell, en la que hablan de algunos elementos que usaron para el UML y las razones por las cuales los tomaron en cuenta. Cada uno de ellos expresa su punto de vista en cuanto a sus aportaciones propias y a las tomadas de otros autores.

Título:	***The ABCs of Software Design***
Autor:	Lisa Dornell
No.:	2
Fecha :	Enero-Marzo 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo habla de la empresa norteamericana *American Buildings Company*, que es una empresa dedicada a la construcción de componentes de metal para estructuras de construcción. Esta empresa decidió hacer una revisión a sus sistemas de cómputo para mejorarlos, y para ello uso en parte tecnología Rose, lo cual quiere decir que uso UML. Se describe como es que esta empresa comenzó el desarrollo de su nuevo sistema y el avance exitoso que lleva hasta el momento usando tecnología orientada a objetos.

Titulo:	***Applying Design Patterns in UML***
Autor:	Joanne Garlow, Chris Holmes y Thomas Mowbray
No.:	2
Fecha :	Enero-Marzo 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo habla acerca de los patrones en UML. Primero da una breve explicación de lo que son los patrones, después numera algunos patrones comunes y útiles para cualquier desarrollo de sistemas, sin importar que sean diferentes. Finalmente explica para que sirven dichos patrones, los cuales son principalmente para el control de proyectos de software.

Título:	***Using Structured Tools in Object-Oriented Development***
Autor:	Efim Oykhman y Yurij Novozhenov
No.:	2
Fecha :	Enero-Marzo 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo habla sobre el uso de bases de datos en el desarrollo de sistemas orientados a objetos. Se menciona que si bien existen Sistemas de Administración de Bases de Datos orientadas a objetos, las personas que hacen desarrollo orientado a objetos prefieren trabajar con Sistemas de Administración de Bases de Datos Relacionales. La explicación es simple las RDBMS proporcionan grandes ventajas conceptuales, además tienen muy buena reputación en cuanto a confiabilidad y tienen una grana ceptación. En cambio las DBMS orientadas a objetos parecen ser muy complicadas debido a que es un campo poco explorado y los desarrolladores prefieren no tomar riesgos.

Título:	***Destroying the Tower of Babel***
Autor:	Christian Buckley & Darren Pulsipher
No.:	2
Fecha :	Enero-Marzo 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo aborda de manera muy general el tema de la documentación. Se mencionan algunos aspectos importantes en cuanto a la documentación de sistemas y se dan algunos consejos para construir la documentación de forma adecuada y sencilla mediante el uso del UML Se menciona la herramienta de modelado Rational Rose, basada en el UML, sin embargo no es la única que existe, aunque ésta ofrece muchas ventajas.

Título:	***The View from the Front: Changes to the UML by the Revision Task Force***
Autor:	James Rumbaugh
No.:	3
Fecha :	Abril-Junio 1998
Ubicación	http://www.researchitect.com/mag/

El autor de este artículo, que es uno de los creadores del UML, explica lo que es el Revision Task Force, RTF (Fuerza de REvisión de Tarea), que es una organización creada a partir de la necesidad de revisar el UML y aceptarlo como estándar esta organización se creo en el Object Management Group, OMG (Grupo de Administración de Objetos), la cual esta encargada de aceptar el estándar para el modelado orientado a objetos, específicamente el UML. Se mencionan los cambios que se han hecho al UML, a partir de la revisión que el grupo citado arriba hace continuamente.

Título:	***What's New in Rational Rose 98i***
Autor:	Naveena Bereny
No.:	3
Fecha :	Abril-Junio 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo trata sobre la nueva versión de Rational Rose, los cambios que ha sufrido y las mejoras que se le han incluido. Esta herramienta es importante porque es de la compañía donde trabajan los creadores del UML, por lo tanto su software de modelado Rational Rose está completamente basado en el UML. Según el artículo los cambios más importantes que se han realizado son la conexión de objetos ActiveX desarrollados en VB, VC++ y VJ++ con bases de datos. El elemento más importante según la autora es el elemento *Internet Web Publisher*, que permite exportar el modelo de diseño creado al HTML visible en el Web.

Título:	***Business Modeling, Tenagers, and Primal Scream Therapy***
Autor:	Christian Buckley & Darren Pulsipher
No.:	3
Fecha :	Abril-Junio 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo trata sobre la dificultad de tratar con los clientes que desean que se les desarrolle un sistema, debido a que estos a veces piden demasiados requerimientos, a tal grado que el proyecto puede salirse de control por la dificultad de comunicación entre el desarrollador y el cliente. En este artículo se analizan algunos de los problemas más comunes con clientes y como manejarlos para llegar a la satisfacción de ambas partes, aun si en algunas ocasiones es necesario ocultar información que de cualquier modo el cliente no entiende.

Título:	***Modeling Large Systems***
Autor:	Magnus Christerson & Nils Undén
No.:	4
Fecha :	Julio-Agosto 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo trata de responder a la siguiente pregunta ¿Por qué los sistemas grandes son tan difíciles de desarrollar?. Según el artículo para tener éxito en el desarrollo de un sistema grande lo que debe hacerse es primero, tomar un enfoque de arriba hacia abajo y después usar la tecnología orientada a objetos. Lo que se sugiere es usar UML (Unified Modleing Language) o RUP (Rational Unified Process) para realizar el modelado de sistemas grandes. A continuación se da una breve

explicación de lo que son cada una de estas herramientas y como usarlas para lograr obtener un buen desarrollo. Finalmente se menciona un pequeño ejemplo.

Título:	*The importance of Teams*
Autor:	Grady Booch
No.:	4
Fecha :	Julio-Agosto 1998
Ubicación	http://www.researchitect.com/mag/

En este articulo Grady Booch toma como base un curso que se dio hace ya 25 años que se llamaba *The Psychology of Computer Programming, computer programming is a human activity* (La psicología de la programación en computadora, la programación es una actividad humana), endonde se resaltaba la importancia de solaborar en equipo, con un grupo de gentes con diferentes cualidades para obtener un producto de software con calidad. Booch menciona que uno de los grandes problemas en el desarrollo del software es que los desarrolladores no trabajan en equipo, y por lo tanto hay tareas duplicadas y ni hay compatibilidad en lo que se está haciendo, debido a la falta de comunicación. El trabajo en equipo implica varios elementos no solamente estar todos juntos, sino estar en comunicación constante y usar el mismo lenguaje de modelado para que las cosas se entiendan como son y no se preste a interpretaciones erróneas.

Título:	***The Web Publisher - To Publish Or Not To Publish***
Autor:	Naveena Berenyu
No.:	4
Fecha :	Julio-Agosto 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo habla del nuevo elemento de Rose 98i, el Web Publisher, el cual permite, entre otras cosas, publicar completamente en el Web, el diseño del proyecto con todos sus detalles. lo cual tiene algunas ventajas, por ejemplo si se está trabajando en un proyecto muy grande, lo más conveniente es tener acceso al diseño actualizado que se está realizando lo más rápido posible, entonces la publicación en el Web sería conveniente, como analista, arquitecto, desarrollador, líder delproyecto y editor de la documentación. en cada caso el autor explica los beneficios de la publicación en el Web del diseño del proyecto.

Título:	***Rose, Modeling, and UML Extensibility***
Autor:	Tom Schultz
No.:	4
Fecha :	Julio-Agosto 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo trata de las aportaciones que ha dado el UML al nuevo enfoque de modelado y visualización. Primero se empieza por definir que es modelado y visualización, después explica que es lo que en realidad se pretende lograr mediante estos elementos en el nuevo paradigma de análisis, diseño y programación orientada a objetos. El autor menciona algunas aspectos de Rose que pueden usarse para diseñar sistemas muy grandes mediante el uso de Stereotypes y Tagged-Values, que son extensiones que se le han hecho al UML.

Título:	***Rational ProductIntegration Focus: Rose and the Rational Unified Process***
Autor:	Per Kroll
No.:	4
Fecha :	Julio-Agosto 1998
Ubicación	http://www.researchitect.com/mag/

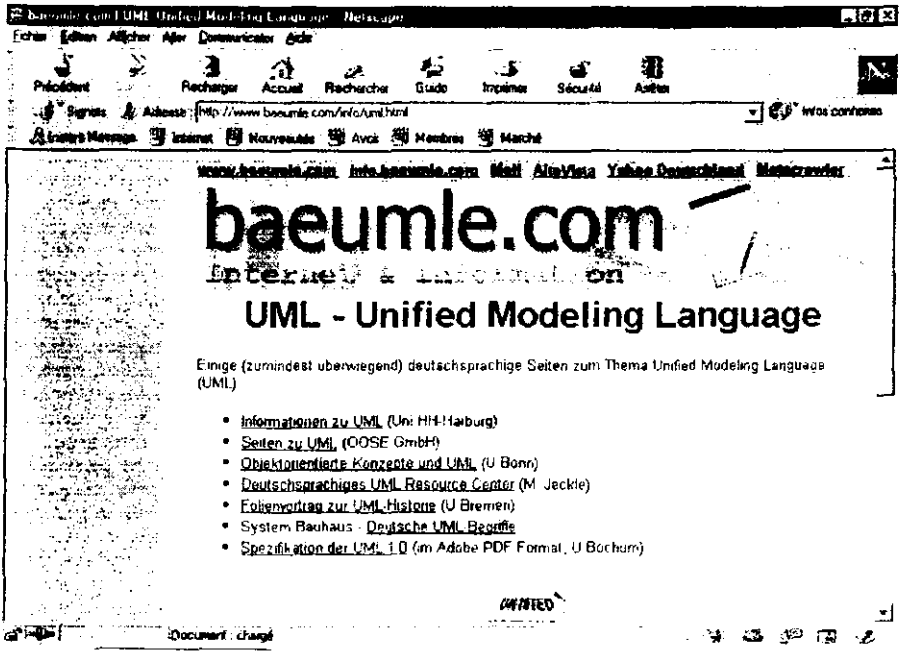
Este artículo habla sobre la importancia de saber usar las nuevas herramientas, porque probablemente existan y las conozcamos pero no sepamos como usarlas. Rational Unified Process es una herramienta que precisamente es una guía para aprender a usar el UML. Este artículo explica de que forma se han integrado estas dos herramientas y la forma en que facilitan el trabajo a los desarrolladores.

Título:	***Landscaping with Rose - Planning a System Architecture***
Autor:	Magnus Christerson y David Norris
No.:	4
Fecha :	Julio-Agosto 1998
Ubicación	http://www.researchitect.com/mag/

Este artículo habla sobre la importancia de diseñar una buena arquitectura, de las consecuencias que traen el diseño de una mala arquitectura y como evitarlo. Una de las cosas que ayudan es las limitaciones que el software Rose tiene, por ejemplo evita mover los objetos sin que haya para eso una razón lógicamente correcta en cuanto al diseño. Se mencionan algunas recomendaciones para diseñar una buena arquitectura aun con sistemas muy grandes.

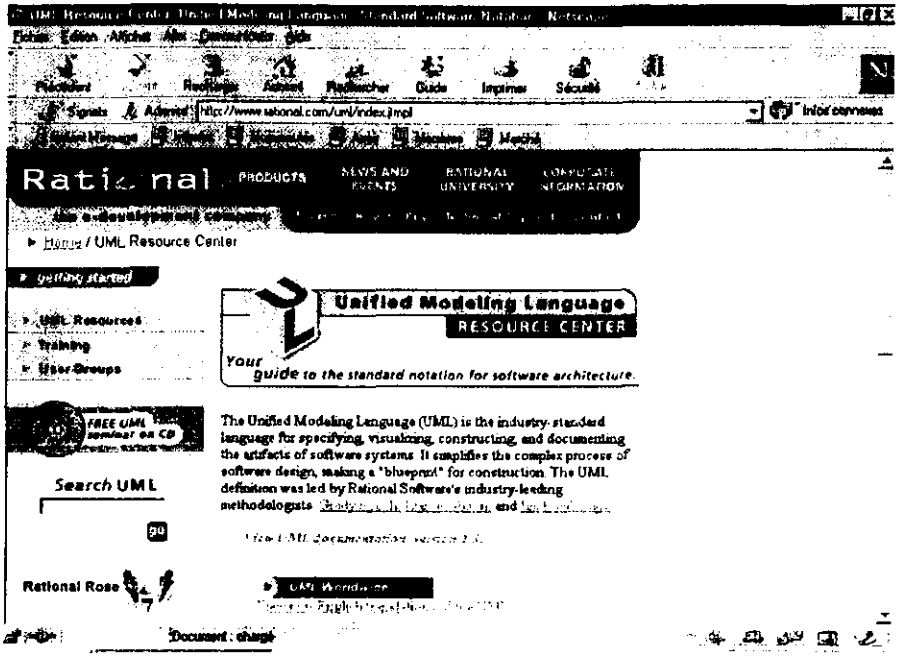
2.4 Internet

- http://www.baeumle.com/info/uml.html



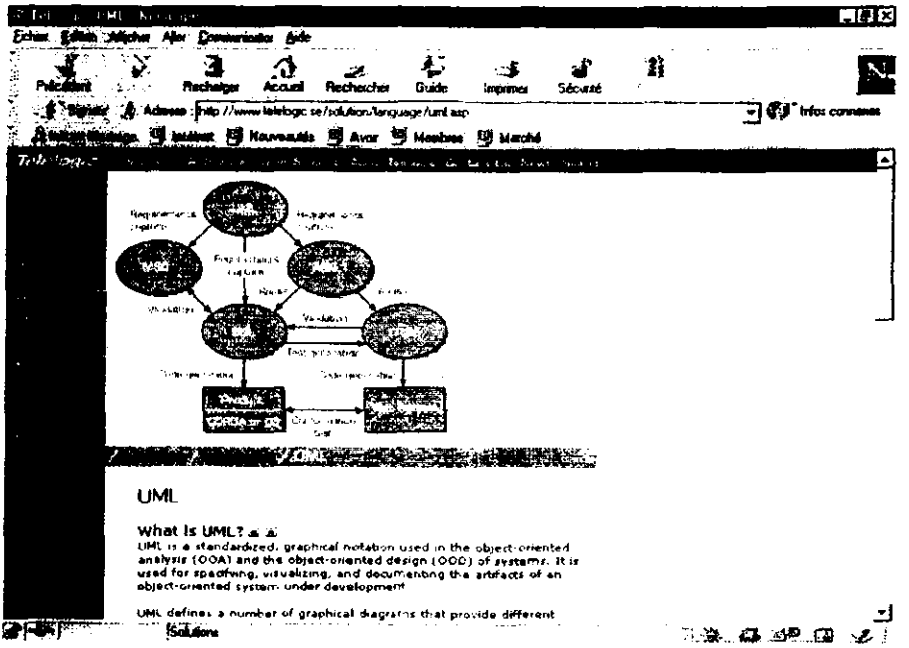
Esta página contiene varias ligas de direcciones a otros sitios de la red en inglés y alemán que tratan sobre el tema del UML (Unified Modeling Language). Las direcciones más importantes contienen información sobre la metodología. También hay un diccionario UML e información sobre la especificación OMG (Object Management Group). También hay sitios donde se puede intercambiar conocimitnos sobre el tema y hacer preguntas a expertos que estén dispuestos a contestar. Finalmente hay una liga a varias herramientas CASe para el modelado de sistemas usando UML.

- http://www.rational.com/uml/index.jtmpl



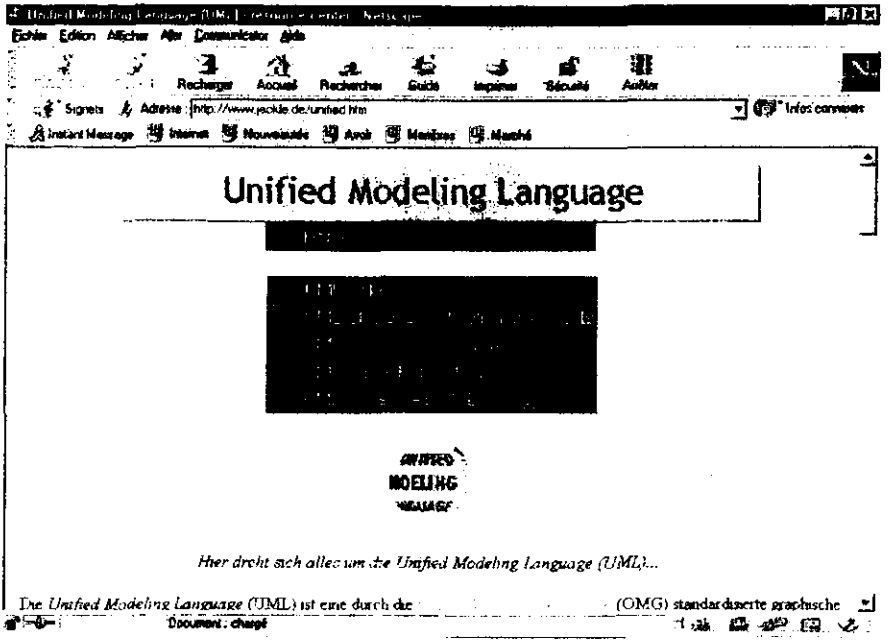
Esta página tiene una liga a la documentación del UML con todas las versiones existentes hasta ahora. También trae ligas con información acerca de los tres autores que crearon el UML Grady Booch, Ivar Jacobson y Jim Rumbaugh.

- http://www.telelogic.se/solution/language/uml.asp



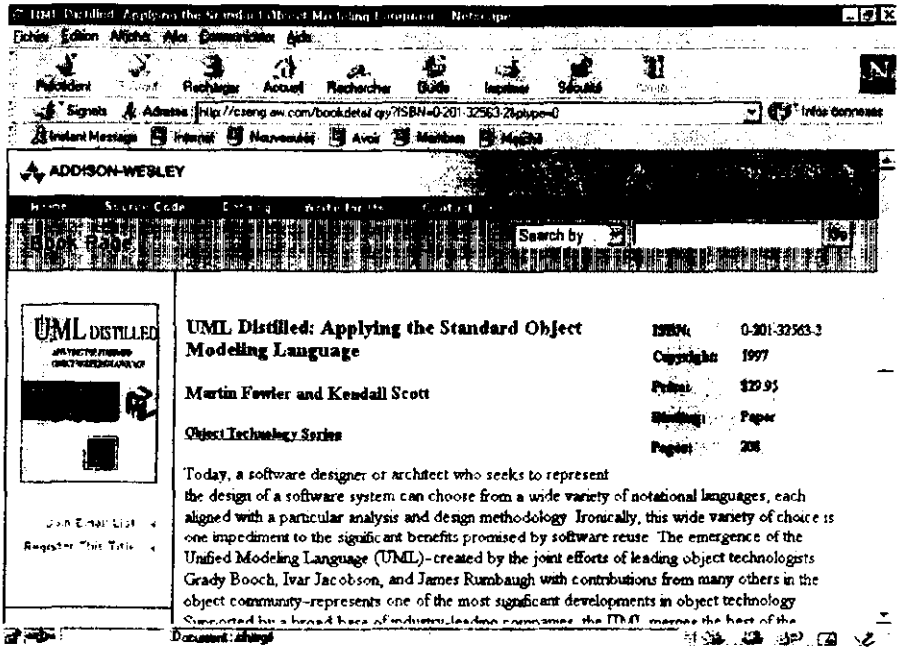
Esta página contiene información general acerca de lo que es UML, cómo surge, cómo se usa y cómo se estandariza.

- http://www.jeckle.de/unified.htm



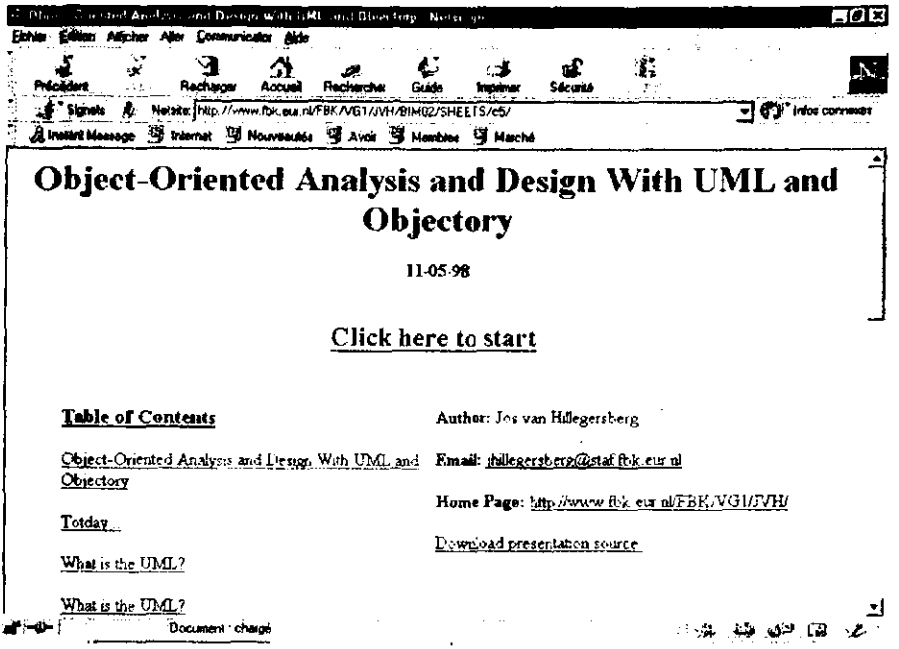
Una vez más su contenido es acerca de las características del UML, herramientas case para modelar sistemas con esta metodología. Y una parte muy interesante son las noticias y nuevas publicaciones sobre el tema, tanto en inglés como en alemán.

- http://cseng.aw.com/bookdetail.qry ?ISBN=0-201-32563-2&ptype=0



Esta página tiene algunos capítulos importantes disponibles del libro **UML Distilled** de Martin Fowler y Kendall Scot.

- http://www.fbk.eur.nl/FBK/VGI/JVH/BIM02/SHEETS/c5/



Object-Oriented Analysis and Design With UML and Objectory

11-05-98

Click here to start

Table of Contents

Object-Oriented Analysis and Design With UML and Objectory

Today

What is the UML?

What is the UML?

Author: Jos van Hillegersberg

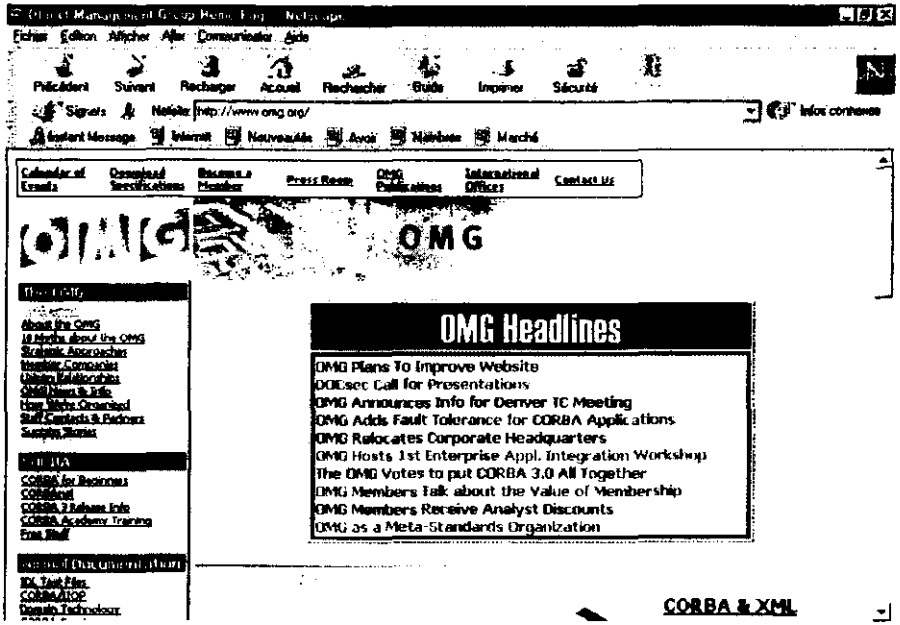
Email: jhillegersberg@staf.fbk.eur.nl

Home Page: http://www.fbk.eur.nl/FBK/VGI/JVH/

Download presentation source

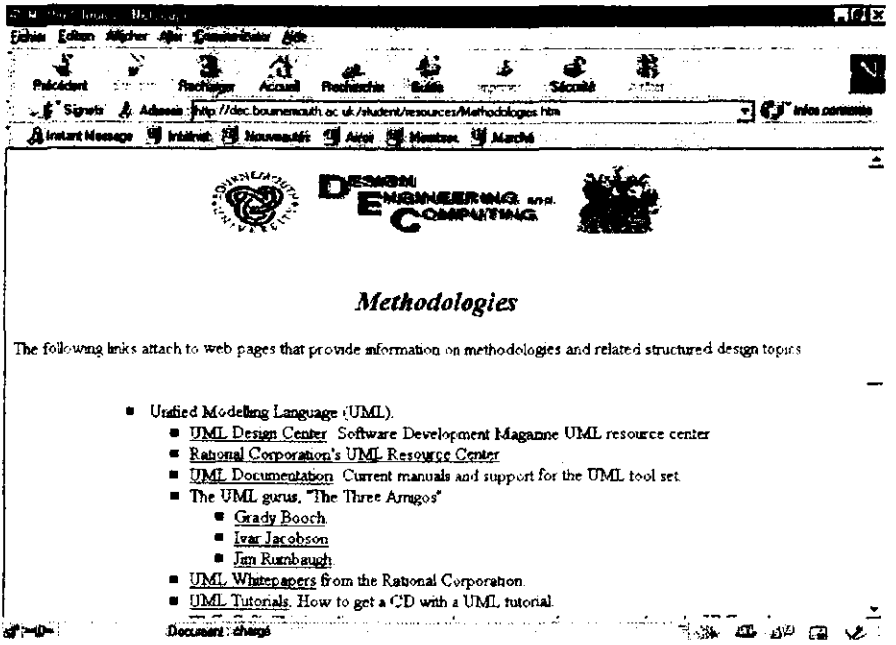
Esta página contiene un poco de historia y antecedentes del UML, así como información de cómo realizar análisis y diseño orientado a objetos usando UML y un producto CASE que se llama **Objectory**, el cual está basado en UML. También define y explica la filosofía y métodos del UML y trae algunas noticias importantes.

- http://www.omg.org/



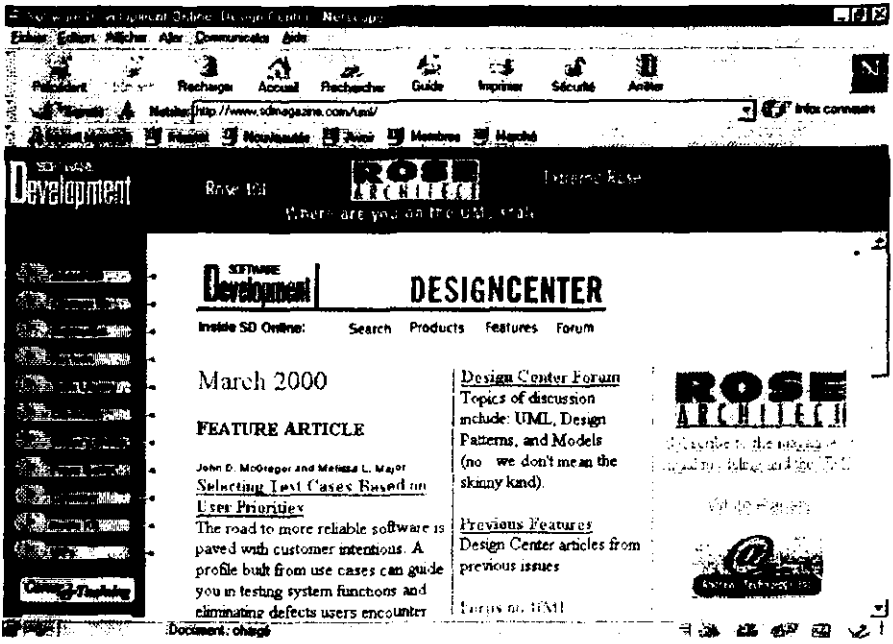
Página oficial de OMG (Object Management Group). Contiene información general sobre el modelado de objetos. Además por ser una organización que rige el modelado de objetos contiene las últimas noticias y actualizaciones del análisis y diseño orientado a objetos así como metodologías, obviamente el centro de atención es UML.

- http://dec.bournemouth.ac.uk/student/resources/Methodologies.htm



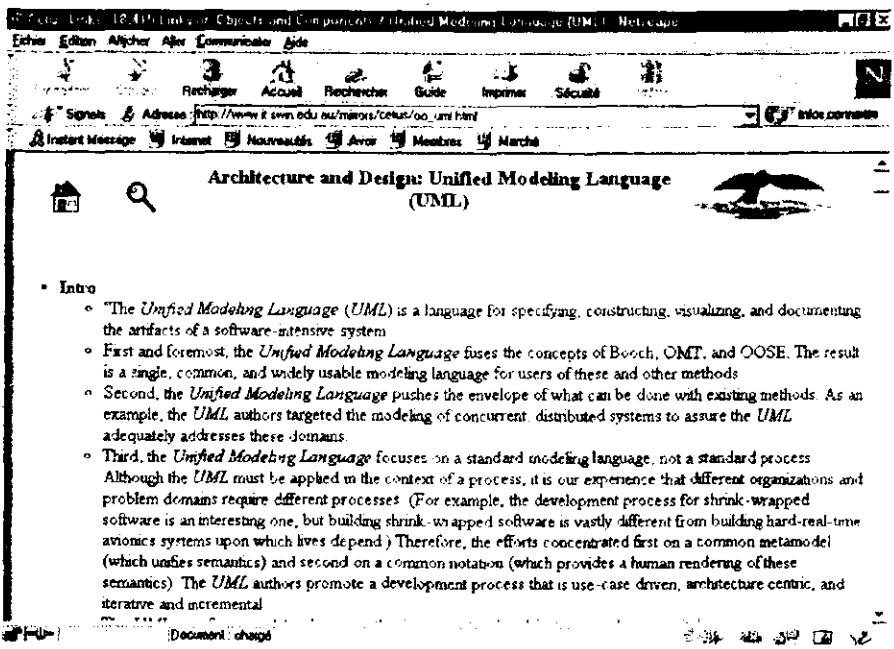
Esta página contiene ligas a diferentes puntos de vista de la metodología UML, una vez más a herramientas CASE para el modelado de objetos. También tiene información importante sobre la realización del análisis y diseño y los riesgos de no llevarlos a cabo correctamente. Aparte del UML, contiene información sobre otras metodologías, que son: SSADM (Structured System Analysis and Design Methodology) y DSDM (Dynamic Systems Development Method).

- http://sdmagazine.com/uml/



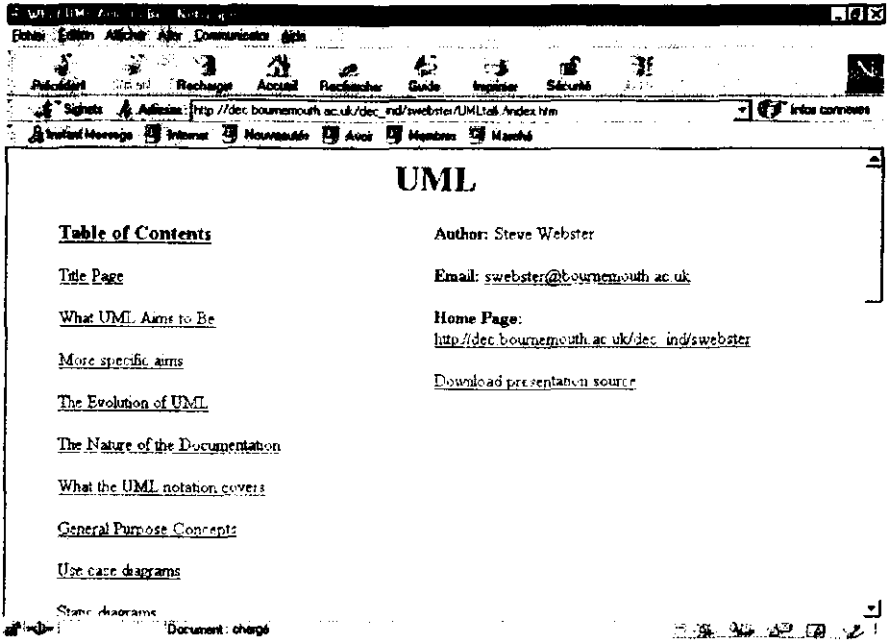
Página de la revista Software Development, que trae bastante información acerca del UML, así como una liga al centro de diseño de UML, donde hay varios artículos de expertos y algunos ejemplos del uso de esta metodología.

- http://www.it.swin.edu.au/mirrors/cetus/oo_uml.html



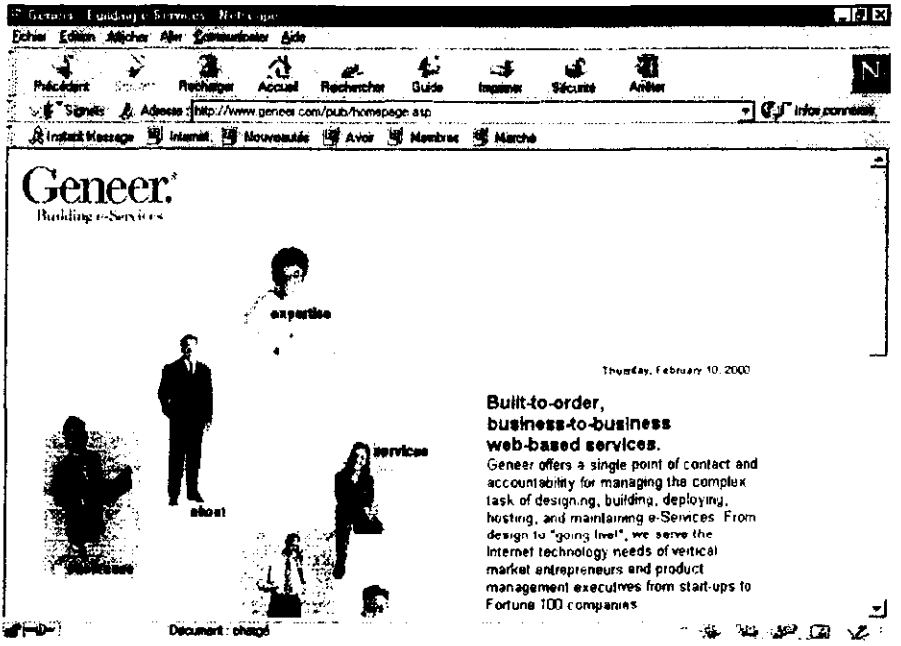
Contiene ligas a tutoriales, estándares, glosarios, ejemplos basados en UML. También una liga a una página que contiene otras metodologías para el análisis y diseño orientado a objetos que no son UML.

- http://dec.bournemouth.ac.uk/dec_ind/swebster/UMLtalk/index.html



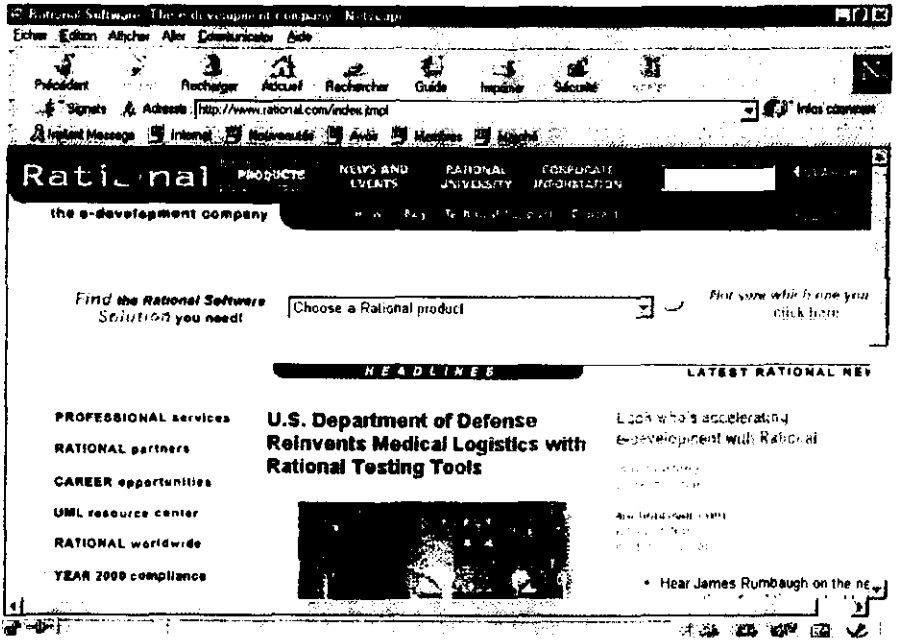
Está página trae información acerca de la evolución del UML y de la importancia de la documentación en la creación de sistemas.

- http://www.geneer.com/pub/homepage.asp



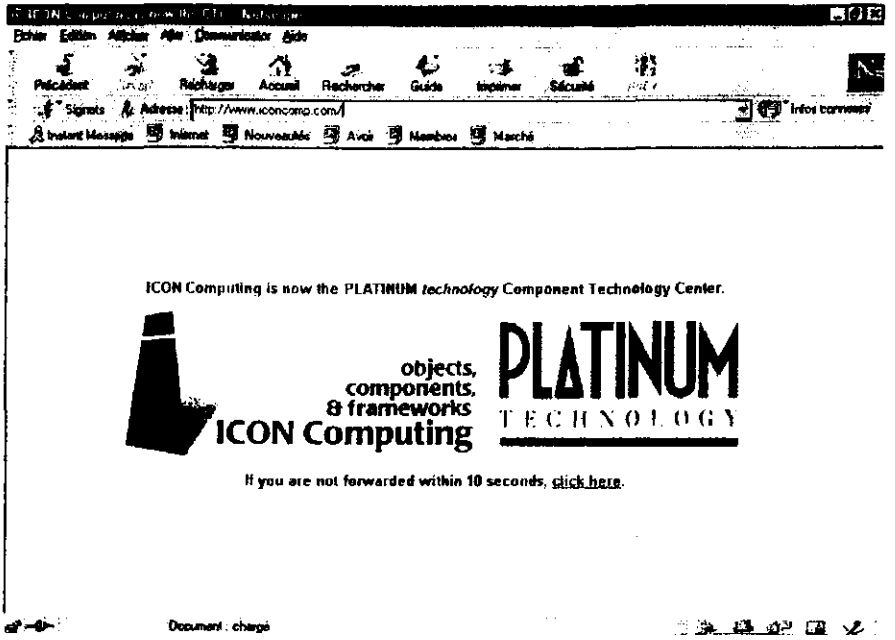
Página de Geneer, que contiene el software Code Science, el cual realiza las fases de análisis, arquitectura, construcción y entrega.

- http://rational.com/index.jtmpl



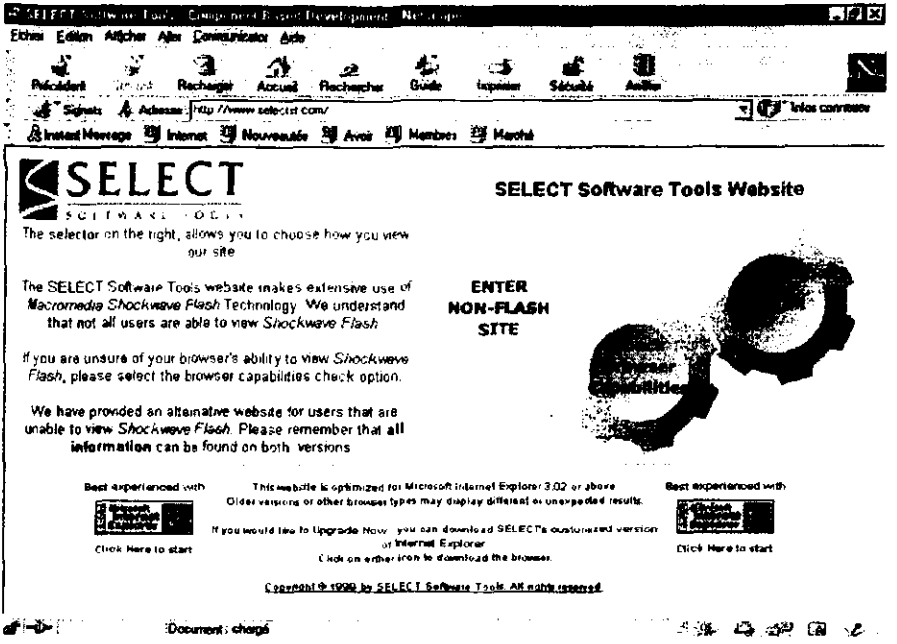
Página de Relational Software, que contiene el software Objectory/RUP (Rational Unified Process), que cubre las fases de incepción, elaboración, construcción y transición.

- http://www.iconcomp.com/



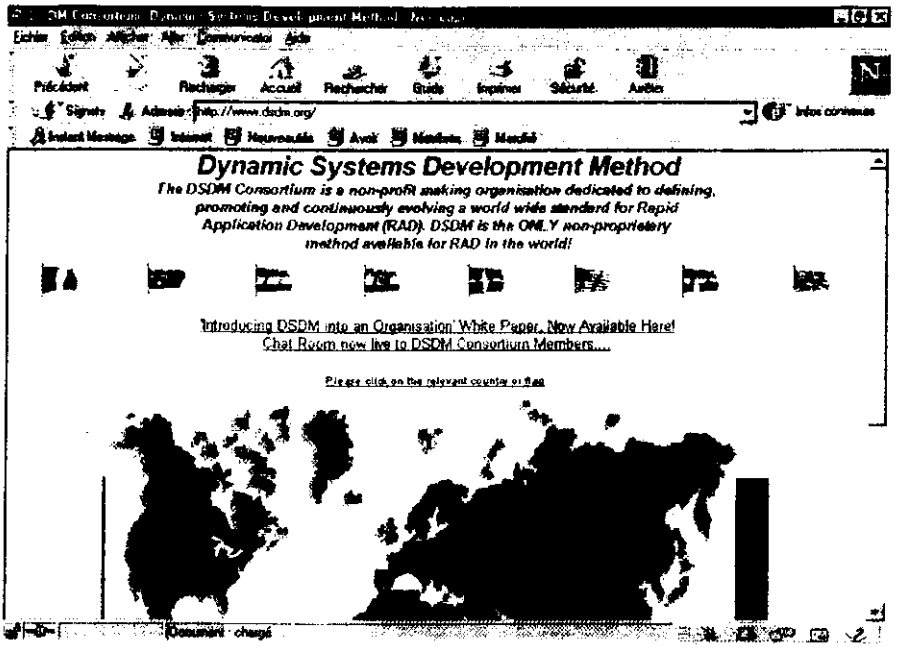
Página de Icon Computing, que contiene el software Catalysis, el cual cubre las fases de requerimientos, especificación del sistema, diseño arquitectónico y diseño interno de componente.

- http://www.selectst.com/



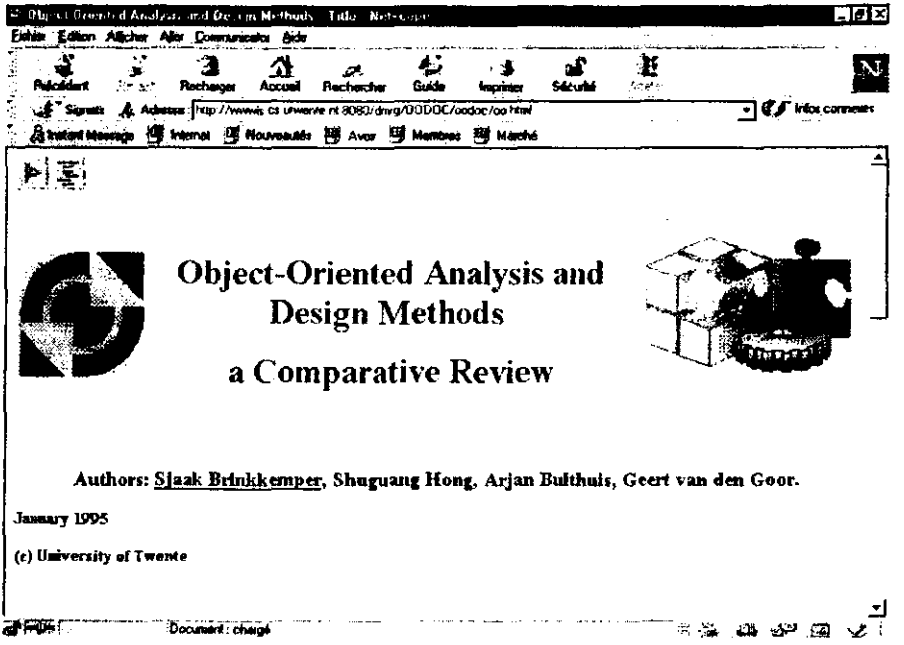
Página de SELECT Software, que contiene le software SELESCT Perspective, el cual cubre las fases de disponibilidad, análisis, prototipos, plan de desarrollo, indremento de diseño y construcción.

- http://www.dsdm.org/



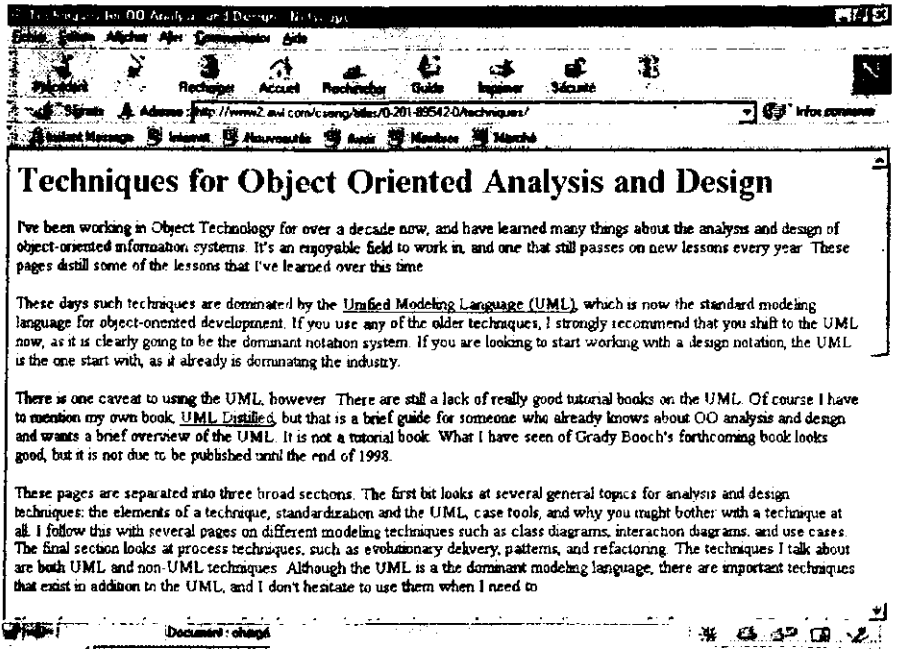
Página de Dynamic Systems Development Method (DSDM) Consortium, que contiene el software DSDM, el cualcubre las fases de estudio de factibilidad, estudio del negocio, iteración del modelo funcional, iteración del modelo y construcción e implementación.

- http://wwwis.cs.utwente.nl:8080/dmrg/OODOC/oodoc/oo.html



Página con un listado de varios métodos orientados a objetos, con explicación y notación de cada uno de ellos. Así como un cuadro comparativo de los métodos incluidos y un par de herramientas CASE, por supuesto orientadas a objetos.

- http://www2.awl.com/cseng/titles/0-201-89542-0/techniques/



Techniques for Object Oriented Analysis and Design

I've been working in Object Technology for over a decade now, and have learned many things about the analysis and design of object-oriented information systems. It's an enjoyable field to work in, and one that still passes on new lessons every year. These pages distill some of the lessons that I've learned over this time.

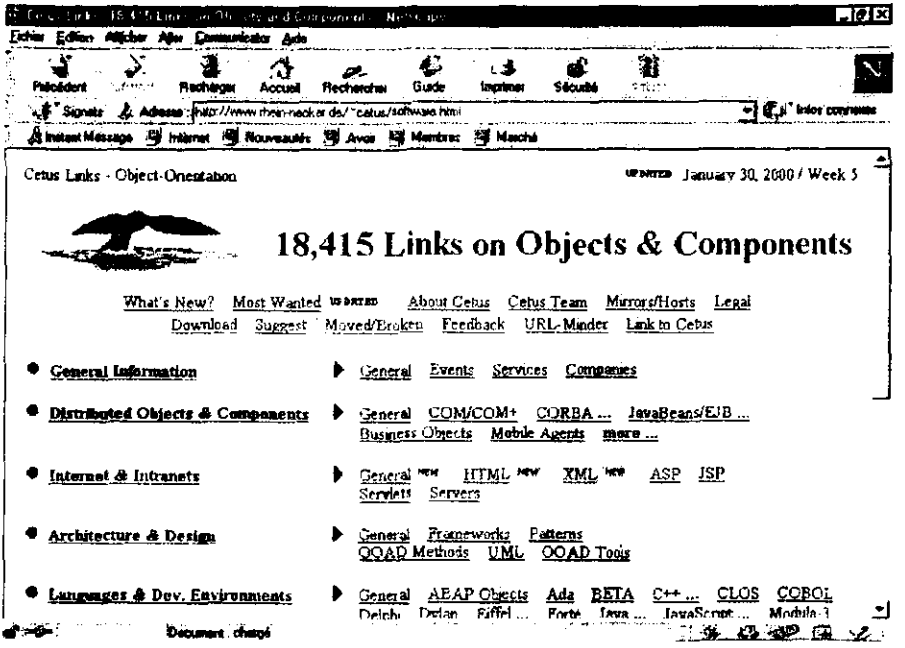
These days such techniques are dominated by the Unified Modeling Language (UML), which is now the standard modeling language for object-oriented development. If you use any of the older techniques, I strongly recommend that you shift to the UML now, as it is clearly going to be the dominant notation system. If you are looking to start working with a design notation, the UML is the one start with, as it already is dominating the industry.

There is one caveat to using the UML, however. There are still a lack of really good tutorial books on the UML. Of course I have to mention my own book, UML Distilled, but that is a brief guide for someone who already knows about OO analysis and design and wants a brief overview of the UML. It is not a tutorial book. What I have seen of Grady Booch's forthcoming book looks good, but it is not due to be published until the end of 1998.

These pages are separated into three broad sections. The first bit looks at several general topics for analysis and design techniques: the elements of a technique, standardization and the UML, case tools, and why you might bother with a technique at all. I follow this with several pages on different modeling techniques such as class diagrams, interaction diagrams, and use cases. The final section looks at process techniques, such as evolutionary delivery, patterns, and refactoring. The techniques I talk about are both UML and non-UML techniques. Although the UML is a the dominant modeling language, there are important techniques that exist in addition to the UML, and I don't hesitate to use them when I need to.

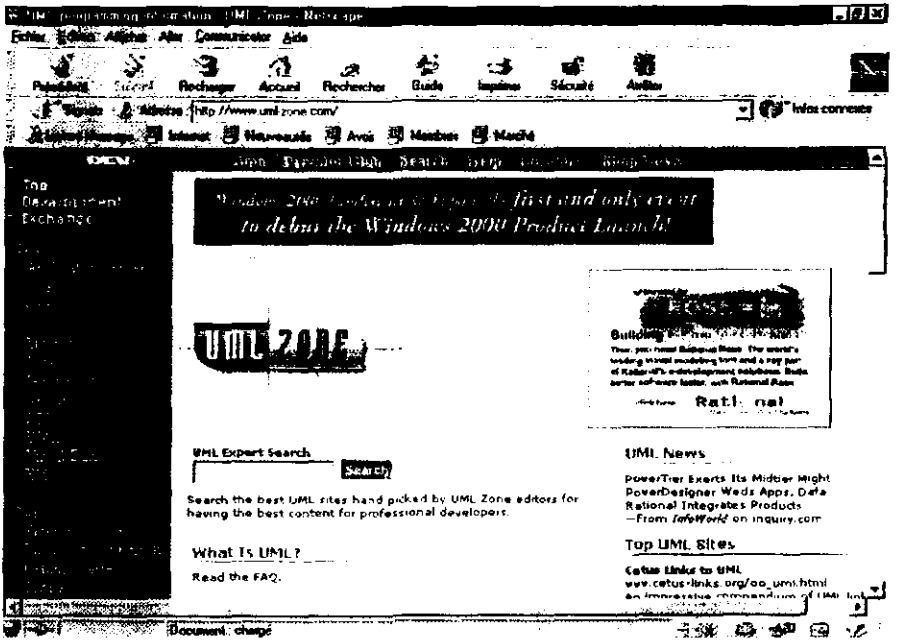
Página que describe de manera general al lenguaje UML, habla de cómo se estandarizó y de la razón por la cual se debe hacer un análisis y un diseño antes de programar un sistema. Se explican los elementos generales del UML, como son: casos de uso, diagramas de clase, tarjetas CRC, diagramas de paquete, diagramas de estado, de interacción y de actividad. Incluye también algunos elementos propios del UML, desarrollo incremental, patrones y *refactoring*.

- http://www.rhein-neckar.de/~cetus/software.html



Página con información de la tecnología orientada a objetos abarca: información general, componentes distribuidos y de objetos, diseño y arquitectura, lenguajes y ambientes de desarrollo, bases de datos y repositorios y finalmente un poco sobre administración de proyectos orientados a objetos.

- http://www.uml-zone.com/



Página que ofrece el producto GDPro para el modelado en UML, pero también incluye novedades sobre el UML, como artículos y las ultimas actualizaciones al UML.

2.5 Participación en proyectos

- **Curso de UML**
  DCAA
  Marzo 1999
  Impartido por el Ing. Alexei Dezotti
  Traducción de la presentación del curso de Análisis y Diseño Orientados a Objetos (UML), proporcionado por Rational Rose en http://www.rational.com/

CAPITULO 3
Marco Conceptual

Marco Conceptual

3.1 Historia del software

La historia del software siempre va acompañada de la historia del hardware; primero se construye la estructura física de una computadora y después el lenguaje que servirá para darle instrucciones, es decir, el software.

A continuación se presenta una tabla con una breve historia del software, que incluye a algunos de sus representantes más importantes.

1600s	Gottfried Wilhelm von Leibniz (1646-1716)	Alemania	Construyó una máquina calculadora que podía sumar y multiplicar. Sin embargo, su principal aportación en el desarrollo del software fue el establecimiento de los fundamentos de la lógica simbólica y del sistema binario.
1800s	Charles Babbage (1792-1871)	Inglaterra	Planteó una solución para calcular e imprimir con precisión las tablas de logaritmos. Ideó un mecanismo para controlar la operación de la máquina con base en tarjetas perforadas. Además de haber construido una sucesión de su máquina analítica, escribió algunos programas para procesarlos, pero no logró ejecutarlos.
	Ada Byron Condesa de Lovelace (1815-1852)	Inglaterra	Traduce, anota y mejora una versión en italiano de la descripción de la máquina de Babbage. Además realiza un sofisticado programa para calcular los números de Bernoulli. Ada es la primera mujer programadora en la historia del software.
	George Boole (1815-1864)	Inglaterra	Fue el creador del álgebra booleana, sistema de símbolos y reglas de procedimiento para ejecutar operaciones no solamente con números sino también con letras, objetos y conceptos en general. Los operadores lógicos más importantes son: AND, OR y NOT. Con estos signos o conectores lógicos, se puede sumar, restar, multiplicar, dividir o realizar otras operaciones como la comparación de números y conceptos.
	Leonardo Torres y Quevedo (1852-1938)	España	Aportó ideas sobre el funcionamiento de la computadora y del software. Acuñó el término *automático*, que utilizó en algunos tratados que escribió sobre autómatas; por ejemplo, *Un ensayo sobre autómatas*. Sus ideas fueron aplicadas en el diseño de mecanismos automáticos y sus conceptos sobre los autómatas pueden considerarse como los antecedentes del software para las máquinas que actualmente se construyen con la moderna tecnología de la robótica.

1900s	Bertrand Rusell (1872-1970)	Gales	Estableció una sólida base de lógica matemática en la que se apoya la teoría de sistemas. Los puntos fundamentales de su libro *Principles of Mathematics* son: 1. Probar que todas las matemáticas tratan exclusivamente con conceptos definibles de un reducido número de conceptos lógicos fundamentales. 2. Demostrar que todas las proposiciones son deducibles de un reducido número de principios lógicos fundamentales. 3. Explicar los conceptos fundamentales que las matemáticas aceptan como indefinibles.
	John Louis Von Neuman (1903-1954)	Hungría	Se involucra con Herman Goldstine para la construcción de la *ENIAC*. Su principal aportación fue el haber concebido la idea de programa almacenado dentro de la computadora; concepto que transformó la idea de las máquinas calculadoras en verdaderas computadoras y sentó las bases para el nacimiento del software.
	Alan Turing (1912-1954)	Inglaterra	Creador de la *Máquina de Turing* que existe únicamente en papel como un conjunto de especificaciones, es el prototipo de la máquina lógica. Su documento *Computing Machinery and Inteligence*, representa uno de los documentos más interesantes en la historia del software, en éste establece su convicción de que las computadoras pueden llegar a tener la capacidad de imitar perfectamente la inteligencia humana y que esto sería posible en el 2000.
	Konrad Zuse	Alemania	En 1945 desarrolla el lenguaje *Plankalkül* (programa de cálculo). También escribió algunos algoritmos para clasificar un programa para evaluar la construcción sintáctica de una fórmula mediante el análisis de la estructura de sus paréntesis de la misma.
	Herman Goldstine y John von Neuman	Inglaterra	En 1946 desarrollan los primeros diagramas de flujo como parte de la técnica para hacer los programas de la computadora *ENIAC*. El diagrama de flujo consistía en una serie de rectángulos unidos por líneas y flechas donde el principio y el fin se indicaban con un pequeño círculo.

1990s	Asociación Alemana de Matemáticas y Mecánica Aplicadas Asociación de Maquinaria de Computación de E.U.	Suiza	En 1958 se crea un *Comité Internacional* para estudiar la posibilidad de construir un lenguaje de programación universal como medio de comunicación entre personas. Así nació el lenguaje *Algol*.
		Francia	En 1967 se crea *Simula*, el primer lenguaje de programación orientada a objetos, sólo fue usado en Europa, nunca llegó a E.U.
	Dennis Ritchie	Laboratorios Bell, Nueva Jersey, E.U.	En 1972 crea el lenguaje de programación *C* de alto nivel que permite tener acceso a la programación a nivel bit.
		E.U.	En 1972 se crea Smalltalk, uno de los mejores lenguajes OO que existen. Sin embargo, fue poco usado, probablemente debido a que las máquinas no eran lo suficiente poderosas.
	Niklaus Wirth	Instituto Federal de Tecnología, E.U.	A finales de 1970 crea el lenguaje *Pascal*, con gran facilidad de compilación en diferentes computadoras.
	Departamento de Defensa de E.U. y Jean Ichbiah de Francia	E.U.	En el período de 1974 a 1983 crean el lenguaje de programación *Ada*, basado en la filosofía de *Algol* y *Pascal*, debido a lo cual es modular y facilita la programación estructurada. En un lenguaje sólido y bien diseñado que se aplica en computadoras grandes.
	Bjarne Stroustrup	E.U.	En 1986 crea *C++*

3.2 Historia de los paradigmas de programación

La forma de comunicarse con la máquina y la forma en la que se usa el software han sido diferentes a través del tiempo y de las diferentes arquitecturas de máquina. He aquí una breve historia de los distintos tipos de lenguajes de programación y sus paradigmas de programación.

1800s	Lenguaje máquina	El lenguaje era extremadamente dependiente de la máquina. El programador virtualmente tenía que pensar como la máquina, conocer el conjunto de sus instrucciones y cómo se representaban los datos.	El paradigma era todo con base en la forma de funcionar de la máquina.	El programador estaba trabajando bajo la premisa básica de la máquina: hay datos, y hay instrucciones que procesan la información. Por lo tanto tenía que pensar como la máquina.
1920s	Lenguaje ensamblador	Al crecer las computadoras en poder, los ensambladores y los lenguajes ensambladorores comenzaron. Esto creó un nivel de abstracción entre la base de la máquina y el programador, permitiéndole a éste último pensar más como ser humano y menos como computadora. El programador ya no tuvo que preocuparse de poner los ceros y los unos correctamente, podía usar mnemónicos, constantes, literales y pseudo-lenguajes. Los lenguajes ensambladores también permitían funcionalidad para aislar y reusar datos e instrucciones a través de subrutinas. Las subrutinas extendieron básicamente el conjunto de instrucciones de la computadora.	El paradigma seguía basándose en la forma de funcionar de la máquina, aunque con algunas facilidades.	Los programadores siguen trabajando bajo la premisa de la máquina en su abstracción: hay datos y hay instrucciones y subrutinas que procesan los datos.
1950s	Lenguaje de alto nivel	Estos lenguajes compiladores e intérpretes crearon otro nivel de abstracción entre el programador y la máquina. El programador podía reposicionar miles de líneas código de ensamblador con una expresión o declaración. Una vez más, los lenguajes de programación dieron otro paso adelante en la forma de pensar de la gente. Además, los lenguajes de alto nivel eran (en teoría) independientes de la máquina. Los lenguajes compiladores e intérpretes también tienen funciones y subrutinas.	El paradigma aún está basado en la forma de funcionar de la computadora, aunque los niveles de abstracción van aumentando.	Los programadores siguen con una premisa básica: hay datos, y hay funciones/subrutinas/expresiones/declaraciones que procesan los datos.

1960s	Lenguajes estructurados	Estos lenguajes (*C*, *Pascal*) incluyeron la ayuda *goto* para localizar y clarificar el código. La productividad del programador creció en un 100%.	El paradigma está aún basado en la forma de funcionar de la máquina, pero se logra implementar el análisis y el diseño estructurado.	Con todos los avances del software, los programadores aún tenían que trabajar bajo la premisa básica de la máquina de Von Neuman: hay datos, hay instrucciones/funciones/subrutinas que procesan datos.
1970s	Lenguajes orientados a objetos	Las características principales son: soporte de módulos, los módulos deben soportar abstracción de datos, que incluyen: Ocultación de información. ▪ Encapsulación. ▪ Manejo automático de memoria (no necesariamente colección de basura). ▪ Datos están definidos por tipos de dato abstracto (ADT, abstract data type) y basados en módulos. Habilidad de extender clases existentes. ▪ Soporte de polimorfismo y ocultamiento dinámico. ▪ Herencia múltiple	Finalmente, se agregó otra capa de abstracción en la parte alta de la máquina. Esto servía para que los programadores pudieran escribir códigos de acuerdo a su pensamiento natural, excluyendo casi en su totalidad la forma en la que trabaja la máquina. Así surgió también el análisis y diseño orientados a objetos.	El análisis OO tiene fuertes raíces en los diagramas ER (entidad-relación), y en el modelado de información, ya en uso en el diseño estructurado. El análisis y el diseño orientado a objetos modela diagramas ER extendidos, agregándoles funcionalidad a las entidades de datos. Además, se incluyó el concepto de herencia a los modelos de análisis y diseño OO. El agregar funcionalidad a los diagramas ER evitó la división de datos/funcionalidad que se daba en los modelos de análisis estructurado.

3.3 Programación orientada a objetos

3.3.1 Breve historia de la programación orientada a objetos

La programación orientación a objetos no es un concepto nuevo, de hecho, su origen se ubican en Noruega a finales de los 60s, en el Centro de Cómputo de Noruega, donde Kristen Nygaard y Ole-Johan Dahl desarrollaron un lenguaje llamado *Simula 67*. Este lenguaje fue el primero en introducir los conceptos de clases, co-rutinas y subclase, como los lenguajes orientados a objetos actuales.

A mediados de los 70s, se desarrolló el lenguaje *Smalltalk* en el Centro de Investigaciones de Xerox Palo Alto. *Smalltalk* es el primer lenguaje, completo y robusto, orientado a objetos. En Smalltalk, cada elemento del lenguaje se implementó como un objeto. El ambiente de programación y la cultura que lo rodea eran orientados a objetos. Incluso ahora, *Smalltalk* es considerado el lenguaje orientado a objetos más puro.

Con los lenguajes *Simula67* y *Smalltalk* se desarrollaron los lenguajes de programación basado en clases que almacenan datos y operaciones juntos. Estos lenguajes que sólo eran conocidos en círculos universitarios, empiezan a ser relativamente accesibles a las compañías grandes hasta los 80s. A inicios de los 80s, Bjarne Stroustrup, de los Laboratorios Bell de AT&T, extendió el lenguaje *C* para crear el lenguaje *C++*, lenguaje que soporta la programación orientada a objetos. Con el mejoramiento de la herramienta y su lanzamiento comercial AT&T y otros vendedores captaron la atención de los clientes respecto a la orientación a objetos. El desarrollo de computadoras personales poderosas y la desaparición del obstáculo precio/desempeño contribuyeron a la propagación de la tecnología orientada a objetos. Específicamente, la máquina Macintosh en 1987 y la máquina Next en 1988, que fueron piedras de toque significativas para el uso de la nueva tecnología orientada a objetos.

A finales de los 1980s, las aplicaciones necesitaban satisfacer requerimientos más sofisticados: usar estructuras de datos y arquitecturas más complejas y ser compatibles con una amplia gama de plataformas. Todos estos factores necesitaban sistemas más complejos y a gran escala. La programación orientada a objetos proporciona una mejor manera de manejar la complejidad de la tecnología, permite programar a niveles más altos de abstracción, del objeto a la clase, a la librería de clase y al final al marco de trabajo de la aplicación entera; los estilos de programación orientada a objetos incrementan el reuso de código y los programas pueden adaptarse más fácilmente a los cambios de circunstancias. Una de las ventajas de la orientación a objetos consiste en tener un desarrollo de software más coherente. Por esto, el diseño y la programación orientada a objetos han tenido un amplio desarrollo desde los 90s.

Aunque los primeros lenguajes de programación orientados a objetos surgieron en los 60s, la creación de metodologías de análisis y diseño no surgió hasta los 80s, su principal precursor fue Grady Booch. Después, otras personas involucradas con el desarrollo de software, comenzaron a crear sus propias metodologías; y aunque todas diferentes, todas trataban de mantener las principales características de la orientación a objetos.

3.4 UML

3.4.1 Historia del UML

Rational Rose Software desarrolló el *UML* en colaboración con Grady Booch, Ivar Jacobson y Jim Rumbaugh, y junto con otros desarrolladores que se han ido incorporando o aportando nuevas ideas. 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.

Aunque los lenguajes orientados a objetos surgieron a mediados de los 70s, no fue sino hasta finales de los 80s cuando varios desarrolladores de metodologías propusieron sus ideas para el análisis y diseño orientado a objetos. Algunas técnicas, ya tradicionales, como los diagramas ER (entidad-relación), influenciaron a estas personas.

El número de lenguajes orientados a objetos se incrementó en un 50% durante el periodo comprendido entre 1989 y 1994. Varios usuarios tuvieron dificultad para encontrar una metodología que cubriera sus expectativas por completo, y así surgió la "guerra de las metodologías". A mediados de 1990, comenzaron a aparecer metodologías muy completas, principalmente la de Grady Booch. Éstas empezaron a combinar sus técnicas con las de otras personas, y surgieron metodologías bastante buenas como la OOSE (Object-Oriented Software Engineering) de Ivar Jacobson, la OMT (Object Modeling Technique) de Jim Rumbaugh y la OOADA (Object Oriented Analysis and Design with Applications) de Grady Booch. Cada una de éstas constituye un método completo, y cada uno de ellos tiene aportaciones interesantes. OOSE tiene un enfoque orientado a casos de uso, que proporciona excelente soporte para la ingeniería de negocio y análisis de requerimientos. OMT es especialmente expresivo para el análisis y para los sistemas de información con muchos datos. OOADA es particularmente bueno durante las fases de diseño y construcción de proyectos.

El desarrollo del *UML* empezó en octubre de 1994, cuando Grady Booch y Jim Rumbaugh, de la corporación Rational Software, empezaron su trabajo de unificación de las metodologías OMT y OOADA. Metodologías que ya estaban teniendo un desarrollo independiente y que ya eran reconocidas como líderes a nivel mundial. Booch y Rumbaugh trabajaron juntos para unificar su trabajo. Como resultado de sus esfuerzos en octubre de 1995 fue liberada la versión 8.0 del Método Unificado (*Unified Method*). En el otoño de 1995, Ivar Jacobson se unió al equipo de unificación de Rational Software con su metodología OOSE. El resultado que se esperaba obtener cambió y fue nombrado Proceso Unificado de Rational (*Rational Unified Process*).

Las razones que estas tres personas tuvieron para unificar sus trabajos en un sólo lenguaje de modelado, fueron tres: primero, estos métodos ya se estaban desarrollando de forma separada e independiente. Era mejor que ellos continuaran la evolución juntos que cada quien por su lado, esto permitía eliminar las diferencias innecesarias que en un futuro confundirían a los usuarios; segundo, unificando las semánticas y la notación se podría estabilizar, en alguna medida, el mercado de orientación a objetos; así los desarrolladores podrían enfocarse al producto terminado, más que a los elementos que deben usar para el desarrollo; y, tercero, esperaban que su colaboración diera como resultado la mejora de todos los métodos anteriores.

Al inicio de la unificación ellos se enfocaron en el establecieron de cuatro objetivos:

- Habilitar el modelado de sistemas (no sólo de software) usando conceptos orientados a objetos

- Establecer un acoplamiento explícito a elementos tanto conceptuales como ejecutables
- Direccionar los elementos en una escala inherente a sistemas complejos de misión crítica
- Crear un lenguaje de modelado que pudiera usarse tanto por hombres como por máquinas

Los tres amigos, como fueron llamados Booch, Jacobson y Rumbaugh sabían que diseñar una notación para el análisis y el diseño orientado a objetos no era lo mismo que diseñar un sistema, ya que lo primero implica algunas dificultades, por ejemplo:

- ¿Deberá la notación incluir especificación de requerimientos? (Si, parcialmente)
- ¿Se deberá extender la notación al nivel de un lenguaje de programación visual? (No)
- Debe de haber un balance entre la expresividad y la simplicidad; si es muy simple, la notación va a limitar el rango de problemas que se pueden resolver; si es muy compleja, la notación va a abrumar al desarrollador. Hay que ser sensitivo en el caso de la unificación de métodos existentes: hacer demasiados cambios, confundirá a los usuarios existentes; entonces si el usuario no avanza con la notación, se perderá la oportunidad de comprometer a un mayor número de usuarios.

Los esfuerzos de Booch, Rumbaugh y Jacobson dieron como resultado las versiones 0.9 y 0.91 del *UML* en junio y octubre de 1996. Durante éste período, los autores del UML invitaron a la toda comunidad a hacer aportaciones. Afortunadamente recibieron muchas aportaciones que se incorporaron, sin embargo todavía era hacer algunos ajustes.

3.4.2 Evolución del *UML*

Varias organizaciones consideraron al *UML* como una estrategia para su negocio en 1996. El Grupo de Administración de Objeto (OMG, Object Managemetn Group) lanzó su Requerimiento de Propuestas (RFP, Request for Proposal), para que las organizaciones que así lo desearan enviaran sus propuestas y así contribuyeran a la definición del *UML*, versión 1.0.

Varias empresas participaron, esto dio como resultado el *UML 1.0*, un lenguaje de modelado bien definido, expresivo, poderoso y aplicable de forma general. Se envió al OMG en enero de 1997 como una respuesta inicial al RFP.

En enero de 1997, 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. El enfoque del *UML* 1.1 tenía como objetivo mejorar la claridad de las semánticas del *UML* 1.0 e incorporar contribuciones de nuevos participantes.

3.4.3 Aportadores al UML

Los aportadores al desarrollo del *UML* contribuyeron con una gran variedad de perspectivas expertas, como: las perspectivas tecnológicas del OMG, modelado de negocios, lenguaje restrictivo, semánticas del estado de las máquinas, tipos, interfaces, componentes, colaboraciones, refinamiento, marcos de trabajo, distribución y metamodelo. El *UML 1.1* es el resultado del esfuerzo de un equipo de trabajo.

Aportadores:

- **HP** es un proveedor líder de soluciones de imagen y cómputo, se enfoca en la capitalización de oportunidades en Internet y en la proliferación de servicios electrónicos. Dave Packard y Bill Hewlett establecieron la compañía en 1939. La aportación de Hewlett-Packard al UML fue establecer una relación entre modelos del UML y elementos de reuso de paquetes para facilitar la construcción de aplicaciones de marcos de trabajo basados en componentes reusables. HP abogó por una estructura base del UML y por mecanismos para definir subconjunos de métodos de reuso y de especificación de dominio para su herramienta *Fusion*. Contribuyó con el modelado de relaciones inter-modelo, el uso de patrones y las relaciones con los servicios de CORBA.
  www.hp.com

- **IBM** es líder en la creación, desarrollo y manufactura de tecnologías de información avanzadas de la industria, sistemas de cómputo, software, sistemas de redes, dispositivos de almacenamiento y microelectrónicos. IBM fue creado en 1911 en Nueva York con el nombre de Computing-Tabulating - Recording Co. (C-T-R), y en 1924 adoptó el nombre de International Business Machines. La principal contribución de IBM al UML es el OCL, Lenguaje de Restricciones del Objeto (Object Constraint Language). OCL fue desarrollado en IBM como un lenguaje para el modelado de negocios y sus derivados del método Syntropy. Éstos se usan con UML para formalizar las semánticas del lenguaje y para facilitar a los usuarios de UML la expresión de restricciones precisas en la estructura del modelo. Otra contribución de IBM al UML es la definición de conceptos fundamentales en la semántica de refinamiento y patrones.
  www.ibm.com

- **i-Logic, Inc.**, fundada en 1978, es una proveedora de soluciones de información y de consultoría de administración de la información., se especializa en información y sistemas de información. i-Logic contribuyó al UML con su gran experiencia en la definición de semánticas, el uso del comportamiento ejecutable, eventos y señales para el UML. i-Logic proporcionó los diagramas de Harel para especificar la jerárquica del comportamiento concurrencial y trabajó en la relación entre modelos de objeto y de comportamiento en el UML.
  www.ilogix.com

- **ICON Computing** contribuyó al UML con una definición de la abstracción y del refinamiento del negocio al código; en el área del modelado del comportamiento de sistemas basados en componentes y en marcos de trabajo. Sus principales áreas técnicas en el UML son: los tipos, la especificación de comportamiento, el refinamiento, la colaboración y la composición de marcos de trabajo reutilizables tomados del método *Catalysis*.
  www.iconcomp.com

- **IntelliCorp** fue fundado en 1984, se dedica a la creación de heramientas de negosio para la industria. IntelliCorp contribuyó al UML en la definición de los diagramas de actividad y flujo de objetos, así como en la formalización para el concepto de roles, y otros aspectos de modelado de objetos dinámico.
  www.intellicorp.com

- **EDS** es una empresa fundada en Inglaterra hace 37 años; se dedica a la creación de sistemas alrededor del mundo. MCI Systemhouse, como líder de integrador de sistemas MCI, ha trabajado para asegurar que UML pueda usarse en un amplio rango de dominios de aplicación. Sus expertos en sistemas de objetos distribuidos, han hecho al UML más escalable y más capacitado para direccionar elementos de distribución y concurrencia. MCI Systemhouse jugó un rol importante en la definición del metamodelo y del glosario.
  www.systemhouse.mci.com

- **Microsoft** es una empresa líder en la creación de soluciones para cómputo, fue fundada en 1975 por Bill Gates. Microsoft proporcionó expertos con la capacidad de construir sistemas basados en componentes, incluyendo componentes de modelado, sus interfaces y su distribución. También se ha enfocado en las relaciones entre UML y estándares como ActiveX y COM; y usan al UML con su tecnología de repositorio.
  www.microsoft.com

- **ObjectTime** es una empresa líder en sistemas embedidos y en sistemas con redes internas embedidas, fue fundada en 1992. ObjectTime contribuyó en las áreas de especificación formal, extensibilidad y comportamiento. Jugó un rol clave en las definiciones de máquinas de estado, comportamiento común, modelado de roles y refinamiento.
  www.objectime.com

- **Oracle** es la segunda empresa en importancia a nivel mundial en bases de datos, fue fundada en 1977. Oracle ayudó en la definición y soporte para los procesos de modelado de negocio y para describir modelos de negocio en UML. Se enfocó a las descripciones de flujo de trabajo, de diagramas de actividad y de objetos de negocios; preparó estereotipos para ajustar el UML al modelado de negocios.
  www.oracle.com

- **Paltinum Technology**, es una empresa que proporciona herramientas para el modelado y diseño de sistemas empresariales, fue fundada en 1998. Paltinum Technology contribuyó en las áreas de mecanismos de extensión, alineamiento con el MetaObject Facility, metamodelos, perspectivas de CDIF e interoperabilidad de herramientas.
  www.platinum.com

- **Ptech** es una empresa que proporciona soluciones de modelado, fue fundada en 1994 en Boston. Ptech contribuyó con expertos en metamodelos y sistemas distribuidos.
  www.ptechinc.com

- **Rational Software** es una compañía de desarrollo enfocada a hacer desarrollo de software de negocios y de infraestructuras con una combinación de herramientas, servicios y prácticas de negocios, fue fundada en 1981. Rational Software definió, técnicamente y administrativamente, el UML original y fue líder de los proyectos UML 1.0 y 1.1. Rational tiene mucha experiencia en la orientación a objetos, sistemas basados en componentes y tecnología de modelado visual.
  www.rational.com/uml/

- **Reich Technologies** contribuyó con su experiencia en colaboraciones y modelado de roles, en las asesorías en la creación de software, uso y creación de herramientas CASE para el desarrollo estructurado pero sobre todo para el desarrollo orientado a objeto.
  www.sn.no

- **Softeam** proporcionó revisiones detalladas del UML durante su evolución.
  www.softeam.fr

- **SterlingSoftware** proporciona software y servicios a nivel mundial para el desarrollo de aplicaciones, inteligencia en los negocios, administración de información, almacenamiento de información, administración de redes y mercados de sistemas federales. SterlingSoftware contribuyó con su experiencia en el modelado de componentes y tipos. Se enfocó en tipos de modelos y especificaciones, en el modelado de negocios y en la definición del UML a estándares. **Texas Instruments Software**, socio de UML, fue adquirido por Sterling Software durante la fase de definición del UML 1.1.
  www.sterling.com

- **Unisys** es una empresa que existe desde hace 112 años. Está dedicada a dar servicios de software y mantenimiento. Unisys tiene un fuerte interés en los meta-meta modelos y en sus relaciones con el UML, incluyendo la formalización de relaciones y restricciones a nivel meta y nivel meta-meta consistentemente. Con la propuesta del UML se enfocó particularmente a la integración del UML con el Meta-Modelo Facility de OMG y con el IDL de CORBA.
  www.unisys.com

3.4.4 *Los tres amigos*

Los tres amigos, como fueron llamados los primeros y más importantes colaboradores del UML, son Grady Booch, Jim Rumbaugh e Ivar Jacobson. A continuación se menciona la trayectoria que han tenido en el desarrollo de sistemas y una pequeña descripción de sus respectivas metodologías.

I. Grady Booch

Grady Booch es uno de los metodologistas líderes a nivel mundial en el desarrollo del software. Trabaja en Rational Software Corporation desde los inicios de esta empresa en 1980. Sus dos contribuciones principales al desarrollo del software son: un modelo iterativo del desarrollo del software y la importancia de la arquitectura del software. Desarrolló el método llamado *Booch*, que es un método líder en el análisis y diseño orientado a objetos, y los componentes *booch*, que son componentes reusables que ayudaron a popularizar precisamente el reuso del software.

Grady también fue uno de los desarrolladores originales de varios productos Rational, incluyendo el *Rational Environment*, que es un ambiente de ingeniería de software, y *Rational Rose*, que es una herramienta de desarrollo visual de la misma empresa.

Ha sido el consultor de numerosos proyectos a nivel mundial, actuando como mentor de arquitectura, ayudando a las organizaciones a desarrollar sistemas de misión crítica y a otros desarrolladores a personalizar sus propios métodos de desarrollo de software. Algunos de estos clientes son el gobierno de Estados Unidos, Alcatel, Andersen Consulting, AT&T, Banker's Trust, Boeing, IBM, MCI, Microsoft, Oracle y Siemens.

También es un prolífico escritor de libros y artículos relacionados con la tecnología de objetos. Ha escrito siete libros, inlcuyendo su último libro *The UML user Guide*, así *como Object-Oriented Analysis and Design*, *Object Solutions: Managing the Object-Oriented Project*, *Best of Booch: Designing Strategies for Object Technology*, *Software Engineering with Ada* y *Software Components with Ada*.

Es miembro de la Asociación Americana para el Avance de la Ciencia (Association for the Advancement of Science), la Asociación para la Maquinaria de Cómputo (Association for Computing Machinery), el Instituto de Ingenieros Eléctricos y Electrónicos (Institute of Electrical and Electronic Engineers) y Profesionales de Cómputo para la Responsibilidad Social (Computer Professionals for Social Responsability).

Grady se graduó como Licenciado en Ciencias de la Computación en 1977 en la Academia Aérea de Estados Unidos y tiene una maestría en Ingeniería en Computación de la Universidad de California en Santa Bárbara en 1979.

Metodología (OOADA)

El método de Booch Análisis y Diseño Orientado a Objetos con Aplicaciones OOADA (Object Oriented Analysis and Design with Aplications), distingue entre la estructura física y la lógica de un sistema y describe para ambos las semánticas: estática y dinámica. El método no es secuencial, el desarrollo del diseño es evolutivo, incremental e iterativo.

El OOADA se divide en un micro y en un macro proceso. El micro proceso representa básicamente las actividades diarias del desarrrollador y consiste en cuatro pasos principales no secuenciales:

- Identificar clases y objetos en cierto nivel de abstracción
- Identificar las semánticas de los objetos y clases
- Identificar las relaciones entre clases y objetos

El macro proceso se usa para controlar al micro proceso, direcciona actividades para todo el equipo de desarrollo en la escala de semanas o meses a la vez; en este macro proceso se realizan cinco actividades:

- Conceptualización. Se establecen los requerimientos
- Análisis. Se desarrolla el modelo del comportamiento deseado.

- Diseño. Se crea la arquitectura.
- Evolución. Se desarrolla la implementación.
- Mantenimiento. Se maneja la evolución de las entregas de las versiones futuras.

Conceptos

Objeto	Un objeto tiene estado, comportamiento e identidad.
Clase	Conjunto de objetos que comparten una estructura común y comportamiento.
Clase abstracta	Una clase sin instancias.
Clase base	La clase más generalizada en una estructura.
Categoría de clase	Una colección de clases (agrupamiento de clases).
Utilería de clase	Una colección de subprogramas, procedimientos o funciones que sirven como operaciones generales en uno o más objetos y que pueden pertenecer a la misma clase o a una diferente.
Metaclase	La clase de una clase, la clase es tratada como un objeto, por lo tanto como instancia de otra clase.
Nota	Descripción textual de hipótesis y decisiones. Una nota puede aplicarse a cualquier elemento del diagrama.
Amigo	Una clase u operación cuya implementación puede hacer referencia a elementos privados de otra clase.
Campo	Un repositorio para parte del estado de un objeto.
Objeto activo/pasivo	Objeto activo es el que se controla a sí mismo y es autónomo. El objeto pasivo es controlado por otro objeto, él mismo no puede tomar sus propias decisiones de control.
Persistencia	La propiedad de un objeto a través del que su existencia trasciende en tiempo y/o espacio. Un objeto puede tener persistencia estática o dinámica.
Estado	Propiedades estáticas y valores actuales de un objeto.
Historia	Indicador del estado más recientemente visitado.
Transición	Un cambio de estado.
Evento	Una ocurrencia que puede dar lugar a una transición.
Acción	Algo que se lleva a cabo cuando ocurre una transición.
Método	Una operación sobre un objeto, definida como parte de la declaración de una clase.
Operación	Una acción que un objeto lleva a cabo sobre otro objeto para obtener cierta reacción. Todos los métodos son operaciones, pero una operación puede ser un método o un subprograma independiente.
Mensaje	Una operación que un objeto lleva a cabo sobre otro.

Protocolo	Una especificación de como puede actuar o reactuar un objeto.
Modulo	Una unidad de código que sirve como bloque de construcción para la estructura física de un sistema. Un módulo consiste de una parte de interfaz y otra de implementación.
Subsistema	Una colección de módulos.
Proceso	La activación de un sólo hilo de control.
Procesador	Pieza de hardware capaz de ejecutar programas.
Dispositivo	Pieza de hardware que no es capaz de ejecutar programas.
Programa principal	Raíz de la que se activa un programa.
Especificación	Archivo que contiene la declaración de entidades.
Cuerpo	Archivo que contiene la definición de entidades.

Relaciones entre Objetos

Ligas

Un objeto colabora con otros objetos a través de sus ligas a estos objetos. Una liga denota la asociación especifica a través de un objeto (el cliente) que recibe servicio de otro objeto (el proveedor), o a través de un objeto que puede navegar en otro. El mensaje se pasa a través de una liga.

Un objeto puede ejecutar tres papeles cuando está unido a una liga:

- Actor: un objeto puede operar sobre otros objetos pero no se puede operar en él.
- Servidor: un objeto que sólo puede operar sobre otros objetos.
- Agente: un objeto que puede tanto operar sobre otros objetos como operar sobre él. Un agente usualmente trabaja para un actor u otro agente.

Cada vez que el objeto envíe un mensaje a otro objeto a través de una liga, se dice que los dos objetos están sincronizados. Hay tres formas de sincronización cuando un objeto activo tiene una liga a un objeto pasivo:

- Secuencial: las semánticas de un objeto pasivo solo pueden garantizarse si hay un solo objeto activo a la vez.
- Protegido: las semánticas de un objeto pasivo se garantizan en un sistema con múltiples hilos de control, pero las exclusiones mutuas deben realizarse con la colaboración de clientes activos.
- Sincronizado: las semánticas de objetos pasivos se garantizan en un sistema con múltiples hilos de control, y el proveedor garantiza exclusión mutua.

Durante el diseño también es importante que algunos objetos sean visibles a otros. Hay cuatro formas en las que un objeto puede ser visible a otro:

- De forma global.

- Que el objeto proveedor sea un parámetro a alguna operación del cliente.
- Que el objeto proveedor sea una parte del objeto cliente.
- Que el objeto proveedor sea un objeto localmente declarado en alguna operación del cliente.

Agregación

La agregación denota una jerarquía "todo-parte" con la habilidad de navegar del todo a sus partes.

Relaciones entre clases

Asociaciones

Una asociación sólo denota una dependencia semántica. La dirección de esta dependencia, y la forma exacta en la que una clase se relaciona a otra son implícitas.

Relaciones de Herencia

La relación de herencia es la relación entre clases en la cual una clase comparte su información y comportamiento definidos con otra Clase (en caso de herencia simple) o con más clases (herencia múltiple). La clase que hereda se llama Subclase y la clase de la cual se hereda se llama Superclase. La Superclase aumenta o restringe la estructura y comportamiento de la Subclase.

Relaciones de Agregación

Las relaciones de agregación entre clases son parecidas a las de herencia. La diferencia entre agregación y herencia es que una relación que puede definirse como "es-un" implica herencia, de lo contrario es una agregación.

Uso de relaciones

Un uso de una relación puede ser un refinamiento de una asociación en la que indicamos qué clase es el cliente y qué clase es el proveedor de ciertos servicios.

Relaciones de instancias

Las relaciones de instancias entre clases se proporcionan para construir clases contenedoras, que quiere decir que las clases cuyas instancias (homogéneas o heterogéneas) son colecciones de otros objetos. Se definen con frecuencia como clases parametizadas que tienen que ser instanciadas (parámetros llenados en) antes de que los objetos puedan ser creados.

Metaclase

Una metaclase es una clase de clase.

Interfaces de clases

En OOADA, una clase consiste de una interfaz(la vista externa) y una implementación (la vista interna). Una clase puede tener tres posibles interfaces:

- Publica. Una declaración que es accesible a todos sus clientes.
- Protegida. Una declaración que es accesible solo a la clase misma, sus subclases y sus amigos.
- Privada. Una declaración que es accesible solo a la clase misma y sus amigos.

Operaciones y Comunicación

Las operaciones pueden ser de cinco tipos:

- Modificadora: una operación que cambia el estado de un objeto

- Selectiva: una operación que tiene acceso al estado de un objeto, pero no cambia su estado
- Repetitiva: una operación que tiene acceso a todas las partes de un objeto en un orden bien definido
- Constructoras: una operación que crea un objeto y/o inicia su estado
- Destructoras: una operación que libera el estado de un objeto y/o destruye al objeto mismo.

El mensaje que pasa entre objetos puede tomar las siguientes formas de sincronización, de las cuales la primera se hará para diagramas secuenciales y las otras para sistemas concurrentes:

- Simple: en diagramas secuenciales el pasar mensajes se hace de forma simple.
- Sincronizado: una operación comienza solo cuando el emisor ha iniciado la acción y el receptor está listo a aceptar el mensaje. Emisor y receptor se esperan hasta que ambos están listos para proceder.
- *Balking*: parecido al sincronizado, pero el emisor abandonará la operación si el receptor no está listo.
- *Time-out*: parecido al sincronizado, pero el emisor solo esperara una cantidad especifica de tiempo para que el receptor este listo.
- Asincrono: una operación comienza sin importar si el receptor esta esperando el mensaje o no.

Técnicas

OOADA proporciona técnicas para las cuatro perspectivas que describe; las vistas lógicas y físicas y las semánticas estáticas y dinámicas.

Las técnicas para la vista estática lógica son:
1. Diagramas de Objeto, que muestran los objetos existentes y las relaciones entre ellos, incluyendo aspectos de visibilidad y sincronización.
2. Diagramas de clase para mostrar las clases existentes y las relaciones entre ellas incluyendo aspectos de cardinalidad, utilerías de clases, concurrencia, persistencia y visibilidad.

Las técnicas para la vista dinámica lógica son:
1. Diagramas de transición de estados para mostrar los Estados de un Objeto, los eventos que causan transiciones y las acciones resultantes de las transiciones.
2. Diagramas de interacción para describir cómo se ejecutan los escenarios en el mismo contexto como un Diagrama de Objeto. La diferencia es que el diagrama de Interacción muestra los aspectos dinámicos, mientras que el diagrama de Objetos muestra los aspectos estáticos.

La técnica para la vista estática física es:
1. Diagramas de módulos son para diseñar el empaquetamiento físico de clases y objetos en módulos.

Las técnicas para la vista dinámica física son:
1. Diagramas de Proceso son para la locación de procesos a procesadores. Los procesos y dispositivos son la plataforma de ejecución del sistema.

Procesos de análisis y diseño

Macroproceso

El macroproceso sirve como un marco de trabajo que controla el proceso micro. Prescribe algunos productos y actividades de tal forma que el equipo de desarrollo puede evaluar riesgos y hacer correcciones tempranas en el proceso micro, para poner énfasis en las actividades de análisis y diseño. El proceso macro representa las actividades del equipo de desarrollo entero en la escala de meses o semanas a la vez.

Las actividades que se realizan son:

Establecer requerimientos **(Conceptualización)**. La conceptualización trata de establecer los requerimientos para el sistema. Se establecen las ideas se establece para alguna aplicación y se validan hipótesis.

La conceptualización es un proceso muy creativo y no debe por lo tanto desempeñarse con reglas de desarrollo rígidas.

Los productos primarios de esta actividad son los prototipos.

Desarrollo del modelo del comportamiento deseado **(Análisis)**. El propósito del análisis es modelar el mundo identificando las clases y objetos que forman el vocabulario del dominio del problema. El comportamiento del sistema se enfatiza con esta fase, al enfocarse en el comportamiento, se identifican los puntos de función de los sistemas para denotar las salidas observables y comportamientos que pueden probarse del sistema.

El análisis consiste del análisis del dominio y la planeación del escenario. En el análisis se identifican las clases y los objetos que son comunes a un dominio de problema particular. La planeación del escenario es la actividad central en el análisis. Todas las actividades se documentan.

Crear una arquitectura **(Diseño)**. El diseño tiene que dirigir una arquitectura para realizar la implementación evolutiva, y establecer elementos estáticos que van a usarse por elementos dispares del sistema. El diseño puede dividirse en planeación arquitectónica, diseño táctico y planeación de versión.

- En la planeación arquitectónica, el objetivo es crear previamente el ciclo de vida de un marco de trabajo de aplicación de dominio específico que pueda refinarse después.
- El diseño táctico consiste en refinar lo elaborado en la planeación arquitectónica.
- La planeación de versión organiza la evolución arquitectónica; es decir los cambios y mejoras que se hagan y lleva control de las versiones que se produzcan. En esta fase se obtiene un plan de desarrollo formal, que identifica el flujo de versiones arquitectónicas, tareas de equipo y evaluación de riesgos.

Desarrollo de la implementación **(Evolución)**. El propósito de la evolución es establecer el crecimiento y cambio en la implementación a través del refinamiento, hasta que se alcance la producción del sistema. La evolución consiste de la aplicación de micro procesos y administración del cambio.

La aplicación del micro proceso empieza con el análisis de requerimientos para la próxima versión, después de la cual se dirige al diseño de una arquitectura, donde se crean clases y objetos necesarios para implementar este diseño. El producto principal es un flujo de versiones ejecutables que representan los refinamientos sucesivos de la primer versión de la arquitectura Los prototipos de comportamiento también se producen para explotar alternativas de diseño o para, en el futuro, investigar partes desconocidas de la funcionalidad del sistema.

Durante el manejo o administración del cambio se pretende reconocer el carácter incremental e iterativo del sistema orientado a objetos. El cambio indisciplinado a jerarquías y protocolos de clase, o mecanismos, deben ser posible, pero el cambio sin restricción es una amenaza para la arquitectura estratégica y el equipo de desarrollo.

Manejo de la evolución de las entregas posteriores (Mantenimiento). El mantenimiento es la actividad del manejo de la evolución de las entregas posteriores. Esta fase es principalmente la continuación de la evolución, con la diferencia de que en esta fase se pueden contemplar cambios en el sistema, por ejemplo requerimientos nuevos o eliminación de errores.

Debido a que las actividades de evolución y mantenimiento se dan de forma paralela y conjunta, las actividades tanto de evolución y mantenimiento son: la aplicación del micro proceso y manejo del cambio.

Microproceso

El micro proceso maneja el flujo de escenarios y productos arquitectónicos que resulten del macro proceso y que hagan referencia al mismo. Toma en cuanta principalmente las actividades diarias del desarrollador o del equipo de desarrollo.

El micro proceso consiste en llevar a cabo las fases tradicionales del análisis y diseño, pero de forma flexible de acuerdo a la oportunidad que se presente para mejorarlas durante el proceso y no hasta que se ha terminado el desarrollo.

Las actividades principales del micro proceso son:

Identificación de Clases y Objetos. La identificación de clases y objetos se hace encontrando clases y objetos significativos en el espacio del problema y los mecanismos que proporcionan el comportamiento requerido de objetos que trabajan juntos para alcanzar alguna función.

Esto puede hacerse con el análisis orientado a objetos, el análisis de comportamiento y/o de casos de uso.

Esta actividad da como resultado un diccionario de datos de clases y objetos candidatos y un documento describiendo el comportamiento del objeto.

Identificar la semántica de clases y objetos. El objetivo es establecer el estado y comportamiento de cada abstracción identificada en la fase previa. Tres actividades se llevan a cabo en esta fase: registro histórico, diseño de clases aisladas y obtención de patrones.

Las semánticas se representan de forma *arriba-abajo* en un diagrama histórico que tienen que ver con los puntos de función del sistema, direccionando elementos estratégicos. Las semánticas también se identifican de *abajo-a-arriba* lo que da como resultado un diseño de clase aislado. Si se obtiene el diseño de clase aislado, ésto quiere decir que es un buen diseño de clase, todavía no un diseño arquitectónico. En la tercera actividad, obtención de patrones, se descubren elementos comunes que puedan identificarse como patrones de comportamiento, lo cual contribuye al reuso de elementos del mismo sistema para otras actividades.

Identificar las relaciones entre clases y objetos. El propósito de identificar las relaciones entre clases y objetos es solidificar los límites de cada abstracción e identificar clases y objetos cooperativos. Este paso consiste en tres actividades: las especificaciones de asociaciones, la identificación de varios colaboradores y el refinamiento de asociaciones.

La identificación de asociaciones es principalmente una actividad de análisis y diseño tempranos. El producto resultante de esta actividad es un diagrama de clase. La identificación de colaboraciones es principalmente una actividad de diseño y clasificación. Las colaboraciones se documentan en diagramas de objetos y módulos. Los diagramas de clase se refinan para mostrar

decisiones tácticas acerca de la herencia, agregación, instanciación y uso. La tercera actividad, el refinamiento de asociaciones, se lleva a cabo en ambas fases la de análisis y diseño. El resultado es una descripción más específica de semánticas y de relaciones.

Implementación de Clases y Objetos. Durante el análisis, el propósito de implementar clases y objetos es refinar las abstracciones existentes y descubrir nuevas clases y objetos en el siguiente nivel de abstracción, que puede iterarse en el proceso micro.

Durante el diseño, el propósito es hacer un refinamiento de todas las actividades que se llevan acabo en el macroproceso, pero con una actividad extra que es la selección de las estructuras y algoritmos que proporcionan las semánticas de abstracciones identificadas con anterioridad.

Las primeras tres fases de microproceso se enfocan en la vista externa de abstracciones; este paso final se enfoca en su vista interna, en los elementos representativos de cada abstracción y en el mapeo de estas representaciones al modelo físico. Para finalmente hacer entrega del producto terminado.

II. Jim Rumbaugh

Jim Rumbaugh es uno de los líderes de metodologías de desarrollo de software. Jim ha trabajado con la metodología orientada a objetos y las herramientas relacionadas durante más de 30 años. Es el jefe de desarrollo de la Técnica de Modelado de Objetos OMT (Object Modeling Technique), un método de análisis y diseño orientado a objetos. Antes de unirse a Rational Software Corporation en 1994, trabajó durante más de 25 años en el Centro de Desarrollo e Investigación de General Electric en Schenectady, Nuev York. Allí desarrollo el lenguaje de programación orientado a objetos DSM, el modelo de árbol de estado de control, la notación de modelado de objetos OMT y el editor gráfico Object Modeling.

Jim también fue uno de los inventores de la arquitectura de cómputo de flujo de datos. Ha trabajado en otras áreas incluyendo semánticas de cómputo, herramientas para la productividad de la programación y aplicaciones usando algoritmos y estructuras de datos complejos.

Es también un escritor y lector prolífico del tema de tecnología de objetos. Encabezó como autor el libro *Object-Oriented Modeling and Design* y *OMT Insights: Perspectives on Modeling from the Journalof Object-Oriented Programming*. Su último libro es *The UML Reference Manual*.

Tiene una licenciatura en Física del MIT, una maestría en Astronomía en Caltech y un Doctorado en Ciencias de la Computación de MIT.

Metodología OMT

El OMT consiste de tres puntos de vista diferentes pero relacionados entre sí: el modelo de objetos, el modelo dinámico y el modelo funcional.

El modelo de objetos es quizá el más importante porque es el modelo inicial, es decir un modelo de análisis. Ayuda a entender el problema. La entrada para esta fase es la especificación de requerimientos, o la definición del problema. La salida es un modelo formal que describe los objetos y sus relaciones.

Como se mencionó anteriormente el OMT consta de tres modelos:

- El modelo de objetos representa los aspectos estáticos y estructurales, "datos" del sistema

- El modelo dinámico representa los aspectos temporales, de comportamiento "control", del sistema
- El modelo funcional representa los aspectos transformacionales "funcional", del sistema.

Estos tres tipos de modelos se llevan a cabo por separado describiendo un aspecto del sistema diferente, sin embargo cada uno de ellos contienen referencias a los otros modelos.

Modelo de objetos

El modelo de objetos proporciona el marco de trabajo esencial de datos en el que los modelos dinámico y funcional pueden basarse, define la estructura de datos en la que los otros modelos operan. Al final los tres modelos se juntan en la implementación, que contiene datos (modelo de objetos), secuencias (modelo dinámico) y operaciones (modelo funcional).

Modelo dinámico

El modelo dinámico captura los aspectos del sistema que tienen que ver con el tiempo y los cambios. Define la estructura de control, es decir los aspectos de un sistema que describen las secuencias de operaciones que ocurren en respuesta a estímulos externos, sin consideración de lo que hacen las operaciones (modelo funcional), en lo que operan (modelo de objetos) o como se implementan. Los conceptos de modelado dinámico principales son: eventos (estímulos externos) y estados (valores y ligas a un objeto). El patrón de eventos, estados y transición de estados para una clase dada, pueden representarse como un diagrama de estados. El modelo dinámico consiste de diagramas de estado múltiples, uno por cada clase con comportamiento dinámico.

Modelo funcional

El modelo funcional muestra las funciones de los valores, sin indicar cómo (implementación), cuándo (modelo dinámico) o porqué se cambian los valores. Las relaciones entre los valores en función se muestran en el diagrama de flujo de datos. Consiste de:

- procesos
- transformación de datos
- actores
- procuradores y consumidores de valores
- almacenamiento de datos
- creadores de tardanza entre creación y uso de datos
- flujos de datos
- relaciones entre procesos, actores y almacenamiento de datos

Relaciones entre los tres modelos

Modelo de objetos y dinámico

El modelo dinámico especifica secuencias de cambios de los objetos representados en el modelo dinámico. Los estados del modelo dinámico pueden estar relacionados a clases de atributos

y ligas de un objeto, los eventos pueden representarse como operaciones en el modelo de objetos. La generalización, agregación y herencia se aplican a estos modelos.

Generalización

Cada subestado de un objeto restringe los valores que un objeto puede tener, esta es una forma especial de generalización, llamada generalización por restricción. La generalización en clases y en estados particionan el conjunto de valores de objetos posibles. Cuando existen diferencias inherentes entre objetos, se modelan como clases diferentes. Cuando las diferencias son temporales, se modelan como estados diferentes de la misma clase.

Agregación

La agregación se da cuando existe un estado compuesto, es decir cuando un objeto puede tener más de un subestado. La agregación existe en dos niveles, agregación de objetos (el estado compuesto se parece a los estados individuales de las partes) y la agregación en objetos (grupos de atributos y ligas detienen el estado compuesto).

Herencia

La herencia es la representación de jerarquías de eventos y de estados. Los eventos heredan atributos (valores de datos) de sus padre(s) y los estados heredan las transiciones posibles. El modelo dinámico de una clase es también heredado por sus subclases.

Modelo de objeto y funcional

Los cuatro componentes del modelo funcional pueden relacionarse con el modelo de objetos.

Procesos

El modelo funcional muestra los procesos que tiene que implementarse en los métodos de los objetos. Los proceso en el modelo funcional muestran los objetos que están relacionados por función. Con frecuencia una entrada o salida es el cliente de un proceso. Las otras entradas son parámetros, es decir proveedores. Estas relaciones cliente-proveedor establecen dependencias de implementación entre clases relacionadas.

Actores

Los actores son los objetos en el modelo de objeto.

Almacenamiento de datos

Son los atributos de objetos representados en el modelo de objetos.

Flujos de datos

Los flujos de datos se representan en el modelo de objeto, consisten en el flujo de datos de los actores a otros actores u objetos por medio de operaciones.

Modelo Dinámico y funcional

La relacione entre estos dos modelos es que el modelo dinámico establece cuándo deben llevarse acabo las operaciones, y el modelo funcional establece cómo se llevan a cabo las

operaciones y cuáles argumentos se necesitan. El modelo funcional solamente especifica la forma en la que actúan los objetos mientras que el modelo dinámico tiene que especificar cuando actúan.

Restricciones futuras

Los objetos tienen una identidad por ejemplo, se distinguen por la existencia inherente y no por propiedades descriptivas que puedan tener. Además de los atributos y operaciones comunes, los objetos en una clase también comparten la semántica. Los objetos pertenecen a cada uno de cierta forma.

Un atributo tiene un valor para cada objeto. La misma operación puede aplicarse para mantener clases diferentes. Cada una de estas clases tiene un método para manejar la operación. Es importante que los métodos para una operación tengan el mismo número y tipos de argumentos y el mismo tipo de resultado de valor.

Las operaciones que no tienen efectos alternos (sólo computan un valor funcional), sin argumentos excepto los objetos, pueden ser considerados como atributos derivados. Cada asociación en el diagrama de clase corresponde a un conjunto de ligas en el diagrama de instancia, así como cada clase corresponde a un conjunto de objetos.

La asociación ternaria es una unidad atómica y no puede subdividirse en asociaciones binarias sin perder información (la asociación ternaria es una asociación entre tres clases).

Un atributo de liga es una propiedad de las ligas a la asociación.

Un rol es un fin de una asociación. Una asociación binaria tiene dos roles, cada uno de los cuales puede tener un nombre de rol diferente. Un nombre de rol es un nombre que únicamente identifica un fin de una asociación.

La generalización es la relación entre una clase y una o más versiones refinadas de esa. La clase refinada se llama la superclase y cada versión refinada se llama subclase.

Procedimiento de modelado

Se llevan a cabo los siguientes pasos para construir un modelo de objeto:

- Identificar clases
- Preparar un diccionario de datos
- Identificar asociaciones (incluyendo agregaciones) entre objetos
- Identificar atributos de objetos y ligas
- Organizar y simplificar clases usando herencia
- Verificar que existan rutas de acceso para likely queries
- Iterar y refinar el modelo
- Agrupar clases en módulos.

Identificar clases

Las clases se pueden identificarse con le siguiente procedimiento:

1. Listar todas las clases candidatas encontradas en la descripción escrita del problema. Esto puede hacerse escribiendo todos los sustantivos de la descripción del problema.
2. Después descartar todas las clases innecesarias e incorrectas.

Preparar un diccionario de datos

Escribir en algunas lineas cada clase, su ámbito, restricciones, atributos, asociaciones y operaciones.

Identificar asociaciones

Cualquier dependencia entre dos o más clases es una asociación. Una referencia de una clase a otra es una asociación. Las asociaciones corresponden con frecuencia a verbos o frases verbales. Estas incluyen locación física, acciones dirigidas, comunicación, propiedad o satisfacción de alguna condición. Algunas recomendaciones son: extraer todos los candidatos de una definición del problema y obtenerlos en papel, no gastar mucho tiempo tratando de distinguir entre asociación y agregación. La agregación es solo una asociación con condiciones extra. Es importante descartar asociaciones innecesarias e incorrectas.

Identificar atributos de objetos y ligas

Los atributos son propiedades de objetos individuales. Los atributos no deben ser objetos; con frecuencia corresponden a sustantivos seguidos por frases posesivas, como "el color del carro" o adjetivos. Los atributos no son prestos a ser descritos en la definición del problema, entonces se deben dibujar en el conocimiento del dominio de la aplicación y del mundo real para encontrarlas. Primero se obtienen los atributos más importantes, los detalles se pueden agregar después. Algunas recomendaciones son: evitar atributos que solo sean para la implementación, asegurarse de darle a cada atributo un nombre significativo, omitir los atributos, identificar los atributos de ligas, después de identificar a los atributos candidatos, eliminar atributos innecesarios e incorrectos.

Organizar y simplificar las clases usando herencia

La herencia puede agregarse en dos formas diferentes:

1. Abajo-arriba: esto quiere decir generalizando aspectos comunes de las clases existentes en una superclase. Se puede descubrir la herencia de abajo-arriba buscando clases con atributos, asociaciones u operaciones similares. Para cada generalización, se debe definir una superclase para compartir elementos comunes. Puede ser necesario refinar cuidadosamente algunos atributos o hasta clases para ajustarse en propiedad (esto es aceptable pero no hay que forzarlo si no encaja, porque quizá no sea la generalización correcta) Algunas generalizaciones se van a sugerir ellas mismas basadas en taxonomías existentes del mundo real; es recomendable usar conceptos existentes siempre que sea posible.

2. Arriba-abajo; por el refinamiento de clases existentes en subclases especializadas. Las especializaciones arriba-abajo con frecuencia se pueden visualizar desde el dominio de la aplicación. Se recomienda buscar frases de sustantivos compuestas de varios adjetivos en el nombre de la clase, evitar refinamiento excesivo, si las especializaciones propuestas son incompatibles con una clase existente, la clase existente puede estar formulada de forma inapropiada.

Herencia múltiple

La herencia múltiple puede usarse para incrementar los elementos compartidos, pero solo si es necesario, ya que incrementa la complejidad conceptual y de implementación. Los atributos y asociaciones deben de asignarse a la clase más general para la que sea apropiado.

Iterar y refinar el modelo

Un modelo de objeto es raramente correcto después de un solo paso. Si se encuentra alguna deficiencia, regresar a la etapa anterior para corregirlo. Algunos refinamientos solo pueden realizarse después de que los modelos dinámicos y funcionales están completos.

Agrupamiento de clases en módulos

Los diagramas pueden dividirse en hojas de tamaño uniforme para la conveniencia del dibujo, impresión y visión. Las clases que tengan relación entre ellas deben agruparse juntas. Un módulo es un conjunto de clases (una o más hojas) que capturan algunos subconjuntos lógicos del modelo entero. Cada asociación debe mostrarse generalmente en una sola hoja, pero algunas clases

deben mostrarse más de una vez para conectar hojas diferentes. Debe intentar minimizar el numero de estas llamadas Clases puente.

Ivar Jacobson

Ivar Jacobson es uno de los lideres en metodologías de desarrollo de software en el mundo. Es vicepresidente de ingeniería de negocios en Rational, es responsable de ayudar a Rational a definir estrategias corporativas y de productos en el área de negocios e ingeniería de sistemas. También trabaja con Grady y Jim refinando y mejorando el proceso UML. Antes de unisre a Rational, Ivar trabajaba en Objectory AB en Suiza, la compañía que él fundó, la cual su fusionó con Rational Software Corporation en 1995.

Las contribuciones de Ivar se basan en cinco áreas principales. Primero, el enfoque de arquitectura céntrica al desarrollo de software. Segundo, su énfasis en el desarrollo de casos de uso, que en 1987 fue parte del proceso de desarrollo de *Objectory*, herramienta de su creación. En 1990 expandió el *Objectory* para entender mejor el contexto de negocios y para la mejor captura de requerimientos. Después de varias generaciones, y después de unir fuerzas con Rational, evolucionó en 1998 en el Proceso Unificado de Rational (Rational Unified Process).

Ivar es el autor principal de tres libros que han tenido mucho éxito*: Object-Oriented Software Engineering--A Use Case Driven Approach, The object Advantage--Business Process Reengineering with Object Technology* y *Software Reuse: Architecture,Process and Organization for Business Success.* Su más reciente libro es *The Unified Software Development Process.* Ivar también ha sido co-autor de varios documentos de la tecnología de objetos.

Metodología (OOSE)

OOSE tiene un enfoque llamado orientado a casos de uso. En este enfoque un modelo de casos de uso sirve como un modelo central del que todos los otros modelos se derivan. Un modelo de caso de uso describe la funcionalidad completa del sistema identificando como todo lo que esta fuera del sistema interactua con el sistema.

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 Los objetos se encuentran, se describen interacciones organizadas y de objetos. Las operaciones de objetos y la vista interna de los objetos se describe también durante el análisis. La construcción contiene el diseño y la implementaron en el código fuente. Es importante que los objetos en la fase de análisis puedan encontrase de nuevo durante la construcción. Los componentes son muy importantes en esta metodología, un componente es una pieza ya definida de código fuente que puede usarse para la implementaron de objeto. Al probar el sistema es verificado, lo que quiere decir que lo correcto del sistema se revisa de acuerdo a sus especificaciones.

Conceptos

Actor	Un actor define un rol que el usuario puede jugar en el intercambio de información con el sistema.

Actor primario	Un actor que usa al sistema directamente, llevando a cabo una o algunas de las tareas principales.
Actor secundario	Un actor que supervisa o mantiene al sistema.
Actor abstracto	Un actor que describe un rol que debe llevarse a cabo contra el sistema. Los actores diferentes deben de heredar de un actor abstracto si tienen roles similares.
Rol	El rol se define por las operaciones de un objeto. Describe el propósito en el que un objeto participa con otro.
Usuario	Es la persona que actualmente usa el sistema. Un usuario es una instancia de un clase Actor.
Caso de uso	Es un curso de eventos completo especificando todas las acciones entre el usuario y el sistema.
Caso de uso abstracto	Una descripción de las cosas comunes en otros casos de uso. Un caso de uso abstracto no será instanciado por si mismo.
Caso de uso concreto	En caso de uso que en realidad será instanciado.
Objeto	Esta caracterizado por un numero de operaciones y un estado que recuerda el efecto de estas operaciones.
Objeto entidad	Un objeto cuya información se almacena por un largo periodo de tiempo, incluso cuando se completa un caso de uso.
Objeto interfaz	Un objeto que contiene funcionalidad de los casos de uso que interactuan directamente con el ambiente.
Objeto control	Un objeto que modela funcionalidad que no esta en ningún otro objeto.
Objeto de interfaz central	Un objeto interfaz que contiene otros objetos interfaz.
Atributo	Un atributo contiene información y su tipo.
Clase	Un grupo de objetos que tienen comportamiento y estructuras de información similares.
Clase abstracta	Una clase que se desarrolla con el propósito principal de ser heredada por otras clases.
Clase concreta	Una clase que se desarrolla con el propósito principal de crear instancias de ella.
Estimulo	Un evento que se envía de un objeto a otro y que inicia una operación.
Mensaje	Estimulo intra-proceso: una llamada normal dentro de un proceso.
Señal	Estimulo inter-proceso se envía entre dos procesos.
Operación	Una actividad dentro de un bloque que puede llevar a un cambio de estado del objeto correspondiente.
Subsistema	Un grupo definido de objetos para estructurar el sistema.

Paquete de servicio	El nivel mas bajo de un subsistema que se ve como una unidad de cambio anatómico.
Bloque objeto de diseño	Una abstracción de la implementaron actual. Los bloques se implementan después en el código fuente.
Estado	La unión de todos los valores describiendo la situación presente.
Transición	El cambio de estado.
Modulo de objeto	La implementaron en código fuente para un bloque. Un bloque puede implementarse en más de un modulo de objeto.
Modulo de objeto publico	Un modulo de objeto que es accesible desde fuera de un bloque.
Modulo de objeto privado	Un modelo de objeto que no es accesible desde fuera de un bloque.

Relaciones entre objetos y clases

Una distinción se hace entre asociaciones de clase y asociaciones de instancia denotando relaciones entre objeto. Las asociaciones de clase se muestran con flechas punteadas, mientras las asociaciones de instancias se representan con flechas completas. Todas las relaciones son siempre unidireccionales.

Asociaciones de clase

Asociación de herencia

Las operaciones y la estructura de la información de una superclase son heredadas por una subclase. Una subclase puede heredar de mas de una superclase (herencia múltiple) Asociación de extensión. Extensión quiere decir insertar una descripción de caso de uso en otra descripción de caso de uso que debe ser un curso completo por si mismo. Esta descripción es por lo tanto independiente de cualquier descripción insertada Asociaciones de instancia:

- Asociación (relación): para modelar relaciones entre objetos, OOSE menciona el rol que un objeto puede desempeñar en relación con otro objeto.
- Asociación de conocimiento: un objeto es un conocido de otro objeto si sabe que el otro objeto existe. Esta es una relación estática, que quiere decir que los objetos no pueden cambiar información entre ellos.
- Asociación consiste de algún objeto: es un tipo especial de asociación que denota que esta compuesta de objetos. También se usa el término agregación.
- Asociación de comunicación: estas asociaciones se usan para modelar las comunicaciones entre objetos. A través de las asociaciones de comunicación los objetos envían y reciben estímulos. La flecha denota la dirección de un estímulo.

Operaciones y comunicación

Las operaciones que tienen que ser llevadas fuera por un objeto entidad pueden incluir: creación y borrado de objeto entidad almacenamiento y direccionamiento de información comportamiento que debe de cambiar si el objeto entidad es cambiado.

Los objetos se comunican enviando estímulos. un estimulo causa una operación para llevarse a cabo en el objeto que recibe. Un estimulo puede ser un mensaje, denotando comunicación interproceso sincronia o asíncrona.

Técnicas

1. **Requerimientos del modelo.** Este modelo delimita el sistema y define su d}funcionalidad. consiste de tres partes:

Modelo de caso de uso, que describe los actores y los casos de uso. Los actores definen roles que lo usuarios pueden ejecutar intercambiando información con el sistema y los casos de uso representan la funcionalidad dentro del sistema. Es un curso completo de eventos especificando todas las acciones entre el usuario y el sistema

Modelo de objeto del dominio del problema, para desarrollas la vista lógica del sistema que puede usarse para hacer una lista de sustantivos que puede ser la base para especificar casos de uso.

Descripciones de interfaz. Es importante que los usuarios estén involucrados en hacer descripciones de interfaz detallada. Por lo tanto estas descripciones deben hacerse en una etapa temprana. La interfaz tiene que capturar la vista lógica del usuario del sistema, ya que los intereses principales es la consistencia de la vista lógica y el comportamiento actual del sistema. Puede manejar cambos las interfaces del usuario y de otros sistemas.

2. El modelo de análisis. Consiste en la estructura el sistema (modelo de requerimientos), modela tres tipos de objetos: objetos de interfaz, objetos entidad y objetos de control. El comportamiento que se modela en el caso de uso se expande entre los objetos en el modelo de análisis. El modelo de análisis proporciona una fundación para el diseño.

3. El modelo de diseño. Consiste en refinar el modelo de análisis y lo adaptara al ambiente de implementaron. Las interfaces de objetos y semánticas de operaciones se definen y se pueden tomar decisiones acerca de Sistemas de Manejo de Bases de Datos y lenguajes de programación. Los bloques se introducen para tipos de objetos para ocultar la implementaron actual. El modelo de diseño consiste de diagramas de interacción y gráficas de transición de estado.

Un diagrama de interacción esta dibujado por cada caso de uso concreto. Describe el caso de uso en términos de objetos comunicantes. Esta comunicación se modela como bloques enviándose estímulos entre ellos, los diagramas de interacción soportan casos de uso con extensiones.

Las gráficas de transición de estado se usan para describir el comportamiento del objeto en términos en los que el estimulo puede recibirse y lo que el estimulo pueda causar.

4. El modelo de implementaron. Consiste del código fuente de los objetos especificados en el modelo de diseño. Es deseable que el bloque pueda ser fácilmente traducido en el modulo de objeto actual.

5. El modelo de prueba. Consiste en probar el modelo de implementaron. Especificación de prueba, que es la prueba de la clase cuando una prueba se ve como un objeto, y el resultado de la prueba, la ejecución de una instancia de la clase prueba, son los conceptos principales. Primero el nivel mas bajo del sistema se prueba, después los casos de uso pueden probarse y finalmente se puede ejecutar una prueba a todo el sistema. El modelo de requerimientos puede servir como una verificación para el modelo de prueba.

Procesos de Análisis y Diseño

Análisis

En el análisis, se producen dos modelos: el modelo de requerimientos y el modelo de análisis. El modelo de requerimientos describe todos los requerimientos funcionales desde la perspectiva del usuario y la forma en la que el sistema va a usarse por los usuarios finales. El modelo de requerimientos se puede estructurar en el modelo de análisis. Se estructura desde una perspectiva lógica en un sistema robusto y fácil de mantener. OOSE no describe pasos de como estos modelos pueden ser obtenidos.

Construcción

La construcción esta dividida en dos pasos, diseño e implementación.

Diseño

El diseño consiste en tres fases. Primero, el ambiente de implementaron tiene que identificarse. Las consecuencias que el ambiente de implementación tendrá en el diseño se estudian. La identificación e investigación del ambiente de implementaron resulta de decisiones de implementaron estratégicas. Segundo, un primer enfoque a un modelo de diseño se desarrolla en el que los objetos de análisis se traducen es objetos de diseño encajando en el ambiente de implementaron. Finalmente, la interacción de objetos en cada caso de uso se describe, lo que da como resultado interfaces de objetos.

Implementaron

En este paso cada objeto se implementa en el lenguaje de programación.

Pruebas

La prueba empieza con la planeación de pruebas en la que se examina si los programas de pruebas existentes y datos pueden usarse, si pueden ser modificados o si se debe hacer un nuevo desarrollo. También, se toman decisiones de subpruebas. Después de la planeación de pruebas, se identifican las pruebas. Se hace una descripción detallada de que debe probarse y cuantas fuentes se necesitan aproximadamente. EL tercer paso es especificar las pruebas. Se obtiene una descripción general de la prueba y su propósito. Además de eso, una descripción de procedimientos detallada de cada paso de haces en la prueba. En el cuarto paso las pruebas se llevan a cabo, cada caso de uso uno por uno. Se usa una tabla de decisión para almacenar los resultados de las pruebas de cada subprueba. Un resultado de prueba se llama salida y puede ser i para lo que esta bien y 0 para lo que fallo. Al agregar factores de peso a las subpruebas de acuerdo a su importancia, la prueba total puede mesurarse y puede comparase a un limite. Si la suma total de las subpruebas exceden el limite entonces se aprueba. En caso de que la prueba falles, se analiza la falla. Después del análisis, se especifican las pruebas y se llevan a cabo una vez mas hasta que se apruebe la prueba. En ese caso la prueba termina. El equipo y todo lo que se uso se almacena y documento para mantener un control y ver si se puede usar en un futuro para otros sistemas.

3.5 Definiciones

3.5.1 Documentación

Definición etimológica

Diccionario Crítico Etimológico Castellano e Hispánico. Gredos, Madrid, 1980.
Documento. Del latín *documentum*, enseñanza, ejemplo, muestra.
Documentar. Del latín *documentáre*, probar algo.
Documentación. Del latín *documentat0Fnis*, ejemplificar.

Definiciones de Diccionarios

Diccionario de la Real Academia de la Lengua Española. Vigésima primera edición. Madrid, 1992.
Documentación. Acción y efecto de documentar. Documento o conjunto de documentos preferentemente de carácter oficial, que sirven para la identificación personal o para documentar o acreditar algo.
Documentar. Justificar la verdad de una cosa con documentos.
Documento. Escrito en que constan datos fidedignos o susceptibles de ser empleados como tales para probar algo.

Otros diccionarios

Moliner, María. Diccionario del uso del español. Gredos, Madrid, 1990.
Documentación. Acción de documentar o documentarse. Conjunto de documentos referentes a algo o alguien.
Documentar. Adjuntar documentos para acreditar una afirmación, para justificar algo.
Documento. Escrito que sirve para justificar o acreditar algo.

Diccionario Enciclopédico Santillana. España, 1992.
Documentación. No hay definición
Documentar. Presentar documentos para justificar, demostrar o fundamentar algo.
Documento. Escrito u otra cosa que informa o ilustra sobre una cuestión o sirve para justificar o acreditar algo.

Diccionario del Español Usual en México. El Colegio de México, México, 1996.
Documentación. Acto de documentar o documentarse. Conjunto de los documentos relacionados con alguna cosa, principalmente de aquellos que permiten identificar o acreditar oficialmente algo o alguien.
Documentar. Reunir y presentar los documentos necesarios para probar alguna cosa, para justificarla, acreditarla o analizarla.
Documento. Escrito que hace constar algo, lo justifica o lo demuestra.

Diccionarios y glosarios especializados

Universidad de Leicester. http://wombat.doc.ic.ac.uk/foldoc

Documentación. Los varios kilogramos de papel que acompañan a un producto de software o hardware para conocer su funcionamiento.

Enciclopedia de términos de cómputo en Ineternet. http://www.techweb.com/encyclopedia/
Documentación. Descripción narrativa y descriptiva de un sistema. Incluye la documentación requerida para describir un sistema de información tanto para el usuario como para el staff técnico.

High-Tech Dictionary. http://www.currents.net/resources/dictionary/dictionary.phtml
Documentación. Instrucciones que acompañan a un programa de software que pueden ser en papel o en manuales electrónicos, archivos README y ayuda en línea.

Definición propia

Documentación. Escrito que de forma narrativa, descriptiva y con ilustraciones muestra el desarrollo de un sistema desde el nacimiento del mismo hasta su etapa final, tomando en cuenta todos los aspectos necesarios para el entendimiento total del mismo.

Sinónimos

Diccionario razonado de sinónimos y contrarios. Vecchi, España 1985.
Documento. Instrumento, Monumento, Título, Diploma, Pliego, Papel, Comunicación, Dato, Carta, Pancarta, Pergamino, Albalá, Albarán, Rollo, Copia, Minnuta, Compulsa, Duplicado.

Antónimos

Diccionario de Sinónimos y Antónimos. Culturas Internacionales, México 1991.
Documento. No hay antónimos.
Documentar. Ignorar, desconocer, desentenderse, olvidar, negar, descuidar.

Definiciones en otras lenguas

Inglés
The Oxford English Reference Dictionary. Oxford University Press, New York, 1996.

Documentation. The accumulation, classification and dissemination of information. The collection of documents relating to a process or event, the written specification and instructions accompanying a computer program.
Documentación. La acumulación, clasificación y diseminación de información. La colección de documentos relacionados a un proceso o evento, la especificación escrita e instrucciones que acompañan a un programa de computadora.

Francés
Le petit Larousse Ilustré. Larousse, Belgique 1994.

***Documentation**. Action de sélectionner, de classser, d'utiliser ou de diffuser des documents. Ensemble de documents relatifs à une question, un ouvrage. Ensemble des opérations, des méthods qui facilitent la collecte, le stockage, la circulation des documents et de l'information.*
Documentación. Acción de seleccionar, clasificar, de utilizar o de propagar los documentos. Conjunto de documentos relativos a un tema, una obra. Conjunto de operaciones, de métodos que facilitan la colecta, almacenamiento, circulación de documentos y de información.

Italiano
Zingarelli, Nicolo, Vocabolario della lingua italiana. Milano 1974.

***Documentazione**. Atto, effetto del documentare e del documentarsi. Ciò che consente di documentare qualsiasi cosa. Elaborazione automatica: insieme di tecniche che permettono la registrazione in forma abbreviata di documenti letterali nelle memorie e la loro sucessiva ricerca automatica.*
Documentación. Acto, efecto de documentar o de documentarse. Lo que permite documentar cualquier cosa. Elaboración automática: conjunto de técnicas que permiten el registro en forma abreviada de documentos literales en la memoria y su sucesiva búsqueda automática.

Alemán
Duden Deutsches Universalwörterbuch. Dudenverlag Mannheim, Deutchland 1989

***Urkunde**. Scrifstück, durch das etwas beglaubigt oder bestätigt wird; Dokument mit Rectskraft.*
Documento. Escrito para dar testimonio de algo o para confirmar algo; documento con registro.

3.5.2 Análisis

Definiciones etimológicas

Diccionario Crítico Etimológico Castellano e Hispánico. Gredos, Madrid, 1980
Análisis. Tomado del griego -;K9@N7?, disoluciNn de un conjunto en partes, derivado de -;K9GM7; 'desatar', y 7ste de 9GM7; 'soltar'.

Definiciones de Diccionarios

Diccionario de la Real Academia de la Lengua Española. Madrid, 1992
Análisis. Distinción y separación de las partes de un todo hasta llegar a conocer sus principios o elementos.
Des de le punto de vista de la informática la definición es: estudio mediante técnicas informáticas, de los límites, características y posibles soluciones de un problema al que se aplica un tratamiento por ordenador.

Otros diccionarios

María Moliner. Diccionario del uso del español. Gredos, España, 1990

Análisis. Procedimiento utilizado para conocer o razonar que consiste en descomponer el total del objeto del conocimiento en partes, o bien en aplicar a un caso en particular un conocimiento o ley general que lo comprende.

Diccionario Enciclopédico Santillana. España, 1992
Análisis. Distinción y separación de las partes de un todo para identificar sus principios o elementos. Examen de alguna cuestión, problema, obra, escrito, etc.

Diccionario del Español USUAL en México. El Colegio de México, 1996
Análisis. Estado detallado de una organización, un procedimiento, un método, una técnica, etc. con el fin de examinar cada una de sus fases y evaluar su eficiencia, particularmente, estudio de la estructura en que se ordenan los datos de una computadora (análisis de datos) y de las operaciones sucesivas que se siguen para manejarlos (Análisis de programa).

Diccionarios y glosarios especializados

Universidad de Leicester. http://wombat.doc.ic.ac.uk/foldoc
Análisis. Proceso de revisar el proceso del negocio para determinar las necesidades y requerimientos funcionales que se deben cubrir en un sistema de cómputo.

Enciclopedia de términos de cómputo en Ineternet. http://www.techweb.com/encyclopedia/
Análisis orientado a objetos. Examen de un problema modelándolo como un grupo de objetos que interactuan entre sí. Un elemento está definido por su clase, sus elementos de datos y su comportamiento.

High-Tech Dictionary. http://www.currents.net/resources/dictionary/dictionary.phtml
Análisis. El análisis de un sistema de información tiene el enfoque se diseñar y modificar los requerimientos del usuario final para cubrir sus necesidades. El análisis de sistemas contempla la investigación de un programa en cuanto a costo y requerimientos, creación de la documentación y pruebas a un prototipo en varias etapas del diseño.

Definición propia

Análisis. Etapa del desarrollo de sistemas que consiste en conocer el ámbito en el que el sistema va a desarrollarse así como las necesidades y requerimientos funcionales que deberán tomarse en cuenta para las etapa posterior de diseño.

Sinónimos

Diccionario razonado de sinónimos y contrarios. Vecchi, España 1985
Análisis. Examen, estudio, observación, comparación.

Antónimos

Diccionario de Sinónimos y Antónimos. Culturas Internacionales, México 1991
Análisis. Síntesis, sinopsis.

Definiciones en otras lenguas

Inglés
The Oxford English Reference Dictionary. Oxford University Press, New York 1996

Analysis. *A detailed examination of the elements or structure of a substance. The act or process of breaking something down into its constituent parts.*
Análisis. Un examen detallado de los elementos o estructura de una substancia. El acto o proceso de dividir algo en las partes que lo constituyen.

Francés
Le petit Larousse Ilustré. Larousse, Belgique 1994

Analyse. *Décomposition d'un corps, d'une substance en ses éléments constitutifs. Ensemble des travaux comprenant l'étude detaillé d'un problème, la conception d'une méthode permettant de le résoudre et la définition prècise du traitement correspondant sur ordinateur.*
Análisis. Descomposición de un cuerpo, de una substancia en sus elementos constitutivos. Conjunto de los trabajos que comprende el estudio detallado de un problema, la concepción de un método que permite resolver y definir de forma precisa el tratamiento correspondiente a una computadora.

Italiano
Vocabolario della lingua italiana. Zingarelli, Milano 1974

Anàlisi. *Método di studio consistente nello scomporre, materialmente o idealmente, un tutto nelle sue componenti per esaminarle una per una, traendone le debite conclusioni. Nell'applicazione dei sistemi elettronici per l'elaborazione dei dati, fase di lavoro, precedente la programmazione in cui vengono esaminate le procedure la cui esecuzione sarà affidata alla machina.*
Análisis. Método de estudio consistente en la descomposición, material o ideal, de un todo en sus componentes para examinarlos uno por uno, extrayendo las debidas conclusiones. En la aplicación de los sistemas electrónicos para el procesamiento de datos, fase del trabajo, que antecede a la programación, en la cual son examinados los procedimientos cuya ejecución realizará la máquina.

Alemán
Duden Deutsches Universalwörterbuch. Dudenverlag Mannheim, Deutchland 1989

Analyse. *Untersuchung bei der etwas zergliedert: eine wissenschaftliche, sorgfältige.*
Análisis. Examen mediante la descomposición de un todo en sus partes más pequeñas: científicamente o cuidadosamente.

3.5.3 Diseño

Definiciones etimológicas

GNmez de Silva, Gaido. Breve diccionario etimolNgico de la lengua espa>ola. El Colegio de M7xico, FCE, M7xico, 1988
Diseño. Traza o delineación de un edificio o de un aparato: derivado del italiano *disegno*, 'diseño, dibujo', de *disegnare*, 'dibujar, indicar', del latín *designare* 'indicar'.

Definiciones de Diccionarios

Diccionario de la Real Academia de la Lengua Española. Madrid, 1992
Diseño. Traza delineación de un edificio o de una figura. Descripción o bosquejo de alguna cosa, hecho por palabras.

Otros diccionarios

María Moliner. Diccionario del uso del español. Gredos, España, 1990
Diseño. Dibujo hecho solo en líneas para representar algo con poco detalle. Descripción de una cosa hecha con palabras a la ligera.

Diccionario Enciclopédico Santillana. España, 1992
Diseño. Delineación o dibujo de edificios, figuras, vestidos, etc. Esquema, resumen.

Diccionario del Español USUAL en México. El Colegio de México, 1996
Diseño. ideal original de algo que se dibuja o proyecta para después elaborarlo. Arte de usar técnicas e instrumentos para representar gráficamente determinado producto u obra.

Diccionarios y glosarios especializados

Universidad de Leicester. http://wombat.doc.ic.ac.uk/foldoc
Diseño. Enfoque que aplican algunas disciplinas, se usa para especificar como crear o hacer algo. Un diseño satisfactorio debe cumplir con las especificaciones funcionales del negocio ; conforme a las limitaciones propias del negocio. En el ciclo de vida del software el diseño está después del análisis y sigue las implementaciones del mismo.

Enciclopedia de términos de cómputo en Ineternet. http://www.techweb.com/encyclopedia/
Diseño orientado a objetos. Transformación de un modelo orientado a objetos a las especificaciones requeridas para crear un sistema. Primero se lleva a cabo el análisis orientado a objetos y después el diseño orientado a objetos, dando como resultado cada vez más detalle en los diagramas y especificaciones.

High-Tech Dictionary. http://www.currents.net/resources/dictionary/dictionary.phtml
Diseño. Forma de desarrollar programas de computadoras desde arriba hacia abajo, primero describiendo las acciones del programa en un nivel alto, y después dividiendo las instrucciones en

elementos más pequeños, los cuales se definen después con más detalle hasta que se completa todo el programa.

Definición propia

Diseño. Etapa del desarrollo de sistemas que precede al análisis, en la cual por medio de descripciones escritas y gráficas se va a representar el funcionamiento del sistema; cumpliendo con las necesidades y requerimientos funcionales considerados en la etapa previa.

Sinónimos

Diccionario razonado de sinónimos y contrarios. Vecchi, España 1985
Diseño. Esbozo, delineación, boceto, bosquejo, croquis, traza.

Antónimos

Diccionario de Sinónimos y Antónimos. Culturas Internacionales, México 1991
Diseño. No se encontró.

Definiciones en otras lenguas

Inglés
The Oxford English Reference Dictionary. Oxford University Press, New York 1996

Design. *A preliminary plan or sketch for the making or production of a building, machine, etc. A scheme of lines or shapes forming a pattern or decoration. A generak arrengement or layout of a product, a work of art, printed material, etc.*
Diseño. Un plan o bosquejo preliminar para realizar o producir un edificio, máquina, etc. Un esquema de líneas o figuras que forman un patrón o decoración, el arreglo o composición de un producto, obra de arte, material impreso, etc.

Francés
Le petit Larousse Ilustré. Larousse, Belgique 1994

Design. *Discipline visant à la creation d'objects, d'environnements, d'ouvres graphiques, etc. , à la fois fonctionels, esthétiques et conformes aux imperátifs d'une production industrielle. Ensemble des objects créés selon ces critères.*
Diseño. Disciplina que tiene que ver con la creación de objetos, ambientes, obras gráficas, etc., que son tanto funcionales, como estéticas y conforme a las necesidades de una producción industrial. Conjunto de objetos creados de acuerdo a sus criterios.

Italiano
Vocabolario della lingua italiana. Zingarelli, Milano 1974

Diségn. *Representazione con linee e segni di figure immaginate o di oggetti reali.*
Diseño. Representación con líneas y signos de figuras imaginadas o de objetos reales.

Alemán
Duden Deutsches Universalwörterbuch. Dudenverlag Mannheim, Deutchland 1989

Zeichnung. *Mit den Mittelm des Zeichenens verfertigte bildliche Darstelluung.*
Diseño. Recurso mediante el cual se representa algo para fabricarlo; rerpesentación ilustrada de algo.

3.5.4 Orientado

Definiciones etimológicas

Barcia, D. Roque. Primer diccionario General Etimológico de la Lengua Española. Seix, Barceolona
Orientado, da. Participio pasivo de orientar.
Orientado. Del italiano *orinetato*, francés *orienté*, catalán *orientat, da*.
Orientar. Del italiano *orientare*, francés *orienter*, catalán *orientar*.

Definiciones de Diccionarios

Diccionario de la Real Academia de la Lengua Española. Madrid, 1992
Orientado. La palabra orientado no está definida.
Orientar. Colocar una cosa en posición determinada respecto a los puntos cardinales. Determinar la posición o dirección de una cosa respecto a un punto cardinal.

Otros diccionarios

María Moliner. Diccionario del uso del español. Ed. Gredos, España, 1990
Orientado. Participio adjetivo.
Orientar. Poner una cosa en cierta dirección.

Diccionario Enciclopédico Santillana. España, 1992
Orientado. La palabra orientado no está definida.
Orientar. Colocar una cosa en cierta dirección. Indicar a alguien el lugar donde se encuentra o la dirección que debe seguir para ir a donde desea.

Diccionario del Español USUAL en México. El Colegio de México, 1996
Orientado. La palabra orientado no está definida.
Orientar. Colocar algo en una posición determinada con respecto a los puntos cardinales o en cierta dirección.
Diccionarios y glosarios especializados

Universidad de Leicester. http://wombat.doc.ic.ac.uk/foldoc
Diseño orientado a objetos. Método de diseño en el que se modela un sistema como una colección de objetos individuales y objetos que cooperan entre sí. Estos objetos se capturan en instancias de una clase dentro de una jerarquía de clases.

Enciclopedia de términos de cómputo en Ineternet. http://www.techweb.com/encyclopedia/
Diseño orientado a objetos. Transformación de un modelo orientado a objetos a las especificaciones requeridas para crear un sistema. Primero se lleva a cabo el análisis orientado a objetos y después el diseño orientado a objetos, dando como resultado cada vez más detalle en los diagramas y especificaciones.

High-Tech Dictionary. http://www.currents.net/resources/dictionary/dictionary.phtml
Orientado a objetos. Relacionado con le uso de objetos ; un objeto en este sentido es un componente que contiene datos e instrucciones para las operaciones que llevan a cabo los datos.

Definición propia

Orientado. En sistemas, que se inclina hacia una tendencia específica, a un modo de hacer las cosas, a un método específico.

Sinónimos

Diccionario razonado de sinónimos y contrarios. Vecchi, España 1985
Orientar. Situar, colocar, disponer, informar, instruir, aconsejar, adiestrar, guiar, dirigir, encausar, encaminar, enderezar.

Antónimos

Diccionario de Sinónimos y Antónimos. Culturas Internacionales, México 1991
Orientar. Desencaminar, desorientar, descarriar, desviar, desaconsejar, desconectar, despistar, equivocar, torcer, ofuscar, extraviar.

Definiciones en otras lenguas

Inglés
The Oxford English Reference Dictionary. Oxford University Press, New York 1996

Orient. *Place or exactly determine the position of with the aid of a compass; settle or find the bearings of.*
Orientar. Poner o determinar exactamente la posición de algo con la ayuda de una brújula; establecer o encontrar las relaciones de algo.

Francés
Le petit Larousse Illustré. Larousse, Belgique 1994

Orienté,e. *Adjetive. Qui a une position, une direction déterminée.*
Orientado. Adjetivo. Que tiene una posición o dirección determinada.

Italiano
Vocabolario della lingua italiana. Zingarelli, Milano 1974

Orientàto. *Participio passato di orientare; anche aggetivo.*
Orientado. Participio pasado de orientar; también se usa como adjetivo.

Alemán
Duden Deutsches Universalwörterbuch. Dudenverlag Mannheim, Deutchland 1989

Orienteren. *Die richtige Richtung finden.*

Orientar. Encontrar la dirección exacta.

3.5.5 Objeto

Definiciones etimológicas

GNmez de Silva, Gaido. Breve diccionario etimolNgico de la lengua espa>ola. El Colegio de México, FCE, México, 1988
Objeto. Algo que puede percibirse con uno o más sentidos (Especialmente si se pude ver y tocar), cosa material: latin *objectus* 'objeto, cosa vista' de *objectus* 'puesto delante, echado delante' y participio pasivo de *objicere obicere* 'echar ante'.

Definiciones de Diccionarios

Diccionario de la Real Academia de la Lengua Española. Madrid, 1992
Objeto. Todo lo que puede ser materia de conocimiento o sensibilidad de parte del sujeto, incluso este mismo. Lo que sirve de materia o asunto al ejercicio de las facultades mentales.

Otros diccionarios

María Moliner. Diccionario del uso del español. Gredos, España, 1990
Objeto. Cosa. Particularmente cosa corpórea y, en especial, de no gran tamaño.

Diccionario Enciclopédico Santillana. España, 1992
Objeto. Cosa especialmente la de carácter material. Aquello a lo que va dirigido una acción, un pensamiento, un sentimiento, etc. Finalidad propósito.

Diccionario del Español USUAL en México. El Colegio de México, 1996
Objeto. Todo lo que pueda ser materia, asunto o cuestión que alguien perciba o conozca particularmente lo que sea real o comprobable por los demás.

Diccionarios y glosarios especializados

Universidad de Leicester. http://wombat.doc.ic.ac.uk/foldoc
Diseño orientado a objetos. Método de diseño en el que se modela un sistema como una colección de objetos individuales y objetos que cooperan entre sí. Estos objetos se capturan en instancias de una clase dentro de una jerarquía de clases.

Enciclopedia de términos de cómputo en Ineternet. http://www.techweb.com/encyclopedia/
Diseño orientado a objetos. Transformación de un modelo orientado a objetos a las especificaciones requeridas para crear un sistema. Primero se lleva a cabo el análisis orientado a objetos y después el diseño orientado a objetos, dando como resultado cada vez más detalle en los diagramas y especificaciones.

High-Tech Dictionary. http://www.currents.net/resources/dictionary/dictionary.phtml

Orientado a objetos. Relacionado con el uso de objetos ; un objeto en este sentido es un componente que contiene datos e instrucciones para las operaciones que llevan a cabo los datos.

Definición propia

Objeto. En sistemas, un objeto es un componente que almacena en sí mismo sus características propias y las instrucciones que debe ejecutar sin importar el medio ambiente en el que se encuentre.

Sinónimos

Diccionario razonado de sinónimos y contrarios. Vecchi, España 1985
Objeto. Materia, cuerpo, cosa, substrato, elemento, masa, concreción, asunto, substancia, esencia, fin término, finalidad, intento, propósito, intención.

Antónimos

Diccionario de Sinónimos y Antónimos. Culturas Internacionales, México 1991
Objeto. Idea, insustancialidad.

Definiciones en otras lenguas

Inglés
The Oxford English Reference Dictionary. Oxford University Press, New York 1996

Object. A material thing that can be seen or touched. Computing a package of information and a description of its manipulation.
Objeto. Una cosa material que puede verse o tocarse. Cómputo de un paquete de información y una descripción de su manipulación.

Francés
Le petit Larousse Ilustré. Larousse, Belgique 1994

Objet. Toute chose concrète, perceptible par la vue, le toucher. Programmation par objets: mode de programmation dans lequel les données et les procédures qui les manipulent sont reagrupées en entités appelées "objets".
Objeto. Toda cosa concreta, perceptible por la vista, que puede tocarse. Programación orientada a objetos: modo de programación en la que los datos y los procedimientos que los manipulan se agrupan en entidades llamadas "objetos".
Italiano
Vocabolario della lingua italiana. Zingarelli, Milano 1974

Oggètto. Tutto ciò che il soggetto conoscente intende come diverso da sé. Correntemente, ognicosa che può essere percipita dai sensi.
Objeto. Todo lo que el sujeto consciente interpreta como diverso de sí. Comúnmente, cualquier cosa que puede ser percibida por los sentidos.

Alemán
Duden Deutsches Universalwörterbuch. Dudenverlag Mannheim, Deutchland 1989

Objekt. *Gegenstand, auf den das Interesse, das Denken.*
Objeto. Sujeto tema de cierto interés o de pensamiento.

UML (Unified Modeling Language)
Lenguaje Unificado de Modelado

3.5.6 Lenguaje

Definiciones etimológicas

Corominas, Joan. Breve Diccionario Etimológico de la Lengua Castellana. Gredos, Madrid, 1973
Lengua. Del latín *lingua* 'órgano humano para comer y pronunciar.
Lenguaje. Derivado de lengua (1220-1250). Manera de hablar.

Definiciones de diccionarios

Diccionario de la Real Academia de la Lengua Española. Madrid, 1992
Lenguaje. Conjunto de sonidos articulados con que el hombre manifiesta lo que piensa o siente. Desde el punto de vista informático: conjunto de signos y reglas que permiten la comunicación con un ordenador.

María Moliner. Diccionario del uso del español. Gredos, España, 1990
Lenguaje. Facultad de emplear sonidos articulados para expresarse, propia del hombre.

Diccionario Enciclopédico Santillana. España, 1992
Lenguaje. Medio o instrumento de comunicación entre los miembros de una misma especie.

Diccionario del Español USUAL en México. El Colegio de México, 1996
Lenguaje. Conjunto de señales o signos que permite a los seres humanos comunicar lo que piensan o sienten, generalmente mediante sonidos articulados en palabras o mediante algún otro medio sensible: lenguaje escrito, de la fotografía, de las manos.

Sinónimos

Diccionario razonado de sinónimos y contrarios. Vecchi, España 1985
Lenguaje. Argot, caló, charla, dialecto, fraseología, habla, idioma, jerga, jeringonza, mímica, terminología, vernáculo.

Antónimos

Diccionario de Sinónimos y Antónimos. Culturas Internacionales, México 1991

Lenguaje. Mudez, mutismo, silencio.

Definiciones en otras lenguas

Inglés
The Oxford English Reference Dictionary. Oxford University Press, New York 1996

Language. The method of human communication, either spoken or written, consisting of the use of words in an agreed way. A system of symbols and rules for writing computer programas or algorithms.
Lenguaje. El método de comunicación humana, ya sea escrito o hablado, que consiste del uso de palabras de una forma convenida. Un sistema de símbolos y reglas para escribir programas de computadora o algoritmos.

Francés
Le petit Larousse Ilustré. Larousse, Belgique 1994

Langage. *Faculté propre à l'homme d'exprimer et de communiquer sa pensée au moyen d'un systeme de signes vocaux ou graphiques. Ensemble de caractères, de symboles et de règles permettant de les assemble, utilisé pour donner des instructions à un ordinateur.*
Lenguaje. Facultad propia del hombre para expresar y comunicar su pensamiento por medio de un sistema de signos vocales o gráficos. Conjunto de caracteres, de símbolos y de reglas que permiten su conjunción y utilización para dar instrucciones a una computadora.

Italiano
Vocabolario della lingua italiana. Zingarelli, Milano 1974

Linguaggio. *Sistema dei signi per mezzo dei quali gli uomini comunicano fra loro. Nella programazione dei sistemi elettronici per l'elaborazione dei dati, insieme di simboli e regole utilizzato per la redazione dei programmi di elaborazione.*
Lenguaje. Sistema de signos por medio del cual los hombres se comunican entre si. En la programación de los sistemas electrónicos para el procesamiento de datos, conjunto de símbolos y de reglas usados para la redacción de los programas.

Alemán
Duden Deutsches Universalwörterbuch. Dudenverlag Mannheim, Deutchland 1989

Sprache. *Das sprechen als Anlage, Möglichkeit, Fähigkeit.*
Lenguaje. Disposición, posibilidad, capacidad de hablar o de comunicarse.

3.5.7 Modelado

Definiciones etimológicas

Corominas, Joan. Breve Diccionario Etimológico de la Lengua Castellana. Gredos, Madrid, 1973
Modelo. Del italiano *modello*, del latín *modéllus*, diminutivo y sinónimo de *modelus*.
Modelar. (1765-83) derivado de modo
Modo. Tomado del latín *módus*, manera, género.
Modelado, da. Participio pasado de modelar.

Definiciones de diccionarios

Diccionario de la Real Academia de la Lengua Española. Madrid, 1992
Modelado. Acción y efecto de modelar.
Modelar. Forma de cera o barro u otra materia blanda una figura o adorno. Configurar o conformar algo no material.

María Moliner. Diccionario del uso del español. Gredos, España, 1990
Modelado. Sin definición. V. MODO
Modo. En sentido amplio modo es cualquiera de los resultados de la combinación de los accidentes de una cosa variable que la hace diferente en cada caso.

Diccionario Enciclopédico Santillana. España, 1992
Modelado. Participio pasado de modelar, también adjetivo. Acción o arte de modelar.
Modelar. Dar forma artística al barro, la cera o a otra materia blanda.
Diccionario del Español USUAL en México. El Colegio de México, 1996
Modelado. Sin definición.
Modelar. Hacer mano a figuras con materiales como la cera, el barro, etc.

Sinónimos

Diccionario razonado de sinónimos y contrarios. Vecchi, España 1985
Modelo. Arquetipo, ejemplo, ideal ,módulo, muestra, norma, original, prototipo, tipo.

Antónimos

Diccionario de Sinónimos y Antónimos. Culturas Internacionales, México 1991
Modelo. Copia, calco, imitación.

Definiciones en otras lenguas

Inglés
The Oxford English Reference Dictionary. Oxford University Press, New York 1996

Mode. *A representation in three dimensions of an existing person or thing or of a proposed structure, on a smaller scale. A simplified description of a system.*
Modelo. Una representación en tres dimensiones de una persona o cosa existente o de una estructura propuesta, en una escala menor. Una descripción simplificada de un sistema.

Francés
Le petit Larousse Ilustré. Larousse, Belgique 1994

Modeler. *Pétrir (de la terre, de la cire, etc.) pour obtenir une certaine forme. Donner une forme, un relif particulier à. Fixer d'après un modèle.*
Modelar. Petrificar (barro, cera, etc.) para obtener alguna forma. Dar una forma, un relieve particular a. Fijar según un modelo.

Italiano
Vocabolario della lingua italiana. Zingarelli, Milano 1974

Modellàto. *Participio passato di modellare; anche aggetivo.*
Modelado. Participio pasado de modelar; también se usa como adjetivo.
Modellàre. *Formare, fare o conformare qual cosa rifacendosi a un preciso modello.*
Modelar. Formar, hacer o conformar algo siguiendo un modelo preciso.

Alemán
Duden Deutsches Universalwörterbuch. Dudenverlag Mannheim, Deutchland 1989

Modell. *Form, Beschaffenheit, Ma§verhältnisse veranschaulichende Ausführung eines vorhandenen oder noch zu schaffenden.*
Modelo. Forma, naturaleza, estado a escala que ilustra o explica la realización de algo existente o no.

3.5.8 Unificado

Definiciones etimológicas

Corominas, Joan. Breve Diccionario Etimológico de la Lengua Castellana. Gredos, Madrid, 1973
Unificar. Del latín *uni* uno más *facere* hacer. Hacer uno.
Unificado. Participio pasado de unificar.

Definiciones de diccionarios

Diccionario de la Real Academia de la Lengua Española. Madrid, 1992
Unificado. Sin definición
Unificar. Hacer de muchas cosas una o un todo, uniéndolas, mezclándolas o reduciéndolas a una misma especia. Hacer que cosas diferentes o separadas formen una organización, produzcan un determinado efecto, tengan una misma finalidad, etc.

María Moliner. Diccionario del uso del español. Gredos, España, 1990
Unificado. Sin definición.
Unificar. Hacer que entre las cosas de cierta clase no haya diferencias en el aspecto de que se trata. Hacer que varias cosas distintas o separadas formen un conjunto u organización o contribuyan a un efecto o una acción común.

Diccionario Enciclopédico Santillana. España, 1992
Unificado. Sin definición
Unificar. Poner juntas personas o cosas distintas o independientes para que formen un todo, contribuyan a una misma acción o finalidad.

Diccionario del Español USUAL en México. El Colegio de México, 1996
Unificado. Sin definición
Unificar. Hacer que varias personas, acciones, opiniones, etc. distintas y aisladas se unan, generalmente para conseguir cierto propósito común a todas ellas.

Sinónimos

Diccionario razonado de sinónimos y contrarios. Vecchi, España 1985
Unificar. Aunar, adunar, agrupar, centralizar, juntar, aislar, separar.

Antónimos

Diccionario de Sinónimos y Antónimos. Culturas Internacionales, México 1991
Unificar. Desunificar, desunir, separar, diversificar, romper.

Definiciones en otras lenguas

Inglés
The Oxford English Reference Dictionary. Oxford University Press, New York 1996
Unify. *Reduce to unity or uniformity.*
Unificar. Reducir a unidad o uniformidad.

Francés
Le petit Larousse Ilustré. Larousse, Belgique 1994

Unifier. *Amener ou ramener à l'unité.*
Unificar. Hacer o conducir a la unidad.

Alemán
Duden Deutsches Universalwörterbuch. Dudenverlag Mannheim, Deutchland 1989

Vereinheitlichen. *Unterschiedliches, normierend, einheitlich machen.*
Unificar. Quitar la diferencia, normalizar, uniformar algo.

Diccionarios y glosarios especializados

Universidad de Leicester. http://wombat.doc.ic.ac.uk/foldoc
UML. (Unified Modeling Language) Lenguaje de modelado de tercera generación. El UML es un método abierto que se usa para especificar, visualizar, construir y documentar los artefactos de un software orientado a objetos bajo desarrollo. El UML representa una compilación de las "mejores prácticas de ingeniería" que han tenido mucho éxito en el modelado de sistemas grandes y complejos. El UML es el resultado de la fusión de los conceptos e ideas de Booch, OMT y OOSE de sus ideas del modelado de sistemas orientado a objetos.

Enciclopedia de términos de cómputo en Ineternet. http://www.techweb.com/encyclopedia/
UML. (Unified Modeling Language) Lenguaje de diseño orientado a objetos del OMG, Object Management Group (Grupo del Manejo de Objetos)). En los 80s se desarrollaron varias metodologías para el diseño orientado a objetos. El UML unifica los métodos más populares es un solo estándar, que incluye a Grady Booch con su trabajo en Rational Software, Object Modeling Technique (Técnica de Modelado de Objeto) de Rumbaugh y Ivar Jacobson con su trabajo en casos de uso.

High-Tech Dictionary. http://www.currents.net/resources/dictionary/dictionary.phtml
UML. Sin definición.

Definición propia

UML. Es un lenguaje para especificar, visualizar, construir y documentar los elementos de los sistemas de software, así como el modelado de negocios y otros sistemas que no son necesariamente

de software. El UML representa la unificación de los métodos de análisis y sistemas orientados a objetos.

3.6 Tendencias

El UML no es propiedad de nadie y está disponible para todos. Se enfoca en las necesidades de los usuarios y de las comunidades científicas, ya que está establecido en métodos con mucha experiencia. Varios desarrolladores, organizaciones y vendedores de herramientas lo usan e incluso participaron en su desarrollo. Debido a que el UML se construyó con semánticas parecidas a las de Booch, OMT, OOSE y otros métodos líderes varias personas se han mostrado muy interesadas por este proyecto.

Hay dos aspectos de 'unificado' que el UML alcanza o logra: Primero, definitivamente termina con muchas de las varias diferencias entre lenguajes de modelado de métodos previos. Segundo, y quizá más importantes, unifica las perspectivas entre diferentes tipos de sistemas (negocios versus software), fases de desarrollo (requerimientos de análisis, diseño e implementación) y conceptos internos.
Debido a lo anterior varias organizaciones alrededor del mundo ya adoptaron al UML como su herramienta base para el desarrollo de sistemas de software. Se espera que le número de organizaciones que lo adopten crezca considerablemente en el tiempo y que motiven a otros desarrolladores, vendedores, organizaciones y autores de libros y cursos a que usen el UML.

El verdadero éxito del UML va a residir en su uso en proyectos exitosos y en la demanda creciente de herramientas de soporte, libros, herramientas, entrenamiento, asesoría, etc.

3.6.1 Evolución Futura del UML

Aunque el UML define un lenguaje preciso, no es una barrera para hacer mejoras futuras a los conceptos de modelado. Se espera que el UML en su forma actual sea la base de varias herramientas, incluyendo aquellas del modelado visual, simulación y ambientes de desarrollo. Se espera también que se desarrollen integraciones de herramientas que se usen como estándares de implementación basados en el UML.
El UML ha integrado varias ideas relacionadas con la orientación a objetos y a logrado obtener un resultado bastante completo y entendible, por lo tanto va a acelerar el uso del OO.

CAPITULO 4
Marco Metodológico

Marco Metodológico

4.1 Variables

1. Independientes

Si se desea realizar el análisis y diseño de un sistema con el paradigma orientado a objetos, usando el Lenguaje de Modelado Unificado UML, que es una notación gráfica; y además cuentan con una guía para llevar acabo el análisis y el diseño paso a paso junto con la documentación.

2. Dependientes

Contará con un documento actualizado de los requerimientos que el cliente haya solicitado, es decir el análisis.

Contará con un documento que explique el diseño total del sistema.

4.2 Hipótesis definitiva

Si se desea realizar el análisis y diseño de un sistema orientado a objetos usando el UML, se cuenta con una guía para hacer el análisis y el diseño con su respectiva documentación, las probabilidades de que se tenga éxito en la realización del sistema son muy grandes.

4.3 Definición del universo

El universo que considero para mi investigación está formado por empresas que realizan el análisis y diseño de sistemas, particularmente orientado a objetos.

4.4 Determinación de la muestra

En base al numero de concurrencias se realiza el tipo de muestra. En el caso de esta herramienta la muestra se ha toma por juicio. Primero investigué qué personas estaban usando la metodología UML o qué personas han adquirido el software Rational Rose, el cual está totalmente basado en el lenguaje UML y es muy popular en la comunidad informática. Son muy pocas las empresas que usan el lenguaje UML, entre estas empresas se encuentran el INE (Instituto Nacional de Ecología), la DCAA (Dirección de Cómputo para la Administración Académica de la UNAM) y el Citibank. Decidí centrarme en estas dos empresas porque tienen un promedio de un año usando la herramienta.

4.5 Definición del método de la investigación

La investigación acerca de los métodos orientados a objetos, específicamente del UML, no están muy desarrollados y por lo tanto su uso no es todavía muy común, sin embargo algunas empresas ya están realizando algunos sistemas con el método orientado a objetos. He contactado a tres empresas que aunque no usan específicamente al UML, si usan métodos orientados a objetos, por lo tanto he decidido realizar encuestas para conocer su punto de vista y finalmente aprobar mi hipótesis.

4.6 Costo de la investigación

La investigación ha tenido una duración aproximada de seis meses, durante ese tiempo los gastos efectuables son los siguientes:

Concepto	Mensual	Total
Equipo de cómputo (realizado una sola vez)		15,000
Papel, tinta para imprimir, discos 3 1/2	200	1,800
Renta de un lugar para realizar la investigación	5,000	30,000
Teléfono	1000	6,000
Luz	300	1,800
Renta de Internet	200	1,200
Honorarios por la investigación	10,000	60,000
Total		115,800

4.7 Construcción del cuestionario

Pregunta	Razón de ser	Respuesta esperada
1. ¿Qué importancia considera que tiene la documentación para el desarrollo de un sistema?	Confirmar que la documentación es muy importante para el desarrollo de un sistema.	La documentación es parte fundamental de un sistema, es tan importante que sin ella el desarrollo del sistema podría resultar complicado.
2. ¿Se realiza la documentación cada vez que se desarrolla un sistema? ¿Porqué?	Saber si la empresa entrevistada tiene el hábito de realizar la documentación por cada sistema que desarrolla debido a que la documentación es parte y soporte del sistema.	Sí, porque es necesario contar con un soporte.
3. ¿En qué etapa de desarrollo se realiza la documentación?	Saber si la documentación se lleva acabo desde el inicio o al final del desarrollo.	En todas las etapas del desarrollo del sistema.
4. ¿Considera que la etapa mencionada en su respuesta anterior es la mejor o considera que puede ser otra?	Conocer la opinión de la persona acerca de la etapa de desarrollo idónea para la documentación.	Sí, la documentación debe realizarse en todas las etapas del desarrollo.
5. ¿Usa alguna herramienta para facilitar la realización de la	Saber si se cuenta con alguna herramienta y cuál, ya que al final de mi trabajo pretendo	Sí. En este caso las variedad de herramientas que pueden usarse son muchas, por lo tanto espero que

documentación? ¿Cuál y porqué?	proponer algunas herramientas que pueden resultar de utilidad.	usen alguna herramienta.
6. ¿Qué ventajas y/o desventajas encuentra usted en la realización de la documentación?	Saber si la persona considera que realizar la documentación tiene más ventajas que desventajas.	Ventajas: facilitar el trabajo de desarrollo. Desventajas: quita tiempo y puede resultar tedioso.
7. ¿Quiénes considera que se benefician con la documentación? ¿Porqué?	Conocer el beneficio que aporta la documentación para las personas involucradas o no con el desarrollo y uso del sistema.	Todos se benefician con la documentación. Porque a todos nos sirve de una forma o de otra.
8. ¿Cuáles considera que serían las consecuencias si no se realiza la documentación o se hace de manera confusa?	Saber la opinión del o los entrevistados, con base en su experiencia, acerca de las consecuencias de una mala documentación o carencia de la misma.	La peor consecuencia es que no se contaría con un soporte para realizar modificaciones o para entender el funcionamiento y desarrollo del sistema.
9. ¿Porqué usa la tecnología orientada a objetos?	Confirmar que en algunos casos la tecnología orientada a objetos puede ser de gran utilidad.	Porque simplifica procesos y facilita el desarrollo si se sabe usar correctamente.
10. ¿Desde cuándo usa el UML?	Saber cuanto tiempo tiene usando el UML, para compararlo con la fecha de su surgimiento.	Espero que la respuesta sea en promedio un año.
11. ¿Específicamente para qué usa al UML?	Comprobar que el UML es útil tanto para el análisis, el diseño y la documentación.	Para la documentación, principalmente, aunque también para el análisis y el diseño.
12. ¿Aplica todos los elementos de la notación UML que proponen sus creadores?	Comprobar que aplicar la notación completa resulta engorroso y complicado.	No, solamente los elementos que considero necesarios para el tipo de sistema que este desarrollando.
13. ¿Qué elementos aplica y porqué?	Saber que elementos usa y compararlos con los elementos que propongo en mi trabajo de tesis.	En esta respuesta no espero ninguna respuesta específica.
14. ¿Considera que se han obtenido beneficios en el desarrollo de sistemas a partir del uso del UML?	Verificar que el uso del UML puede aportar beneficios al desarrollo del sistema, específicamente en su etapa de documentación.	Si
15. ¿Cuál considera que es la razón por la que ha obtenido o no beneficios?	Comprobar que el buen uso de la metodología puede aportar beneficios y su mal uso puede causar problemas.	La principal razón es el uso adecuado de la metodología.
16. ¿Ha sido complicado incorporar la notación UML y porqué?	Saber en qué grado y en qué aspectos resulta complicado incorporar la notación UML.	Si, porque implica un cambio de forma de pensar de estructurado a orientado a objetos.
17. ¿El uso del UML ha sido de utilidad para realizar la documentación de los sistemas?	Saber si el esfuerzo de incorporar la notación UML ha valido la pena.	Si.
18. ¿De qué forma ha sido o no de utilidad?	Conocer específicamente los beneficios que han obtenido con el uso de dicha notación.	Ha sido de utilidad porque el UML sugiere diagramas que son útiles para la documentación, simplificación y comprensión de procesos.
19. ¿Considera que el uso del UML ha facilitado el análisis, diseño y documentación de los sistemas? ¿Porqué?	Concluir si el UML en su conjunto cumple con los objetivos para los que fue creado, que son facilitar el análisis, diseño y documentación de sistemas orientados a objetos y en algunos casos, sin proponérselo, de sistemas estructurados.	Sí, porque el UML simplifica las cosas y establece diagramas que pueden usarse de forma flexible.

4.8 Cuestionario piloto

El cuestionario fue aplicado, a modo de prueba, para garantizar que las preguntas planteadas fueran claras y no hubiera grandes dudas ni necesidades de explicaciones durante su aplicación real. Afortunadamente la evaluación de la comprensión del cuestionario fue satisfactoria, no se presentaron problemas de entendimiento. La duración de la aplicación del cuestionario fue aproximadamente de 35 a 45 minutos.

4.9 Cuestionario definitivo

El cuestionario definitivo, que después de la evaluación realizada, permanece sin variaciones al original.

4.10 Realización de la investigación

Los entrevistados son:
INE Instituto Nacional de Ecología. Departamento de Sistemas. Sr. Carlos Roldán
DCAA Dirección de Cómputo para la Administración Académica. Departamento de Estructuración. Lic. Edgar González
Banco Citibank. Departamento de Sistemas. Lic. Juan Torres

Los resultados de cada uno de lo tres cuestionarios se encuentra en el Anexo 1.

4.11 Conclusión sobre los resultados

1. ¿Qué importancia considera que tiene la documentación para el desarrollo de un sistema?	
INE	Es fundamental, es la parte esencial de todo sistema ya que sin esta se podría decir que existirá un margen de error en el desarrollo del sistema de 80% al 90%.
DCAA	Tiene mucha importancia porque es la base del sistema.
Citibank	La documentación es tan importante como cualquier otro elemento del sistema.

Conclusión. No hay ninguna duda de que la documentación es sumamente importante para el desarrollo de los sistemas, de no realizarla hay una fuerte posibilidad de errores, debido a que la documentación es parte del desarrollo de todo sistema.

2. ¿Se realiza la documentación cada vez que se desarrolla un sistema? ¿Porqué?	
INE	Si. Porque se muestran las necesidades, el análisis, las actividades y toda la información relacionada con el sistema que se realizo para llevar a acabo el desarrollo de dicho sistema; desde su inicio hasta su final.
DCAA	Si. Porque la documentación es parte del sistema, si el sistema llega a fallar o es necesario realizar un cambio, la documentación ayuda a hacerlo con facilidad y a observar tanto el desarrollo mismo del sistema en papel como las modificaciones que se le hagan.
Citibank	Si. Es parte fundamental de todo sistema, debemos saber qué estamos trabajando y cómo lo estamos haciendo.

Conclusión. Todos los entrevistados realizan la actividad de documentación cuando desarrollan un sistema, lo cual indica que ésta actividad es imprescindible para ellos; de hecho los tres coinciden en que además de que la documentación es parte del desarrollo de cualquier sistema, también sirve como guía para saber lo que se hace y cómo se hace. Es indudable pues, que la documentación es de gran utilidad.

3. ¿En qué etapa de desarrollo se realiza la documentación?	
INE	Dentro de la metodología utilizada por el Instituto se realiza al principio un documento conocido como DOVI (Documento de Visión) que es la visión general del sistema que se desea desarrollar. Al término de éste se comienza con otro documento que es más técnico y que es conocido como ESFU (Especificación funcional). Cabe mencionar que también son requeridos los manuales de procedimientos de las áreas donde se va a realizar el sistema, documentos relacionados, etc.
DCAA	Se realiza desde el inicio del análisis y recopilación de información para tener un soporte sólido en cuanto al tipo de sistema a desarrollar y no termina hasta que el sistema está terminado. En realidad todas las etapas son documentadas paralelamente.
Citibank	La documentación se realiza durante todo el desarrollo del sistema y al final se afinan detalles de presentación, pero la información se actualiza sobre la marcha.

Conclusión. De acuerdo a los entrevistados, en realidad no existe una etapa definida para realizar la documentación, ésta se lleva acabo durante todo el desarrollo del sistema: desde los requerimientos hasta que el sistema está terminado.

4. ¿Considera que la etapa mencionada en su respuesta anterior es la mejor o considera que puede ser otra?	
INE	Hasta ahora nos ha funcionado realizar la documentación desde el principio y después en todas las etapas.
DCAA	Considero que lo más conveniente es documentar todas las etapas del sistema en el momento en que se estén realizando, como se menciono con anterioridad.
Citibank	De la forma que lo hemos hecho para nosotros ha funcionado bien.

Conclusión. De acuerdo a la experiencia de los entrevistados la documentación debe realizarse en todas las etapas del desarrollo de un sistema, de esta forma obtenido resultados satisfactorios.

5. ¿Usa alguna herramienta para facilitar la realización de la documentación? ¿Cuál y porqué?	
INE	No, en realidad la única herramienta que usa el instituto es Word para crear dicha documentación.
DCAA	Usamos Word, Visio porque facilita la creación de diagramas y últimamente Rational Rose porque facilita la creación de diagramas usando el UML.
Citibank	No usamos ninguna herramienta en especial, solamente un procesador de textos y una herramienta para graficar.

Conclusión. Dos de los entrevistados no usan ninguna herramienta especifica para realizar la documentación, el entrevistado en la DCAA menciona Rational Rose que está basado en el UML, como herramienta de ayuda para la realización de los diagrama que forman parte de la documentación.

6. ¿Qué ventajas y/o desventajas encuentra usted en la realización de la documentación?	
INE	**Ventajas:** Facilita el desarrollo de sistemas Disminuye el tiempo de desarrollo de sistemas (Si se compara con el tiempo que puede llevarse al realizar modificaciones no estimadas con una documentación no existente) Ayuda a comprender bien el problema Si existen modificaciones se pueden realizar más fácilmente. **Desventajas:** En sistemas o programas pequeños, quita tiempo Es una tarea muy laboriosa y difícil Implica coordinación entre las personas dedicadas al desarrollo, ya que se debe comunicar cualquier cambio, ésto conlleva a la desventaja siguiente No siempre se lleva actualizada
DCAA	**Ventajas:** Se registra lo que el cliente quiere que haga el sistema Ayuda a no olvidar los objetivos y alcances del sistema Es más fácil hacer cambios y de ser necesario regresar a la versión anterior sin tantas dificultades Contamos con un soporte en papel del sistema La documentación da confianza al desarrollador y al cliente **Desventajas:** Realizar la documentación es laborioso y tardado Si no se realiza de manera adecuada puede perjudicar en lugar de ayudar Es necesario trabajar en equipo todo el tiempo, lo cual a veces resulta difícil Es difícil mantenerlo actualizado
Citibank	**Ventajas:** La ventaja más importante es que logramos controlar el proceso de desarrollo del sistema **Desventajas:** La única desventaja es que el equipo no este bien organizado, pero en ese caso todo el desarrollo del sistema se encuentra en riesgo

Conclusión. La ventaja más importante en la que coinciden los entrevistados es que la documentación ayuda a entender el sistema y a tener un control sobre la forma en la que se realiza y las modificaciones que sufre durante el proceso hasta que está terminado. De no existir la documentación actualizada de principio a fin se pierde la oportunidad de tener información confiable acerca del sistema que se está desarrollando. Una de las desventajas es que la documentación puede resultar tardada y tediosa; sin embargo considero que esto no sucedería si el equipo de desarrollo se encuentra bien organizado, lo cual es otra desventaja que es propia del equipo de trabajo más que de la documentación misma. Es necesario aclarar que la buena o mala organización del equipo de trabajo no solo repercute en la documentación, sino en todo el desarrollo del sistema.

7. ¿Quiénes considera que se benefician con la documentación? ¿Porqué?	
INE	El beneficio es para todos, es decir, beneficia tanto al usuario final del sistema, a los desarrolladores, a los analistas y a los líderes del proyecto.
DCAA	Nos beneficiamos todos, porque la documentación es para todos, para los analistas, desarrolladores, programadores, líderes de proyecto, usuarios finales, etc.; y todos pueden utilizarla de acuerdo a sus tareas.
Citibank	Primero, los beneficiados son los propios desarrolladores porque se mantienen al tanto de lo que ellos mismos están haciendo y después por supuesto los usuarios finales porque conocen los alcances y limitaciones del sistema.

Conclusión. La respuesta es muy clara: la documentación nos beneficia a todos, desde los desarrolladores hasta los usuarios finales. Por medio de la documentación, aún sin ver el sistema, es posible conocerlo, saber el grado de desarrollo en el que se encuentra, conocer las limitaciones y alcances del mismo. Es necesario aclarar que hay diferentes tipos de documentación, en este caso la idea es resaltar la importancia de la documentación durante el desarrollo del sistema.

8. ¿Cuáles considera que serían las consecuencias si no se realiza la documentación o se hace de manera confusa?	
INE	a) El desarrollador no podrá realizar un buen sistema b) El sistema o programa tendrá un número considerable de errores c) Complicará cualquier modificación
DCAA	La peor consecuencia es que el sistema no sería completamente confiable; los errores serían más difíciles de encontrar y sería muy difícil mantener un seguimiento a las modificaciones que se le hicieran. Además el cliente tampoco tendría una guía para entender y conocer el sistema.
Citibank	La consecuencia es que se tiene un sistema incompleto, es como si tuviéramos un sistema sin interface de usuario. Es necesario entender que la documentación es parte del desarrollo de cualquier sistema.

Conclusión. La documentación es parte del desarrollo de todo sistema, no realizarla se traduce en un sistema incompleto y por lo tanto con errores.

9. ¿Porqué usa la tecnología orientada a objetos?	
INE	Porque hasta donde hemos probado facilita el desarrollo de sistemas, de una manera más rápida, al principio es my difícil, pero una vez comprendida es fácil utilizar las herramientas ya creadas.
DCAA	Porque simplifica las cosas, no en todos los casos claro, sólo en algunos.
Citibank	Porque en algunas ocasiones esta tecnología nos ha facilitado el desarrollo de los sistemas.

Conclusión. La tecnología orientada a objetos facilita el desarrollo de sistemas en general, aunque en algunos casos probablemente otras tecnologías sean más recomendables. La adopción de la tecnología orientada a objetos puede no ser fácil al principio pero una vez adoptada resulta muy útil.

10. ¿Desde cuándo usa el UML?	
INE	Aproximadamente un año dos meses dentro del Instituto Nacional de Ecología.
DCAA	Desde hace un año aproximadamente, aunque no lo usamos de manera constante.
Citibank	Aproximadamente una año y medio.

Conclusión. El promedio de uso de la herramienta es de un año, por lo tanto la experiencia que los entrevistados expresarán en las siguientes respuestas será en base a un año de trabajo con el UML.

11. ¿Específicamente para qué usa al UML?	
INE	Para el análisis de sistemas que se desarrollan dentro del instituto.
DCAA	Para el análisis y diseño de sistemas, incluso sin importar que no sean orientados a objetos.
Citibank	Principalmente para tener una visión global del sistema en diagramas, así como para el análisis y diseño y finalmente para crear el código de forma automática con el uso de Rational Rose.

Conclusión. Todos los entrevistados usan el UML para el análisis, solamente dos para el diseño también y uno solo para la creación del código mediante la herramienta Rational Rose. El UML está conformado en su mayoría por diagramas: diagramas generales y diagramas para usos específicos que son de gran utilidad para las etapas de análisis y diseño de sistemas.

12. ¿Aplica todos los elementos de la notación UML que proponen sus creadores?	
INE	No exactamente, de hecho existen puntos que a veces omitimos e incluso creemos que dicha metodologia tal y como está sólo sirve para el desarrollo de grandes sistemas, más no asi para desarrollos pequeños.
DCAA	No, solamente aplicamos los elementos que consideramos de utilidad, algunos no los conocemos, se requiere más tiempo para conocerlos a todos y algunos de ellos no nos son de utilidad.
Citibank	No, algunos de ellos no son de utilidad para nuestro desarrollo.

Conclusión. Todos los entrevistados coinciden en no utilizar todos los elementos del UML. El UML es muy grande e incluso complicado, una gran ventaja es que se pueden tomar únicamente elementos de utilidad de acuerdo a las necesidades de cada sistema sin que eso afecte el desarrollo o uso de la herramienta.

13. ¿Qué elementos aplica y porqué?	
INE	Dentro del documento Visión (DOVI), incluimos el diagrama de casos de uso, definición de actores y flujo de eventos. Dentro del documento de especificación funcional, incluimos la descripción de escenarios, los escenarios secundarios, el diagrama de colaboración y correlación , para después tratar de definir las clases y los objetos. Cabe mencionar que las clases Boundary, Entity y Control realmente no las utilizamos porque creemos que son muy particulares de la metodología y no se apegan en todos los casos.
DCAA	Aplicamos el diagrama de casos de uso, la definición de actores y de casos de uso, el flujo de eventos, el diagrama de colaboración. Los usamos porque facilitan el trabajo de documentación incluso para los sistemas que no son orientados a objetos.
Citibank	Casos de uso, diagramas de clases, diagramas de interacción, diagramas de paquete, diagramas de estado y diagramas de actividad; sin embargo no usamos todos los elementos propuestos en cada diagrama, únicamente usamos los más sencillos.

Conclusión. De acuerdo a las respuestas de los entrevistados, éstos usan los diagramas más sencillos y útiles del UML, sin que ésto ocasione ningún tipo de problema.

14. ¿Considera que se han obtenido beneficios en el desarrollo de sistemas a partir del uso del UML?	
INE	Si claro, se ve en el tiempo de desarrollo de sistemas y la manera en que estos funcionan.
DCAA	Si una vez comprendida la metodología (al menos la parte de ella que utilizamos) resulta más rápido y sencillo.
Citibank	Si, el uso del UML ha facilitado algunas de nuestras tareas y las hemos sistematizado de una forma más simplificada.

Conclusión. Todos los entrevistados expresan que han obtenido beneficios al usar el UML. Lo cual indica, al menos en esta pequeña muestra, que el uso del UML puede facilitar el desarrollo de sistemas.

15. ¿Cuál considera que es la razón por la que ha obtenido o no beneficios?	
INE	Si existen beneficios y son porque normalmente se aprovechan tanto los documentos como los códigos de las aplicaciones ya desarrolladas.
DCAA	Se han obtenido beneficios porque la metodología no es tan rígida y se pueden combinar diferentes elementos de utilidad sin confundirlos.
Citibank	Yo creo que la principal razón es que hemos aprendido a aprovechar los elementos de utilidad de la herramienta.

Conclusión. Son tres las principales razones por las que se han obtenido beneficios con el uso del UML: la flexibilidad de la herramienta, la facilidad de reuso que ofrece y la capacidad de los desarrolladores de adoptar la herramienta. Esta última razón depende más de la capacidad de los usuarios de adaptarse a herramientas, pero las dos primeras son propias de la herramienta misma.

16. ¿Ha sido complicado incorporar la notación UML y porqué?	
INE	Si claro que si, porque aún tenemos presente el paradigma del desarrollo de sistemas con otras metodologías no orientadas a objetos.
DCAA	Si ha sido complicado porque nuestra forma de pensar está muy acostumbrada a el análisis y diseño estructurados y es difícil deshacerse de este tipo de razonamiento de tal forma que no interfiera con la orientación a objetos.
Citibank	Si fue un poco complicado al principio porque la herramienta tenía elementos que es su momento no era tan comunes, de hecho aún lo son, entonces tuvimos que aprender cosas nuevas y a re-aprender cosas olvidadas.

Conclusión. En las preguntas anteriores los entrevistadores han coincidido en que el UML les ha facilitado el desarrollo de sistemas, en esta pregunta los tres entrevistados expresan que si ha sido complicado adoptar la notación UML, debido principalmente al cambio de paradigma que implica la orientación a objetos en la que principalmente se basa el UML.

17. ¿El uso del UML ha sido de utilidad para realizar la documentación de los sistemas?	
INE	Si
DCAA	Si
Citibank	Si

Conclusión. El UML sí es de utilidad para realizar la documentación de los sistemas, todos loa entrevistados coinciden en ésto de acuerdo a sus experiencias.

18. ¿De qué forma ha sido o no de utilidad?	
INE	Bueno es más fácil que otras metodologías.
DCAA	Ha sido de utilidad porque los diagramas son muy expresivos y fáciles de usar.
Citibank	Ha sido de utilidad por su flexibilidad y sobre todo porque al hacer la actualización en un diagrama los demás se actualizan automáticamente.

Conclusión. El UML ha sido de utilidad principalmente por sus diagramas y por la estructura de la herramienta.

19. ¿Considera que el uso del UML ha facilitado el análisis, diseño y documentación de los sistemas? ¿Porqué?	
INE	Si. Lo complicado es hacer el primer sistemas con la metodología, después puede ser utilizado para el desarrollo de otros sistemas y es más fácil identificar las necesidades del usuario y la documentación del mismo.
DCAA	Si. Porque la herramienta ayuda a mantener la documentación actualizada, sin embargo no es útil para realizar toda la documentación, solamente una parte de ella usando únicamente algunos elementos.
Citibank	Si. Porque la herramienta fue diseñada para esas actividades entre otras y funciona de forma adecuada y satisfactoria.

Conclusión. El UML facilita el análisis, diseño y documentación del desarrollo de sistemas. El análisis y el diseño son las partes más importantes porque es donde se define lo que será el sistema, la documentación es importante en todas las etapas porque sirve de guía de desarrollo. Las tres actividades son partes vitales del desarrollo de cualquier sistema, las cuales se realizan con facilidad mediante el uso de la herramienta UML.

4.13 Aprobación de la hipótesis

Con la investigación realizada se aprueba la hipótesis que dice de la siguiente manera:

Si se desea realizar el análisis y diseño de un sistema orientado a objetos usando el UML, se cuenta con una guía para hacer el análisis y el diseño con su respectiva documentación, las probabilidades de que se tenga éxito en la realización del sistema son muy grandes.

Considero conveniente agregar que el sistema para el que se aplica la hipótesis anterior puede también ser no orientado a objetos, siempre y cuando se tomen en cuenta únicamente los elementos que resulten útiles.

CAPITULO 5
Marco Instrumental

Marco Instrumental

5.1 Propuestas de acción

Las propuestas de acción que de acuerdo al trabajo de tesis van a tomarse son las siguientes:

1. Proposición de incluir el tema: *Documentación del análisis y diseño de sistemas orientados a objetos con UML*, en la materia *INFORMÁTICA VI* del 7o semestre de la carrera de Lic. en Informática de acuerdo al plan de estudios 98.
2. Publicación del resultado de la tesis en la revista ***Emprendedores*** de la Facultad de Contaduría y Administración.
3. Proposición de presentar el tema de la tesis en algún seminario, relacionado con el tema, que la Facultad de Contaduría y Administración organice.
4. Enviar currículum vitae y trabajo de tesis al proveedor de la herramienta Rational Rose basada en el UML.
5. Enviar resultados del trabajo a usuarios del UML.

5.2 Plan y programa de trabajo

El plan y programa de trabajo en base a las acciones propuestas es el siguiente:

1. Para la propuesta de incluir el tema de mi tesis en la materia *INFORMÁTICA VI*, envié una carta a la directora de la carrera Lic. en Informática de la Facultad de Contaduría y Administración explicándole los motivos por los que considero importante la inclusión del tema y también un temario especificando donde incluirlo. (Anexo 2)
2. Para la propuesta de la publicación del artículo con la conclusión de la tesis, envíe una carta al encargado de publicaciones de la Facultad de Contaduría y Administración anexando el artículo para su próxima publicación. (Anexo 3)
3. Para la propuesta del tema en seminario, también envíe una carta a la directora de la carrera Lic. en Informática de la Facultad de Contaduría y Administración en donde propongo mis conocimientos para alguna oportunidad que llegue a presentarse. (Anexo 4)
4. El Curriculum Vitae y el trabajo del proveedor han sido ya enviados al proveedor de la herramienta con una carta explicativa. (Anexo5)
5. Finalmente los usuarios de la herramienta UML, que entreviste y cuya ayuda me fue tan valiosa también han recibido el trabajo de tesis con una carta de agradecimiento por su ayuda. (Anexo 6)

Conclusiones

Conclusiones

La historia es el relato de los acontecimientos que se dan a través del tiempo. La historia de la humanidad, se supone, debe servirnos para aprender de nosotros mismos y no cometer los mismos errores. La historia de las computadoras y de los sistemas computarizados, al contrario, refleja una evolución vertiginosa que apenas percibimos; de tal forma que resulta imposible girar la mirada hacia atrás porque nos perderíamos lo que sucede en el momento. Así, la creación de nuevos sistemas que faciliten nuestras actividades se lleva a cabo, hoy en día, de una manera muy rápida. Por lo tanto es necesario que hagamos uso de todas aquellas herramientas que nos faciliten el desarrollo de sistemas.

A través del desarrollo de este trabajo de tesis me he dado cuenta que precisamente debido a la premura con la que son solicitados los sistemas, éstos sufren de carencias importantes. La principal es que el análisis y el diseño no son realizados con el cuidado necesario; de acuerdo a las entrevistas realizadas, los desarrolladores opinan que el análisis y el diseño son la columna vertebral de cualquier sistema sobre lo que debe trabajarse para obtener los resultados deseados. Si a esto le agregamos que la documentación de dichas etapas no se realiza en algunas ocasiones, entonces aunque conozcamos la esencia del sistema, perdemos los detalles primordiales que pueden marcar grandes diferencias que en apariencia son muy sutiles y casi imperceptibles. Y es aquí donde entramos al tema sobre el que trata este trabajo de tesis, la documentación es parte esencial de cualquier sistema desarrollado con cualquier paradigma de programación.

En base a mi investigación y a las entrevistas que tuve oportunidad de realizar, concluyo que:

1. El desarrollo de un sistema implica varias etapas, mismas que deben realizarse en su totalidad y con la mayor precisión posible, así como con la ayuda de herramientas que faciliten dichas tareas.

2. Dos etapas importantes del desarrollo de cualquier sistema, sin restarle importancia al resto, son el análisis y el diseño, que son la base de conocimiento para proceder a las etapas siguientes.

3. La documentación es una etapa clave en el desarrollo de cualquier sistema, que se lleva acabo desde el inicio, durante y al finalizar la realización de cualquier sistema. Es una etapa clave, porque es en ésta donde se plasman los objetivos, alcances y limitaciones del sistema, así como el desarrollo del mismo, sus modificaciones, cualquier alteración posterior al lanzamiento del sistema y la identificación de posibles errores.

4. La documentación del análisis y diseño de sistemas siempre debe realizarse con la mayor precisión posible para identificar errores o interpretaciones equivocadas de lo que debe ser el sistema antes de proceder a la programación y etapas posteriores.

5. Para realizar la documentación del análisis y diseño de sistemas el uso de herramientas adecuadas facilitan en gran medida estas tareas.

Conclusión de la hipótesis

El uso de la herramienta UML para la realización de la documentación del análisis y diseño de sistemas orientados a objetos o estructurados, utilizando únicamente los elementos útiles (en el Anexo 7 se proponen algunos elementos de UML) de acuerdo a las necesidades de los desarrolladores, incrementa las posibilidades de éxito en el desarrollo del sistema.

Anexos

ANEXO 1

Respuestas al cuestionario

INE (Instituto Nacional de Ecología)

1. ¿Qué importancia considera que tiene la documentación para el desarrollo de un sistema?
Es fundamental, es la parte esencial de todo sistema ya que sin esta se podría decir que existirá un margen de error en el desarrollo del sistema de 80% al 90%.
2. ¿Se realiza la documentación cada vez que se desarrolla un sistema? ¿Porqué?
Si. Porque se muestran las necesidades, el análisis, las actividades y toda la información relacionada con el sistema que se realizo para llevar a acabo el desarrollo de dicho sistema; desde su inicio hasta su final.
3. ¿En qué etapa de desarrollo se realiza la documentación?
Dentro de la metodología de este se comienza con otro documento que es más técnico y que es conocido como ESFU (Especificación funcional). Cabe mencionar que también son requeridos los manuales de procedimientos de las áreas donde se va a realizar el sistema, documentos relacionados, etc.
4. ¿Considera que la etapa mencionada es la mejor o considera que puede ser otra?
Considero que la etapa de documentación es fundamental en todo sistema, independientemente de la metodología que sea utilizada, de hecho pienso que todas las etapas debieran ser documentadas.
5. ¿Usa alguna herramienta para facilitar la realización de la documentación? ¿Cuál y porqué?
No, en realidad la única herramienta que usa el instituto es Word para crear dicha documentación.
6. ¿Qué ventajas y/o desventajas encuentra usted en la realización de la documentación?
Ventajas: Facilita el desarrollo de sistemas Disminuye el tiempo de desarrollo de sistemas (Si se compara con el tiempo que puede llevarse al realizar modificaciones no estimadas con una documentación no existente) Ayuda a comprender bien el problema Si existen modificaciones se pueden realizar más fácilmente.
Desventajas: En sistemas o programas pequeños, quita tiempo Es una tarea muy laboriosa y difícil

Implica coordinación entre las personas dedicadas al desarrollo, ya que se debe comunicar cualquier cambio, ésto conlleva a otra desventaja No siempre se lleva actualizada
7. ¿Quiénes considera que se benefician con la documentación? ¿Porqué?
El beneficio es para todos, es decir, beneficia tanto al usuario final del sistema, a los desarrolladores, a los analistas y a los líderes del proyecto.
8. ¿Cuáles considera que serían las consecuencias si no se realiza la documentación o se hace de manera confusa?
a) El desarrollador no podrá realizar un buen sistema b) El sistema o programa tendrá un número considerable de errores c) Complicará cualquier modificación
9. ¿Porqué usa la tecnología orientada a objetos?
Porque hasta donde hemos probado facilita el desarrollo de sistemas, de una manera más rápida, al principio es my difícil, pero una vez comprendida es fácil utilizar las herramientas ya creadas.
10. ¿Desde cuándo usa el UML?
Aproximadamente un año dos meses dentro del Instituto Nacional de Ecología.
11. ¿Específicamente para qué usa al UML?
Para el análisis de sistemas que se desarrollan dentro del instituto.
12. ¿Aplica todos los elementos de la notación UML que proponen sus creadores?
No exactamente, de hecho existen puntos que a veces omitimos e incluso creemos que dicha metodología tal y como está sólo sirve para el desarrollo de grandes sistemas, más no así para desarrollos pequeños.
13. ¿Qué elementos aplica y porqué?
Dentro del documento Visión (DOVI), incluimos el diagrama de casos de uso, definición de actores y flujo de eventos. Dentro del documento de especificación funcional, incluimos la descripción de escenarios, los escenarios secundarios, el diagrama de colaboración y correlación , para después tratar de definir las clases y los objetos. Cabe mencionar que las clases Boundary, Entity y Control realmente no las utilizamos porque creemos que son muy particulares de la metodología y no se apegan en todos los casos.
14. ¿Considera que se han obtenido beneficios en el desarrollo de sistemas a partir del uso del UML?

Si claro, se ve en el tiempo de desarrollo de sistemas y la manera en que estos funcionan.
15. ¿Cuál considera que es la razón por la que ha obtenido o no beneficios?
Si existen beneficios y son porque normalmente se aprovechan tanto los documentos como los códigos de las aplicaciones ya desarrolladas.
16 ¿Ha sido complicado incorporar la notación UML y porqué?
Si claro que si, porque aún tenemos presente el paradigma del desarrollo de sistemas con otras metodologías no orientadas a objetos.
17 ¿El uso del UML ha sido de utilidad para realizar la documentación de los sistemas?
Si
18 ¿De qué forma ha sido o no de utilidad?
Bueno es más fácil que otras metodologías.
19. ¿Considera que el uso del UML ha facilitado el análisis, diseño y documentación de los sistemas? ¿Porqué?
Si. Lo complicado es hacer el primer sistemas con la metodología, después puede ser utilizado para el desarrollo de otros sistemas y es más fácil identificar las necesidades del usuario y la documentación del mismo.

DCAA (Dirección de Cómputo para la Administración Académica)

1. ¿Qué importancia considera que tiene la documentación para el desarrollo de un sistema?
Tiene mucha importancia porque es la base del sistema.
2. ¿Se realiza la documentación cada vez que se desarrolla un sistema? ¿Porqué?
Si. Porque la documentación es parte del sistema, si el sistema llega a fallar o es necesario realizar un cambio, la documentación ayuda a hacerlo con facilidad y a observar tanto el desarrollo mismo del sistema en papel como las modificaciones que se le hagan.
3. ¿En qué etapa de desarrollo se realiza la documentación?

Debe realizarse en todas las etapas desde el análisis y el diseño hasta la programación y evaluación.
4. ¿Considera que la etapa mencionada es la mejor o considera que puede ser otra?
Considero que lo más conveniente es documentar todas las etapas del sistema en el momento en que se estén realizando.
5. ¿Usa alguna herramienta para facilitar la realización de la documentación? ¿Cuál y porqué?
Usamos Word, Visio y últimamente para algunas cosas Rational Rose.
6. ¿Qué ventajas y/o desventajas encuentra usted en la realización de la documentación?
Ventajas: Se registra lo que el cliente quiere que haga el sistema Ayuda a no olvidar los objetivos y alcances del sistema Es más fácil hacer cambios y de ser necesario regresar a la versión anterior sin tantas dificultades Contamos con un soporte en papel del sistema La documentación da confianza al desarrollador y al cliente **Desventajas:** Realizar la documentación es laborioso y tardado Si no se realiza de manera adecuada puede perjudicar en lugar de ayudar Es necesario trabajar en equipo todo el tiempo, lo cual a veces resulta difícil Es difícil mantenerlo actualizado
7. ¿Quiénes considera que se benefician con la documentación? ¿Porqué?
Nos beneficiamos todos, porque la documentación es para todos, para los analistas, desarrolladores, programadores, líderes de proyecto, usuarios finales, etc.; y todos pueden utilizarla de acuerdo a sus tareas.
8. ¿Cuáles considera que serían las consecuencias si no se realiza la documentación o se hace de manera confusa?
La peor consecuencia es que el sistema no sería completamente confiable; los errores serían más difíciles de encontrar y sería muy difícil mantener un seguimiento a las modificaciones que se le hicieran. Además el cliente tampoco tendría una guía para entender conocer al sistema.
9. ¿Porqué usa la tecnología orientada a objetos?
Porque simplifica las cosas, no en todos los casos claro, sólo en algunos.
10. ¿Desde cuándo usa el UML?

Desde hace un año aproximadamente, aunque no lo usamos de manera constante.
11. ¿Específicamente para qué usa al UML?
Para el análisis y diseño de sistemas, incluso sin importar que no sean orientados a objetos.
12. ¿Aplica todos los elementos de la notación UML que proponen sus creadores?
No, solamente aplicamos los elementos que consideramos de utilidad, algunos no los conocemos, se requiere más tiempo para conocerlos a todos y algunos de ellos no nos son de utilidad.
13. ¿Qué elementos aplica y porqué?
Aplicamos el diagrama de casos de uso, la definición de actores y de casos de uso, el flujo de eventos, el diagrama de colaboración. Los usamos porque facilitan el trabajo de documentación incluso para los sistemas que no son orientados a objetos.
14. ¿Considera que se han obtenido beneficios en el desarrollo de sistemas a partir del uso del UML?
Si una vez comprendida la metodología (al menos la parte de ella que utilizamos) resulta más rápido y sencillo.
15. ¿Cuál considera que es la razón por la que ha obtenido o no beneficios?
Se han obtenido beneficios porque la metodología no es tan rígida y se pueden combinar diferentes elementos de utilidad sin confundirlos .
16. ¿Ha sido complicado incorporar la notación UML y porqué?
Si ha sido complicado porque nuestra forma de pensar está muy acostumbrada a el análisis y diseño estructurados y es difícil deshacerse de este tipo de razonamiento de tal forma que no interfiera con la orientación a objetos.
17. ¿El uso del UML ha sido de utilidad para realizar la documentación de los sistemas?
Si
18. ¿De qué forma ha sido o no de utilidad?
Ha sido de utilidad porque los diagramas son muy expresivos y fáciles de usar.
19. ¿Considera que el uso del UML ha facilitado el análisis, diseño y documentación de los sistemas? ¿Porqué?
Si. Porque la herramienta ayuda a mantener la documentación actualizada, sin embargo no es útil

para realizar toda la documentación, solamente una parte de ella usando únicamente algunos elementos.

City Bank

1. ¿Qué importancia considera que tiene la documentación para el desarrollo de un sistema?
La documentación es tan importante como cualquier otro elemento del sistema.
2. ¿Se realiza la documentación cada vez que se desarrolla un sistema? ¿Porqué?
Si. Es parte fundamental de todo sistema, debemos saber qué estamos trabajando y cómo lo estamos haciendo.
3. ¿En qué etapa de desarrollo se realiza la documentación?
La documentación se realiza durante todo el desarrollo del sistema y al final se afinan detalles de presentación, pero la información se actualiza sobre la marcha.
4. ¿Considera que la etapa mencionada es la mejor o considera que puede ser otra?
De la forma que lo hemos hecho para nosotros ha funcionado bien.
5. ¿Usa alguna herramienta para facilitar la realización de la documentación? ¿Cuál y porqué?
No usamos ninguna herramienta en especial, solamente un procesador de textos y una herramienta para graficar.
6. ¿Qué ventajas y/o desventajas encuentra usted en la realización de la documentación?
Ventajas: La ventaja más importante es que logramos controlar el proceso de desarrollo del sistema **Desventajas:** La única desventaja es que el equipo no este bien organizado, pero en ese caso todo el desarrollo del sistema se encuentra en riesgo
7. ¿Quiénes considera que se benefician con la documentación? ¿Porqué?
Primero, los beneficiados son los propios desarrolladores porque se mantienen al tanto de lo que ellos mismos están haciendo y después por supuesto los usuarios finales porque conocen los alcances y limitaciones del sistema.

8. ¿Cuáles considera que serían las consecuencias si no se realiza la documentación o se hace de manera confusa?
La consecuencia es que se tiene un sistema incompleto, es como si tuviéramos un sistema sin interface de usuario. Es necesario entender que la documentación es parte del desarrollo de cualquier sistema.
9. ¿Porqué usa la tecnología orientada a objetos?
Porque en algunas ocasiones esta tecnología nos ha facilitado el desarrollo de los sistemas.
10. ¿Desde cuándo usa el UML?
Aproximadamente una año y medio.
11. ¿Específicamente para qué usa al UML?
Principalmente para tener una visión global del sistema en diagramas, así como para el análisis y diseño y finalmente para crear el código de forma automática con el uso de Rational Rose.
12. ¿Aplica todos los elementos de la notación UML que proponen sus creadores?
No, algunos de ellos no son de utilidad para nuestro desarrollo.
13. ¿Qué elementos aplica y porqué?
Casos de uso, diagramas de clases, diagramas de interacción, diagramas de paquete, diagramas de estado y diagramas de actividad; sin embargo no usamos todos los elementos propuestos en cada diagrama, únicamente usamos los más sencillos.
14. ¿Considera que se han obtenido beneficios en el desarrollo de sistemas a partir del uso del UML?
Si, el uso del UML ha facilitado algunas de nuestras tareas y las hemos sistematizado de una forma más simplificada.
15. ¿Cuál considera que es la razón por la que ha obtenido o no beneficios?
Yo creo que la principal razón es que hemos aprendido a aprovechar los elementos de utilidad de la herramienta.
16. ¿Ha sido complicado incorporar la notación UML y porqué?
Si fue un poco complicado al principio porque la herramienta tenía elementos que es su momento no era tan comunes, de hecho aún lo son, entonces tuvimos que aprender cosas nuevas y a re-aprender cosas olvidadas.

17. ¿El uso del UML ha sido de utilidad para realizar la documentación de los sistemas?
Si
18. ¿De qué forma ha sido o no de utilidad?
Ha sido de utilidad por su flexibilidad y sobre todo porque al hacer la actualización en un diagrama los demás se actualizan automáticamente.
19. ¿Considera que el uso del UML ha facilitado el análisis, diseño y documentación de los sistemas? ¿Porqué?
Si. Porque la herramienta fue diseñada para esas actividades entre otras y funciona de forma adecuada y satisfactoria.

ANEXO 2

Ing. y Mta. Graciela Bribiesca
Coordinadora de la Licenciatura en
Informática de la FCA
P r e s e n t e.

Por medio de la presente me permito proponer la inclusión de un tema en la materia **INFORMÁTICA VI** de la carrera de Lic. en Informática, plan 98 que se imparte en esta Facultad de Contaduría y Administración. Propongo la inclusión del tema porque me parece de gran importancia en base a los conocimientos que adquirí durante la realización de mi tesis.

El tema de mi tesis es la documentación del análisis y el diseño de sistemas orientados a objetos; en ésta propongo la utilización de una herramienta que se llama Unified Modeling Language UML (Lenguaje de Modelado Unificado), que es una herramienta de notación gráfica creada por los principales analistas y desarrolladores de sistemas orientados a objetos. Considero que el tema de mi trabajo de tesis se adapta a los objetivos de la materia y la enriquece.

La tesis se llama ***Documentación del análisis y diseño de sistemas orientados a objetos con UML***, que fue dirigida por el Dr. Ricardo Rivera Soler.

Adjunto la propuesta del temario en la página siguiente.

A t e n t a m e n t e
CD. Universitaria, D.F., a 27 de junio del 2000.

Jenny Jiménez Herrada
No. cta. 8700415-3

Propuesta de inclusión de tema en materia

Propongo incluir el tema del UML Unified Modeling Language (Lenguaje de Modelado Unificado) en la materia Informática VI del plan 98, que se cursa en el séptimo semestre. El temario es el siguiente:

PROGRAMA DE LA ASIGNATURA: *INFORMÁTICA VI*

	CLAVE:
PLAN: 98.	CRÉDITOS: 8
LICENCIATURA: INFORMÁTICA.	SEMESTRE: 7o
ÁREA: INFORMÁTICA.	HRS. CLASE: 2
REQUISITOS: INFORMÁTICA V.	HRS. POR SEMANA: 4

TIPO DE ASIGNATURA: OBLIGATORIA (X) OPTATIVA ( )

OBJETIVO GENERAL DEL CURSO:

AL FINALIZAR EL CURSO, EL ALUMNO SERÁ CAPAZ DE REALIZAR EL DISEÑO DE UN SISTEMA A PARTIR DE LOS RESULTADOS OBTENIDOS DURANTE LA ETAPA DE ANÁLISIS Y QUE SERVIRÁ COMO MODELO PARA LA IMPLANTACIÓN DEL MISMO.

TEMAS	HORAS SUGERIDAS
	8
	10
I. PLATAFORMA TEÓRICO-CONCEPTUAL.	10
II. FUNDAMENTOS DEL DISEÑO DE SISTEMAS.	10
III. METODOLOGÍAS DE DISEÑO DE SISTEMAS.	12
IV. DOCUMENTACIÓN DE SISTEMAS.	12
V. PRUEBA Y EVALUACIÓN DE SISTEMAS.	
VI. MANTENIMIENTO A SISTEMAS.	6
EVALUACIÓN	
TOTAL	68

Propongo que el tema se incluya en los temas **III. METODOLOGÍAS DE DISEÑO DE SISTEMAS** y **IV. DOCUMENTACIÓN DE SISTEMAS**. En el primero presentar al UML como síntesis de las metodologías del análisis y diseño de sistemas orientados a objetos; haciendo mención a sus creadores, la forma en que lo crearon y sus principales fundamentos. Y en el segundo, el UML como herramienta gráfica de apoyo para la documentación, incluyendo los diagramas que propongo en mi tesis por ser éstos, según mi opinión, los más sencillos y útiles.

BIBLIOGRAFÍA

En la bibliografía básica del programa de estudios 98, se incluye el libro *Diseño orientado a objetos, con aplicaciones* de Grady Booch. Este es el único libro cuyo tema principal es la orientación a objetos.

Para la inclusión de tema arriba mencionado, propongo incluir en la bibliografía el libro *UML Distilled, applying the standard object modeling language* de Martin Fowler, así como el sitio en internet **www.rational.com**, que es donde se encuentra la notación completa del UML.

ANEXO 3

Lic. Adrián Méndez
Coordinador de publicaciones
periódicas de la FCA
P r e s e n t e.

Por medio de la presente me permito presentarle un artículo que presenta los resultados de mi tesis para su consideración y de ser posible, para su publicación en la revista *Emprendedores*.
La tesis que realicé, para obtener el título de Lic. en Informática, ha sido dirigida por el Dr. Ricardo Rivera Soler y tiene como título ***Documentación del análisis y diseño orientados a objetos con UML***. El tema de mi tesis es la documentación de las etapas del análisis y diseño del desarrollo de sistemas orientados a objetos, con el uso de una herramienta que se llama Unified Modeling Language UML (Lenguaje de Modelado Unificado).
El objetivo de la publicación del artículo, es dar a conocer los resultados de mi trabajo de investigación y alentar a los que estén interesados a investigar más acerca del tema.

A t e n t a m e n t e
CD. Universitaria, D.F., a 27 de junio del 2000.

Jenny Jiménez Herrada
No. cta. 8700415-3

La documentación del análisis y el diseño de sistemas con UML

La documentación del análisis y el diseño de los sistemas que se desarrollan son tan importantes como su programación y su funcionamiento correcto. Al respecto varios autores coinciden y recomiendan realizar una documentación sencilla, clara y actualizada.

Entonces, ¿porqué a los desarrolladores no les gusta hacer la documentación de los sistemas? Según mi investigación la documentación es una tarea aburrida y tardada, que aunque sabemos que es necesaria y útil tratamos de evitar hasta que la necesitamos.

La diferencia entre tener un sistema bien documentado y no tener la documentación de un sistema o de tenerla incorrecta es muy grande. La documentación representa el soporte del sistema, con ella es posible saber la manera en que fue desarrollado y por lo tanto muestra las posibilidades de cambios y actualizaciones, así como la identificación de errores. Los desarrolladores entrevistados, el INE (Instituto Nacional de Ecología), la DCAA (Dirección de Cómputo para la Administración Académica), coinciden en la importancia que tiene la documentación y la realizan siempre que desarrollan un sistema.

El hecho de realizar la documentación de un sistema es ya complicado, pero si a esto le agregamos que el sistema sea orientado a objetos y además que se use una herramienta poco común creada por los principales desarrolladores de sistemas orientados a objetos, el hecho se complica aún más por las siguientes razones.

Primero, el paradigma de orientación a objetos es de difícil adopción debido a que tenemos ya una forma de pensar y de razonar los sistemas de forma estructurada; el cambio a un razonamiento orientado a objetos implica un cambio radical que no siempre logra adaptarse al tipo de sistema que se este desarrollando.

Segundo, la herramienta Unified Modeling Language UML (Lenguaje de Modelado Unificado) es muy grande y por eso de difícil estudio.

En mi trabajo de tesis propongo usar únicamente los elementos de UML que resulten útiles y que no compliquen más la tarea de documentar. Recomiendo algunos diagramas que considero de utilidad, algunos de los cuales ya se realizan en el análisis y diseño estructurado aunque de forma distinta.

Es muy importante tener la posibilidad de contar con una guía que nos permita tener una idea de cómo usar el UML para realizar una parte tan importante del desarrollo de sistemas como es la documentación del análisis y diseño orientados a objetos. En mi tesis hago una propuesta al respecto que considero que resulta de utilidad, tanto para aquellas personas que conocen el UML y lo quieren usar para el desarrollo de sus sistemas, como para aquellas personas que quieren conocer al UML.

ANEXO 4

Ing. y Mta. Graciela Bribiesca
Coordinadora de la Licenciatura en
Informática de la FCA
P r e s e n t e.

Por medio de la presente me permito poner a disposición de usted y de la Facultad de Contaduría y Administración los conocimientos que adquirí durante la realización de mi tesis para impartir un seminario o participar en alguna conferencia.

El tema de mi tesis es la documentación del análisis y el diseño de sistemas orientados a objetos; en ésta propongo la utilización de una herramienta que se llama Unified Modeling Language UML (Lenguaje de Modelado Unificado), que es una herramienta de notación gráfica creada por los principales analistas y desarrolladores de sistemas orientados a objetos. En mi trabajo resalto la importancia de la documentación y propongo la utilización de algunos diagramas de la herramienta mencionada anteriormente para facilitar dicha tarea.

La tesis se llama ***Documentación del análisis y diseño de sistemas orientados a objetos con UML***, que fue dirigida por el Dr. Ricardo Rivera Soler.

A t e n t a m e n t e
CD. Universitaria, D.F., a 27 de junio del 2000.

Jenny Jiménez Herrada
No. cta. 8700415-3

ANEXO 5

Sr. Carlos Ortega
Proveedor de Rational Rose en México
Abits S.A. de C.V.
P r e s e n t e.

Por medio de la presente me permito poner a disposición de usted el trabajo de tesis que realicé para obtener mi título de Lic. en Informática en la Facultad de Contaduría y Administración de la Universidad Nacional Autónoma de México.

El tema de mi tesis es la documentación del análisis y el diseño de sistemas orientados a objetos; en ésta propongo la utilización de la herramienta Unified Modeling Language UML (Lenguaje de Modelado Unificado), en base a la cual ustedes distribuyen la herramienta gráfica Rational Rose, misma que es utilizada por las personas que entrevisté, quienes manifiestan haber obtenido resultados satisfactorios.

La tesis se llama ***Documentación del análisis y diseño de sistemas orientados a objetos con UML,*** que fue dirigida por el Dr. Ricardo Rivera Soler. Pongo a su disposición también mi Curriculum Vitae.

A t e n t a m e n t e
CD. Universitaria, D.F., a 27 de junio del 2000.

Jenny Jiménez Herrada
No. cta. 8700415-3

ANEXO 6

Presente.

Por medio de la presente me permito poner a su disposición el trabajo de tesis que realice y en el cual su gran ayuda me fue tan valiosa.

El tema de mi tesis, como ya sabe, es la documentación del análisis y el diseño de sistemas orientados a objetos; mediante la utilización del Unified Modeling Language UML (Lenguaje de Modelado Unificado). En mi trabajo resalto la importancia de la documentación y propongo la utilización de algunos diagramas de la herramienta mencionada anteriormente para facilitar dicha tarea.

La tesis se llama ***Documentación del análisis y diseño de sistemas orientados a objetos con UML***, que fue dirigida por el Dr. Ricardo Rivera Soler.

Esperando que dicho trabajo le sea de utilidad, les agradezco profundamente de nuevo la ayuda que me proporcionaron para la realización de la misma.

Atentamente
CD. Universitaria, D.F., a 27 de junio del 2000.

Jenny Jiménez Herrada
No. cta. 8700415-3

ANEXO 7

Propuesta de Documentación

Documentación del análisis y diseño orientado a objetos con UML

¿Por qué análisis y diseño orientado a objetos con UML? y ¿por qué documentar dicho análisis y diseño?, finalmente lo más importante de los sistemas es la codificación del mismo.
El análisis y el diseño son la parte fundamental de todos los sistemas, si éstas actividades no se realizan de forma correcta lo más probable es que el sistema tenga muchas deficiencias. Antes de empezar a programar hay que saber qué es lo que se programará y de qué forma hacerlo; un análisis y un diseño deben proporcionar al programador una visión clara del funcionamiento del sistema y de su estructura, para garantizar que el programador obtenga la solución al problema planteado por el cliente.

Durante las fases de análisis y diseño, que es cuando se intenta proponer una solución, se realizan muchos cambios ya que tanto los analistas como los diseñadores buscan la mejor solución al problema de acuerdo a los requerimientos proporcionados por el cliente. Por lo tanto es muy importante que todos los miembros del equipo cuenten con la documentación del sistema actualizada.

Pero, ¿qué es la documentación? No es un documento con cientos de páginas que nadie lee y nadie entiende, tampoco es un documento *bonito* que parece una tesis; la documentación es un conjunto de documentos, no muchos, que de manera clara y sencilla representan la estructura del sistema y su funcionamiento. Para que los miembros del equipo puedan entender de manera clara y sencilla la documentación, primero es necesario que esté escrita con una notación y vocabulario que todos conozcan y entiendan (en este caso el UML); de nada sirve una documentación que únicamente entienden algunos miembros del equipo.

La idea de hacer una documentación es facilitarle el trabajo a todos, y no complicarlo más; la documentación que propongo está basada en diagramas. No es necesario perder mucho tiempo haciendo la documentación, ésta debe realizarse de forma paralela a las actividades de análisis y diseño, y no cuando estas actividades se han terminado y nadie recuerda cómo era la solución ni quiere hacer la tediosa tarea de documentar.

La documentación es un mal necesario que puede sacarnos de aprietos.

Lo más importante es identificar un método que se adecue al tipo de software que se pretende desarrollar. Existe una corriente muy fuerte en favor del uso de un método mínimo, de tal forma que se use únicamente lo indispensable para el desarrollo del sistema que en particular se esté desarrollando. Yo estoy de acuerdo con ello, son más valiosos algunos documentos que nos den la idea general del sistema que un conjunto de diagramas que a la larga lejos de facilitar el desarrollo lo entorpecen y dificultan.

El UML

El UML es sólo una notación estándar. Define un numero de diagramas que se pueden dibujar para describir un sistema, y lo que quieren decir estos diagramas. No describe el proceso que

se usa para construir el software. Una descripción del proceso como tal, o método, incluiría una lista de tareas que se deben llevar a cabo, el orden en el que se deben hacer, las entregas producidas, los requerimientos por cada tarea, etc. La metodología formal tradicional consiste de ambas, notación y un método.

Es importante destacar que el UML es bastante complicado, debido a que trata de proporcionar diagramas para todo tipo de situación en la que uno pueda encontrarse. Por lo tanto si apenas se inicia el desarrollo con el UML es recomendable ser selectivo y concentrarse en usar sólo la notación básica. En este trabajo de tesis no tomo en cuenta todos los tipos de diagramas que la última versión del UML ofrece, únicamente propongo algunos diagramas básicos, a partir de los cuales es posible realizar otros más elaborados; sin embargo sugiero que la documentación sea lo más sencilla y mínima posible, con la finalidad de no perder mucho tiempo elaborándola y entendiéndola.

El uso del UML, como de cualquier otro método, depende de lo que se está desarrollando y cuánta ceremonia se quiere hacer al proceso de software.

Análisis y Diseño

Una de las principales características del desarrollo de sistemas usando el UML es el desarrollo incremental o iterativo el cual, considero que es importante para análisis y diseño orientado a objetos.

Varias de las metodologías existentes para el desarrollo de sistemas orientados a objetos proponen una serie de pasos más o menos rígidos para el análisis y el diseño, lo cual a veces resulta de poca ayuda como guía de desarrollo por la naturaleza de la orientación a objetos. Los autores de dichas metodologías han propuesto una lista detallada de actividades que deben dar como resultado algunos productos (diagramas en su mayoría). El proceso de cada autor en detalle parece ser muy diferente, sin embargo hay una relación común; todos, o casi todos, proponen un desarrollo incremental e iterativo.

El desarrollo incremental ha sido muy criticado por algunos desarrolladores, ya que, argumentan, promueve el desarrollo sin control. En realidad el desarrollo incremental puede dar, en el caso de orientación a objetos, mucho más control que el desarrollo tradicional en cascada.

La esencia del desarrollo incremental es que el desarrollo se hace como una serie de incrementos. Cada incremento representa un ciclo completo del análisis y diseño; que se va enriqueciendo hasta llegar al sistema final. La idea principal del desarrollo incremental es tomar el caso de uso general, dividirlo en sus partes más pequeñas, realizar el análisis y el diseño para cada una de estas partes y obtener una parte terminada cada vez, hasta que se complete el diseño del caso de uso general. Cada vez que se ha completado una parte, ésta debe funcionar correctamente antes de continuar. Con lo anterior se logra tener capacidad de prueba desde el inicio del desarrollo del sistema, entonces, es más fácil de controlar un proyecto incremental que uno en cascada, ya que no hay forma de equivocarse con inestables *análisis y diseño completos*.

Recomendaciones para el desarrollo incremental:

- Identificar los elementos de mayor riesgo primero y aplicarles el desarrollo incremental
- Desarrollar los casos de uso de mayor prioridad lo más pronto posible
- Hacer un prototipo que le muestre al cliente las interfaces del sistema y su navegación

Al final de la iteración:

- Determinar que riesgos se han reducido o eliminado
- Determinar si se han descubierto nuevos riesgos
- Actualizar el plan de iteración para los riesgos faltantes

Por las características antes mencionadas propongo que el análisis y el diseño se realicen de manera conjunta, esto es, el desarrollo empieza desde su forma más básica creciendo continuamente y enriqueciéndose hasta alcanzar el diseño para la solución del problema.

El lenguaje de notación UML cuenta con una gran variedad de diagramas, algunos de ellos bastante complejos para representar el análisis y el diseño. En este trabajo de tesis, como mencione con anterioridad, voy a tomar en cuenta los diagramas y las técnicas básicos, debido a que la totalidad de la notación UML es bastante amplia.

Las técnicas y diagramas que a continuación se describen pueden realizarse en el orden en el que aparecen, sin embargo como seguramente la primera propuesta de diseño no será la mejor ni la única, se debe regresar continuamente a los diagramas previos para hacer los ajustes necesarios.

Los elementos básicos de una documentación del análisis y el diseño orientado a objetos usando el UML son los siguientes:

I. Un diagrama de casos de uso con interacciones claras
II. Un diagrama de interacción que muestre cómo colaboran los objetos
III. Un diagrama de clases y una breve descripción de cada clase
IV. Un diagrama de paquete que facilite el manejo de varias clases
V. Un diagrama de estados que muestre los diferentes estados de un objeto

A continuación se describen las técnicas y los diagramas básicos de la notación UML para realizar el análisis y el diseño orientado a objetos así como la descripción de la propuesta de la documentación que debe realizarse en cada caso.

I. Diagrama de casos de uso

Un caso de uso es una representación gráfica de la vista externa del sistema y de sus interacciones con el mundo exterior; es parecido al diagrama de contexto de los métodos tradicionales.

El principal objetivo de los diagramas de caso de uso es representar la comunicación entre la funcionalidad y comportamiento del sistema con el cliente o usuario final. Los diagramas de caso de uso están compuestos por:

- **Actores.** Un actor representa a cualquier cosa que interactua con el sistema, puede intercambiar información con el mismo o simplemente ser un receptor. Los actores son roles que pueden ser ejecutados por varias personas, u otros sistemas de computo. Una persona puede desempeñar varios roles y un rol puede tener a muchas personas ejecutándolo; para identificar a los actores no se debe pensar en las personas que ejecutan las operaciones, simplemente se debe pensar en roles.

- **Casos de uso.** Un caso de uso es una secuencia de acciones (un flujo de eventos completo) que el sistema desempeña y cuyo resultado observable desemboca en un actor particular. Un caso de uso es iniciado por un actor que invoca cierta funcionalidad en el sistema.

Lo más importante es identificar los objetivos del usuario, no las funciones del sistema. Para hacer lo anterior se deben considerar las tareas de los usuarios como casos de uso, y preguntarse cómo puede desempeñarlas un sistema de computadora.

Los casos de uso se representan por un óvalo y los actores por un dibujo que representa a una persona.

Algunas preguntas útiles para identificar a los casos de uso y a los usuarios son:

- ¿Cuál es el objetivo del usuario (las operaciones que quiere que haga el sistema)?
- ¿Falta algún objetivo (operación) o están todos los que el usuario desea?
- ¿Hay objetivos (operaciones) que estén mal representadas?
- ¿Se consideraron operaciones de administración y mantenimiento del sistema?
- ¿Qué actores son necesarios para cada caso de uso?
- ¿Quién proporciona, usa y borra información del sistema?
- ¿El sistema usa alguna fuente externa?
- ¿Un actor ejecuta varios roles diferentes? ¿Varios actores ejecutan el mismo rol?
- ¿El diagrama muestra una vista general del sistema?
- ¿Es posible hacer algunos cambios para simplificar el diagrama?

Recomendaciones:

- Ambos, los casos de uso y los actores deben tener nombres bien definidos, por ejemplo los casos de uso pueden tener nombres de operaciones como: Cálculo de nómina, Registro de participantes; los actores pueden tener nombres como: Usuario, Administrador.
- Es recomendable que los casos de uso no se relacionen entre sí, porque no pueden actuar por si mismos necesitan a un actor como intermediario, aunque hay algunas excepciones.
- Debe evitarse que los casos de uso sean ejecutados por más de un actor, ya que no se sabe qué funciones realiza cada uno de ellos.
- Es recomendable describir cada uno de los casos de uso y de los actores, aunque tengan nombres representativos, es necesario saber exactamente que función tiene cada uno de ellos en el sistema.

Relaciones entre casos

En las recomendaciones se mencionó que debe evitarse que dos casos de uso estén relacionados directamente, debe haber un intermediario, el actor; sin embargo la notación UML menciona dos tipos de relación entre casos de uso:

- **Relaciones de extensión.** La relaciones de extensión ocurre cuando se tiene un caso de uso que es similar a otro caso de uso pero el primero tiene más actividades que el segundo. Lo recomendable es construir un caso de uso de la forma más simple y poner el comportamiento poco común o diferente en otra parte. De esta forma se evita tener un caso de uso demasiado complicado que después sea difícil de manejar o bien tener varios casos de uso muy similares, que puedan llegar a confundirse en un momento dado. Cada vez que el caso de uso simple tenga

necesidad de una variación en su comportamiento, éste puede relacionarse con otro caso de uso que tenga el comportamiento poco común o las variaciones. En este caso el actor puede desempeñar a ambos casos de uso: al más simple y al que contiene la variación del comportamiento.

- **Relaciones de uso.** La relación de uso ocurre cuando se tiene comportamiento que es común a muchos casos de uso y no se quiere estar copiando esa descripción del comportamiento. Entonces se crea un caso de uso general que pueda llamarse cada vez que un caso de uso lo necesite. A diferencia de la relación de extensión, en la relación de uso el actor no puede desempeñar al caso de uso general, solamente puede desempeñar al caso de uso que llama al caso de uso general.

Reglas para el uso de relaciones de extensión y relaciones de uso:
- Usar relaciones de extensión cuando se está describiendo una variación de comportamiento normal.
- Usar relaciones de uso cuando se está repitiendo comportamiento en dos o más casos de uso separados y se quiere evitar la repetición.

Yo propongo hacer un diagrama de casos de uso general de todo el sistema, el cual por supuesto omitirá algunos detalles, después, de ser posible, hacer un diagrama de caso de uso detallado por cada caso de uso, a lo cual se le llama escenario.

Un escenario es una instancia de un caso de uso. Cada caso de uso tiene un conjunto de escenarios. Los escenarios se dividen en:
- Escenarios primarios: que muestran la forma en la que el sistema debe funcionar
- Escenarios secundarios: muestra las excepciones del escenario principal

Cada escenario debe tener una descripción escrita de forma sencilla y usando un vocabulario común.

La documentación propuesta del diagrama de casos de uso contiene:
- Una breve descripción del caso de uso general
- Una descripción de cada caso de uso
- Una descripción de cada actor
- Una descripción de cada escenario, es decir el detalle de cada caso de uso

La documentación debe estar escrita en un lenguaje común tanto para el equipo de desarrollo como para los clientes o usuarios del sistema.

II. Diagrama de interacción

Los diagramas de interacción son modelos que describen como colaboran grupos de objetos, es decir, son la representación gráfica de las interacciones de objetos.

Los diagramas de interacción pueden ser de dos formas, ambos proporcionan una vista diferente de la misma interacción:

- **Diagramas de secuencia:** muestran las interacciones de objetos a través del tiempo. El diagrama muestra a los objetos que participan en la interacción y la secuencia de los mensajes que se intercambian. Los objetos se dibujan en rectángulos con sus nombres subrayados. Un diagrama de secuencia contiene: objetos con sus líneas de vida, mensajes intercambiados entre objetos en orden secuencial y enfoque de control (opcional). El enfoque de control representa el tiempo relativo en el que el objeto esta direccionando mensajes. Se pueden agregar notas para proporcionar más información al diagrama.

- **Diagramas de colaboración:** muestran las interacciones de objetos enfocándose en el flujo de datos. Es una forma alterna de representar los mensajes intercambiados por un grupo o conjunto de objetos. Los diagramas de colaboración contienen: objetos, ligas entre objetos, mensajes intercambiados entre objetos y flujo de datos entre objetos si existe. Los diagramas de colaboración simplemente numeran los mensajes en secuencia. Además se puede usar una letra para mostrar ligas de concurrencia. Las ligas en los diagramas de colaboración se representan con una línea que conecta objetos indicando la ruta de comunicación de los mismos.
  Una liga de interacción en un diagrama de colaboración pude anotarse con:
  - Una flecha que apunta del objeto cliente al objeto proveedor
  - El nombre del mensaje con una lista opcional de parámetros y/o un valor de regreso con datos
  - Un numero de secuencia opcional que muestra el orden relativo con el que se envía el mensaje

Considero que lo más importante de los diagramas de interacción es su simplicidad, el UML propone representar los diagramas con un mayor grado de complejidad para que describan el comportamiento y lo definan; desde mi punto de vista los diagramas más simples (aunque no logren representar en su totalidad el comportamiento y su definición) son los más valiosos.

Los diagramas de interaccion deben usarse cuando se quiere ver el comportamiento de varios objetos en un solo caso de uso. Son útiles para mostrar las colaboraciones entre objetos.

Identificación de Clases

A partir del diagrama de casos de uso es posible identificar las clases que actúan en el sistema; para lo cual existe una técnica muy útil que aunque no es parte de la metodología del UML, es usada con frecuencia por su facilidad y los buenos resultados que proporciona.

Tarjetas CRC

A finales de 1980 en uno de los centros más grandes de tecnología de objetos, los laboratorios de investigación Tektronix, en Portland Oregon, se encontraban los principales usuarios de Smalltalk, y ahí se desarrollaron varias ideas de la tecnología orientada a objetos.

Ward Cunningham y Kent Beck, estaban involucrados en la enseñanza del conocimiento de Smalltalk. A partir de la cuestión de como enseñar objetos crearon la técnica de Tarjetas CRC (Class-Responsability-Collaboration).

En lugar de usar diagramas para desarrollar modelos, ellos representaban modelos en tarjetas de 4x6cm y en lugar de representar atributos y métodos representaban responsabilidades. Una responsabilidad es una descripción a alto nivel del propósito de la clase. La idea es capturar el propósito de la clase en algunas sentencias. Una responsabilidad es una descripción de alto nivel del propósito de la clase en algunas sentencias. La elección de una tarjeta es deliberado --no esta permitido escribir mas de lo que cabe en una tarjeta.

Las responsabilidades de la clase son:

- Conocimiento interno de la clase
- Servicios que proporciona la clase

Ejemplo :

Nombre de la clase

Responsabilidad	**Orden**	Colaboración
Revisar productos en almacén	Línea de Orden	
Determinar precio	Línea de Orden	
Revisar pago válido	Cliente	
Despachar a dirección entregada		

Las colaboraciones se refieren a otras clases que colaboran para completar la responsabilidad, sin embargo en un principio es posible omitir está parte. Lo más importantes es identificar de forma clara las responsabilidades.

Uno de los beneficios es que alienta a la discusión entre desarrolladores. Son efectivas cuando se revisa un diagrama de Casos de Uso para ver como van a funcionar las clases. Un error muy común es crear largas listas de responsabilidades de bajo nivel. Y este no es el objetivo. Las responsabilidades principales deben caber en una sola tarjeta, una tarjeta con más de tres responsabilidades es cuestionable.

Recomendaciones para el uso de tarjetas CRC:

- Usarlas en equipo, reunión de todo el equipo de desarrollo
- Elegir un grupo de personas para que tomen y ejecuten el rol de un escenario
- Asignar una tarjeta por cada objeto en el escenario
- Asignar a cada participante un conjunto de tarjetas, la persona se convierte en la *clase*

- Los participantes actúan como si fueran los escenarios definidos
- En cada tarjeta se anotan responsabilidades y colaboraciones
- Se crea una tarjeta para cada objeto descubierto

Algunas personas consideran que el uso de tarjetas CRC es maravilloso, otros piensan que es poco apropiado; lo más recomendable es hacer una prueba con el equipo de desarrollo y evaluar los resultados que se obtengan. Los resultados no son buenos cuando el equipo tiende a detallar demasiado el proceso desde el inicio o cuando el equipo no cuenta con definiciones claras para identificar clases.

Una de las ventajas del uso de tarjetas CRC es que es fácil identificar elementos que pueden convertirse en patrones de reuso, así como jerarquías de clases. Es muy importante, para la identificación de clases, contar con el diagrama del análisis de casos de uso para el sistema.

Una vez identificadas las clases es importante elegir nombres significativos para las mismas, ya que varios términos pueden referirse al mismo objeto; el lenguaje natural es muy ambiguo. Por lo tanto es necesario tratar de ser lo más claros y precisos posibles en la designación de nombres de clases.

Las clases se representan con rectángulos con tres niveles. El primer nivel tiene le nombre de la clase, el segundo muestra la estructura, es decir los atributos; y el tercero muestra el comportamiento, es decir las operaciones. Las ultimas dos secciones pueden omitirse si en el diagrama no es necesario mostrarlas.

III. Diagrama de clase

Un diagrama de clase describe los tipos de objetos en el sistema y los varios tipos de relaciones estáticas que existen entre ellos. Hay tres tipos principales de relaciones estáticas:

- Asociaciones: por ejemplo, un estudiante puede inscribirse en un numero de cursos.
- Subtipos: un alumno de universidad es un tipo de estudiante.
- Agregación: un disco duro es parte de una computadora.

Los diagramas de clase también muestran los atributos y operaciones de una clase y las restricciones que se aplican a la forma en la que los objetos se conectan.

El UML propone tres perspectivas para dibujar diagramas de clase:

- Conceptual: en este caso se dibuja un diagrama que represente los conceptos del dominio del problema. Estos conceptos se van a relacionar naturalmente a las clases para implementarlas, pero con frecuencia no es un mapeo directo. El modelo se dibuja con poco o nada que ver con el software que se va a implementar, y generalmente es independiente del lenguaje.

- Especificación: ahora si tomamos en cuenta al lenguaje de programación, pero estamos viendo las interfaces del software, no la implementacion, por lo tanto se debe tener cuidado de enfocarse en la interfaz del software y no su implementación, es decir el enfoque debe ser a los tipos y no a las clases. El desarrollo orientado a objetos pone mucho énfasis en la diferencia entre tipo y clase,

pero en la practica no se toma en cuenta. Es importante separar interfaz (tipo) e implementacion (clase). Los tipos representan una interfaz que puede tener cualquier implementacion debido al ambiente de desarrollo, caracteristicas de desempeño o vendedor. Las clases representan la implementación misma del sistema en el lenguaje de programación.

- Implementacion: en este caso en realidad tenemos clases y estamos representando la implementacion. Esta es quizá la perspectiva más usada, pero en muchos sentidos la perspectiva de especificación es mejor.

Entender la perspectiva es crucial para ambos el dibujo y la lectura de diagramas de clase. Cuando dibujamos un diagrama, debemos hacerlo a partir de una perspectiva única y clara, cuando leemos un diagrama debemos asegurarnos de conocer en que perspectiva dibujó el diseñador. Ese conocimiento es esencial si queremos interpretar el diagrama propiamente. Desafortunadamente las líneas entre las perspectivas no están muy bien definidas y la mayoría de los modeladores no son cuidadosos en la elección de la perspectiva que pretenden usar.

Asociaciones

Las asociaciones representan relaciones entre instancias de clases (un estudiante estudia en una universidad, una universidad tiene varios departamentos). La interpretación de las asociaciones varía con la perspectiva.

Desde la perspectiva conceptual las asociaciones representan relaciones conceptuales entre clases. El diagrama indica que una solicitud de una materia solo puede venir de un estudiante y que un estudiante puede hacer varias solicitudes de materias en el tiempo.

Desde la perspectiva de la especificación las asociaciones son responsabilidades. El diagrama implica que hay uno o más métodos asociados al estudiante que indicarán qué solicitudes de materias ha hecho el estudiante. De forma similar, hay métodos en las solicitudes de materias que indicarán qué estudiante hizo la solicitud y qué características tiene la misma. Estas responsabilidades no implican estructuras de datos. En un diagrama de especificación no es posible saber cuál es la estructura de datos de la clase, el diagrama solo indica la interfaz, nada más.

Desde la perspectiva de implementación sí sabemos que hay apuntadores en ambas direcciones entre las clases relacionadas. En este caso no podemos saber nada acerca de la interfaz. En un diagrama de implementación se indica que la solicitud de materia tiene un apuntador a un estudiante pero el estudiante no apunta a la solicitud. Por lo tanto es muy importante saber en qué perspectiva nos enfocamos para construir un modelo e interpretarlo correctamente.

Una asociación es una conexión semántica bi-direccional entre clases. Esto implica que hay una liga entre objetos en las clases asociadas. Las asociaciones se representan en diagramas de clase con una línea que conecta a las clases asociadas. Los datos pueden fluir en ambas direcciones a través de la liga.

Para clarificar su significado a veces se nombran las asociaciones. El nombre está representado como un nivel en la línea de asociación, con frecuencia es un verbo o frase verbal. En

algunos casos el nombre puede ser un rol, que denota el propósito o capacidad con la que una clase se asocia a otra clase. Los nombres de roles son con frecuencia sustantivos o frases de sustantivos. Un nombre de rol se escribe en la línea que asocia a las dos clases. Una asociación puede tener dos nombres de roles, uno por cada dirección, si la asociación es bi-direccional.

Agregación

La agregación se especifica a partir de la asociación en la que un todo está relacionado con su o sus partes. La agregación se conoce como una relaciónparte-de. Una agregación está representada como una asociación con un diamante al lado de la clase que denota al agregado (el todo).

¿Asociación o agregación?

- Si dos objetos están fuertemente limitados por una relación del todo a una parte. La relación es una agregación.
- Si dos objetos se consideran usualmente como independientes, aunque esten ligados. La relación es una asociación.

Atributos

En un diagrama conceptual el atributo de nombre de un estudiante indica que el estudiante tiene un nombre.

En el diagrama de especificación, este atributo indica que el objeto estudiante puede decir su nombre y tiene alguna forma de establecer su nombre.

En el diagrama de implementación, el alumno tiene un campo para su nombre.

La sintaxis del UML para los atributos es : visibility name : type = defaultValue

donde visibility es + (publico), #(protegido) o - (privado)

Operaciones

Las operaciones son los procesos que una clase sabe que puede ejecutar, corresponden a los métodos en las clases.

Desde la perspectiva conceptual las operaciones indican las principales responsabilidades de una clase.

Desde la perspectiva de especificación las operaciones corresponden a los métodos públicos en un tipo, no se muestran las operaciones que solamente manipulan atributos, ya que pueden inferirse; se muestran las operaciones que indican si un atributo es de solo lectura o **inmutable** (Que el valor nunca cambia).

Desde la perspectiva de implementación se muestran operaciones privadas y protegidas.

La sintaxis del UML para las operaciones es :

visibility name (parameter-list) : return-type-expression {property-string}

donde
visibility es + (publico), #(protegido) o (privado)
name es una cadena
parameter-list contiene (opcional) argumentos cuya sintaxis es la misma que para los atributos
return-type-expression es una especificacion dependiente del lenguaje de programación que vaya usarse y es opcional
property-string indica los valores de propiedad que se aplican a la operación dada.

Generalización

Es muy común hacer subtipos en los diagramas ER, para su uso en OO, ya que hay una correspondencia inmediata a la herencia en la programacion OO.

Un ejemplo típico de generalización es estudiantes de tiempo completo o de medio tiempo de una universidad. Tienen diferencias pero también muchas similitudes. Las similitudes pueden ponerse en una clase Estudiante general (el supertipo) con Estudiante de tiempo completo y de medio tiempo.

Desde la perspectiva conceptual, el estudiante de tiempo completo es un subtipo de estudiante, la idea es que todo lo que este en Estudiante (asociaciones, atributos, operaciones) esté también en el Estudiante de Tiempo Completo.
Desde el punto de vista de especificación, la interfaz del Estudiante (subtipo) de tiempo completo debe incluir todos los elementos de la interfaz de Estudiante (supertipo).

Desde el punto de vista de la implementación la herencia es la que se toma en cuenta en los lenguajes de programación. El estudiante de medio tiempo (subclase) hereda todos los métodos y campos de Estudiante (supercalse).

Uno de los problemas de la genaralización es que si queremos hacer cambios posteriores es muy inestable, por lo tanto, desde el principio debemos asegurarnos de que la generalización conceptual sea posible.

No hay que olvidar que la herencia y el hacer subclases en los lenguajes OO es un enfoque de implementacion. Se dice que la subclase hereda los datos y operaciones de una superclase. Esto tiene mucho en común con los subtipos, pero hay diferencias importantes. El hacer subclases es solo una forma de implementar los subtipos. Las subclases también pueden usarse sin los subtipos. Los lenguajes más recientes tratan de diferenciar lo que es la interfaz-herencia (subtipos) y la implementacion-herencia (subclases).

Es posible, por ejemplo, que el subtipo al representarse en la interfaz tenga la misma estructura que el supertipo, solo que realiza las operaciones en forma diferente, pero la estructura de

sus datos se ve igual, es decir tienen la misma interfaz. En cambio una subclase muestra una interfaz totalmente diferente que su superclase, esto es en la implementación.

Es importante tomar en cuenta las restricciones, por ejemplo un alumno de tiempo completo puede inscribirse a 8 materias por periodo, pero un alumno de medio tiempo solo puede inscribirse a 4 materias por periodo como máximo. Estas restricciones pueden mostrarse, de acuerdo con el UML con {}, o bien con texto tratando de ser lo más breve posible.

Operación

Cada clase tiene un conjunto de responsabilidades, éstas se representan por operaciones. Una operación es un servicio que puede ser requerido por un objeto o que puede provocar cierto comportamiento. Una operación debe desempeñar una función simple y cohesiva. Las operaciones deben nombrarse de acuerdo a la salida que dan y no a la serie de pasos que están atrás de la operación.

Las operaciones primitivas son aquellas que solo pueden implementarse con lo interno de la clase, todas las operaciones de una clase son típicamente primitivas.

Atributos

Un atributo es una definición de datos que contiene una clase, no tiene comportamiento, no es un objeto. Los nombres de atributos son sustantivos simples o frases de sustantivos, deben ser únicos por cada clase, cada atributo debe tener una definición clara y concisa.

Cada objeto tiene un valor para cada atributo definido en la clase a la que pertenece. Los atributos de cada clase son diferentes dependiendo de quien la llame.
Los atributos se muestran en el segundo nivel del rectángulo que representa a la clase.

Los atributos derivados son los atributos cuyos valores pueden calcularse con base en otros atributos. Se usan cuando no hay tiempo de recalcular el valor cada vez que se necesite.

Cada atributo tiene: el tipo de valor y un valor inicial opcional. Solo deben tomarse en cuenta los atributos relevantes al problema.

Los atributos y operaciones pueden mostrarse en cada clase, o pueden mostrarse en otro diagrama para evitar el acumulamiento de información.

Los diagramas de clase son la columna vertebral de casi todos los métodos OO. El problema es que son tan ricos que pueden resultar muy complejos en su uso. Algunas recomendaciones son:

- No tratar de usar todas las varias notaciones que se ofrecen. Iniciar con lo simple: clases, asociaciones, atributos y generalización. Introducir otras notaciones solo cuando sea necesario.
- Aclarar la perspectiva de dibujo de los modelos.
- No dibujar modelos para todo, concentrarse en las áreas clave. Es mejor tener algunos diagramas útiles y actualizados que varios modelos olvidados y obsoletos.

IV. Diagrama de paquete

Para comprender este tipo de diagramas primero debemos saber lo que es un paquete, un paquete es un mecanismo de propósito general para la organización de elementos en grupos. Los paquetes son útiles para los sistemas muy grandes; ya que resulta difícil de dibujar un modelo de clase para un sistema de grandes dimensiones, así como difícil de comprender, debido a las múltiples ligas entre clases que deben existir. Se debe dividir el diagrama en varias hojas, pero aunque para el diseñador esto resulta mas fácil de entender para el programador resulta difícil tomar en cuenta tantas ligas.

Los paquetes del UML están basados en las categorías de clases de Booch. En esta técnica las clases que se relacionan entre sí se encuentran en un paquete, si una clase desea usar otra clase en el mismo paquete, todo esta bien. Si una clase quiere usar una clase en un paquete diferente debe dibujarse una dependencia a ese paquete. Todo el dibujo del sistema es el dibujo de paquetes y sus dependencias, el objetivo es mantener las dependencias a un mínimo.
Esta técnica puede aplicarse a cualquier elemento y no solamente a las clases. Aunque en el UML se usan diagramas de paquetes para un diagrama que muestra paquetes de clases y las dependencias entre ellas. Estrictamente hablando, paquetes y dependencias son elementos en un diagrama de clase, entonces un diagrama de paquetes es solo una forma de diagrama de clase.

Dependencias

Una dependencia existe cuando entre dos elementos, en el que un cambio en la definición de un elemento puede causar cambios en el otro.

Las dependencias de clases pueden existir por varias razones:

- una clase envía un mensaje a otra
- una clase tiene a otra como parte de sus datos
- una clase menciona a otra como parámetro de una operación. Si la clase cambia su interfaz, entonces cualquier mensaje que envíe será inválido.

Una dependencia entre dos paquetes existe si existe alguna dependencia entre dos clases en los paquetes. Con los paquetes las dependencias no son transitivas. Un ejemplo de transitivos es si Carmen es mas joven que Rosa y Rosa es mas joven que Alma, entonces podemos deducir que Carmen es más joven que Alma.

Los paquetes están relacionados unos con otros usando relaciones de dependencia. Si una clase en un paquete *habla* con una clase en otro paquete entonces se agrega una relación de dependencia a nivel de paquete. Los diagramas de escenarios y diagramas de clases son evaluados para determinar relaciones de paquetes.

Con un paquete, las clases pueden ser publicas o privadas. Las clases publicas pueden ser vistas por paquetes que tienen visibilidad, los tipos privados solo pueden usarse por clases en el mismo paquete. Los paquetes pueden ser globales, en los que todas sus clases son visibles a otros paquetes.

V. Diagrama de estado

Los diagramas de transición de estado han sido usados desde el inicio del modelado orientado a objetos. Los diagramas de estado son una técnica familiar para describir el comportamiento de un sistema, debido a que ya se han usado en el diseño estructurado de sistemas. Describen todos los estados posibles que un objeto en particular puede tener y como cambian los estados de ese objeto como resultado de eventos que operan en el mismo. Proporcionan una definición del comportamiento formal y explícita. Sin embargo estos diagramas suponen que debemos definir todos los estados posibles del sistema, lo que es adecuado para sistemas pequeños, pero no para sistemas grandes; los diagramas de transición de estado serian tan complejos que no serian prácticos para su uso. Para combatir este problema, los métodos orientados a objetos definen diagramas de transición de estados para cada clase. Esto elimina bastante el problema ya que cada clase es lo suficientemente simple para tener un diagrama de transición de estado comprensible.

La variedad más popular de diagramas de transición de estados en los métodos orientados a objetos es el de Harel. Fue introducido por Rumbaugh, tomado por Booch y mas tarde adoptado por el UML. Este es una de las formas más poderosas y flexibles de un diagrama de transición de estados.

Un diagrama de transición de estados debe tener los siguientes elementos clave:

- Un inicio marcado por un círculo coloreado
- Acciones que estén asociadas con las transiciones y que se consideran procesos que ocurren rápidamente y no pueden interrumpirse
- Actividades que estén asociadas con estados y puedan llevarse mas tiempo, no suceden rápidamente, una actividad puede ser interrumpida por un evento
- Un guardia que es una condición lógica que regresara solamente cierto o falso. Solo se puede tomar una transición de un estado dado, por lo tanto lo que se pretende es que los guardias sean mutuamente excluyentes

Glosario

Glosario

Abstracción.	Acto o resultado de eliminar ciertas distinciones entre los objetos con el fin de ver sus aspectos comunes.
Actividad.	Proceso que implica una producción y consumo específicos. Su producto es una función de los productos que emplea. La actividad es un proceso continuo.
Asociación.	Medio de vincular tipos de objeto de manera significativa.
Atributo.	Asociación identificable que un objeto tiene con algún otro objeto o conjunto de objetos y está representada dentro de un tipo de objeto.
Clase.	Conjunto de objetos con atributos semejantes.
Clasificación.	Acto o resultado de determinar que un objeto es un tipo de objeto específico.
CRC.	Class Responsability Collaboration (Colaboración de Responsabilidad de Clases).
DCAA.	Dirección de Cómputo para la Administración Académica.
Diagrama.	Exhibición gráfica.
Escenario.	Especificación de comportamiento que expresa un proceso particular como una serie de eventos y operaciones.
Estado.	Colección de asociaciones que tiene un objeto.
Implementación.	La palabra no existe en el idioma español, es una palabra tomada del vocablo inglés *implementation*, nosotros la hemos adoptado en la jerga informática y la usamos para expresar la puesta en funcionamiento de un sistema una vez que este ha sido terminado.
Incremental.	Acción de crecimiento constante.
INE.	Instituto Nacional Electoral.
Interacción.	Actividad que existe entre un grupo de objetos.
Iteración.	Repetición de cosas análogas.
Método Metodología.	Especificación de los pasos necesarios para realizar una operación.

MIT	Massachusetts Institute of Technology (Instituto de Tecnología de Massachusetts).
Objeto.	Cualquier cosa a la que se le aplique un concepto.
OMG.	Object Modeling Group (Grupo de Modelado de Objetos).
OMT.	Object Modeling Technique (Técnica de Modelado de Objetos).
OOADA.	Object Oriented Analysis and Design with Applications (Análisis y Diseño Orientados a Objetos con Aplicaciones).
OOSE.	Object Oriented Software Engineering (Ingeniería de Software Orientada a Objetos).
Paradigma.	Forma específica en la que se realiza una acción o conjunto de acciones.
Proceso.	Secuencia ordenada de actos. Algo realizado para llegar a un resultado particular.
Sistema.	Dominio compuesto por aquellos objetos que forman un todo de acuerdo con cierto plan o propósito.
Subtipo.	Tipo de objeto especializado. Todas las propiedades que se aplican a un tipo de objeto también se aplican a sus subtipos. Sin embargo, un subtipo tiene características adicionales.
Supertipo.	Tipo de objeto generalizado con características más generales que sus subtipos. Todos los casos de un tipo de objeto lo son de su supertipo, pero no al contrario.
UML.	Unified Modeling Language (Lenguaje de Modelado Unificado).

Bibliografía

Bibliografía

- Booch Grady, *Análisis y Diseño Orientado a Objetos con Aplicaciones*, Addison Wesley/Dias de Santos, México, 1996
- Cameron Wills Alan, Francis D'Souza Desmond, *Objects, Components, and Frameworks with UML*, Addison-Wesley, Estados Unidos de America, 1999
- Fowler Martin, Scott Kendall, **UML** *Distilled Applying the Standard Object Modeling Language*, Estados Unidos de América, 1997
- Pooley Rob, Stevens Perdita, *Using UML Software Engineering*, Addison-Wesley, Gran Bretaña, 1999
- Runmabugh James, Blaha Michael, Premerlani William, Frederick Eddy, Lorensen William, *Object-Oriented Modeling and Design*, Prentice Hall, Estados Unidos de América, 1991

Internet

- http://www.rational.com/index.jtmpl