UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD CUAJIMALPA
Casa abierta al tiempo

ISBN: 978-607-477-442-9

Notas del curso: ANÁLISIS DE REQUERIMIENTOS • Dra. Maria del Carmen Gómez Fuentes

Notas del curso:

ANÁLISIS DE REQUERIMIENTOS

UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD CUAJIMALPA
Casa abierta al tiempo

Dra. Maria del Carmen Gómez Fuentes

Año 2011

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD CUAJIMALPA

DIVISIÓN DE CIENCIAS NATURALES E INGENIERÍA

MATERIAL DIDÁCTICO

NOTAS DEL CURSO
ANÁLISIS DE REQUERIMIENTOS

AUTOR:
María del Carmen Gómez Fuentes

Departamento de Matemáticas Aplicadas y Sistemas

IBSN: 978-607-477-442-9

2011

Editor
María del Carmen Gómez Fuentes

Departamento de Matemáticas Aplicadas y Sistemas.
División de Ciencias Naturales e Ingeniería
Universidad Autónoma Metropolitana, Unidad Cuajimalpa

Editada por:
UNIVERSIDAD AUTONOMA METROPOLITANA
Prolongación Canal de Miramontes 3855,
Quinto Piso, Col. Ex Hacienda de San Juan de Dios,
Del. Tlalpan, C.P. 14787, México D.F.

NOTAS DEL CURSO: ANALISIS DE REQUERIMIENTOS

Primera edición 2011

ISBN: 978-607-477-442-9

Impreso en México
Impreso por Publidisa Mexicana S. A. de C.V.
Calz. Chabacano No. 69, Planta Alta
Col. Asturias C.P.

Objetivos.

1.- Conocer y ubicar la importancia de la definición formal de los requerimientos.
2.- Conocer los paradigmas(los métodos y los modelos) existentes para el análisis de los requerimientos.
3.- Reconocer la estrecha relación existente entre el nivel de definición de los requerimientos y los modelos de ciclo de vida.
4.- Definir y analizar los requerimientos de un proyecto de software.

Contenido.

Capítulo I: Introducción.

I.1.- Definición de *Requerimientos* y de *Análisis de Requerimientos*.
I.1.1- Tipos de requerimientos.
I.1.2- Características de los requerimientos.
1.2.- Métodos generales de entrevistas.

Capítulo II: Procesos de la ingeniería de requerimientos.
II.1.- Estudios de viabilidad.
II.2.- Obtención y análisis de requerimientos.
II.3.- Especificación de requerimientos.
II.4.- Validación de requerimientos.
II.5.- Gestión de requerimientos. (Manejo de los cambios de requerimientos durante la construcción).
II.6.- Principales riesgos de la etapa de recolección de requerimientos.

Capítulo III: Especificación de requerimientos.
III.1.- Introducción.
III.2.- Principios de Especificación.
III.3.- Requerimientos funcionales y no funcionales.
III.4.- El dominio de la información.
III.5.- La documentación
III.5.1.- Recomendación para la Especificación de Requerimientos de Software de la IEEE.
III.5.2.- Estructura de una Especificación de requerimientos.

Capítulo IV: Relaciones entre administración de requerimientos y modelos de ciclos de vida.

IV.1.- El modelo en cascada.
IV.2.- Modelos evolutivos.
IV.3.- El modelo de componentes reutilizables.
IV.4.- El Proceso Unificado.

Capítulo V: Artefactos de modelado para el Desarrollo Estructurado de Sistemas

V.1.- Las principales metodologías estructuradas para el desarrollo de software.
V.2.- Diagramas de Flujo de Datos (DFD).
V.3.- Diccionario de Datos (DD).
V.4.- Diagramas Entidad-Relación (DER).
V.5.- Diagramas de Transición de Estados (DTE).
V.6.- Balanceo de Modelos.

Capítulo VI: Artefactos de modelado para el Desarrollo Orientado a Objetos.

VI.1.- Metodologías orientadas a objetos para el desarrollo de software.
VI.2.- El lenguaje UML.
VI.2.1.- Diagramas de clases.
VI.2.2.- Diagramas de casos de uso.
VI.2.3.- Diagramas de secuencia
VI.3.- Las herramientas CASE.

Capítulo VII: Métodos de comunicación.

VII.1.- Desarrollo Conjunto de Aplicaciones (JAD).
VII.2.- Prototipos.
VII.2.1.- Prototipo de la Interfaz de Usuario.

Apéndice A: Metodología propuesta para el laboratorio de la UEA "Análisis de Requerimientos".
Apéndice B: "Especificación de Requerimientos para un juego de ajedrez"
Apéndice C: "Especificación de Requerimientos para un Sistema de Información Geográfica"
Apéndice D: "Especificación de Requerimientos para un Visualizador Molecular."

Capítulo I: Introducción.

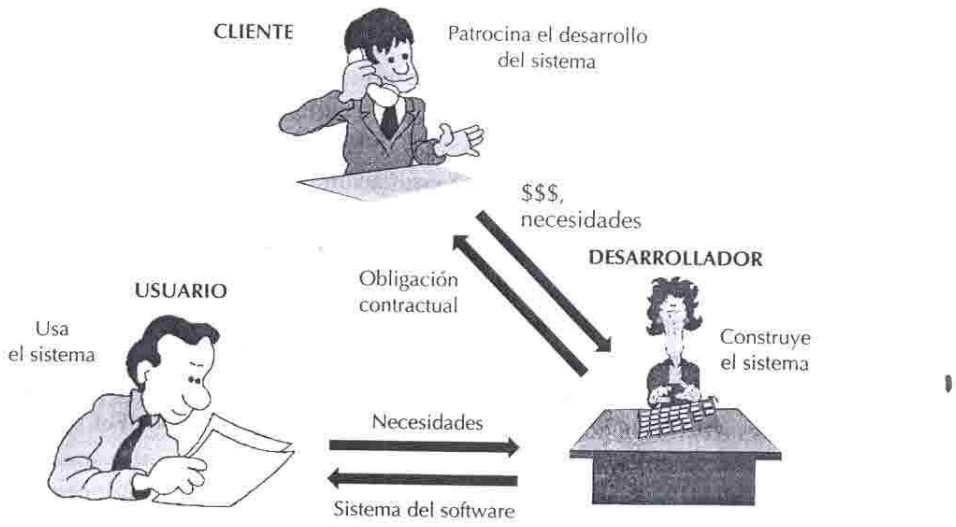


Figura 1.1 Participantes en el desarrollo de software (Pleeger 2002)

I.1.- Definición de *Requerimientos* y de *Análisis de Requerimientos*.

Requerimientos: Los requerimientos especifican qué es lo que el sistema debe hacer (sus funciones) y sus propiedades esenciales y deseables. La captura de los requerimientos tiene como objetivo principal la comprensión de lo que los clientes y los usuarios esperan que haga el sistema. Un requerimiento expresa el propósito del sistema sin considerar como se va a implantar. En otras palabras, los requerimientos identifican el **qué** del sistema, mientras que el diseño establece el **cómo** del sistema.

La captura y el análisis de los requerimientos del sistema es una de las fases más importantes para que el proyecto tenga éxito. Como regla de modo empírico, el costo de reparar un error se incrementa en un factor de diez de una fase de desarrollo a la siguiente, por lo tanto la preparación de una especificación adecuada de requerimientos reduce los costos y el riesgo general asociado con el desarrollo [Norris & Rigby, 1994].

Análisis de requerimientos: Es el conjunto de técnicas y procedimientos que nos permiten conocer los elementos necesarios para definir un proyecto de software. Es una tarea de ingeniería del software que permite especificar las características operacionales del software, indicar la interfaz del software con otros elementos del sistema y establecer las restricciones que debe cumplir el software.

La especificación de requerimientos suministra al técnico y al cliente, los medios para valorar el cumplimiento de resultados, procedimientos y datos, una vez que se haya construido.
La tarea de análisis de los requerimientos es un proceso de descubrimiento y refinamiento, el cliente y el desarrollador tienen un papel activo en la ingeniería de requerimientos de software. El cliente intenta plantear un sistema que en muchas ocasiones es confuso para él, sin embargo, es necesario que describa los datos, que especifique las funciones y el comportamiento del sistema que desea. El objetivo es que el desarrollador actúe como un negociador, un interrogador, un consultor, o sea, como persona que consulta y propone para resolver las necesidades del cliente.

El análisis de requerimientos proporciona una vía para que los clientes y lo desarrolladores lleguen a un acuerdo sobre lo que debe hacer el sistema. La especificación, producto de este análisis proporciona las pautas a seguir a los diseñadores del sistema.

"La carencia de buenos requisitos ha sido la causa del fracaso de proyectos con presupuestos de millones de dólares, ha impedido el desarrollo productivo, y ha sido el mayor contribuyente de los costes elevados del mantenimiento del software" (Dr. Raymond Yeh in the forward to System and Software Requirements Engineering, IEEE Computer Society Press Tutorial, Editors, M. Dorfman, and R.H Thayer, 1990)

I.1.2.- Tipos de requerimientos.

Según el estándar internacional de Especificación de Requerimientos IEEE830, los documentos de definición y especificación de requerimientos deben contemplar los siguientes aspectos resumidos por [Pfleeger, 2002] como se indica a continuación:

Ambiente físico

- ¿Dónde esta el equipo que el sistema necesita para funcionar?
- ¿Existe una localización o varias?
- ¿Hay restricciones ambientales como temperatura, humedad o interferencia magnética?

Interfaces

- ¿La entrada proviene de uno o más sistemas?
- ¿La salida va a uno o más sistemas?
- ¿Existe una manera preestablecida en que deben formatearse los datos?

Usuarios y factores humanos

- ¿Quien usará el sistema?
- ¿Habrá varios tipos de usuario?
- ¿Cuál es el nivel de habilidad de cada tipo de usuario?
- ¿Qué clase de entrenamiento requerirá cada tipo de usuario?
- ¿Cuán fácil le será al usuario comprender y utilizar el sistema?
- ¿Cuán difícil le resultará al usuario hacer uso indebido del sistema?

Funcionalidad

- ¿Qué hará el sistema?
- ¿Cuándo lo hará?
- ¿Existen varios modos de operación?
- ¿Cómo y cuando puede cambiarse o mejorarse un sistema?
- ¿Existen restricciones de la velocidad de ejecución, tiempo de respuesta o rendimiento?

Documentación

- ¿Cuánta documentación se requiere?
- ¿Debe estar en línea, en papel o en ambos?
- ¿A que audiencia está orientado cada tipo de información?

Datos

- ¿Cuál será el formato de los datos, tanto para la entrada como para la salida?
- ¿Cuán a menudo serán recibidos o enviados?
- ¿Cuán exactos deben ser?
- ¿Con qué grado de precisión deben hacerse los cálculos?
- ¿Cuántos datos fluyen a través del sistema?
- ¿Debe retenerse algún dato por algún período de tiempo?

Recursos

- ¿Qué recursos materiales, personales o de otro tipo se requieren para construir, utilizar y mantener el sistema?
- ¿Qué habilidades deben tener los desarrolladores?
- ¿Cuánto espacio físico será ocupado por el sistema?
- ¿Cuáles son los requerimientos de energía, calefacción o acondicionamiento de aire?
- ¿Existe un cronograma prescrito para el desarrollo?
- ¿Existe un límite sobre la cantidad de dinero a gastar en el desarrollo o en hardware y software?

Seguridad

- ¿Debe controlarse el acceso al sistema o a la información?
- ¿Cómo se podrán aislar los datos de un usuario de los de otros?
- ¿Cómo podrán aislarse los programas de usuario de los otros programas y del sistema operativo?
- ¿Con qué frecuencia deben hacerse copias de respaldo?
- ¿Las copias de respaldo deben almacenarse en un lugar diferente?
- ¿Deben tomarse precauciones contra el fuego, el daño provocado por agua o el robo?

Aseguramiento de la calidad

- ¿Cuáles son los requerimientos para la confiabilidad, disponibilidad, facilidad de mantenimiento, seguridad y demás atributos de calidad?
- ¿Cómo deben demostrarse las características del sistema a terceros?
- ¿El sistema debe detectar y aislar defectos?
- ¿Cuál es el promedio de tiempo prescrito entre fallas?
-¿Existe un tiempo máximo permitido para la recuperación del sistema después de una falla?
- ¿El mantenimiento corregirá los errores, o incluirá también el mejoramiento del sistema?
- ¿Qué medidas de eficiencia se aplicarán al uso de recursos y al tiempo de respuesta?
- ¿Cuán fácil debe ser mover el sistema de una ubicación a otra o de un tipo de computadora a otro?

I.1.3.- Características de los requerimientos.

Los requerimientos permiten que los desarrolladores expliquen cómo han entendido lo que el cliente pretende del sistema. También, indican a los diseñadores qué funcionalidad y que características va a tener el sistema resultante. Y además, indican al equipo de pruebas qué demostraciones llevar a cabo para convencer al cliente de que el sistema que se le entrega es lo que solicitó. Las características de los requerimientos mencionados en el estándar IEEE830 los explica [Pfleeger, 2002] como sigue:

Deben ser correctos.

Tanto el cliente como el desarrollador deben revisarlos para asegurar que no tienen errores.

Deben ser consistentes.

Dos requerimientos son inconsistentes cuando es imposible satisfacerlos simultáneamente.

Deben estar completos.

El conjunto de requerimientos está completo si todos los estados posibles, cambios de estado, entradas, productos y restricciones están descritos en alguno de los requerimientos.

Deben ser realistas.

Todos los requerimientos deben ser revisados para asegurar que son posibles.

¿Cada requerimiento describe algo que es necesario para el cliente?

Los requerimientos deben ser revisados para conservar sólo aquellos que inciden directamente en la resolución del problema del cliente.

Deben ser verificables.
Se deben poder preparar pruebas que demuestren que se han cumplido los requerimientos.

Deben ser rastreables.
¿Se puede rastrear cada función del sistema hasta el conjunto de requerimientos que la establece?

I.2.- Métodos generales de entrevistas.

La entrevista es una forma de recoger información de otra persona a través de una comunicación interpersonal que se lleva a cabo por medio de una conversación estructurada, [Braude, 2003] distingue las siguientes fases:

- *Preparación:* El entrevistador debe documentarse e investigar la situación de la organización, analizando los documentos de la empresa disponible. Hay que intentar minimizar el número de entrevistados, hay que considerar las entrevistas de cortesía, analizar el perfil de los entrevistados, definir el objetivo y el contenido de la entrevista, planificar el lugar y la hora en la que se va a desarrollar la entrevista es conveniente realizarla en un lugar confortable. Algunos proponen enviar previamente el entrevistado un cuestionario y un pequeño documento de introducción al proyecto de desarrollo.

- *Realización:* Hay tres fases:
  Apertura: Presentarse e informar al entrevistado sobre la razón de la entrevista.
  Desarrollo: Cumplir las reglas del protocolo, hay que llegar a un acuerdo sobre como se va a registrar la información obtenida.
  Terminación: Se termina recapitulando la entrevista agradeciendo el esfuerzo y dejando abierta la posibilidad de volver a contactar para aclarar conceptos o bien citándole para otra entrevista.

- *Análisis:* Consiste en leer las notas, pasarlas en limpio, reorganizar la información, contrastarlas con otras entrevistas o fuentes de información, evaluar como ha ido la entrevista.

Las entrevistas con los involucrados con el sistema son parte de la mayoría de los procesos de la ingeniería de requerimientos. En estas entrevistas, el equipo de la ingeniería de requerimientos hace preguntas sobre el sistema que utilizan y sobre el sistema a desarrollar. Los requerimientos provienen de las respuestas a estas preguntas.
Las entrevistas pueden ser de dos tipos:

1. *Entrevistas cerradas*: donde los entrevistados responden a un conjunto predefinido de preguntas.

2. *Entrevistas abiertas*: donde no hay un programa predefinido. El equipo de la ingeniería de requerimientos examina una serie de cuestiones con los involucrados con el sistema y, por lo tanto, desarrolla una mejor comprensión de sus necesidades.

En la práctica, las entrevistas son una mezcla de estos dos tipos. Las respuestas a algunas preguntas pueden conducir a otras cuestiones que se discuten de una forma menos estructurada. Las discusiones completamente abiertas rara vez salen bien; la mayoría de las entrevistas requieren algunas preguntas para empezar y para mantener la entrevista centrada en el sistema a desarrollar.

Las entrevistas sirven para obtener una comprensión general de lo que hacen los futuros usuarios del sistema, cómo podrían interactuar con el sistema y las dificultades a las que se enfrentan con los sistemas actuales. A la gente le gusta hablar de su trabajo y normalmente se alegran de verse implicados en las entrevistas. Sin embargo, no son de tanta utilidad para la comprensión de los requerimientos del dominio de la aplicación, tampoco son una técnica eficaz para obtener conocimiento sobre los requerimientos y restricciones organizacionales debido a que existen sutiles poderes e influencias entre los involucrados en el sistema. En general, la mayoría de la gente es reacia a discutir cuestiones políticas y organizacionales que pueden influir en los requerimientos. Por otra parte, hay que destacar que para la mayoría de las personas, la entrevista es un compromiso adicional sobre su cargada lista de trabajos pendientes. Algunos autores proponen mandar previamente un cuestionario que debe llenar el entrevistado y un pequeño documento de introducción al proyecto de desarrollo. El cuestionario permite que el entrevistado conozca los temas que se van a tratar y pueda conseguir con anticipación información que no tenga a disposición inmediata.

Los buenos entrevistadores poseen dos características importantes:

1.- No tienen prejuicios, evitan ideas preconcebidas sobre los requerimientos y están dispuestos a escuchar a los entrevistados. Si el entrevistado propone requerimientos sorprendentes, están dispuestos a cambiar su opinión del sistema.

2.- Ayudan al entrevistado a empezar las discusiones con una pregunta, una propuesta de requerimientos o sugiriendo trabajar juntos en un prototipo del sistema. Preguntar al cliente por lo que quiere de manera general normalmente no proporciona información útil. Para la mayoría de la gente es mucho más fácil hablar de algo en particular que en términos generales.

Las entrevistas, son una técnica general para obtener información. Se pueden complementar las entrevistas individuales con entrevistas en grupo o grupos de discusión. Las ventajas de utilizar grupos de discusión es que los usuarios se estimulan entre sí para proporcionar información y pueden terminar discutiendo diferentes formas que han desarrollado para utilizar los sistemas. más adelante, en el capítulo VII, se expone el "Desarrollo de Conjunto de Aplicaciones" (JAD).

La información de las entrevistas complementa otras informaciones sobre el sistema además de los documentos, observaciones de los usuarios, etcétera. Las entrevistas tienden a omitir información esencial, por lo que deberían ser usadas junto con otras técnicas de obtención de requerimientos.

Recomendaciones generales.

Es común que haya varios interesados en dar su opinión, lo primero es decidir a quién entrevistar. En lugar de intentar que se dedique el mismo tiempo a todos, lo cual puede resultar en requerimientos contradictorios y esfuerzo desperdiciado, [Braude, 2003] recomienda seleccionar uno o quizá dos individuos principales, entrevistarlos y después solicitar comentarios de otros interesados clave. Es preferible que haya dos entrevistadores en cada sesión, pues un entrevistador típico tiende a perder puntos. Grabar la entrevista suele ayudar, pero debe pedirse permiso de antemano.

El objetivo principal es minimizar el número necesario de personas a entrevistar para obtener una visión lo más completa posible sobre el sistema a desarrollar, sin embargo, hay que considerar también las entrevistas de "cortesía", por ejemplo, al jefe de la unidad que se analiza o de quien depende el sistema, quien aportará una visión estratégica que podría ser de interés, pero sobre todo, de quien se persigue obtener el permiso y el apoyo para poder entrevistar al resto del personal. Hay que tener en cuenta que se eliminan muchas dificultades para entrevistar a los empleados si su jefe avala la iniciativa [Piattini et al., 2004].

Una manera de manejar las entrevistas:

Antes de la entrevista.

1. Enumerar y dar prioridad a los clientes que se entrevistarán.
2. Programar una entrevista con tiempos de inicio y terminación fijos.

En la entrevista

1. No ser pasivo, investigar y animar, persistir en entender *deseos* y explorar *necesidades*.
2. Examinar casos de uso, flujos de datos y/o diagramas de estado.
3. Tomar notas exhaustivas.
4. Programar una reunión de seguimiento.

Después de la entrevista.

1. Bosquejar la especificación de los requerimientos.

Enviar correos electrónicos a los clientes para obtener sus comentarios.

EL ENTREVISTADO PUEDE PRESENTAR:

- ☹ PASIVIDAD, INHIBICION
- ☹ NO ACEPTACION
- ☹ RECHAZO
- ☹ AGRESIVIDAD

EL ENTREVISTADOR DEBE POSEER:

- ☺ TRATO CORDIAL
- ☺ CONOCIMIENTO DE TECNICAS DE COMUNICACIÓN
- ☺ ACTITUD PARA ESCUCHAR SIN PREJUICIOS
- ☺ EXPERIENCIA PRÁCTICA
- ☺ INTERÉS POR EL TEMA

No basta con hacer preguntas, es importante la forma en que se plantea la conversación y la relación que se establece.

II.3.- Importancia de la definición formal de requerimientos.

El análisis y especificación de requerimientos puede parecer una tarea relativamente sencilla, pero las apariencias engañan. Puesto que el contenido de comunicación es muy alto, abundan los cambios por mala interpretación o falta de información. El dilema con el que se enfrenta un ingeniero de software puede ser comprendido repitiendo la sentencia de un cliente anónimo: "Sé que crees que comprendes lo que piensas que he dicho, pero no estoy seguro de que entendiste lo que yo quise decir". En la *tabla 1.1* [Pfleeger, 2002] ilustra el conflicto que encontró (Scharer, 1990) cuando los desarrolladores y los usuarios se limitan a ver el problema desde su particular punto de vista sin tomar en cuenta la situación del otro.

Como ven los desarrolladores a los usuarios.	**Como ven los usuarios a los desarrolladores.**
Los usuarios no saben lo que quieren	Los desarrolladores no comprenden las necesidades operacionales.
Los usuarios no pueden articular lo que quieren.	Los desarrolladores ponen demasiado énfasis en la técnica.
Los usuarios tienen muchas necesidades motivadas políticamente.	Los desarrolladores pretenden decirnos como hacer nuestro trabajo.
Los usuarios lo quieren todo bien y ahora	Los desarrolladores no pueden traducir nuestras necesidades claramente establecidas a un sistema exitoso.

Los usuarios son incapaces de priorizar sus necesidades.	Los desarrolladores dicen “no” todo el tiempo.
Los usuarios rehúsan tomar responsabilidades por el sistema	Los desarrolladores siempre están por encima del presupuesto
Los usuarios son incapaces de proporcionar un enunciado utilizable de las necesidades.	Los desarrolladores siempre están atrasados.
Los usuarios no están comprometidos con los proyectos de desarrollo de sistemas.	Los desarrolladores piden a los usuarios tiempo y esfuerzo, aún en detrimento de sus obligaciones primarias importantes.
Los usuarios no tienen voluntad de colaborar.	Los desarrolladores establecen estándares no realistas para la definición de los requerimientos.
Los usuarios no pueden mantener el cronograma.	Los desarrolladores son incapaces de responder rápidamente a los legítimos cambios de las necesidades.

Tabla 1.1: Usuarios y Desarrolladores: como se ven el uno al otro (Scharer 1990).

En realidad existen muchos contribuyentes al conjunto de los requerimientos. Cada uno tiene una visión particular del sistema de cómo debe funcionar, a menudo estas visiones son conflictivas. Una de las muchas habilidades de un analista de requerimientos es la capacidad para comprender cada punto de vista y capturar los requerimientos de una manera que refleje los intereses de cada participante.

Capítulo II: Procesos de la Ingeniería de Requerimientos.

En cursos previos de Ingeniería de Software se aprende que existen diferentes modelos para desarrollar sistemas de software (cascada, evolutivos, etc.) y la obtención de requerimientos se ve como un subproceso de este desarrollo. Sin embargo, visto por separado, el Análisis de Requerimientos es todo un proceso al cual [Sommerville, 2005] llama "Ingeniería de Requerimientos" cuya meta es crear y mantener un documento de requerimientos del sistema. Este proceso general consta de cuatro subprocesos:

- El estudio de viabilidad, que evalúa si el sistema es útil para el negocio.
- Obtención y análisis de requerimientos.
- Especificación de requerimientos: transformación de los requerimientos en formularios estándar.
- Validación: verificar que los requerimientos realmente definen el sistema que quiere el cliente.

En la *figura 2.1* se muestra el proceso de ingeniería de requerimientos.

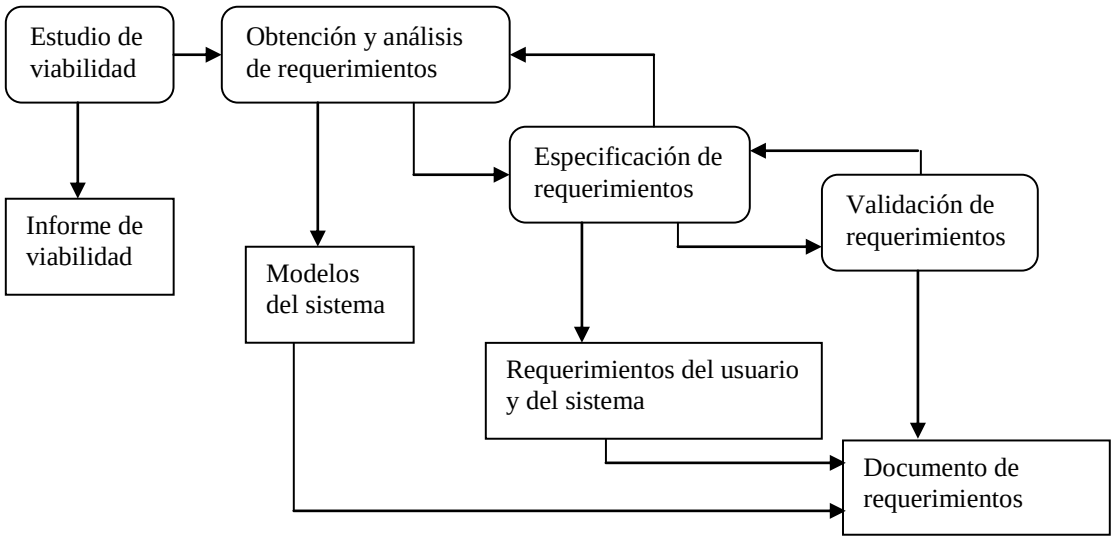


Figura 2.1: Proceso de ingeniería de requerimientos (Sommerville 2005).

II.1.- Estudios de viabilidad.

[Sommerville, 2005] define el estudio de viabilidad como un estudio corto y orientado a resolver las siguientes preguntas:

1.- ¿El sistema contribuye a los objetivos generales de la organización o empresa?
2.- ¿El sistema se puede implantar utilizando tecnología actual dentro de las restricciones de tiempo y presupuesto?
3.- ¿El sistema puede integrarse a otros sistemas existentes en la empresa?

Para ayudar a responder las preguntas del estudio de viabilidad, se tienen algunos ejemplos de preguntas posibles:

- ¿Cómo se las arreglaría la organización o empresa si no se implantara el sistema?
- ¿Cuáles son los problemas con los procesos actuales y como ayudaría un sistema nuevo a aliviarlos?
- ¿Cuál es la contribución directa que hará el sistema a los objetivos y requerimientos del negocio?
- ¿Se puede obtener y transferir la información a otros sistemas de la organización?
- ¿El sistema requiere tecnología que no se ha utilizado previamente en la organización?
- ¿A que debe ayudar el sistema y a qué no necesita ayudar?

El estudio de viabilidad no debe requerir más de dos o tres semanas. El resultado de este estudio es un informe que recomiende si vale o no la pena seguir con la ingeniería de requerimientos y el proceso de desarrollo del sistema. En el informe se pueden proponer cambios en el alcance, el presupuesto o sugerir requerimientos adicionales de alto nivel.

II.2.- Obtención y análisis de requerimientos.

La siguiente etapa del proceso de ingeniería de requerimientos es la obtención y análisis de requerimientos. En esta actividad, los ingenieros de software trabajan con los clientes y los usuarios finales del sistema para determinar el dominio de la aplicación, qué servicios debe proporcionar el sistema, el rendimiento requerido del sistema, las restricciones hardware, etcétera.

[Sommerville, 2005] presenta el modelo de la *figura 2.2* de (Robertson y Robertson, 1999) para mostrar que los requerimientos pueden extraerse de muchas maneras, sugiere ser creativos en la forma de averiguar qué es lo que los clientes quieren, y propone:

- Revisar la situación actual.

- Trabajar en el ámbito del usuario para comprender el contexto, los problemas y las relaciones.
- Entrevistar a los usuarios actuales y potenciales.
- Realizar un video para mostrar como podría funcionar el nuevo sistema.
- Investigar en documentos existentes.
- Conducir tormentas de ideas con los usuarios actuales y potenciales.
- Observar las estructuras y los patrones.

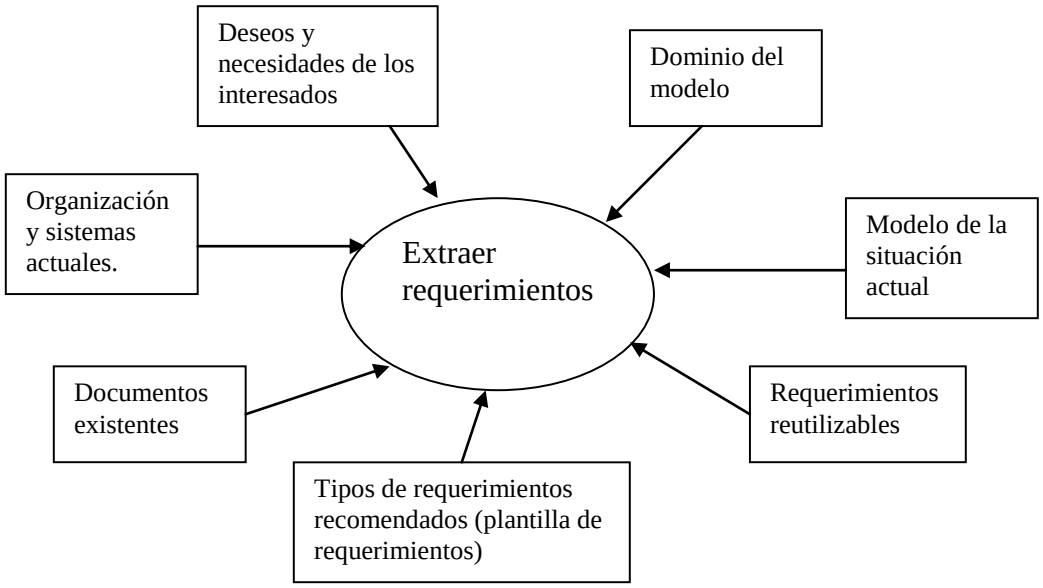


Figura 2.2: Posibles fuentes requerimientos (Robertson y Robertson, 1999).

La obtención y análisis de requerimientos pueden afectar a varias personas de la organización. El término *stakeholder* (sin traducción al español) se utiliza en la Ingeniería de Software para referirse a cualquier persona o grupo que se verá afectado por el sistema, directa o indirectamente.

Según [Sommerville, 2005], obtener y comprender los requerimientos de los stakeholders es difícil por varias razones:

- Los stakeholders a menudo no conocen lo que desean obtener del sistema informático excepto en términos muy generales. Pueden hacer demandas irreales o resultarles difícil expresar lo que quieren que haga el sistema.
- Los ingenieros de requerimientos, sin experiencia en el dominio del cliente, deben comprender los requerimientos que los stakeholders expresan con sus propios términos y con un conocimiento implícito de su trabajo.
- Diferentes stakeholders tienen requerimientos distintos. Es necesario descubrir las concordancias y los conflictos entre éstos.

- Los factores políticos pueden influir en los requerimientos del sistema. Por ejemplo, los directivos pueden solicitar requerimientos específicos del sistema que incrementarán su influencia en la organización.
- Pueden emerger nuevos requerimientos de nuevos stakeholders que no habían sido consultados previamente.

El *descubrimiento de requerimientos* es el proceso de recoger información sobre el sistema propuesto y los existentes extrayendo esta información del usuario y del sistema. Las fuentes de información durante la fase del descubrimiento de requerimientos incluyen la documentación, los stakeholders del sistema y la especificación de sistemas similares. Las técnicas de descubrimiento de requerimientos, son varias, pueden usarse entrevistas, escenarios, prototipos y etnografía.

La *figura 2.3* explica porqué el proceso de los requerimientos es crítico para el buen desarrollo de software.

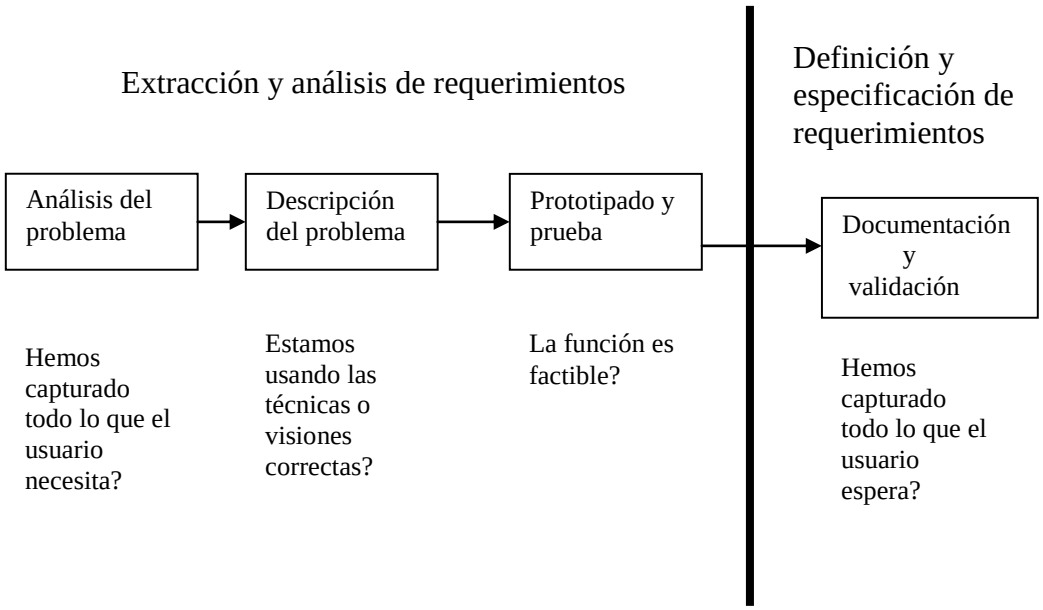


Figura 2.3: El proceso de determinación de los requerimientos (Fleeger, 2002).

II.3.- Especificación de requerimientos.

En el capítulo III se describe con detalle la Especificación de Requerimientos.

II.4.- Validación de requerimientos.

La validación de requerimientos sirve para demostrar que éstos realmente definen el sistema que el cliente desea. Asegura que los requerimientos están completos, son exactos y

consistentes. Debe garantizar que lo descrito es lo que el cliente pretende ver en el producto final. Esta validación es importante porque la detección de errores durante el proceso de análisis de requerimientos reduce mucho los costos. Si se detecta un cambio en los requerimientos una vez que el sistema esta hecho, los costos son muy altos, ya que significa volver a cambiar el diseño, modificar la implementación del sistema y probarlo nuevamente. [Pfleeger, 2005] plantea que las verificaciones que deben llevarse a cabo durante el proceso de validación, son las siguientes:

- *Verificación de validez*. El análisis puede identificar que se requieren funciones adicionales o diferentes a las que pidieron los stakeholders.
- *Verificación de consistencia*. No debe haber restricciones o descripciones contradictorias en el sistema.
- *Verificación de completitud*. El documento de requerimientos debe incluir requerimientos que definan todas las funciones y restricciones propuestas por el usuario del sistema.
- *Verificación de realismo*. Asegurar que los requerimientos pueden cumplirse teniendo en cuenta la tecnología existente, el presupuesto y el tiempo disponible.
- *Verificabilidad*. Para reducir la posibilidad de discusiones con el cliente, los requerimientos del sistema siempre deben redactarse de tal forma que sean verificables. Esto significa que se debe poder escribir un conjunto de pruebas que demuestren que el sistema a entregar cumple cada uno de los requerimientos especificados.

Existen varias técnicas de validación de requerimientos, estas son: *revisiones de requerimientos*, *construcción de prototipos* y *generación de casos de prueba*.

Una *revisión de requerimientos* es un proceso manual en la que intervienen tanto el cliente como personal involucrado en el desarrollo del sistema, ésta puede ser formal o informal, y tiene el fin de verificar que el documento de requerimientos no presente anomalías ni omisiones. En una revisión formal, los revisores deben tomar en cuenta:

- Que el requerimiento se pueda verificar de modo realista.
- Que las personas que adquieren el sistema o los usuarios finales comprendan correctamente el requerimiento.
- Que tan adaptable es el requerimiento? Es decir, ¿puede cambiarse el requerimiento sin causar efectos de gran escala en los otros requerimientos del sistema?

La *construcción de prototipos* consiste en mostrar un modelo ejecutable del sistema a los usuarios finales y a los clientes, así éstos pueden experimentar con el modelo para ver si cumple con sus necesidades reales.

Los requerimientos deben poder probarse, es por esto que debe hacerse una *generación de casos de prueba*. Si una prueba es difícil o imposible de diseñar, normalmente significa que los requerimientos serán difíciles de implantar y deberían ser considerados nuevamente.

En resumen, la validación pretende asegurar que los requerimientos satisfarán las necesidades del cliente.

I.5.- Gestión de requerimientos. (Manejo de los cambios de requerimientos durante la construcción).

En la práctica, en casi todos los sistemas los requerimientos cambian. Las personas involucradas desarrollan una mejor comprensión de lo que quieren que haga el software; la organización que compra el sistema cambia; se hacen modificaciones a los sistemas de hardware, software y al entorno organizacional. El proceso de organizar y llevar a cabo los cambios en los requerimientos se llama *gestión de requerimientos*.

El objetivo del analista es reconocer los elementos básicos de un sistema tal como lo percibe el usuario/cliente. El analista debe establecer contacto con el equipo técnico y de gestión del usuario/cliente y con la empresa que vaya a desarrollar el software. El gestor del programa puede servir como coordinador para facilitar el establecimiento de los caminos de comunicación.

Una vez que un sistema se ha instalado, inevitablemente surgen nuevos requerimientos. Es difícil para los usuarios y clientes del sistema anticipar qué efectos tendrá el sistema nuevo en la organización. Cuando los usuarios finales tienen experiencia con un sistema, descubren nuevas necesidades y prioridades.

Las personas que pagan por el sistema y los usuarios de éste, rara vez son la misma persona. Los clientes del sistema imponen requerimientos debido a las restricciones organizacionales y de presupuesto. Éstos pueden estar en conflicto con los requerimientos de los usuarios finales y, después de la entrega, pueden tener que añadirse nuevas características de apoyo al usuario para que el sistema cumpla con sus objetivos.

Otro motivo por el que cambian los requerimientos es que en ocasiones el entorno de negocios y técnico del sistema cambian después de la instalación. Puede ser que se introduzca un nuevo hardware, o puede ser que surja la necesidad de que el sistema interactúe con otros sistemas. También cambian las prioridades del negocio, las legislaciones y las regulaciones, y esto debe estar reflejado en sistema.

La gestión de requerimientos es el proceso de comprender y controlar los cambios en los requerimientos del sistema [Sommerville, 2005].

Es necesario mantenerse al tanto de los requerimientos particulares y mantener vínculos entre los requerimientos dependientes de forma que se pueda evaluar el impacto de los cambios en los requerimientos. El proceso de gestión de requerimientos debe empezar cuando esté disponible una versión preliminar del documento de requerimientos. Hay que establecer un proceso formal para implantar las propuestas de cambios y planear como se van a gestionar los requerimientos que cambian durante el proceso de obtención de requerimientos.

En la *figura 2.4* se muestra como se tiene una mejor comprensión de las necesidades de los usuarios conforme se va desarrollando la definición de los requerimientos. Esta nueva comprensión retroalimenta al usuario, quien puede proponer entonces un cambio en los requerimientos.

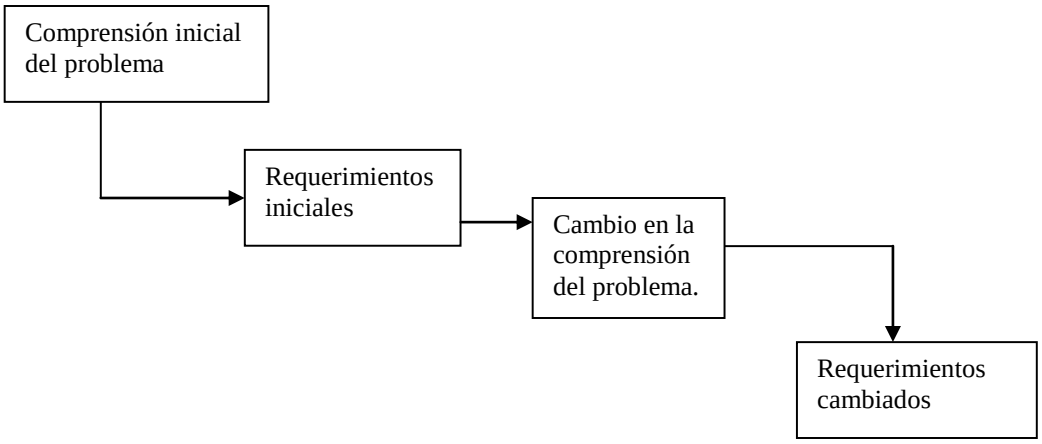


Figura 2.4 : Evolución de los requerimientos (Sommerville 2005).

Desde una perspectiva evolutiva, los requerimientos se dividen en dos clases:

- *Requerimientos duraderos*. Son los relativamente estables, están relacionados directamente con la actividad principal de la organización.
- *Requerimientos volátiles.* Cambian durante el proceso de desarrollo o después de que éste se haya puesto en funcionamiento.

La *gestión del cambio en los requerimientos* (*figura 2.5*) se debe aplicar a todos los cambios propuestos en los requerimientos. La ventaja de utilizar un proceso formal para gestionar el cambio es que todos los cambios propuestos son tratados de forma consistente y que los cambios en el documento de requerimientos se hacen de forma controlada.

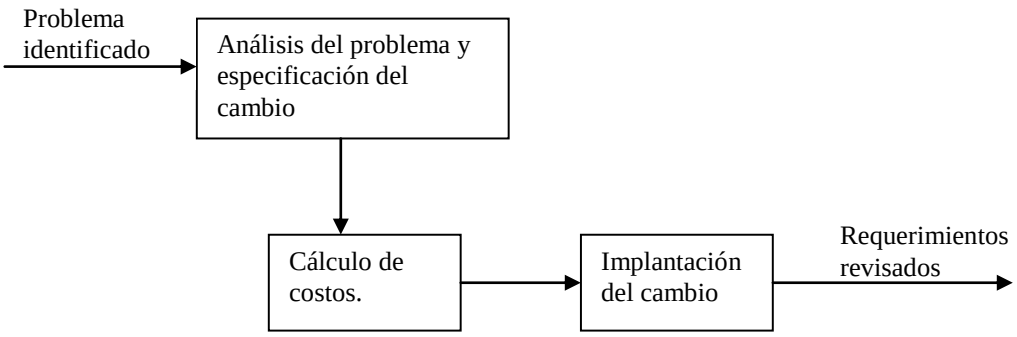


Figura 2.5: Gestión de cambios en los requerimientos (Sommerville 2005).

El proceso empieza con la identificación de un problema en los requerimientos o, algunas veces con una propuesta específica de cambio, dicha propuesta o el problema, se analiza para verificar que sea válido. Después se valora el efecto del cambio propuesto para

calcular su costo. Este costo se estima tomando en cuenta las modificaciones al documento de requerimientos, y si es el caso, al diseño e implementación del sistema. Al finalizar esta etapa se toma una decisión sobre si se continúa con el cambio de requerimientos. En caso afirmativo, se lleva a cabo la implantación del cambio.

El documento de requerimientos debe organizarse haciendo sus secciones tan modulares como sea posible, de tal modo que se puedan hacer cambios en él sin tener que hacer grandes reorganizaciones o redactar nuevamente gran cantidad del mismo. Así se podrán cambiar o reemplazar secciones individuales sin afectar a otras partes del documento.

Algunas veces pasa que se requiere de forma urgente un cambio en los requerimientos. Entonces surge la tentación de hacer ese cambio directamente en el sistema y luego modificar de forma retrospectiva el documento de requerimientos. Esto conduce casi inevitablemente a que la especificación de requerimientos y la implementación del sistema se desfasen. Una vez que se han hecho los cambios en el sistema, los del documento de requerimientos se pueden olvidar o se hacen de forma que no concuerdan con los cambios del sistema.

Ejemplo de un problema en los requerimientos (caso real).

"El desastre del cohete Ariane-5, fue provocado por la reutilización de una sección de código del Ariane-4. Nuseibeh (1997) analiza el problema desde el punto de vista de la reutlización de los requerimientos. Esto es, muchos ingenieros de software sienten que pueden obtener grandes beneficios reutilizando las especificaciones de requerimientos, (y también el diseño, el código y los casos de prueba relacionados) de sistemas anteriormente desarrollados. Las especificaciones candidatas son identificadas contemplando los requerimientos de funcionalidad o de comportamiento que son iguales o similares, haciendo después las modificaciones donde fueran necesarias. En el caso del Ariane-4, el sistema inercial de referencia (SRI) realizaba la mayoría de las funciones que necesitaba el Ariane-5.

Sin embargo, Nuseibeh señala que, aunque la funcionalidad que se necesitaba era similar a la del Ariane-4, había aspectos de Ariane-5 que eran significativamente diferentes. En particular la funcionalidad del módulo SRI que continuaba después del lanzamiento no se necesitaba para el Ariane-5. Es así que, si la validación de los requerimientos se hubiese realizado correctamente, los analistas habrían descubierto que las funciones activas después del lanzamiento no podían rastrearse hasta un requerimiento en la especificación o en la definición del Ariane-5. Es decir que la validación de los requerimientos podría haber jugado un papel crucial en la prevención de la destrucción del cohete." [Pleeger, 2002].

II.6.- Principales riesgos de la etapa de recolección de requerimientos.

El desarrollo de proyectos complejos de software lleva implícita la existencia de ciertos riesgos que si no se toman en cuenta y se analizan con cuidado, retrasarán considerablemente la entrega del producto o incluso podrían llegar a causar la cancelación del proyecto.

Un riesgo se puede definir de manera sencilla como:

- "la probabilidad de que una circunstancia adversa ocurra" [Sommerville, 2005]
- "un problema potencial que puede ocurrir o no" [Pressman, 2006]

A continuación se muestran algunos elementos de riesgo identificados por [McConnell, 1997] que pueden perjudicar la etapa de recolección de requerimientos:

• Los clientes no saben lo que quieren.
• Los clientes no quieren comprometerse a tener un conjunto de requerimientos escritos.
• Los clientes insisten en establecer nuevos requerimientos una vez que se han fijado la planificación y el coste.
• La comunicación con los clientes es lenta.
• Los clientes no participan en las revisiones o son incapaces de hacerlas.
• Los clientes no están preparados técnicamente.
• Los clientes no dejan realizar el trabajo a la gente.
• Los clientes no entienden el proceso de desarrollo de software.

Además [Soto & González, 2010] identifican los siguientes:

• No se comprende claramente el alcance del sistema.
• No se logran identificar con claridad los productos resultantes del proyecto.
• Ni los integrantes del equipo de desarrollo ni el cliente logran especificar de forma apropiada el área de aplicación.

Establecer buenas relaciones con los clientes permite identificar mejor los riesgos y controlarlos durante el desarrollo del proyecto.

Capítulo III: Especificación de requerimientos.

III.1.- Introducción.

La Especificación *es un documento que define, de forma completa, precisa y verificable, los requisitos, el diseño y el comportamiento u otras características, de un sistema o componente de un sistema.*

Para realizar bien el desarrollo de software es esencial tener una especificación completa de los requerimientos. Independientemente de lo bien diseñado o codificado que esté, un sistema pobremente especificado decepcionará al usuario y hará fracasar el desarrollo.

La forma de especificar tiene mucho que ver con la calidad de la solución. Los ingenieros de software que se han esforzado en trabajar con especificaciones incompletas, inconsistentes o mal establecidas han experimentado la frustración y confusión que invariablemente se produce. Como se ilustra en la *figura 3.1*, las consecuencias se padecen en la calidad, oportunidad y completitud del software resultante.

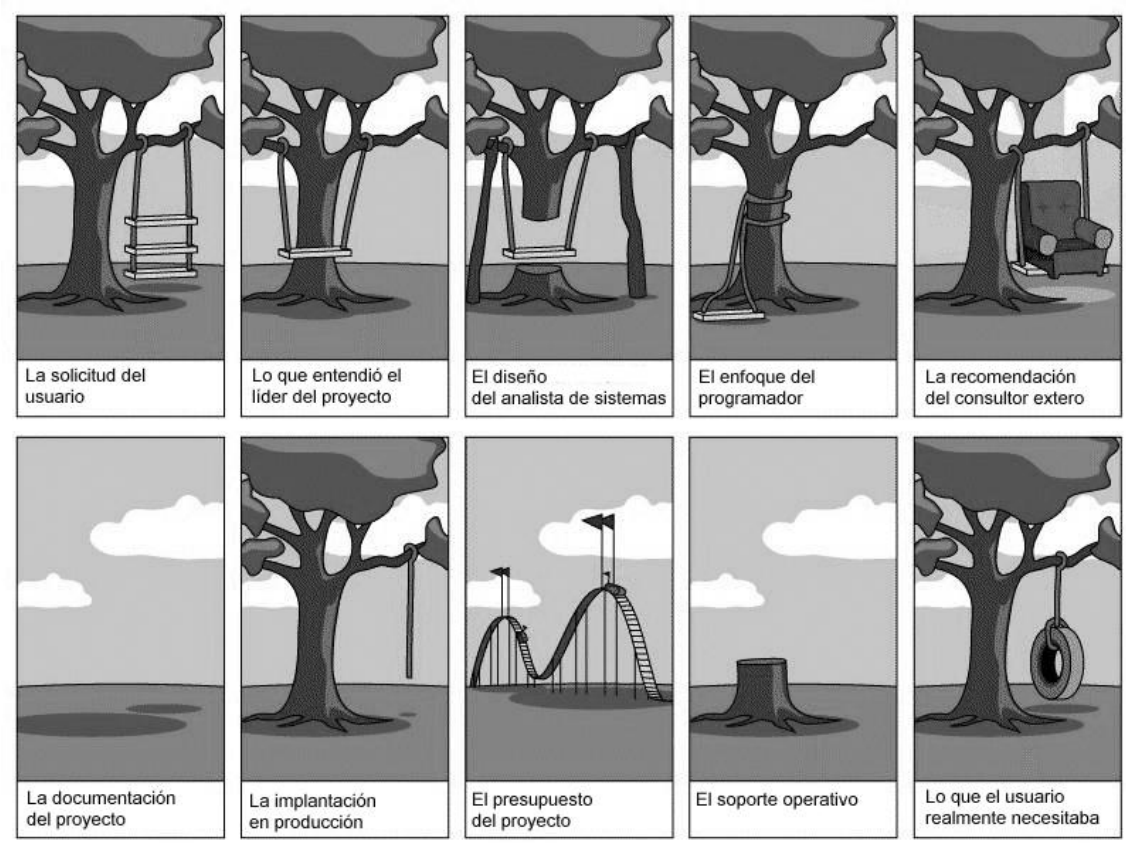


Figura 3.1: Problemas en el desarrollo de sistemas (ilustración anónima).

Tipos de especificaciones.- Los requerimientos de software pueden ser analizados de varias formas diferentes. Las técnicas de análisis pueden conducir a una especificación en papel que contenga las descripciones gráficas y el lenguaje natural de los requerimientos del software. La construcción de prototipos conduce a una especificación ejecutable, esto es, el prototipo sirve como una representación de los requerimientos. Los lenguajes de especificación formal conducen a representaciones formales de los requerimientos que pueden ser verificados o analizados.

III.2.- Principios de Especificación

La especificación, independientemente del modo en que se realice, puede ser vista como un proceso de representación. Los requerimientos se representan de forma que conduzcan finalmente a una correcta implementación del software. [Sommerville, 2005] plantea que una buena especificación debe procurar:

- *Separar funcionalidad de implementación*. Una especificación es una descripción de lo que se desea, en vez de cómo se realiza. Esto en la práctica puede llegar a no suceder del todo, sin embargo es un buen lineamiento a seguir.
- *Una especificación debe abarcar el entorno en el que el sistema opera.* Similarmente, el entorno en el que opera el sistema y con el que interactúa debe ser especificado.
- *Debe ser modificable.* Ninguna especificación puede ser siempre totalmente completa. El entorno en el que existe es demasiado complejo para ello. Una especificación es un modelo, una abstracción, de alguna situación real (o imaginada). Por tanto, será incompleta. Además, al ser formulada existirán muchos niveles de detalle.

III.3.- Requerimientos funcionales y no funcionales.

La especificación debe contener los requerimientos del sistema, la IEEE-830, 1998 divide los requerimientos en funcionales y no funcionales, a continuación se describe en que consiste cada uno de estos grupos de requerimientos.

Los *requerimientos funcionales* describen una interacción entre el sistema y su ambiente, describen cómo debe comportarse el sistema ante determinado estímulo. Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, también pueden declarar explícitamente lo que el sistema no debe hacer.

*Los requerimientos **funcionales** de un sistema describen lo que el sistema debe hacer.*

Los *requerimientos no funcionales:* describen una restricción sobre el sistema que limita nuestras elecciones en la construcción de una solución al problema. Restringen los servicios o funciones ofrecidas por el sistema. Incluyen restricciones de tiempo, el tipo de proceso de desarrollo a utilizar, fiabilidad, tiempo de respuesta, capacidad de almacenamiento.

*Los requerimientos **no funcionales** ponen límites y restricciones al sistema.*

Tipos de Requerimientos No Funcionales:

[Sommerville, 2005] desglosa en la *figura 3.2* los tipos de requerimientos no funcionales. Los tres grupos generales son: requerimientos del producto, organizacionales y externos, de cada grupo se derivan los particulares.

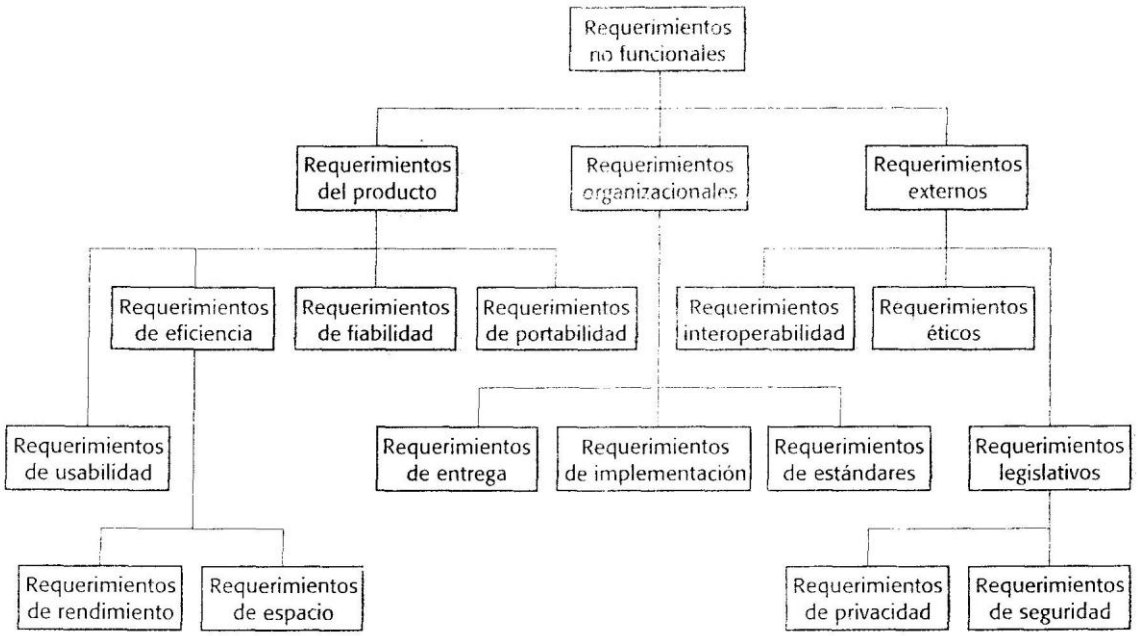


Figura 3.2 Tipos de requerimientos no funcionales (Sommerville, 2005)

- **Requerimientos del Producto:**

Especifican el comportamiento del producto. *Ejemplos: rapidez de la ejecución, capacidad de memoria, fiabilidad, etc.*

- **Requerimientos Organizacionales:**

Derivan de políticas y procedimientos existentes en la organización del cliente y del desarrollador. *Ejemplos: Estándares de procesos, métodos de diseño, lenguajes de programación, métodos de entrega, etc.*

- **Requerimientos Externos:**

Se derivan de factores externos al sistema y de sus procesos de desarrollo. *Ejemplos: Requisitos de interoperatividad, legislativos, éticos, etc.*

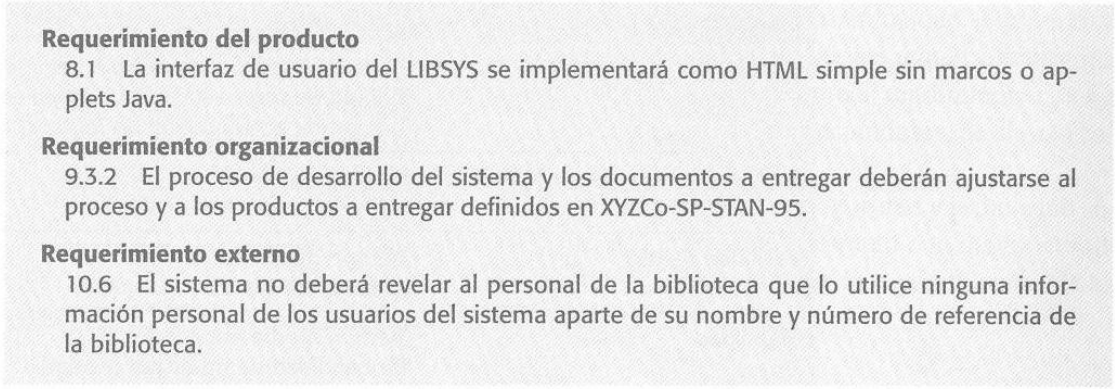


Figura 3.3: Ejemplos de requisitos no funcionales (Sommerville, 2005).

III.4.- El dominio de la información.

El software se construye para procesar datos; para transformar datos de una forma a otra; es decir, para aceptar una entrada, manipularla de alguna forma y producir una salida. El dominio de la información debe contener:

- *El flujo de información*.- representa la manera en la que los datos cambian conforme pasan a través de un sistema, esto se muestra, como se verá en el capítulo V con "Diagramas de Flujo de Datos".

- *El contenido de la información*.- depende de la aplicación específica, por ejemplo, si se trata de un sistema de nomina, la información consistirá en los nombres de los empleados, sueldos, etc., la información es diferente para un sistema que administra un hospital, para una biblioteca, un aeropuerto, etc.

- *La estructura de la información*.- muestra la forma en la que se construye la información, es decir, sus partes y como se subdivide cada una de las partes.

En el capítulo V: "Metodologías del análisis estructurado de requerimientos" y en el capítulo VI: "Metodologías del análisis de requerimientos orientado a objetos", se exponen las metodologías más utilizadas para especificar el dominio de la información.

III.5.- La documentación

Existen dos tipos de documentos de requerimientos, el documento de *especificación de requerimientos* cubre exactamente lo mismo que el documento de *definición de los requerimientos*, la diferencia entre estos dos tipos de documentos la marca el lector, es decir, a quien están dirigidos, el nivel en el que están escritos depende si se trata del cliente o de las personas que van a desarrollar el sistema, esto se muestra con claridad en la *figura 3.4*:

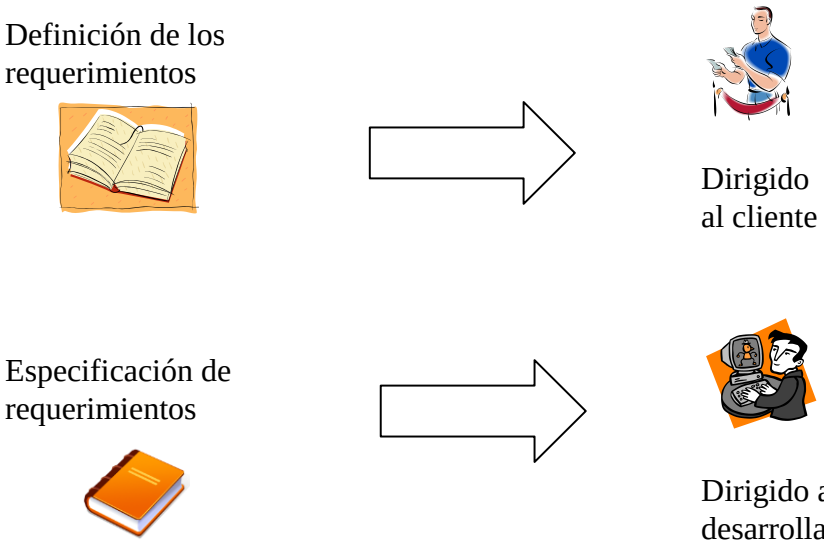


Figura 3.4: Tipos de documentos de requerimientos.

Normalmente, la *definición de los requerimientos* está redactada en lenguaje natural, mientras que la *especificación de los requerimientos* se redacta de una forma más técnica, por ejemplo, puede definir un requerimiento que se hizo en lenguaje natural, como una serie de ecuaciones, un diagrama de flujo de datos, casos de uso, etc.

A continuación se muestra un ejemplo de [Pfleeger, 2005] de un requerimiento redactado en la *definición de los requerimientos*:

> 4.1.3.1.- INICIAR EL RASTREO DE UNA IMAGEN. Se debe hacer un procesamiento lógico para iniciar un RASTREO DE IMAGEN. La entrada debe ser una TRANSFERENCIA DE DATOS. La salida puede ser un a IDENTIDAD DE IMAGEN, el ESATADO DE LOS DATOS ó HOIQ. Cuando sea apropiado, el procesamiento lógico debe poder identificar una nueva instancia de la IMAGEN, el estado de esta nueva instancia de imagen debe ser IMAGEN EN RASTREO. Cuando se requiera el envío de pulsos, se debe ingresar un registro dentro de HOIQ el cual debe alimentar a los procedimientos de envío de pulsos.

En la *especificación*, este mismo requerimiento puede ser redactado en algún lenguaje formal de especificación y ligarse al documento de *definición de los requerimientos*, a continuación se muestra un ejemplo del estándar RSL. Aquí cabe señalar que organizaciones como el IEEE y otras organizaciones internacionales tienen estándares para el formato y contenido de los documentos de requerimientos, como por ejemplo el RSL

(RAISE Specification Language, RAISE significa: Rigorous Approach to Industrial Software Engineering).

ALFA: INICIAR_RASTREO_IMAGEN.
ENTRADAS: TRANSFERENCIA_DE_DATOS.
SALIDAS: HOIQ, ESTADO_DE_DATOS, IDENTIDAD_IMAGEN.
SE CREA: IMAGEN.
EL ESTADO ES: IMAGEN_EN_RASTREO.
DESCRIPCIÓN: (4.1.3.1) la solicitud de un pulso se hace por medio de un registro formal dentro del HOIQ el cual alimenta a los procedimientos de envío de pulsos.

En el ejemplo anterior puede observarse que para el cliente es más fácil comprender la descripción dada en el documento de definición, sin embargo tendría más dificultad con la especificación escrita en RSL.

Otro ejemplo de modelo para hacer especificaciones es el SDL (Specification and Description Languaje). SDL es un método formal que incluye conceptos orientados a objetos. Hay herramientas comerciales disponibles para dar soporte al diseño, la depuración y el mantenimiento de los enunciados SDL. A continuación se muestra un ejemplo de una especificación formal usando SDL[Pfleeger, 2005].

```
STATE SEL_W0RKING;
  INPUT Clear(line);
    DECISION ((cond[1ine] == OOS)
     || (cond[WORK] == OOS));
    (TRUE):
      TASK 'badinput =1;
      NEXTSTATE-
    ENDDECISION;
    TASK  ' cond [line] = NORMAL';
    DECISION ((line == PROT) &&
      (cond[WORK] > cond[PROT]));

    (TRUE):
      NEXTSTATE SEL_PROTECTION;
    ENDDECISION;
    NEXTSTATE-; ENDSTATE
SEL_WORKING;
```

Existen diferentes lenguajes de especificación además de SDL y RSL, podemos mencionar: Modechart, VFSM, Esterel, Lotos, Z y C. Cuando se opta por usar un lenguaje formal de especificación, hay que estudiar que tanto se adopta el lenguaje a los criterios que se consideren importantes en los requerimientos, por ejemplo, para sistemas de lanzamiento de un cohete se toma en cuenta que con el lenguaje de especificación exista capacidad de comprobación, simulación y seguridad de ejecución.

III.5.1.- Recomendación para la Especificación de Requerimientos de Software de la IEEE.

Existe una organización muy importante a nivel internacional llamada IEEE (Institute of Electrical and Electronics Engineers, en español le llaman comunmente la I triple E). Esta organización, produce estándares que se aplican en muchas industrias del mundo. La IEEE, edita revistas de divulgación científica con un prestigio muy alto, y también organiza congresos muy importantes en el ámbito internacional. Por lo general, los libros de texto que hablan acerca de los requerimientos de software, incluyendo estas notas, se basan en un estándar emitido por la IEEE qué se aprobó en 1998, llamado:

IEEE Std 830-1998

Std es la forma de abreviar "Standard" en inglés y el número de la especificación es 830, fue aprobada en 1998 y es una revisión de un estándar previo aceptado en 1993 , Por las siglas en inglés, SRS que significan: Software Requirements Specifications, se acostumbra llamar SRS al documento de especificación.

En el **IEEE Std 830-1998** se habla sobre las características que deben tener los requerimientos (correctos, consistentes, completos, realistas, rastreables y verificables), los tipos de requerimientos (funcionales y no funcionales), así como lo que se debe tomar en cuenta al elaborarlos (ambiente físico, interfaces, usuarios y factores humanos, funcionalidad, documentación, datos, recursos, seguridad y aseguramiento de la calidad). En resumen, este estándar recomienda lo que hemos visto hasta ahora a lo largo del curso. Lo más importante del **IEEE Std 830-1998** es que define la estructura que debe tener una especificación de requerimientos, esta estructura se explica en la siguiente sección.

Normalmente, es necesario contar con un permiso para poder tener acceso al estándar **IEEE Std 830-1998** vía Internet, sin embargo, existe una traducción al español de este estándar cuyo acceso es libre, y se puede consultar en

http://www.unap.cl/metadot/index.pl?id=20061&isa=Item&field_name=item_attachment_file&op=download_file

Otra forma de tener acceso a este link, es buscando en google:

IEEE-STD-830-1998: ESPECIFICACIONES DE LOS REQUISITOS

III.5.1.- Estructura de una Especificación de requerimientos.

La **IEEE Std 830-1998** define la siguiente *estructura para una especificación de requerimientos*, cada una de las secciones mencionadas a continuación se detalla y se explica en el documento 830:

1. Introducción
   1.1. Objetivo

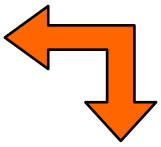
1.2. Ámbito
1.3. Definiciones, Siglas y Abreviaturas
1.4. Referencias
1.5. Visión Global
2. Descripción general
2.1. Perspectiva del producto
2.2. Funciones del producto
2.3. Características del usuario
2.4. Limitaciones generales
2.5. Supuestos y dependencias

3. Requerimientos específicos

Apéndices
Índice

La **IEEE Std 830-1998** es parte de los estándares que es necesario cubrir cuando se pretende cumplir con las normas de calidad, por lo tanto, esta estructura se respeta en la mayoría de las especificaciones de requerimientos en cualquier parte del mundo cuando se elaboran sistemas de software a nivel industrial.

En la *figura 3.5* se muestra gráficamente la estructura recomendada por la IEEE para una especificación de requerimientos.



1.- Introducción
2.- Descripción General.
*3.- **Requerimientos Específicos.***
Apéndices.
Índice

*3.- **Requerimientos Específicos**.*
3.1 Requisitos Funcionales.
3.2 Requisitos de Interfaz Externa.
3.3 Requisitos de Ejecución.
3.4 Restricciones de diseño.
3.5 Atributos de calidad → Mantenimiento, reutilización
3.6 Otros Requisitos.

Figura 3.5: Estructura de la especificación de requerimientos según IEEE std 830-1998.

En cuanto a la sección 3 *requerimientos específicos*, la **IEEE Std 830-1998** propone ocho plantillas diferentes a elegir, y son las siguientes:

A.1.- Plantilla de SRS organizada por *el modo* (versión 1).
A.2.- Plantilla de SRS organizada por *el modo* (versión 2).
A.3.- Plantilla de SRS organizada por *la clase del usuario*.
A.4.- Plantilla de SRS organizada por *el objeto*.
A.5.- Plantilla de SRS organizada por *el rasgo o característica*.
A.6.- Plantilla de SRS organizada por *el estímulo*.
A.7.- Plantilla de SRS organizada por *la jerarquía funcional*.
A.8.- Plantilla de SRS con *organización múltiple*.

A manera de ejemplo, se muestra la estructura de la sección 3 utilizando la plantilla de SRS organizada por *el modo* (versión 1).

3. Los requerimientos específicos

3.1 requerimientos de las interfaces externas
3.1.1 interfaz con el usuario
3.1.2 interfaz con el hardware
3.1.3 interfaz con el software
3.1.4 interfaces de comunicaciones

3.2 requerimientos funcionales

3.2.1 modo 1
3.2.1.1 requerimiento 1.1 funcional
.
.
.
3.2.1.n requerimiento 1.n Funcional

3.2.2 modo 2
.
.
.

3.2.m Modo m
3.2.m.1 requerimiento Funcional m.1
.
.
.
3.2.m.n requerimiento Funcional m.n

3.3 Requerimientos del desarrollo

3.4 Restricciones del diseño
3.4.1. Acatamiento de estándares
3.4.2. Limitaciones hardware

3.5 Atributos de sistema de software

3.6. Otros requerimientos

Una vez obtenidos y analizados los requerimientos, se crean el **SyRS** (***System Requirements Specification***) y el **SRS** (***Software Requirements Specification***, los estándares que se utilizan para escribir estos documentos se muestran en la *figura 3.6*.

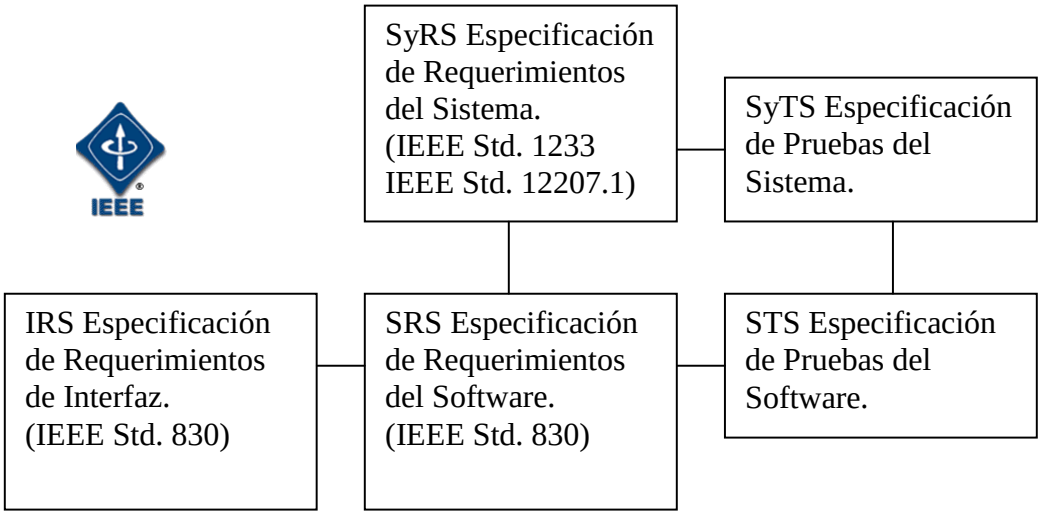


Figura 3.6: Visión global de los estándares de la IEEE.

Figura 3.7: ¿Por qué es necesaria una buena especificación en los requerimientos?

Capítulo IV: Relaciones entre administración de requerimientos y modelos de ciclos de vida.

Según [Pressman, 2002] la ingeniería de software contiene tres elementos básicos: *i)* la *metodología* (o conjunto de *métodos*) los cuales establecen como construir el software. *ii)* Los *paradigmas o modelos* que definen la secuencia en la que se aplican los métodos y *iii)* Las *herramientas* utilizadas para dar soporte a los métodos.

Los *métodos* de la ingeniería de software abarcan las siguientes tareas: Planeación y estimación del proyecto, Recolección de los requerimientos, Análisis de los requerimientos del problema, Diseño de las estructuras de datos, arquitectura de los programas y procedimientos algorítmicos, Codificación, Prueba, Implantación, y Mantenimiento.

Un *modelo* es una secuencia de pasos a seguir para alcanzar el final de un proyecto. Al modelo o proceso de desarrollo de software se le conoce como *ciclo de vida del software*, porque describe la vida de un producto de software desde su concepción hasta su implantación, entrega, utilización y mantenimiento. [Pfleeger, 2005] menciona que los procesos son importantes porque imponen consistencia y estructura sobre un conjunto de actividades. Estas características son útiles cuando se sabe cómo hacer algo bien y se desea asegurar que otros lo hagan de la misma manera. Un proceso es más que un procedimiento. Un procedimiento es como una receta: una manera estructurada de combinar herramientas y técnicas para generar un producto. Sin embargo, un proceso es un conjunto de procedimientos organizado de tal modo que los productos se construyen para satisfacer un conjunto de metas o estándares. El proceso puede sugerir que se seleccione entre varios procedimientos, con tal de que se cumpla con la meta propuesta. Cuando se desarrolla software a gran escala, el ingeniero asume alguno de los roles del proceso de desarrollo (ver figura 4.1), sin embargo, a pequeña escala, el ingeniero asume cada uno de los roles, conforme se va necesitando.

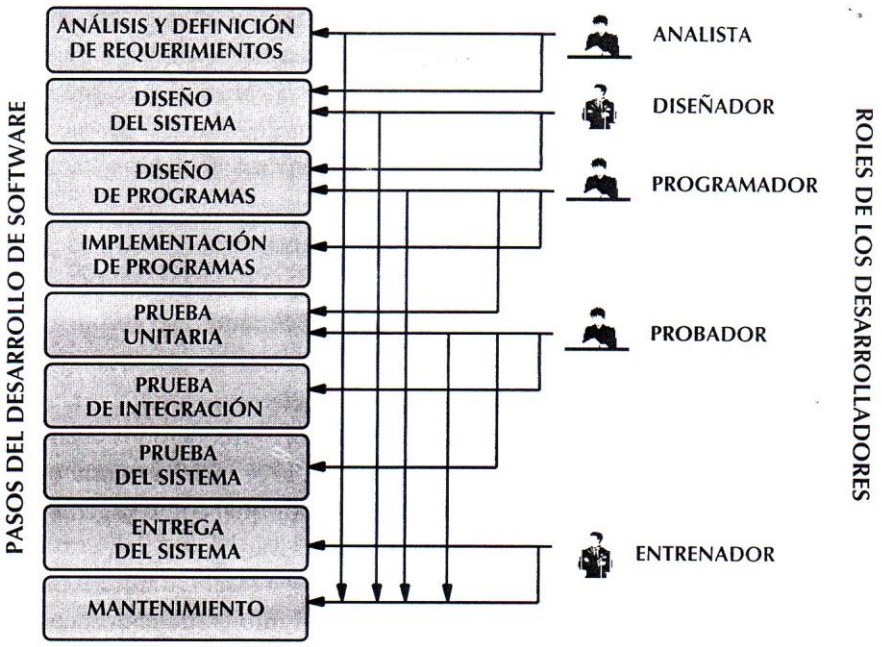


Figura 4.1: Roles de los desarrolladores de software (Pleeger 2002)

Un **proceso de desarrollo de software** es el conjunto estructurado de las actividades requeridas para elaborar un sistema de software, estas actividades son: especificación de requerimientos, diseño, codificación, validación (pruebas) y mantenimiento. Al proceso de desarrollo de software también se le conoce como ciclo de vida del software porque describe la vida de un producto de software; primero nace con la especificación de los requerimientos, luego se lleva a cabo su implantación, que consiste en su diseño, codificación y pruebas, posteriormente el producto se entrega, y sigue viviendo durante su utilización y mantenimiento. Cuando el producto evoluciona se le hacen modificaciones que generan nuevas versiones. La vida del sistema de software termina cuando éste se deja de utilizar.

Por otra parte, un **modelo de desarrollo de software** es una representación abstracta de este proceso [Sommerville, 2005]. Al modelo de desarrollo también se le llama paradigma del proceso. Hay una gran variedad de paradigmas o modelos de desarrollo de software, cada libro que trata este tema [Braude, 2003; McConnell, 1997; Pfleger, 2002; Pressman, 2002; Weitzenfeld, 2004], entre otros, elige los que considera más importantes y desafortunadamente las opiniones son muy diversas, sin embargo, Sommerville [Sommerville, 2005], clasifica sabiamente todos los procesos de desarrollo de software en tres modelos o paradigmas generales que no son descripciones definitivas de los procesos del software sino más bien, son abstracciones de los modelos que se pueden utilizar para desarrollar software:

- *El modelo en cascada*. Representa a las actividades fundamentales del proceso de desarrollo de software como fases separadas y consecutivas. Estas

actividades son: especificación, implantación (diseño, codificación, validación) y mantenimiento.

- ☞ *Modelo evolutivo*. Entrelaza las actividades de especificación, desarrollo y validación. Un sistema inicial se desarrolla rápidamente a partir de especificaciones abstractas. Éste se refina basándose en las peticiones del cliente para producir un sistema que satisfaga sus necesidades.
- ☞ *Modelo de componentes reutilizables*. Se basa en la existencia de un número significativo de componentes reutilizables. El proceso de desarrollo del sistema se enfoca en integrar estos componentes en el sistema en lugar de desarrollarlos desde cero.

Estos tres paradigmas o modelos de procesos genéricos se utilizan ampliamente en la práctica actual de la ingeniería del software. No se excluyen mutuamente y a menudo se utilizan juntos, especialmente para el desarrollo de sistemas grandes. Independientemente del modelo que se elija, siempre se presentará un reto fundamental: *el análisis de los requerimientos y la elaboración de la Especificación del sistema de software a desarrollar*.

IV.2.1.- El modelo en cascada.

El modelo en cascada (*figura 4.2*), presenta una visión muy clara de cómo se suceden las etapas durante el desarrollo, y sugiere a los desarrolladores cuál es la secuencia de eventos que podrán encontrar.

También conocido como ciclo de vida del software. Consta de 5 etapas, que son las actividades fundamentales en cualquier desarrollo de software:

- *Análisis y definición de requerimientos*. Se definen los servicios, metas y restricciones del sistema a partir de consultas con los clientes y usuarios. Con esta información se produce el documento de "Especificación del Sistema".
- *Diseño del sistema y del software*. El proceso de diseño del sistema divide los requerimientos en software o hardware. Establece una arquitectura completa del sistema. El diseño de software identifica y describe las abstracciones fundamentales del sistema software y sus relaciones.
- *Implementación y validación de unidades*. Durante esta etapa, el diseño del software se lleva a cabo como un conjunto de unidades de programas. La prueba de unidades implica verificar que cada una cumpla su especificación.
- *Integración y validación del sistema*. Los programas o las unidades individuales de programas se integran y prueban como un sistema completo para asegurar que se cumplan los requerimientos del software. Después de las pruebas, el sistema de software se entrega al cliente.
- *Funcionamiento y mantenimiento*. Por lo general (aunque no necesariamente), ésta es la fase más larga del ciclo de vida. El sistema se instala y se pone en funcionamiento

práctico. El mantenimiento implica corregir errores no descubiertos en las etapas anteriores del ciclo de vida y mejorar la implantación de las unidades del sistema.

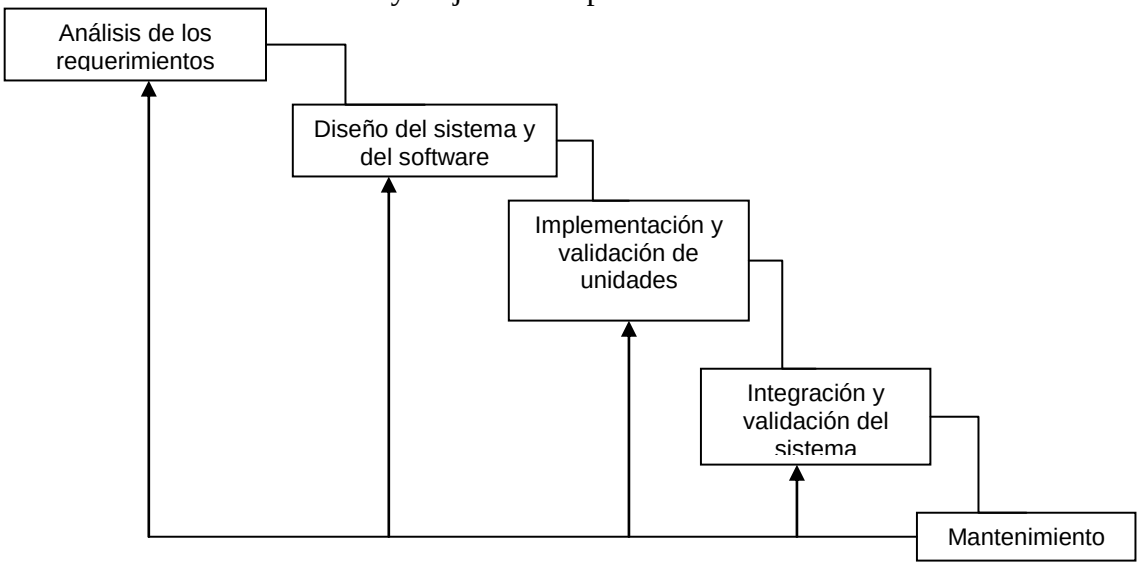


Figura 4.2: El modelo en cascada (Sommerville, 2005)

El resultado de cada fase es uno o más documentos aprobados (firmados). La siguiente fase no debe empezar hasta que la fase previa haya finalizado. En la práctica, estas etapas se superponen y proporcionan información a las otras. Durante el diseño se identifican problemas con los requerimientos, durante el diseño de código se encuentran problemas y así sucesivamente. El principal problema de este modelo es su inflexibilidad al dividir el proyecto en distintas etapas. Se deben hacer compromisos en las etapas iniciales, lo que hace difícil responder a los cambios en los requerimientos del cliente. Por lo tanto, el modelo en cascada solo se debe utilizar cuando los requerimientos se comprenden bien y sea improbable que cambien radicalmente durante el desarrollo del sistema. Sin embargo este modelo este modelo es muy importante porque define las etapas que se siguen en los procesos de software.

Según [Pfleeger, 2005], Curtis, Krasner, Shen e Iscoe (1987) hacen notar que la limitación principal del modelo en cascada reside en que no trata al software como un proceso de resolución de problemas. El modelo en cascada deriva del mundo del hardware y presenta una visión de manufactura sobre el desarrollo del software. Pero la manufactura produce un artículo en particular y lo reproduce muchas veces. El software no se desarrolla de la misma manera; en cambio, evoluciona a medida que el problema se comprende y se evalúan las alternativas. Así el software es un proceso de creación, no de fabricación. La creación implica intentar un poco de esto y de aquello, como desarrollar y evaluar prototipos, valorar la factibilidad de los requerimientos, comparar varios diseños, aprender

a partir de los errores, eventualmente, establecer una solución satisfactoria del problema en cuestión.

IV.2.2.- Modelos evolutivos.

Basándonos en la clasificación de Sommerville, los modelos evolutivos son iterativos. Se caracterizan por la forma en que permiten a los ingenieros del software desarrollar versiones cada vez más completas del software.

El desarrollo evolutivo se basa en la idea de desarrollar una implementación inicial, exponiéndola a los comentarios del usuario y refinándola a través de las diferentes versiones hasta que se desarrolla un sistema adecuado. Las actividades de especificación, desarrollo y validación se entrelazan en vez de separarse, con una rápida retroalimentación entre éstas. Existen dos tipos de desarrollo evolutivo:

1. *Desarrollo exploratorio*. Donde el objetivo del proceso es trabajar con el cliente para explorar sus requerimientos y entregar un sistema final. El desarrollo empieza con las partes del sistema que se comprenden mejor. El sistema evoluciona agregando nuevos atributos propuestos por el cliente.
2. *Prototipos desechables.* Donde el objetivo del proceso de desarrollo evolutivo es comprender los requerimientos del cliente y entonces desarrollar una definición mejorada de los requerimientos para el sistema. El prototipo se centra en experimentar con los requerimientos del cliente que no se comprenden del todo.

Los modelos evolutivos (o de prototipos) tienen como objetivo principal reducir el riesgo y la incertidumbre en el desarrollo, los requerimiento y/o el diseño requieren la investigación repetida par asegurar que el desarrollador, el usuario y el cliente tengan una comprensión unificada tanto de lo que se necesita como de lo que se propone como solución.

El modelo evolutivo suele ser más efectivo que el modelo en cascada para la producción de sistemas, ya que satisface las necesidades inmediatas de los clientes. Tiene la ventaja de que la especificación del sistema se puede desarrollar de forma creciente. Sin embargo, tiene dos problemas principales:

1. *El proceso no es visible*. Los administradores tienen que hacer entregas regulares para medir el progreso. Si los sistemas se desarrollan rápidamente, no es rentable producir documentos que reflejen cada versión del sistema.
2. *A menudo los sistemas tienen una estructura deficiente*. Los cambios continuos tienden a corromper la estructura del software. Incorporar cambios en él se convierte cada vez más en una tarea difícil y costosa.

Para sistemas pequeños y de tamaño medio (hasta 500,000 líneas de código), el modelo evolutivo de desarrollo es el mejor. Los problemas del desarrollo evolutivo se hacen particularmente agudos para sistemas grandes y complejos con un período de vida largo, donde diferentes equipos desarrollan distintas partes del sistema. Es difícil establecer una

arquitectura del sistema estable, porque se hace difícil integrar las contribuciones de los equipos.

Para sistemas grandes, se recomienda un proceso mixto que incorpore las mejores características del modelo en cascada (como son que la documentación se produce en cada fase y que este cuadra con otros modelos del proceso de ingeniería) y del desarrollo evolutivo. Las partes bien comprendidas se pueden especificar y desarrollar utilizando un proceso basado en el modelo en cascada. Las otras partes del sistema, como la interfaz de usuario, que son difíciles de especificar por adelantado, se deben desarrollar siempre utilizando un enfoque de programación exploratoria.

IV.2.2.1.- Ejemplos de modelos evolutivos.

El modelo incremental.- Entrega el software en partes pequeñas, pero utilizables, llamadas *incrementos*. En general, cada incremento se construye sobre aquel que ya ha sido entregado.

El modelo iterativo.- Se entrega el esqueleto de un sistema completo desde el principio, y luego cambia la funcionalidad de cada subsistema con cada versión nueva.

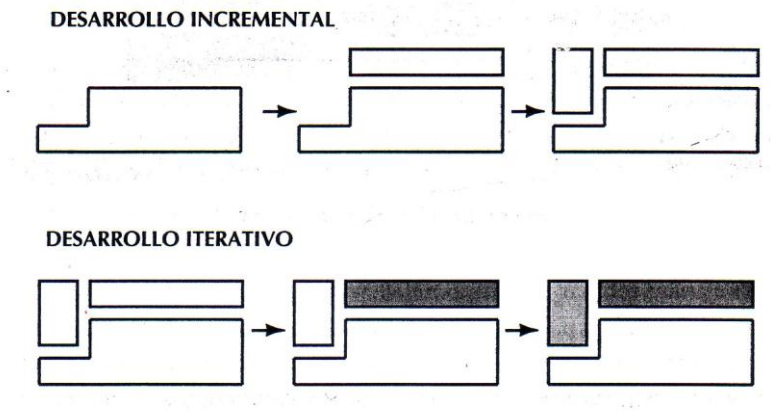


Figura 4.3 Modelos incremental e iterativo (Pleeger 2002)

El modelo en espiral.- Es un modelo de proceso de software evolutivo que conjuga la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial. Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo. Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema diseñado. Este modelo enfatiza ciclos de trabajo, cada uno de los cuales estudia el riesgo antes de proceder al siguiente ciclo. Cada ciclo comienza con la identificación de los objetivos, soluciones alternativas, restricciones asociadas con cada alternativa y, finalmente, se procede a su evaluación.

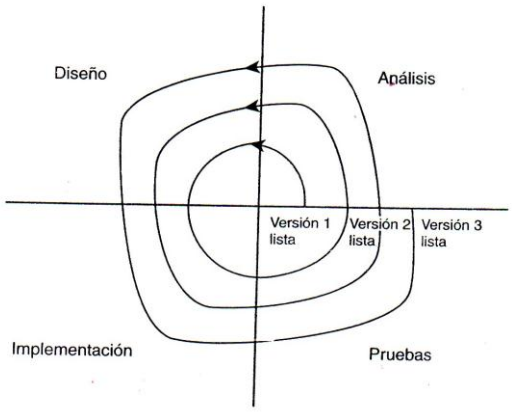


Figura 4.4 Modelos en espiral (Weitzenfeld, 2004)

Otros ejemplos de modelos evolutivos son: *Entrega por Etapas*, *Prototipado Evolutivo* y *Entrega Evolutiva* [Mc Connell, 1997].

IV.2.3.- Modelo de componentes reutilizables.

En el desarrollo y mantenimiento del software, a menudo se saca ventaja de los aspectos comunes de las aplicaciones, reutilizando elementos de desarrollo previos. Por ejemplo, se usa el mismo sistema operativo o el mismo sistema de gestión de base de datos de un proyecto de desarrollo a otro, en lugar de construir uno nuevo cada vez. Del mismo modo, cuando se construye un sistema similar, pero no igual a lo que se ha hecho antes, se reutilizan conjuntos de requerimientos, partes de diseños y grupos de guiones de prueba o de datos.

Según [Sommerville, 2005], en la mayoría de los proyectos existe algo de reutilización de software. Por lo general, esto sucede informalmente cuando las personas que trabajan en el proyecto conocen diseños de código similares al requerido. Los buscan, los modifican según lo creen necesario y los incorporan en el sistema. Las etapas de especificación de requerimientos y de validación son comparables con los otros procesos, sin embargo, las etapas intermedias en el proceso orientado a la reutilización son diferentes, estas etapas son:

- *Análisis de componentes*. Consiste en encontrar componentes que sirvan para implementar la especificación de requerimientos. En general los componentes que se utilizan solo proporcionan parte de la funcionalidad requerida por lo que se necesita modificarlos.
- *Modificación de requerimientos*. Con la información que se tiene de los componentes ya identificados, se analizan los requerimientos. Si es posible, se modifican los requerimientos para que concuerden con los componentes disponibles. Si las modificaciones no son posibles entonces se lleva a cabo nuevamente el análisis de componentes para buscar soluciones alternativas.

- *Diseño del sistema con reutilización*. Se diseña o se reutiliza un marco de trabajo para el nuevo sistema teniendo en cuenta los componentes que se reutilizan y los componentes que serán completamente nuevos.
- *Desarrollo e integración*. El software que no se puede adquirir externamente se desarrolla y los componentes reutilizables se integran. En este modelo, la integración de los sistemas es parte del desarrollo más que una actividad separada.

El modelo de componentes reutilizables tiene la ventaja obvia de reducir la cantidad de software a desarrollarse y así reduce los costos y los riesgos, sin embargo, los compromisos en los requerimientos son inevitables, y esto puede dar lugar a un sistema que no cumpla las necesidades reales de los usuarios. Más aún, si las nuevas versiones de los componentes reutilizables no están bajo el control de la organización que los utiliza, se pierde el control sobre la evolución del sistema.

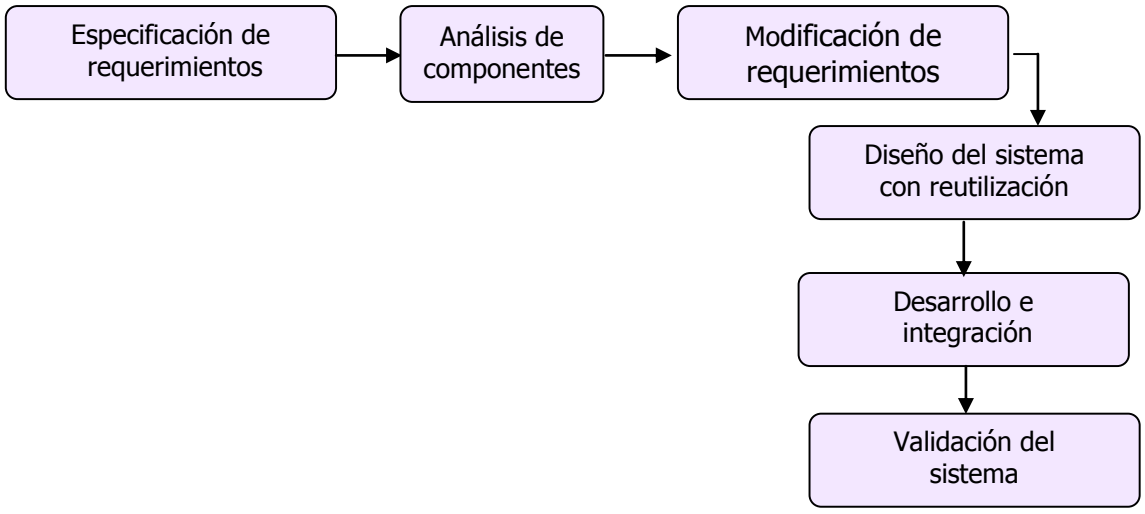


Figura 4.5 Modelo de componentes reutilizables (Sommerville, 2005)

Conclusión del capítulo: Los requerimientos están estrechamente relacionados con el modelo de ciclo de vida con el que se desarrolle el proyecto. En el modelo en cascada, los requerimientos tienen que estar bien definidos desde el inicio del proyecto y la probabilidad de que cambien debe ser mínima. Si se trabaja con los modelos evolutivos, los requerimientos se trabajan al inicio de cada iteración para aumentarlos, corregirlos o redefinirlos. Cuando el paradigma a utilizar es el modelo basado en componentes, es necesario cuidar que la modificación de requerimientos no produzca un sistema que no cumple con las necesidades reales de los usuarios. El alcance de la recolección de los requerimientos cambia considerablemente con el ciclo de vida.

Capítulo V: Artefactos de modelado para el Desarrollo Estructurado de Sistemas

V.1 Las principales metodologías estructuradas para el desarrollo de software.

En las *Metodologías Estructuradas* para el desarrollo de sistemas la unidad básica de construcción es la *función*, es decir, modelan a un sistema en términos de conjuntos de instrucciones que ejecutan una tarea. En otras palabras, las metodologías estructuradas se enfocan principalmente en la descomposición funcional de un sistema. El objetivo es lograr una definición completa del sistema en términos de funciones, estableciendo los datos de entrada y salida correspondientes.

Existen tres herramientas gráficas de modelado, también llamadas artefactos, que sirven para construir una especificación de los requerimientos del usuario usando una metodología estructurada, éstas son: los Diagramas de Entidad-Relación (DER), los Diagramas de Flujo de Datos (DFD) y los Diagramas de Transición de Estados (DET). Cada uno de ellos brinda una visión diferente del sistema. El primero pone énfasis en los datos y sus relaciones, el segundo centra la atención en la funcionalidad del sistema y el tercero en el comportamiento dependiente del tiempo. El Diccionario de Datos (DD) es un complemento a estas herramientas, el DD nos permite definir con un mayor grado de detalle los datos presentes en los diagramas. Trataremos cada una de estas herramientas en las secciones V.2, V.3, V.4 y V.5.

Las principales metodologías estructuradas para el desarrollo de sistemas de software son la de Pressman [Pressman, 2002], la de Yourdon [Yourdon, 1993] y la de Kendall, [Kendall, 1997], aunque cada una de estas metodologías tiene sus particularidades todas incluyen la especificación de los requerimientos y, a partir de estos la elaboración de un diseño. Las metodologías estructuradas comparten un conjunto de principios fundamentales, que son los siguientes.

- Representar y comprender el dominio de la información, así como el dominio funcional de un problema.
- Subdividir el problema de forma tal que se descubran los detalles de una manera progresiva (o jerárquica). La partición se aplica para reducir la complejidad.
- Representar al sistema lógica y físicamente.

Mecanismos para el análisis del dominio de la información. Estos mecanismos se concentran en el flujo de datos y en su contenido o estructura. El flujo de datos representa datos de entrada a los que se les aplican ciertas funciones para transformarlos en los datos

de salida. El contenido de los datos puede representarse explícitamente usando un mecanismo de diccionario o, implícitamente, con la estructura jerárquica de los datos.
Representación funcional. Las funciones se describen normalmente como transformaciones o procesos de la información. Cada función puede ser representada usando una notación específica. Una descripción de la función puede desarrollarse usando el lenguaje natural, un leguaje procedural con reglas sintácticas informales o un lenguaje de especificación formal.

Definición de interfaces. Es importante definir tanto las interfaces del sistema con el usuario, como las interfaces entre los diferentes módulos del sistema.

Mecanismos para subdividir el problema (*Partición).* Normalmente los problemas son demasiado grandes y complejos para ser comprendidos como un todo. Por esta razón, partimos o dividimos los problemas en partes que puedan ser fácilmente comprendidas, y establecemos interfaces entre las partes. Durante el análisis de requerimientos, el dominio funcional y el dominio de la información del software pueden ser particionados. La partición descompone un problema en sus partes constituyentes. Se hace una representación jerárquica de la función o información partiendo un elemento superior horizontal o verticalmente.

a).- verticalmente, se incrementan los detalles.
b).- horizontalmente, se descompone funcionalmente el problema.

En la figura 5.*1* se puede observar gráficamente esta descomposición vertical y horizontal de un problema.

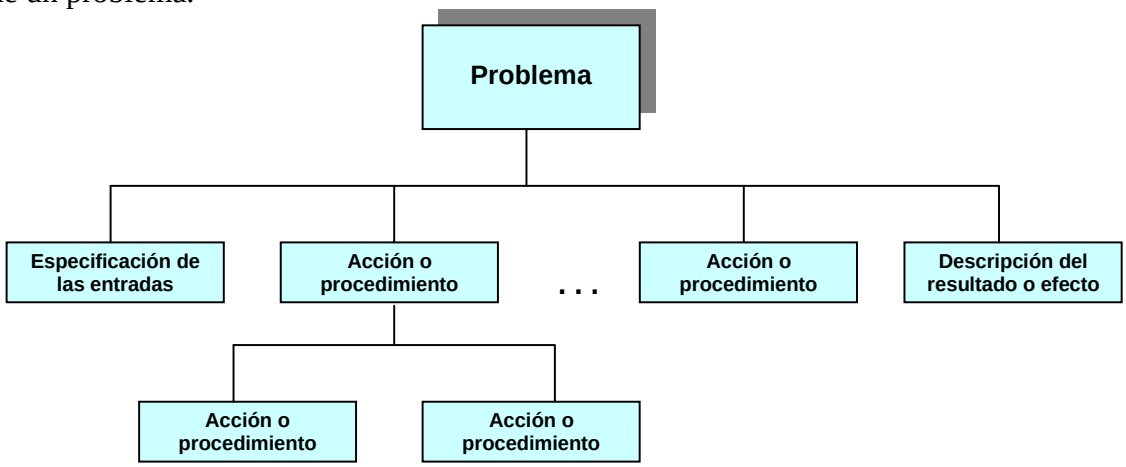


Figura 5.1: Descomposición vertical y horizontal de un problema.
Material de clase elaborado por Dr. Pedro Pablo (DMAS).

Representación de visiones físicas y lógicas. La visión lógica de los requerimientos del software presenta las funciones que han de realizarse y la información que ha de procesarse independientemente de los detalles de implementación. La visión física de los

requerimientos del software presenta una manifestación del mundo real de las funciones de procesamiento y las estructuras de información.

V.2.- Diagramas de Flujo de Datos (DFD).

Las herramientas gráficas más importantes del análisis estructurado son los Diagramas de Flujo de Datos (DFD). Un diagrama de flujo de datos (DFD), es una técnica gráfica que describe el flujo de información y las transformaciones que se aplican a los datos, conforme se mueven de la entrada a la salida, visualiza a un sistema como una red de procesos conectados entre sí. Los Diagramas de Flujo de datos son una notación operacional semi-formal que ha sido ampliamente adoptada para la especificación de sistemas de información. Un DFD es independiente del tamaño y de la complejidad del sistema, consiste en un diagrama en forma de red que representa el flujo de datos y las transformaciones que se aplican sobre ellos al moverse desde la entrada hasta la salida del sistema. El DFD se apoya en otras 2 técnicas: Diccionario de Datos, y Especificaciones de procesos.

Los elementos que componen a un DFD se representan como se indica en la *figura 5.2* y son los siguientes:

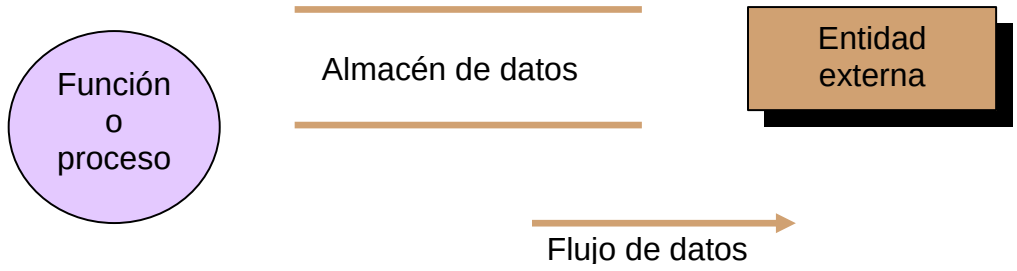


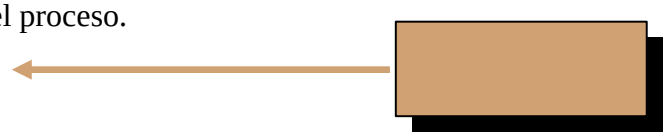
Figura 5.2: Elementos del Diagrama de Flujo de Datos.

- *Función o Proceso*. Representa la transformación del flujo de datos. Muestra cómo una o más entradas se transforman en salidas. Su nombre comienza con un verbo y es lo suficientemente largo y claro para que cualquier persona entienda de qué se trata. Dichas funciones van numeradas para diferenciarlas en un mismo nivel mostrando la jerarquía entre los niveles. Se representa por un círculo en cuyo interior está el nombre y el número de la función.
- *Entidad Externa:* Representa el origen o el destino de la información del sistema. Los flujos que parten o llegan a ellas definen el interfaz entre el sistema y el mundo exterior. Su representación gráfica es un rectángulo o cuadrado con el nombre.
- *Almacenamiento*. Son los datos pasivos; generalmente archivos, tablas, etc. Los almacenes de datos representan información del sistema almacenada de forma temporal por tanto representan datos en reposo; deben tener un nombre

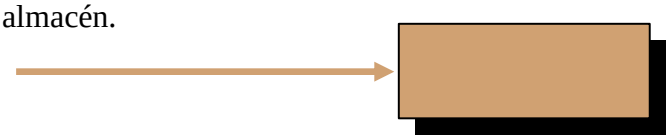
representativo, su representación gráfica son dos líneas paralelas con el nombre en medio, o también se representan con un rectángulo con el nombre adentro.

- *Flujo de dato*. Está representado por una flecha que indica su dirección, va del origen al destino. Los datos siempre van hacia y/o desde una función. Los flujos de datos representan datos en movimiento, los que se mueven hacia y desde almacenes simples no necesitan nombre si transportan toda la información del registro. Cuando se lee o escribe una porción de los elementos se debe especificar el nombre en el flujo. Existen tres tipos: *Flujo de entrada*, *Flujo de salida* y *Flujo de diálogo*. Los procesos pueden introducir o recuperar datos en los almacenes:

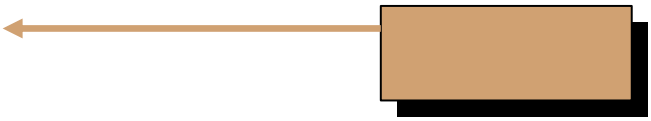
  - *Flujo de salida o consulta*: Indica la utilización de la información del almacén con el proceso.



  - *Flujo de entrada o actualización:* Indica que el proceso va a alterar la información del almacén.



  - *Flujo de diálogo:* Representa como mínimo un flujo de consulta y uno de actualización que no tienen relación directa.



La conexión Entidad Externa- Almacén y viceversa solo es posible con almacenes externos que sirven de interfaz entre el sistema y una entidad externa. La conexión entre procesos mediante un flujo de datos es posible siempre y cuando el proceso destino comience cuando el proceso origen finaliza. Los flujos de datos también se pueden dividir en *flujo síncrono* y *flujo asíncrono.*

El flujo de la información representa la manera en la que los datos cambian conforme pasan a través de un sistema. En la *figura* 5.*3*, la entrada se transforma en datos intermedios y más adelante se transforma en la salida.

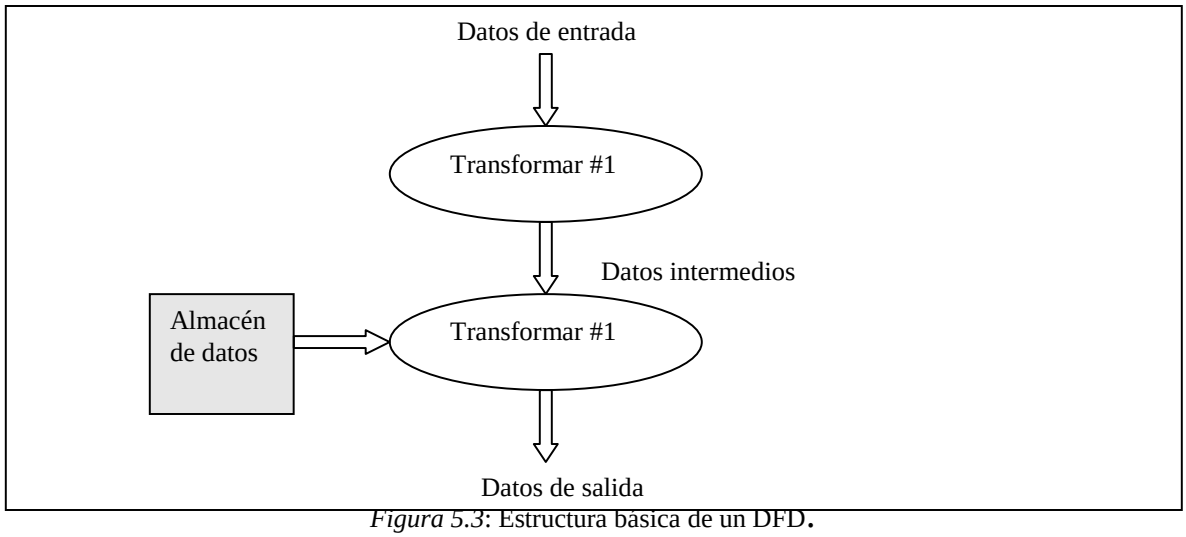


Figura 5.3: Estructura básica de un DFD**.**

Un modelo de flujo de datos puede aplicarse a cualquier sistema basado en computadora independientemente del tamaño o complejidad (*figura 5.4*). El sistema acepta entradas de distintas formas; aplica un hardware, software y elementos humanos para transformar la entrada en salida; y produce una salida en distintas formas. La entrada puede ser una señal de control transmitida por un transductor, una serie de números escritos por un operador humano, un paquete de información transmitido por un enlace a red, o un voluminoso archivo de datos almacenado en memoria secundaria. La transformación puede comprender una sencilla comparación lógica, un complejo algoritmo numérico, o un método de inferencia basado en la regla de un sistema experto. La salida puede encender un sencillo led o producir un informe de 200 páginas.

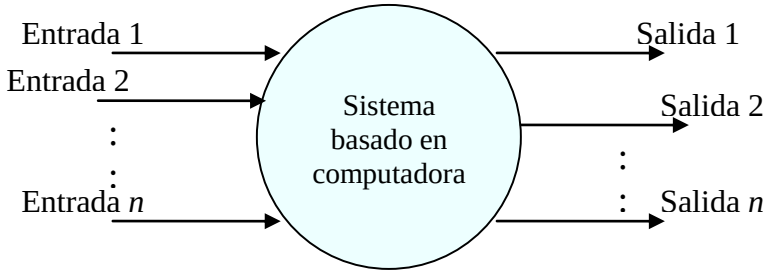


Figura 5.4: Cualquier sistema computacional se puede representar mediante flujo de datos.

Características de un buen DFD.

Las Funciones o Procesos: Cada proceso debe tener Entrada y Salida, es decir, el proceso debe ser capaz de generar los flujos de salida a partir de los de entrada

- El proceso no crea datos nuevos solo los transforma.

- El proceso no debe perder información.

Almacenes:

- Cada dato que sale primero debe entrar.
- No crean datos nuevos.

DFD por niveles.

Cuando el DFD es muy complejo y tiene muchas funciones, se organiza un DFD global en una serie de niveles, de modo que cada DFD de nivel inferior proporcione más detalles sobre un proceso del DFD de nivel superior.

Diagrama de Contexto: cuando construimos un DFD por niveles, el primer DFD consta de una sola burbuja, que representa el sistema completo y los flujos de datos muestran la comunicación entre el sistema y las entidades externas. A este DFD especial se le conoce como *Diagrama de Contexto*, representa el sistema de forma global, solo pueden aparecer entidades externas, flujos de datos y un único proceso que representa el sistema en su conjunto. Pueden aparecer almacenes de datos cuando son compartidos entre nuestro sistema y el exterior

Diagrama del sistema: Se le conoce como *diagrama del Sistema* o *diagrama de nivel 0*, representa las funciones principales a realizar así como la relación entre ellas. Las funciones de este diagrama deben ser lo más independientes entre sí porque esto facilita la descomposición de cada una por analistas diferentes. El DFD de nivel 0 muestra los procesos de más alto nivel del sistema y sus principales interfaces.

Procesos o funciones Primitivas: Son los procesos que no se descomponen en más diagramas de nivel inferior, ya sea porque no se puede o bien porque no interesa.

Consistencia entre niveles: Partiendo de una función de nivel superior que representa el sistema completo se va descendiendo por medio de burbujas a niveles más detallados mediante un razonamiento Top-Down hasta llegar a niveles en que las burbujas ya no se subdividen Estas burbujas se numeran de manera adecuada. Cada burbuja *i* de un nivel particular se asocia con una figura *i* del nivel siguiente (si es que existe). Por ejemplo, la burbuja 2 del DFD de la figura 0 (nivelo 0), se asocia con la figura 2 del nivel siguiente. Las burbujas en la figura *j* se numerarán *j*.1, *j*.2, *j*.3, etc. Por ejemplo, las burbujas de la figura 3 (obtenidas por la explosión de la burbuja 3 de la figura 0) se numerarán 3.1, 3.2, etc.; las burbujas obtenidas por la explosión de la burbuja 3.1, se numerarán 3.1.1, 3.1.2, 3.1.3, etc. En la *figura 5.5* se ve un ejemplo de explosiones.

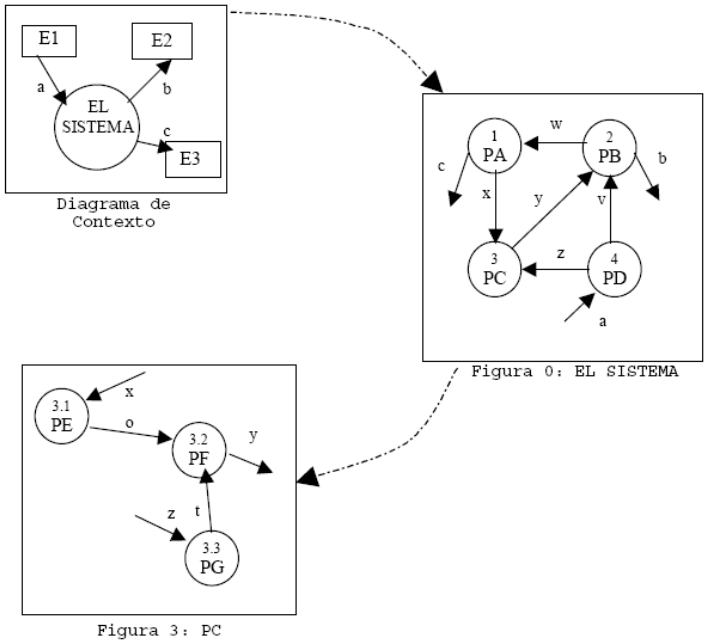


Figura 5.5: Ejemplo de explosiones en un DFD (Yourdon, 1993)

El nombre de la figura se hereda de la burbuja correspondiente a su explosión. Por ejemplo, si la burbuja 2 de la figura 0 se llama VALIDAR DATOS, entonces la figura 2 se deberá etiquetar "figura2: VALIDAR DATOS" y así sucesivamente para todos los niveles.

¿Cuántos niveles debe tener un DFD? Dependerá del sistema. Una regla que puede aplicarse es no poner más de media docena de procesos y almacenes relacionados. Debe caber todo en una sola hoja.

¿Deben desarrollarse todas las partes del sistema con el mismo número de niveles? No. Pueden existir partes más complejas que otras que necesiten un mayor número de niveles de partición. Pero por otro lado, si por ejemplo, al explotar el diagrama de contexto obtenemos 2 burbujas, donde la burbuja 1 es primitiva (no necesita ser explotada) y la burbuja 2 debe ser explotada en 7 niveles, significa que el modelo está desequilibrado y probablemente algunas de las porciones de la funcionalidad asignada a la burbuja 2 deban ser asignadas a otra nueva burbuja o a la burbuja 1.

¿Cómo asegurar que los niveles de un DFD sean consistentes entre sí? Se sigue una regla simple: "*los flujos de entrada y salida de una burbuja en un nivel dado deben corresponder con los que entran y salen de toda la figura asociada a dicha burbuja en el nivel inmediato inferior".* Los DFDs de la *figura 5.5* son consistentes entre sí.

¿Cómo se muestran los almacenes en los diversos niveles? La regla es la siguiente: "*mostrar un almacén en el nivel más alto donde por primera vez sirve de interfaz entre dos o más procesos, luego mostrarlo en cada nivel inferior que describa más a fondo cada una de dichas burbujas.*

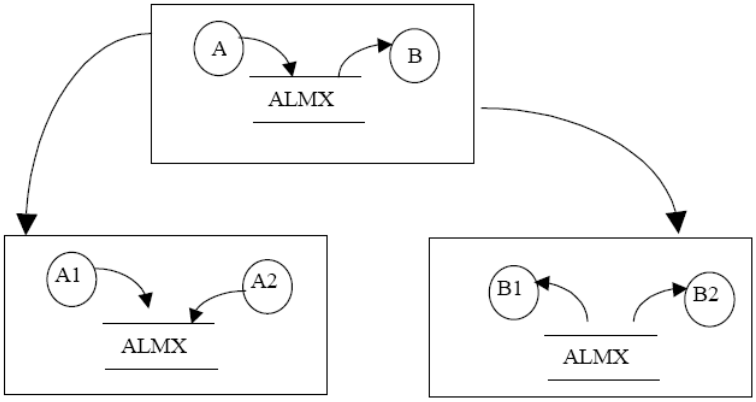


Figura 5.6: Ejemplo de almacén en diferentes niveles (Yourdon, 1993).

Guía práctica para la construcción de un DFD.

- Escoger nombres significativos para todos los componentes del DFD.
- Numerar los procesos. Esto sirve para referirse a ellos de una manera abreviada y para construir la numeración jerárquica por niveles.
- Redibujar los DFDs tantas veces como sea necesario.
- Evitar los DFDs excesivamente complejos. En lo posible deberá caber en una sola página.
- Asegurarse que el DFD sea lógicamente consistente. Las principales reglas de consistencia son:
    - Evitar sumideros infinitos, es decir, DFDs donde solo existen flujos de entrada y ninguno de salida.
    - Evitar burbujas de generación espontánea. Es decir, aquellas burbujas que tienen salida pero no tienen entradas. Estas son sospechosas, sin embargo podrían existir, por ejemplo, la generación de números aleatorios.
    - Evitar los flujos y procesos no etiquetados. Esto podría estar ocultando un error o falta de comprensión del problema.
    - Tener cuidado con los almacenes de solo lectura o solo escritura. Son sospechosos. Una excepción a esta regla son los almacenes externos al sistema que sirven de interfaz con algún otro sistema. O también el caso en el que los DFDs son muy grandes y al particionarlos podríamos encontrarnos con que una parte del sistema está modelada por un DFD en el cual un almacén es accedido como solo lectura, pero luego en otra parte del sistema, existe un DFD que accede al mismo almacén para escritura.

Ejemplo: A continuación se muestra el DFD de una visita al médico. El médico y el paciente se representan como entidades externas, los almacenes son los registros del paciente (su expediente) y los registros contables (los que tienen la información de los precios). En el DFD se observan dos procesos: el *examen médico*, que arroja tanto un diagnóstico y una medicación, como una lista de exámenes y servicios practicados, esta última sirve como flujo de entrada al segundo proceso, que es el de *contabilización*, que tiene como flujo de salida la factura para el paciente. Los flujos de datos son las flechas que sirven como entradas o salidas de los procesos, en cada flecha se pone el nombre del flujo que representa.

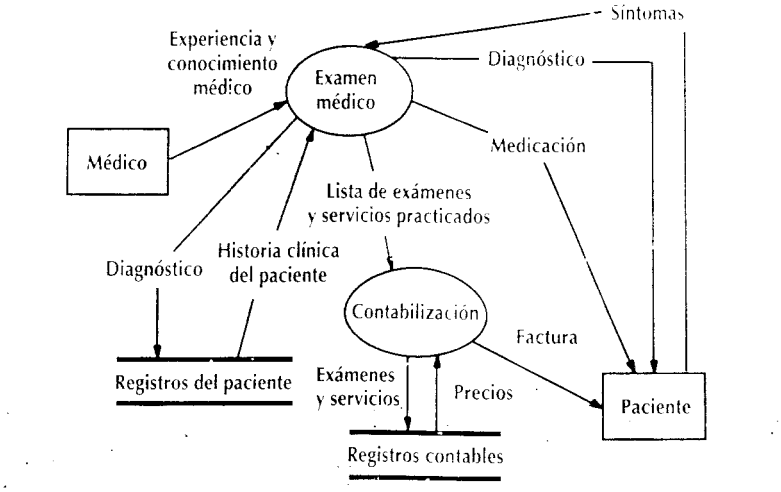


Figura 5.7: Diagrama de Flujo de Datos de la visita al médico (Pleeger 2002)

Para mostrar un ejemplo de un Diagrama de Flujo de Datos con Diferentes Niveles se tomó el Modelo del Sistema Integrado de Gestión Educativa (SIGEDU), de la Universidad del CEMA (Maglione y Placentino, 2001), el cual es un Sistema Informático que integra la información de las distintas áreas de la Universidad del CEMA, como por ejemplo la elaboración de programas de estudio, administración de recursos, administración de alumnos, sistema arancelario, biblioteca, docentes y graduados.

En la construcción de este modelo informático de gestión universitaria se utilizaron las técnicas de modelado de sistemas haciendo uso de las herramientas: diagrama de flujo de datos (DFD), diagrama entidad relación (DER) y diccionario de datos (DD). A continuación, se muestra el Diagrama de Contexto (una sola burbuja) en la *figura 5.8*.

Diagrama de Contexto

SIGEDU

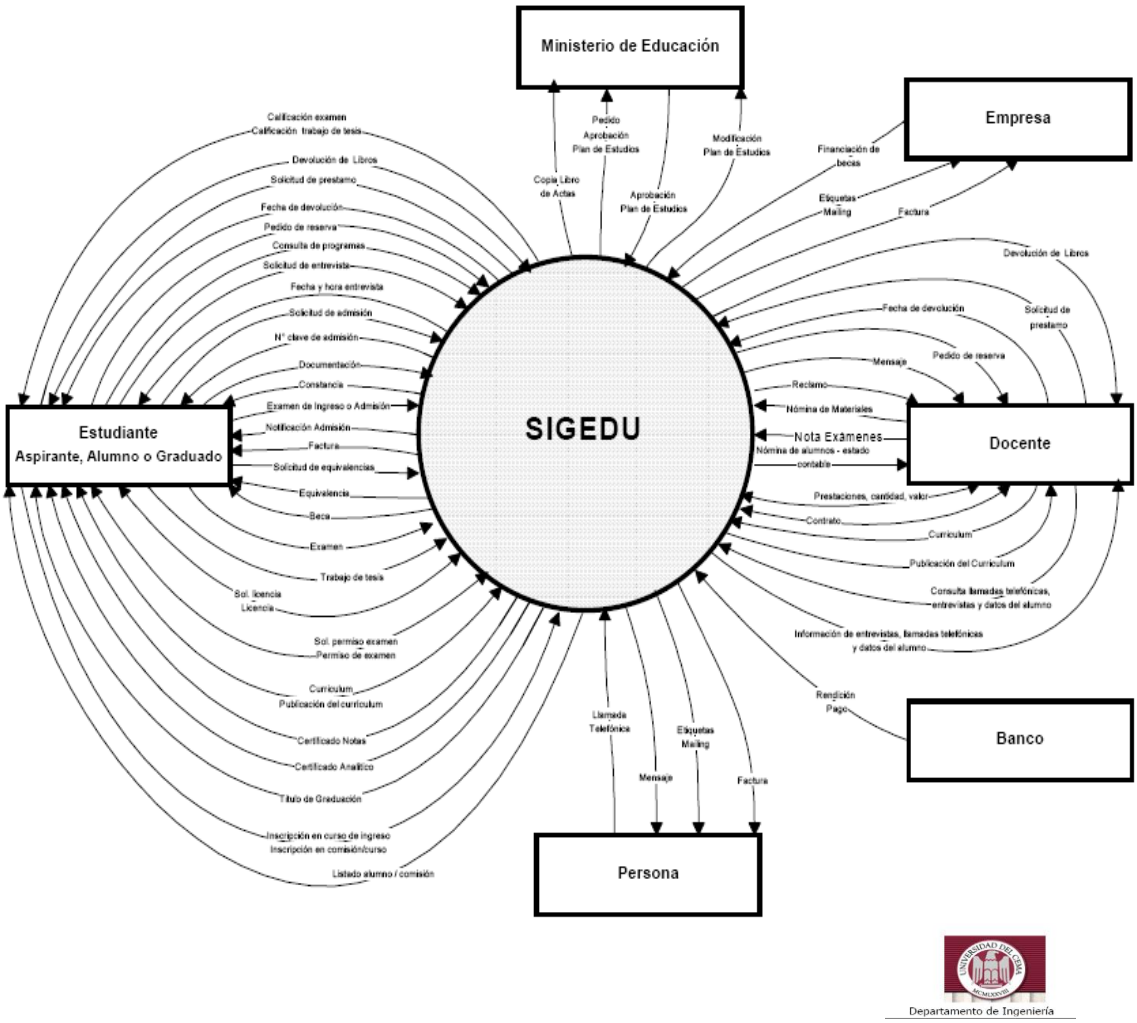


Figura 5.8: Diagrama de Contexto SIGEDU (Maglione y Placentino, 2001)

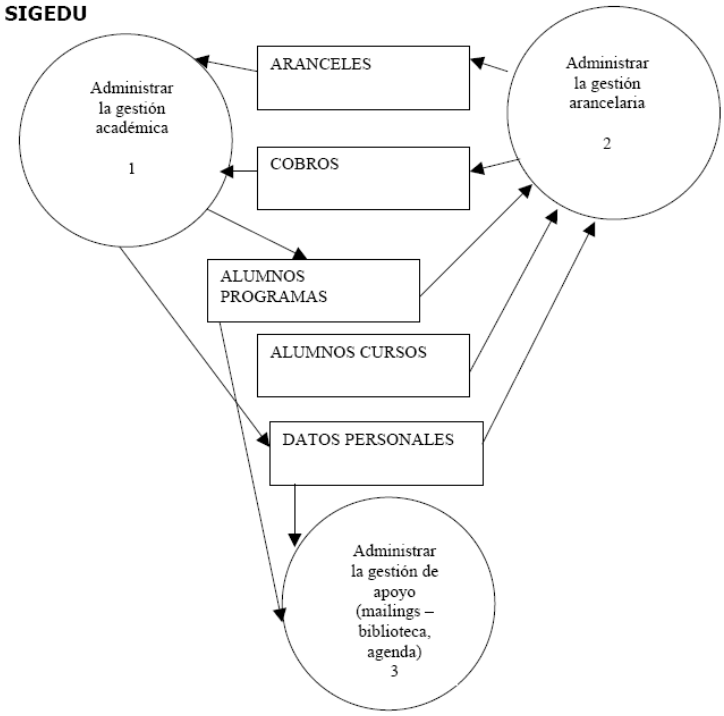


Figura 5.9: Diagrama del sistema (nivel 0) de SIGEDU (Maglione y Placentino, 2001)

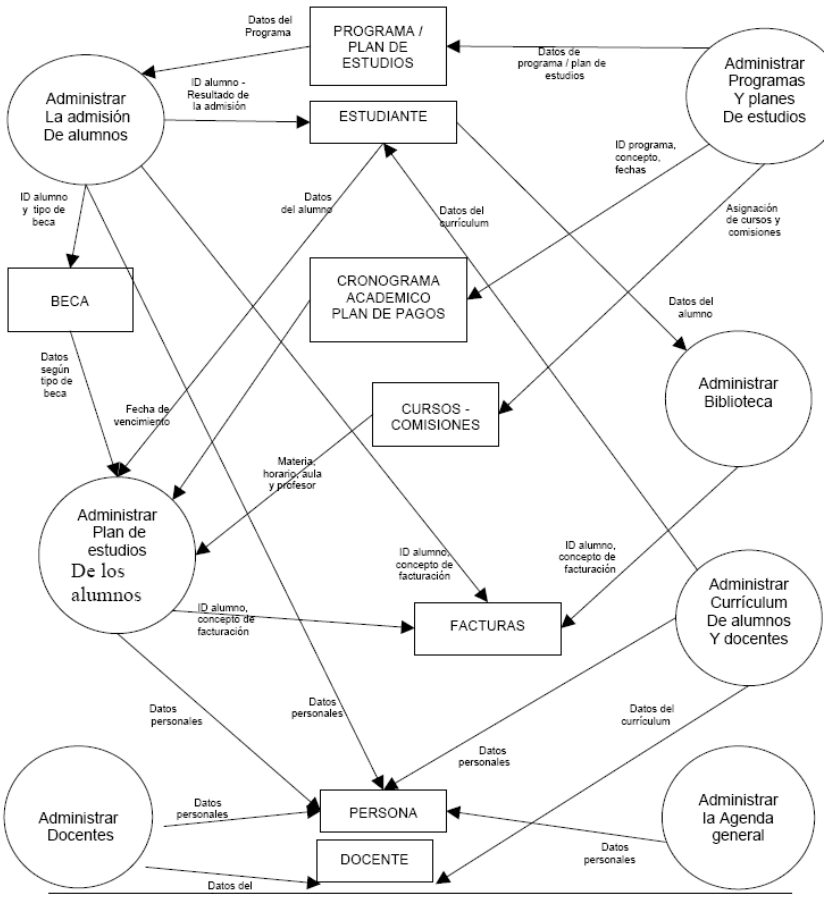


Figura 5.10: Diagrama de nivel 1 Burbuja 1, Administrar Gestión Académica
(Maglione y Placentino, 2001)

V.3.- Diccionario de Datos.

El Diccionario de Datos es una lista organizada de todos los datos pertinentes al sistema con definiciones precisas y rigurosas para que tanto el usuario como el analista tengan un entendimiento común de todas las entradas, salidas, componentes de almacenes y cálculos intermedios.

Un análisis del dominio de la información puede ser incompleto si solo se considera el flujo de datos. Cada flecha de un diagrama de flujo de datos representa uno o más

elementos de información. Por tanto, el analista debe disponer de algún otro método para representar el contenido de cada flecha de un DFD.

Se ha propuesto el diccionario de datos como una gramática casi formal para describir el contenido de los elementos de información.

El diccionario de datos contiene las definiciones de todos los datos mencionados en el DFD, en una especificación del proceso y en el propio diccionario de datos. Los datos compuestos (datos que pueden ser además divididos) se definen en términos de sus componentes; los datos elementales (datos que no pueden ser divididos) se definen en términos del significado de cada uno de los valores que puede asumir. Por tanto, el diccionario de datos esta compuesto de definiciones de flujo de datos, archivos (datos almacenados) y datos usados en los procesos (transformaciones).

El Diccionario de Datos (DD) es un listado organizado de todos los datos del sistema, con definiciones precisas. En las entradas de un DD tendremos:

- Nombre, significado y composición de los flujos y almacenes que se muestran en un DFD. Cuando un paquete de datos de los almacenes o que viaja por un flujo, es compuesto, se describirá cada una de sus partes, las cuales a su vez, si son complejas deberán aparecer como una nueva entrada del diccionario y así sucesivamente.
- Nombre, significado y composición de las entidades dependientes e independientes que se muestran en el DER (diagrama de Entidad-Relación).

Notación.

Existen varias notaciones alternativas, nosotros adoptaremos la siguiente:

=	**Composición**: "*esta compuesto de*" o "*es equivalente a* "
+	**Inclusión**: y
()	**Opción:** significa que el componente encerrado es opcional, es decir, lo que esta entre paréntesis puede estar presente o ausente.
{}	**Iteración**: cero o más ocurrencias de lo que se encuentra entre las llaves.
[]	**Selección:** selección de una de las opciones encerradas entre corchetes y separadas por el símbolo "|".
texto	**Comentario:** El texto entre los dos asteriscos es un comentario aclarativo a una entrada del DD.
@	**Identificador**: Se utiliza para señalar un campo o conjunto de campos que identifican cada ocurrencia de un almacén.
|	separa opciones alternativas de construcción.

Ejemplo.

nombre=*nombre de una persona*
Primer nombre + (segundo nombre) + apellido
primer nombre = {carácter válido}
segundo nombre = {carácter válido}

apellido = {carácter válido}
carácter válido = [A-Z | a-z | ’ | - | |]

Es importante que el diccionario esté completo y sea consistente. Para controlar la completitud debemos verificar que se hayan definido todos los datos presentes en los diagramas; para la consistencia deberemos controlar que no exista más de una definición para un mismo dato y que no hayan entradas fantasmas, es decir, que no correspondan a ningún datos en los diagramas ni tampoco sean parte componente de otro dato definido en el diccionario.

V.4.- Diagramas Entidad-Relación (DER).

Sirven para modelar un almacenamiento de datos. En muchos casos, los datos manipulados por el sistema determinan las acciones que se realizan. Puede ser útil definir los requerimientos concentrándose en los datos en lugar de las funciones. La *abstracción de datos* es una técnica para describir para qué son los datos, en lugar de cuál es su apariencia o como se denominan.

Para describir los datos se utiliza el diccionario de tipo de datos. La idea central es categorizar los datos y agrupar los elementos semejantes.

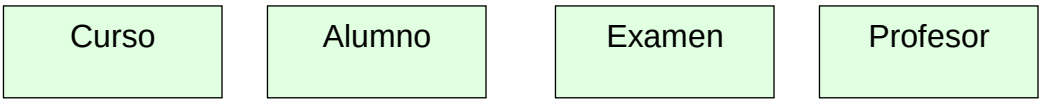
El Modelo Entidad /Relación fue desarrollado por Chen en 1976. Es un modelo muy utilizado en el campo de diseños de bases de datos. Su principal ventaja es que es traducible casi automáticamente a un esquema de bases de datos bajo modelo relacional.

El aspecto de datos es más estable que el funcional en la mayoría de los sistemas; también es mucho más difícil pensar en modelo de datos. La mayor dificultad se presenta en poder establecer la estructura de los objetos y las relaciones entre los mismos.

Elementos del Modelo Entidad/ Relación (MER)

El MER tiene sus propias estructuras que son los diagramas entidad / Relación (DER).

Entidades: Una entidad es un objeto real o abstracto de interés en una organización acerca del cual se puede y se quiere guardar información. Por ejemplo:



Asociado al concepto de entidad surge el de *Ocurrencia de entidad* que es una realización concreta de la misma. Por ejemplo, si las *entidades* son libro, editorial, autor, documento. Las *ocurrencias para la entidad* editorial serían: McGraw-Hill, Addison-Wesley, Alfaomega.

Atributos: Un atributo es una propiedad o característica asociada a una entidad y común a todas las ocurrencias de la misma. Por ejemplo, para la entidad Alumno se pueden tener los

atributos: nombre, grupo y calificación, o para la entidad Curso se pueden tener los atributos: unidad, nombre UEA, Carrera.

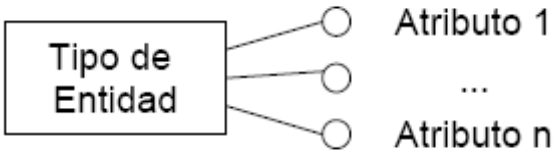
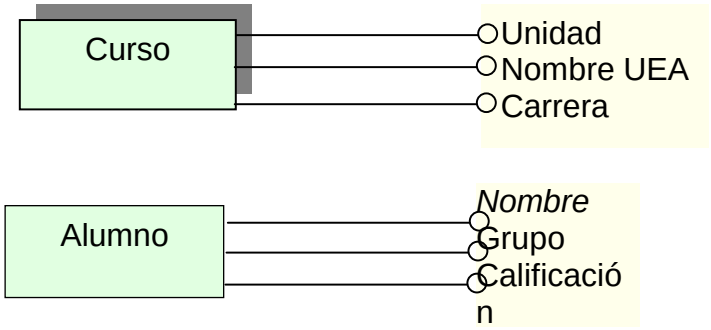
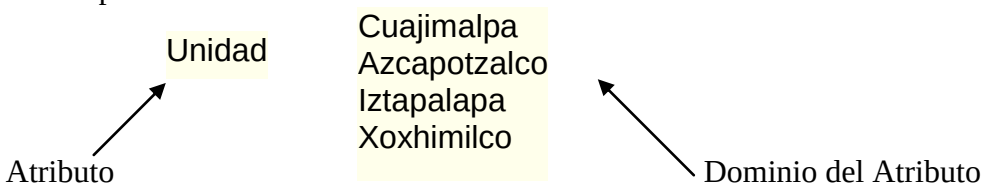


Figura 5.11: Tipo de Entidad y sus Atributos (De Miguel y Piattini, 2004).

Por ejemplo:



Asociado al concepto de Atributo surge el concepto de *dominio*. *Dominio* es el conjunto de valores permitidos para un atributo.



Existen 2 tipos de Atributos:

- *Atributo* ***identificador***: Distingue una ocurrencia de entidad del resto de ocurrencias. Por ejemplo *nombre* del alumno.
- *Atributo* ***descriptor***: Caracteriza una ocurrencia pero no la distingue del resto de ocurrencias de la entidad. Por ejemplo *grupo* y *calificación* del alumno.

Relaciones: Una relación es una asociación entre entidades. Entre dos entidades puede existir más de un tipo de relación. Un *objeto asociativo* es un elemento que sirve para relacionar objetos. Para que exista una instancia del mismo, deben existir instancias de todos los objetos que relaciona. Asociado al concepto de Relación surge el concepto de *ocurrencia de relación* que es la asociación concreta de ocurrencias de entidad de las diferentes entidades. . Por ejemplo: "Autor *escribe* Documento" o "Editorial *edita* Libro" (ver *Figura 5.13*).

Su representación es:

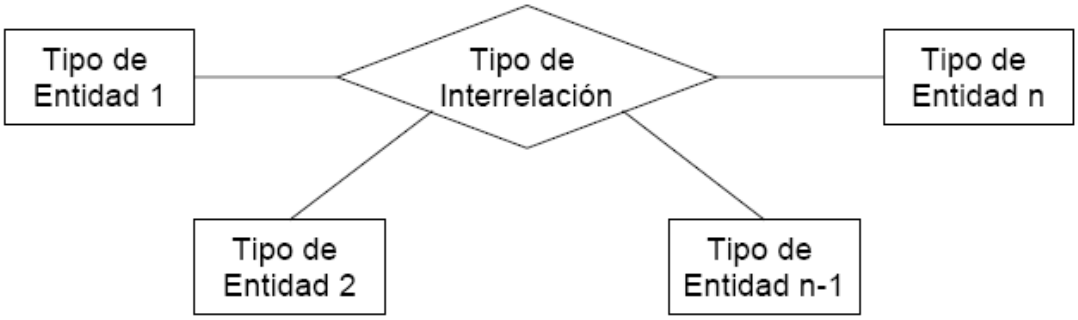


Figura 5.12: Tipo de Interrelación y Tipos de Entidad Relacionadas (De Miguel y Piattini, 2004).

.

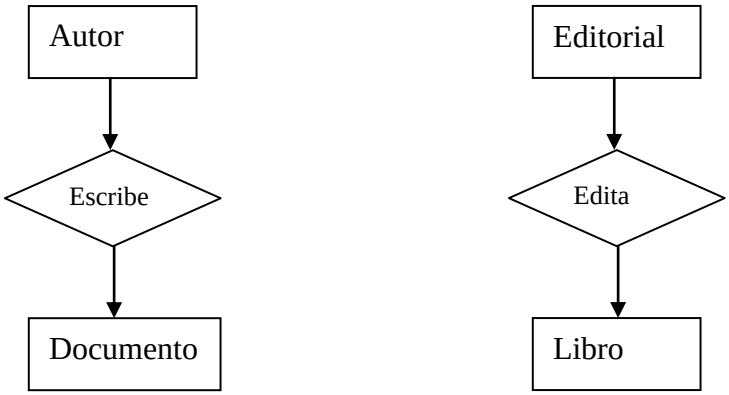


Figura 5.13: Ejemplos de Tipo de Interrelación entre entidades.
Material de clase elaborado por Dr. Ovidio Peña (DMAS)

En la *Figura 5.14*: se muestra un ejemplo de Diagrama de Entidad Relación, para el mismo sistema: SIGEDU expuesto como ejemplo en la sección IV.1.- Diagramas de Flujo de Datos.

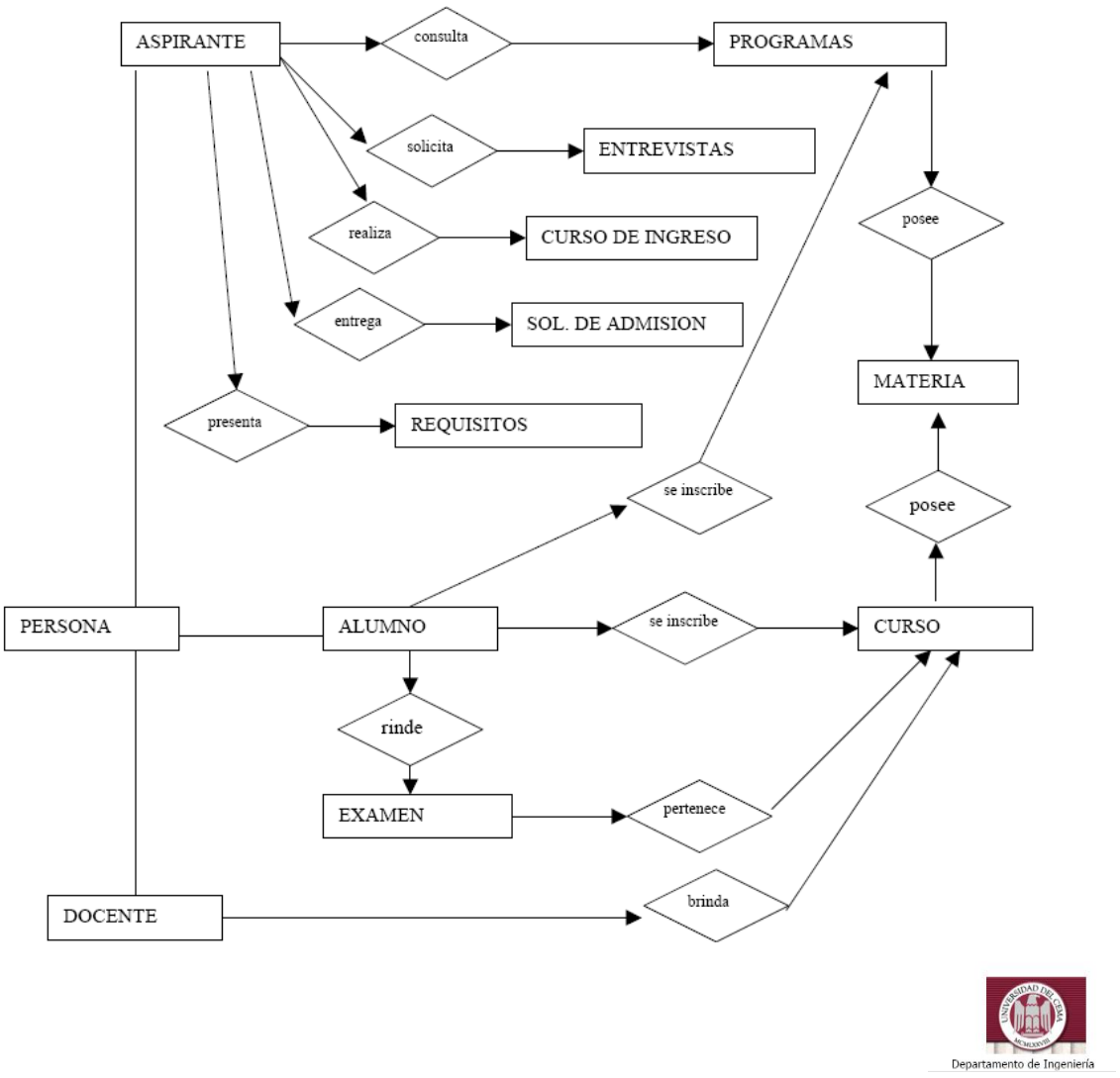


Figura 5.14: Diagrama Entidad-Relación de SIGEDU (Maglione y Placentino, 2001)

En estas notas no se profundiza sobre los Diagramas Entidad-Relación ya que este es un tema de la UEA *Bases de Datos*. Para mayor referencia consultar (Piattini & Castaño, 1998; Piattini et al., 2004).

V.5.- Diagramas de Transición de Estados (DTE).

Se utilizan cuando el aspecto más importante a considerar es el comportamiento del sistema a través del tiempo.

Los Diagramas de Transición de Estados (DTE) es una notación operacional semi-formal que permite construir modelos de comportamiento dependientes del tiempo.

Los principales componentes de un DTE son los estados y las transiciones (flechas que representan cambios de estado). Además, tenemos las condiciones y las acciones asociadas a las transiciones.

Estados.

Para representar cada uno de los estados en los cuales se puede encontrar un sistema normalmente se utiliza un rectángulo o un círculo. Un estado observable del sistema corresponde a períodos en los cuales el sistema esta esperando que algo ocurra en el ambiente externo o esta esperando que alguna actividad que se está realizando en ese momento cambie a otra.

Entonces, decimos que "*un estado representa algún comportamiento del sistema que es observable y que perdura durante algún período de tiempo*".

Algunos ejemplos de estados pueden ser:

- Esperando que el usuario ingrese una contraseña.
- Acelerando el motor.
- Mezclando los ingredientes.

Transiciones.

Para representar los cambios del sistema de un estado a otro, usamos una flecha conectando los dos estados involucrados. Por ejemplo, en la *figura 5.15* se ve un DTE con tres estados y tres transiciones.

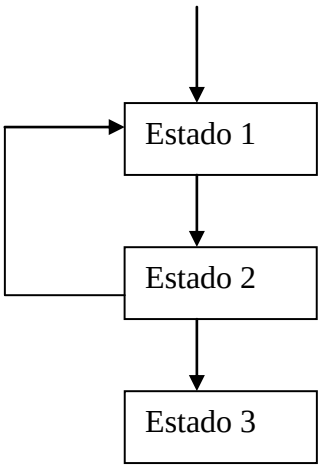


Figura 5.15: Ejemplo de Diagrama de Transición de Estados (DTE).

Las transiciones son las flechas que conectan dos estados involucrados y representan los cambios del sistema de un estado a otro. Nos dicen si se puede alternar entre dos estados o si una vez llegando a un estado, ya no es posible regresar al estado anterior. El sistema modelado por el DTE de la *figura 5.15* puede alternar entre el Estado 1 y el Estado 2, pero

una vez que pasó del Estado 2 al Estado 3 no puede volver más a ninguno de los otros dos estados.

Dos de los estados tienen características particulares. El Estado 1 es el **estado inicial** del sistema. Se le reconoce por la flecha de entrada que no viene de ningún estado anterior. Un DTE sólo puede tener un estado inicial. El Estado 3 es un estado final. Son **estados finales** aquellos estados que no tienen ninguna flecha de salida (o tienen una flecha de salida que no lleva a ningún otro estado), es decir, aquellos estados en los que una vez que se entró no se puede salir. Pueden existir múltiples estados finales. Estos son mutuamente excluyentes. Esto significa que no se puede terminar en dos o más estados finales, es decir, si se termina en un estado es imposible terminar en otro al mismo tiempo.

Condiciones y acciones.

Existen dos elementos más asociados a una transición: las condiciones que se deben satisfacer para que se produzca un cambio de estado y la s acciones que el sistema lleva a cabo cuando se realiza el cambio de estado. Estos dos elementos se representan como se muestra en la *figura 5.16*.

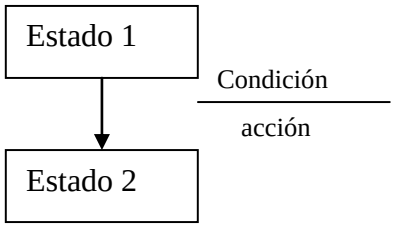


Figura 5.16: Condiciones y acciones en un DTE.

Particionando un DTE.

Al igual que sucede con los DFDs, un DTE puede ser tan complejo que no sea posible visualizarlo cómodamente en una página. En este caso podemos descomponerlo por niveles. Cualquier estado complejo puede dar origen a un nuevo nivel el cual muestre un nuevo DTE con un estado inicial y estados finales.

El estado inicial corresponde al estado de nivel superior, es decir, se entra en el estado inicial del DTE de nivel inferior cuando se entra al correspondiente estado compuesto del nivel superior. Los estados finales del DTE de nivel inferior corresponden a las condiciones de salida del nivel superior. En la *figura 5.17* se muestra un DTE de nivel superior que modela el comportamiento de un cajero automático y en la *figura 5.18* se muestra un DTE de nivel inferior que ilustra como opera el estado compuesto CONSULTAS.

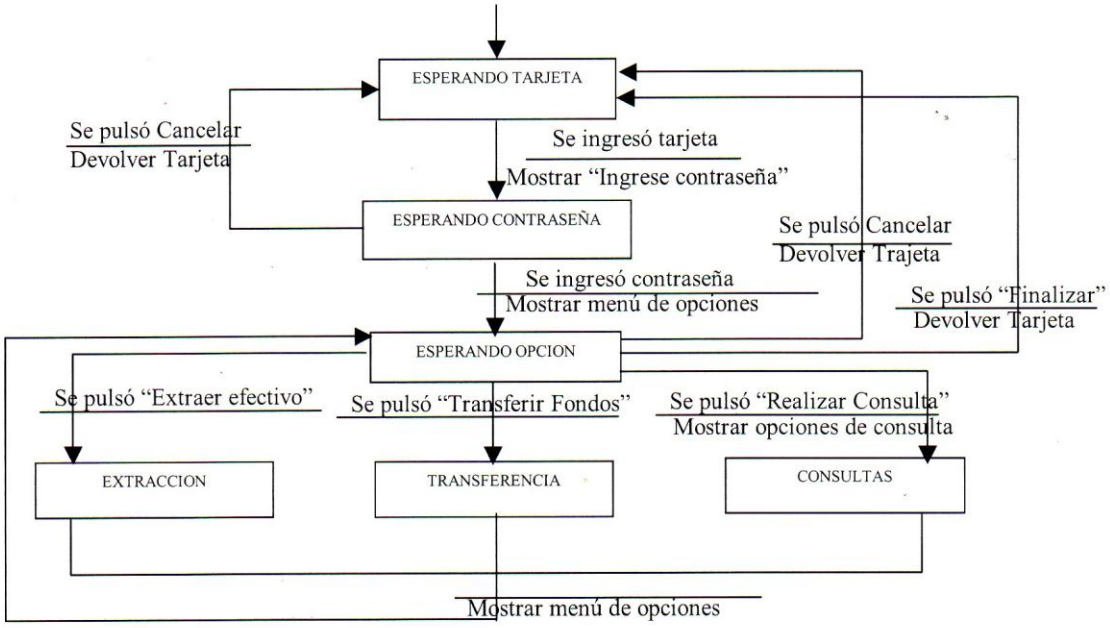


Figura 5.17: DTE de nivel superior para un cajero automático (Yourdon, 1993)

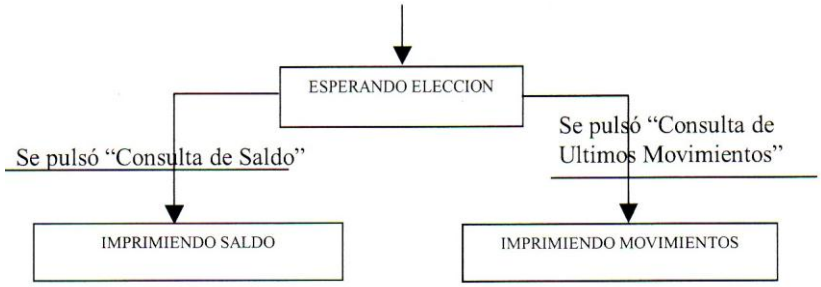


Figura 5.18: DTE de nivel inferior, correspondiente al estado CONSULTAS (Yourdon, 1993).

Ejemplos.- Para enriquecer el contenido de esta sección, se muestran más ejemplos de Diagramas de Transición de Estados. Cabe hacer notar que existen ligeras variaciones en la manera de expresar los DFD, DER y DET, esto se debe a que esta disciplina es relativamente nueva y todavía no existe un estándar universal, sin embargo, en esencia, se representa lo mismo.

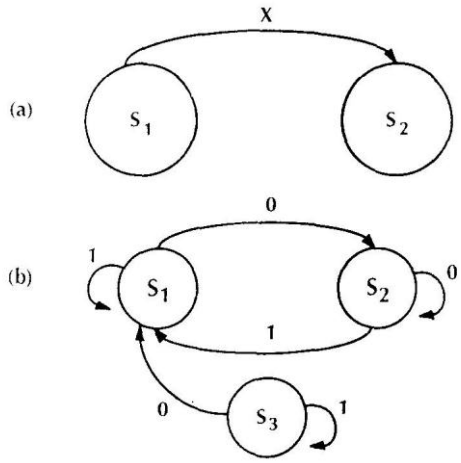


Figura 5.19: más ejemplos de Diagramas de Transición de Estados (Pfleeger 2002)

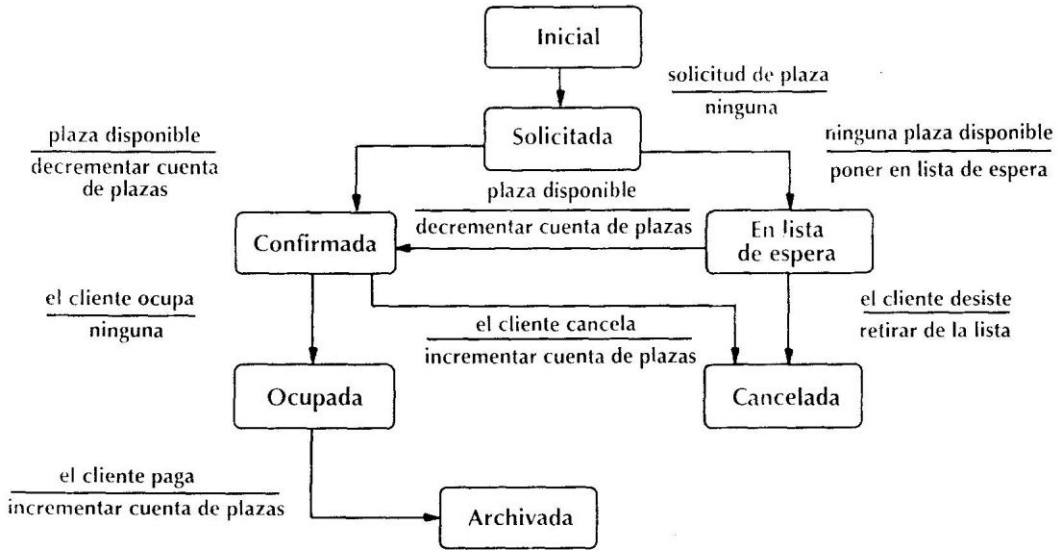


Figura 5.20: Diagrama de transiciones para reservas de hotel (Pleeger 2002).

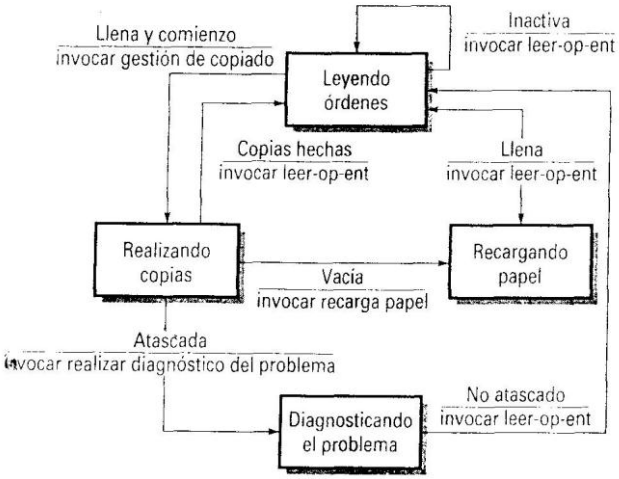


Figura 5.21: Diagrama de transición de estados simplificado para el software de una fotocopiadora (Presuman 2002)

Nota: Los Diagramas de transición de estado son especialmente útiles para ilustrar cuando un sistema reacciona de diferente manera ante un mismo estímulo, esta reacción depende del estado en el que se encuentra el sistema.

V.6.- Balanceo de Modelos.

Cuando reunimos todos los modelos del sistema, corremos el riesgo de que aparezcan inconsistencias entre ellos. Esto suele suceder cuando se trata de proyectos particularmente grandes, donde distintos grupos de personas han trabajado sobre diferentes modelos. Una especificación estructurada, en la cual se ha verificado que todos los modelos sean consistentes entre sí se dice que está *balanceada*.

Hay dos tipos de errores comunes que se suelen detectar en la actividad de balanceo:

- Uno es una *definición faltante* de algún elemento; por ejemplo un almacén de datos definido en un DFD que no se encuentra en el DD.
- El otro tipo son las *inconsistencias entre modelos*, la misma realidad se describe de maneras contradictorias en modelos diferentes.

Balancear un modelo de procesos es comprobar que la información que entra y sale de un proceso de nivel n es consistente con la información que entra y sale del DFD en el que se descompone.

Balanceo del DFD con el DD.

- Cada flujo de datos y cada almacén de datos deben estar definidos en el DD. En caso contrario se dice que el dato está *indefinido*.

- Cada dato y almacén que se define en el DD debe encontrarse en alguna parte del DFD. Si no aparece se dice que es un dato *fantasma*.

Balanceo del DFD con el DER.

- Cada almacén del DFD debe corresponderse con una entidad dependiente o independiente del DER.
- Los nombres de objetos en el DER deben coincidir con los nombres de los almacenes de datos del DFD. La convención a seguir es usar el plural para los nombres de los almacenes en el DFD y el singular para las entidades en el DER. Por ejemplo, si en el DFD tenemos un almacén CLIENTES, en el DER deberá existir una entidad CLIENTE; esto lleva a una definición en el diccionario de datos como sigue:
  CLIENTES = {CLIENTE}
  CLIENTE = nombre + domicilio + teléfono +….
  Nombre = ….

Capítulo VI: Artefactos de modelado para el Desarrollo Orientado a Objetos.

VI.1.- Metodologías orientadas a objetos para el desarrollo de software.

En las *Metodologías Orientadas a Objetos* para el desarrollo de sistemas la unidad básica de construcción es la *clase*, es decir, modelan a un sistema en términos de objetos.

Se identifican inicialmente los objetos del sistema para luego especificar su comportamiento. Existen un gran número de metodologías orientadas a objetos. Éstas utilizan diferentes *herramientas de modelado*, las cuales se detallan en la sección VI.2. La *Tabla 6.1* contiene las principales metodologías para el desarrollo de sistemas Orientado a Objetos. En estas notas haremos hincapié en la metodología del *Proceso Unificado.*

SIGLAS	MÉTODO
RDD	Responsibility-Driven Design
CRC	Tarjetas Clase-Responsabilidad-Colaboración
OOAD	Object-Oriented Análisis and Design
OOD	Object-Oriented Design
OMT	Object Modeling Technique
OOSE	Object Oriented Software Engineering
OOK/MOSES	Object-Oriented Knowledge
OOSA	Object-Oriented System Analysis
OBA	Object Behavior Analysis
OORA	Object-Oriented Requirements Analysis
Synthesis	Synthesis Method
OOSD	Object Oriented System Development
OOAD/ROSE	Object-Oriented Analysis & Design
FUSION	Object-Oriented Development
UP	Unified Software development Process

Tabla 6.1: Diferentes métodos de desarrollo de software orientado a objetos (Weitzenfeld, 2004).

Las principales metodologías orientadas a objetos para el desarrollo de software son:

- Modelado y Diseño Orientado a Objetos (OMT: Object Modeling Technique) [Rumbaugh, 1996].
- Análisis y Diseño Orientado a Objetos (OOAD: Object-Oriented Analysis & Design) [Booch, 1998].
- El Proceso Unificado (UP: Unified Software development Process) [IBM, Rational Unified Process].

El Proceso Unificado.- El Rational Unified Process (RUP), llamado en español el Proceso Unificado Racional, reúne elementos de todos los modelos de procesos genéricos (cascada, evolutivo, reutilización) y plantea buenas prácticas para la especificación y el diseño.
El proceso unificado se aplica al diseño orientado a objetos. Utiliza una combinación de desarrollo incremental e iterativo.

Reúne elementos de todos los modelos de procesos genéricos (cascada, evolutivo, reutilización) y plantea buenas prácticas para la especificación y el diseño. El proceso unificado se describe desde tres perspectivas:

1.- *Una perspectiva dinámica*.- Muestra las fases del modelo sobre el tiempo.
2.- *Una perspectiva estática*.- Muestra las actividades que tienen lugar durante el proceso de desarrollo, se denominan *flujos de trabajo.*
3.- *Una perspectiva práctica*.- Sugiere buenas prácticas a utilizar durante el proceso.

Las fases del Proceso Unificado están relacionadas con asuntos de negocios más que con asuntos técnicos. Considera la cuatro "P" del desarrollo de software: Personas, Proyecto, Producto y Proceso.

- Personas: El cliente, el usuario y el desarrollador.
- Proyecto: Todo el proceso de producir y asegurar la calidad del producto.
- Producto: El sistema a ser desarrollado.
- Proceso: El que se sigue para el desarrollo del producto final.

A continuación se describe cada una de las fases del Proceso Unificado desde la **perspectiva dinámica**:

1.- *Inicio*. El objetivo de esta fase es establecer un caso de negocio para el sistema. Se deben identificar todas las entidades externas (personas y sistemas) que interactuarán con el sistema y definir estas interacciones. Esta información se utiliza para evaluar la aportación que el sistema hace al negocio. Si la aportación es de poca importancia, se puede cancelar el proyecto después de esta fase

2.- *Elaboración.* Los objetivos son: desarrollar una comprensión del dominio del problema, establecer un marco de trabajo arquitectónico para el sistema, desarrollar el plan del proyecto e identificar los riesgos clave del proyecto. Al terminar esta fase, se debe tener un modelo de los requerimientos del sistema (se especifican los casos de uso UML), una descripción arquitectónica y un plan de desarrollo de software.

3.- *Construcción*. Comprende fundamentalmente el diseño del sistema, la programación y las pruebas. Durante esta fase se desarrollan e integran las partes del sistema. Al terminar esta fase, se debe tener un sistema de software operacional y la documentación correspondiente lista para entregarla a los usuarios.

4.- *Transición*. Consiste en mover el sistema desde la comunidad de desarrollo a la comunidad del usuario y hacerlo trabajar en el entorno real. Esto se deja de lado en la mayor parte de los modelos de proceso de software, pero es en realidad una

actividad de alto costo y a veces problemática. Cuando se termina esta fase se debe tener un sistema de software documentado que funciona correctamente en su entorno operativo.

Cada una de estas fases se divide a su vez en varias iteraciones (la de inicio sólo consta de varias iteraciones en proyectos grandes). Estas iteraciones ofrecen como resultado un *incremento* del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo.

Desde la **perspectiva estática** del Proceso Unificado (UP por sus siglas en inglés), cada iteración mencionada anteriormente consta de varias actividades que tienen lugar durante el proceso de desarrollo. Éstas se denominan *flujos de trabajo* y son similares a las etapas del modelo en cascada, pero además se toman en cuenta otros aspectos.

El Proceso Unificado se diseñó junto con el UML (Lenguaje de Modelado Unificado, por sus siglas en inglés) que es un lenguaje de modelado orientado a objetos. El Proceso Unificado define los componentes que se utilizarán para construir el sistema y las interfaces que conectarán los componentes utilizando el UML. El UP se basa en la especificación de requerimientos de un sistema mediante *casos de uso.* En el capítulo V se estudiará con más detalle los casos de uso, por el momento diremos que un *caso de uso* es: "una secuencia de eventos iniciada por el usuario".

En la *tabla 6.2* se muestran los flujos de trabajo estáticos en el proceso unificado.

Flujo de trabajo	*Descripción*
Modelado del negocio	Los procesos del negocio se modelan utilizando casos de uso de negocio
Requerimientos	Se definen los actores que interactúan con el sistema y se desarrollan casos de uso para modelar los requerimientos del sistema
Análisis y diseño	Se crea y documenta un modelo de diseño utilizando modelos arquitectónicos, modelos de componentes, modelos de objetos y modelos de secuencias.
Implementación	Los componentes del sistema se estructuran y se implantan en subsistemas. La generación automática de código de los modelos del diseño ayuda a acelerar este proceso.
Pruebas	Son un proceso iterativo que se lleva a cabo conjuntamente con la implementación. Cuando se termina la implementación, se deben hacer las pruebas del sistema.
Despliegue	Se crea un "release" del producto, se distribuye a los usuarios y se instala en su lugar de trabajo.
Gestión de configuración y cambios	Este flujo de trabajo de soporte gestiona los cambios del sistema.

Gestión del proyecto	Este flujo de trabajo de soporte gestiona el desarrollo del sistema
Entorno	Este flujo de trabajo se refiere a hacer herramientas software apropiadas disponibles para los equipos de desarrollo de software.

Tabla 6.2: flujos de trabajo estáticos en el proceso unificado (Sommerville, 2005).

En cuanto a la **perspectiva práctica**, se recomiendan seis buenas prácticas aconsejables en el desarrollo de sistemas:

1.- *Desarrollar el software de forma iterativa*. Planificar incrementos del sistema basándose en las prioridades del usuario y desarrollo. Entregar las características del sistema que tengan la más alta prioridad al inicio del proceso de desarrollo.

2.- *Gestionar los requerimientos*. Documentar explícitamente los requerimientos del cliente y mantenerse al tanto de los cambios en estos requerimientos. Analizar el impacto de los cambios en el sistema antes de aceptarlos.

3.- *Utilizar arquitecturas basadas en componentes*. Estructurar la arquitectura del sistema en componentes reutilizables en la mayor medida posible.

4.- *Modelar el software visiblemente*. Utilizar modelos gráficos UML para presentar vistas estáticas y dinámicas del software.

5.- *Verificar la calidad del software*. Asegurar que el software cumple con los estándares de calidad de la organización.

6.-*Controlar los cambios del software*. Gestionar los cambios del software usando un sistema de gestión de cambios y procedimientos y también herramientas de gestión de configuraciones.

El Proceso Unificado no es un proceso apropiado para todos los tipos de desarrollo, sin embargo representa una nueva generación de procesos genéricos. Las innovaciones más importantes son la separación de fases y los flujos de trabajo, y el reconocimiento de que la utilización del software en un entorno de usuario es parte del proceso. Las fases son dinámicas y tienen objetivos. Los flujos de trabajo son estáticos y son actividades técnicas que no están asociadas con fases únicas sino que pueden utilizarse durante el desarrollo para alcanzar los objetivos de cada fase. En la *figura 6.1* se muestra una visión global de cómo se combinan las fases dinámicas del proceso unificado, con los flujos de trabajo estáticos. Además se pueden observar los incrementos en cada fase.

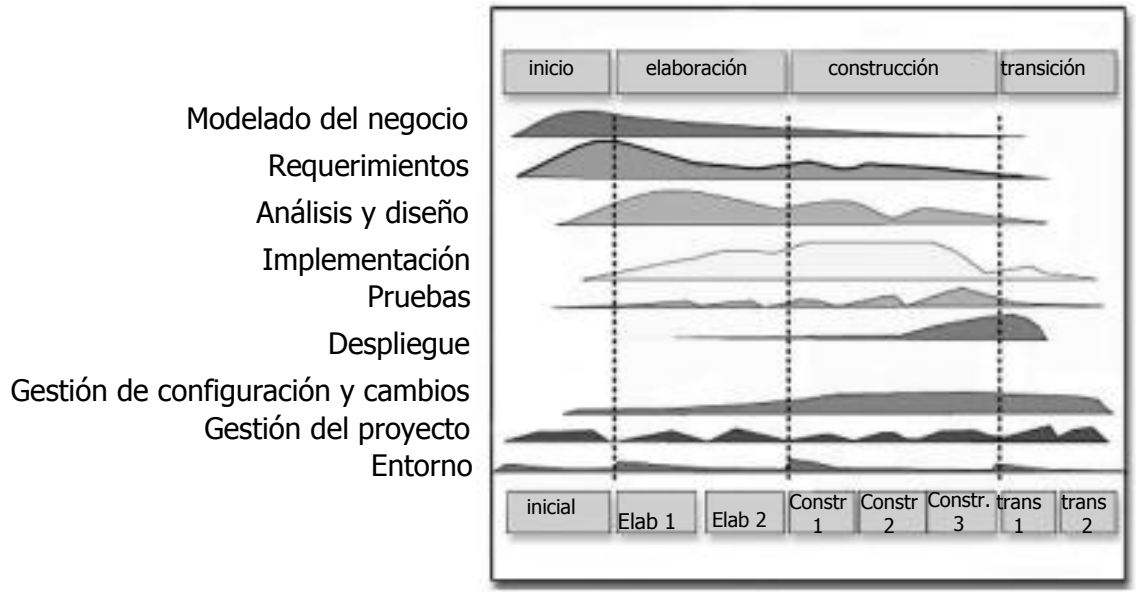


Figura 6.1 Combinación de las fases con los flujos de trabajo en el Proceso Unificado.

VI.2.- Herramientas de modelado: el lenguaje UML.

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, (*Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group), esta asociación se encarga de la definición y mantenimiento de estándares para aplicaciones de la industria de la computación. UML es un lenguaje gráfico que permite especificar, modelar, construir y documentar los elementos que forman un sistema software, principalmente orientado a objetos, sin embargo UML no está diseñado exclusivamente para software orientado a objetos.

A continuación se especifica cada una de las palabras del UML:

- **Lenguaje:** el UML es un lenguaje. Existen reglas sobre cómo deben agruparse los elementos del lenguaje y el significado de esta agrupación.
- **Modelado:** el UML es visual. Mediante su sintaxis se modelan distintos aspectos del mundo real, que permiten una mejor interpretación y entendimiento de éste.
- **Unificado**: unifica varias técnicas de modelado en una única.

La notación UML se deriva y unifica las tres metodologías de análisis y diseño Orientado a Objetos más extendidas:

- Metodología de *Grady Booch* para la descripción de conjuntos de objetos y sus relaciones.
- Técnica de modelado orientada a objetos de *James Rumbaugh* (OMT: Object-Modeling Technique).
- Aproximación de *Ivar Jacobson* (OOSE: Object- Oriented Software Engineering) mediante la metodología de casos de uso (*use case*).

UML no es un método de desarrollo, lo que significa que no sirve para determinar qué hacer en primer lugar o cómo diseñar el sistema, sino que simplemente ayuda a visualizar el diseño y a hacerlo más accesible para otros.

UML se compone de muchos elementos de esquematización que representan las diferentes partes de un sistema de software. Los elementos UML se utilizan para crear diagramas, que representan alguna parte o punto de vista del sistema, UML contiene 13 tipos diferentes de diagramas. Para comprenderlos de manera concreta, es útil clasificarlos por su jerarquía.
Los ***Diagramas de Estructura*** muestran cuales son los elementos que deben existir en el sistema modelado:

- Diagrama de clases
- Diagrama de componentes
- Diagrama de objetos
- Diagrama de estructura compuesta (UML)
- Diagrama de despliegue
- Diagrama de paquetes

Los ***Diagramas de Comportamiento*** muestran lo que debe suceder en el sistema modelado:

- Diagrama de actividades
- Diagrama de casos de uso
- Diagrama de transición de estados

Los ***Diagramas de Interacción*** son un subtipo de diagramas de comportamiento, que están enfocados al flujo de control y de datos entre los elementos del sistema modelado:

- Diagrama de secuencia
- Diagrama de colaboración
- Diagrama de tiempos (UML)
- Diagrama de vista de interacción (UML)

En las tres secciones siguientes se define y ejemplifica un tipo de diagrama por cada uno de los tres grupos mencionados anteriormente; los Diagramas de Clases pertenecen al grupo de los ***Diagramas de Estructura***, los *Diagramas de Casos de Uso* pertenecen al grupo de los ***Diagramas de Comportamiento*** y finalmente, los *Diagramas de secuencia* pertenecen al grupo de los ***Diagramas de Interacción*.**

VI.2.1.- Diagramas de clases.

Los Diagramas de Clases pertenecen al grupo de los ***Diagramas de Estructura***, muestran las diferentes clases que componen un sistema y cómo se relacionan unas con otras. Se dice que los diagramas de clases son diagramas "estáticos" porque muestran las clases, junto con sus métodos y atributos, así como las relaciones estáticas entre ellas: qué clases "conocen" a qué otras clases o qué clases "son parte" de otras clases, pero no muestran los métodos mediante los que se invocan entre ellas. En la *figura 6.2* podemos observar un ejemplo de diagrama de clases. Los *diagramas de clases* sirven para describir los componentes esenciales de la arquitectura de un sistema. A diferencia de los Diagramas de Flujo de Datos, los Diagramas de Clases muestran relaciones de asociación entre clases y no flujo de datos entre ellas.

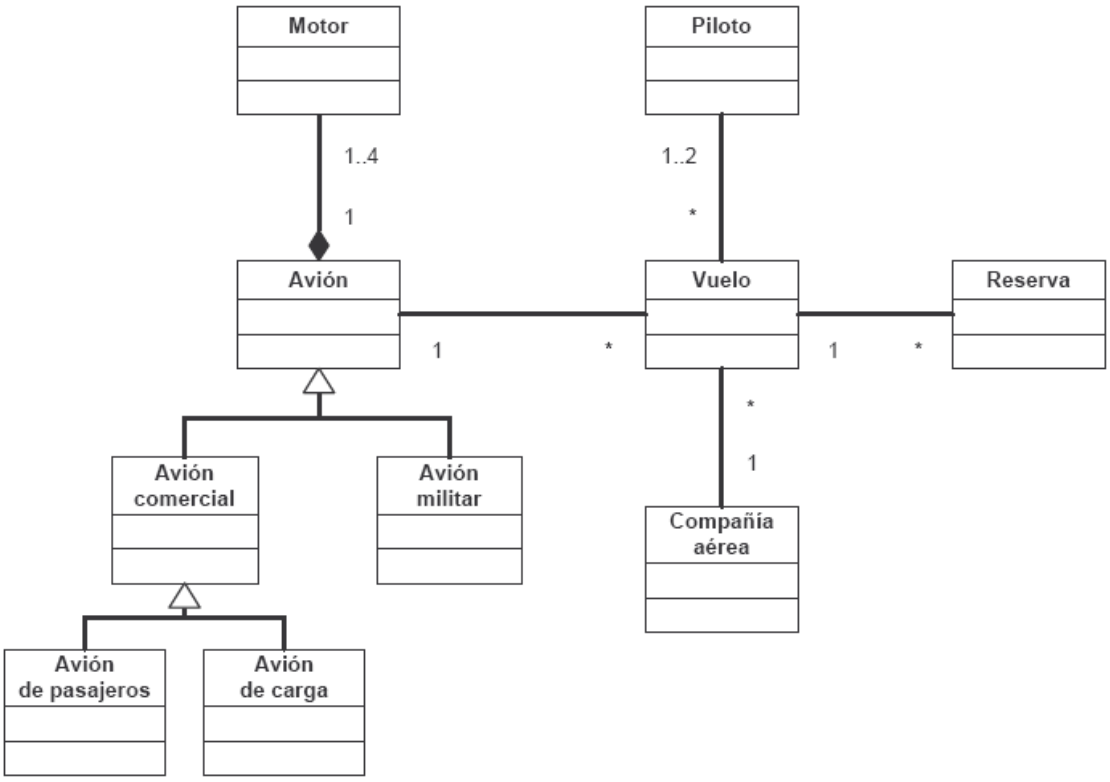


Figura 6.2: Diagrama de clases de un sistema de aviación.

Los *Diagramas de Subsistemas*. Se usan para describir agrupaciones de clases en un sistema

VI.2.2.- Diagramas de casos de uso.

Los *Diagramas de Casos de Uso* pertenecen al grupo de los ***Diagramas de Comportamiento*** Un caso de uso es una interacción entre el sistema y una entidad externa. En su forma más simple, un caso de uso identifica el tipo de interacción y los actores involucrados. Primero se identifican los eventos externos a los que el sistema en desarrollo debe responder, y en segundo lugar, se relacionan estos eventos con los actores y los casos de uso. En las figuras *6.3* y *6.4* podemos observar ejemplos. Los *Diagramas de Casos de Uso* especifican un sistema en término de su funcionalidad. A diferencia de las metodologías estructuradas, los diagramas de casos de uso no se descomponen en funciones de programación

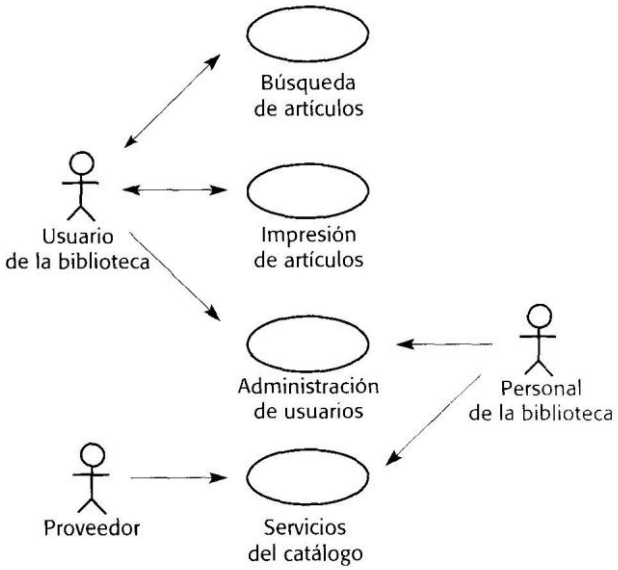


Figura 6.3: Casos de Uso (Sommerville 2005)

Los *Diagramas de Transición de Estado*. Describen los cambios de estado en los objetos, son similares a los DTE de la metodología estructurada, éstos también pertenecen a los Diagramas de Comportamiento.

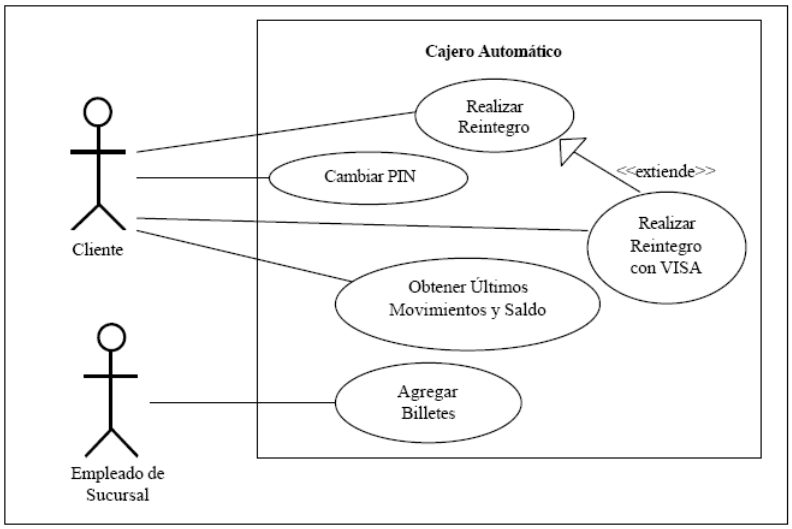


Figura 6.4: Casos de Uso para los actores Cliente y Empleado de Sucursal.

VI.2.3.- Diagramas de secuencia.

Los *Diagramas de secuencia* pertenecen al grupo de los ***Diagramas de Interacción***, sirven para describir los aspectos dinámicos del sistema, mostrando el flujo de eventos entre objetos en el tiempo. Muestran el intercambio de mensajes (es decir la forma en que se invocan) en un momento dado. Ponen especial énfasis en el orden y el momento en que se envían los mensajes a los objetos.

Los objetos están representados por líneas intermitentes verticales, con el nombre del objeto en la parte más alta. El eje de tiempo también es vertical, incrementándose hacia abajo, de forma que los mensajes son enviados de un objeto a otro en forma de flechas con los nombres de la operación y los parámetros. En las *figuras 6.5* y 6.*6* podemos observar ejemplos.

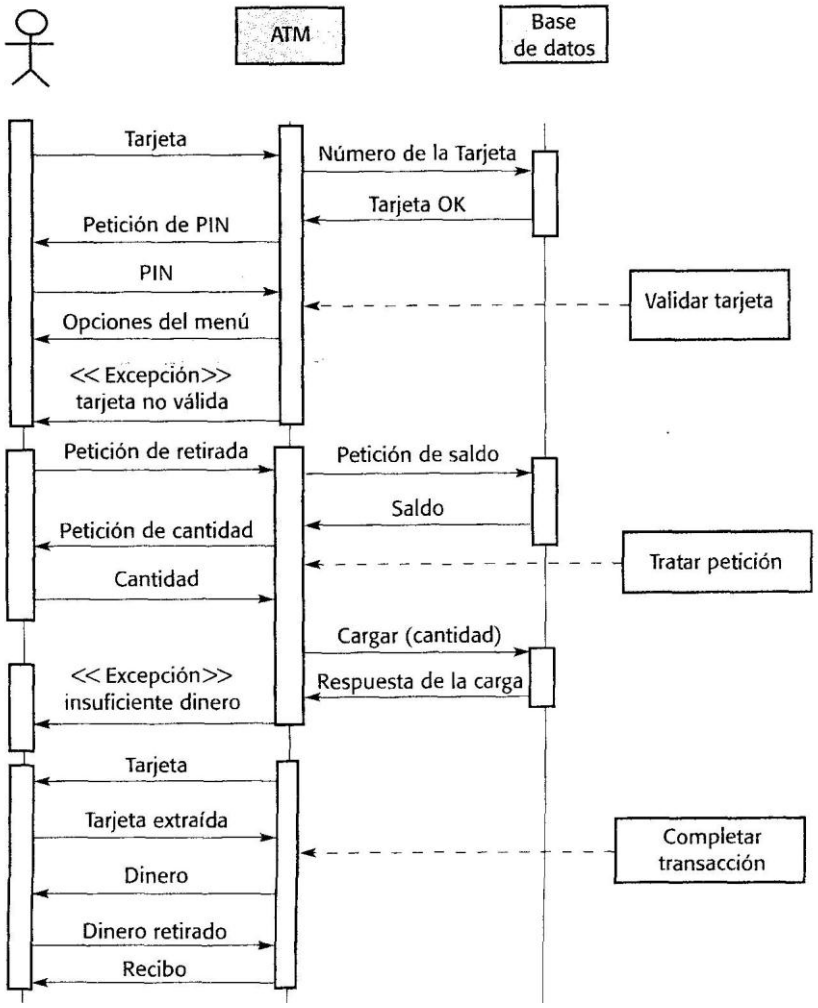


Figura 6.5: Diagrama de secuencia de la retirada de dinero de un cajero automático (Sommerville 2005)

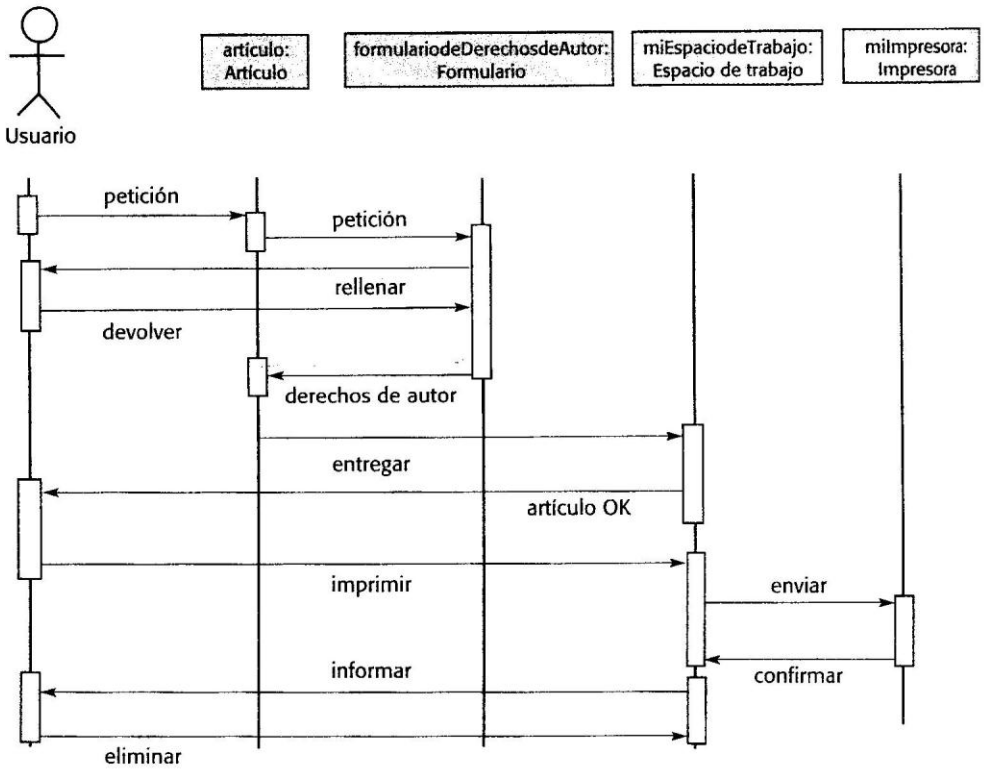


Figura 6.6: Diagrama de secuencia del sistema para la impresión de artículos de una biblioteca (Sommerville 2005).

Los *Diagramas de Colaboración*. Se utilizan para describir la comunicación entre objetos de un sistema y también pertenecen al grupo de al grupo de los ***Diagramas de Interacción***.

VI.3.- Las herramientas CASE.

El rápido incremento en las necesidades de software en las empresas, causó que los desarrolladores de software empezaran a utilizar herramientas automatizadas como apoyo para minimizar la carga.

Hubo un gran auge en la creación de software cuando se empezó a generar con herramientas automatizadas, sin embargo, esto causó serios problemas pues existían millones y millones de líneas de código que necesitaban ser mantenidas y actualizadas. La industria de las computadoras no podía cubrir el incremento de la demanda con los métodos que se estaban usando. Esto fue reconocido como una crisis de software. Para superar este problema en el proceso de desarrollo de software se introdujeron metodologías para crear estándares de desarrollo y se creó un soporte automatizado para el desarrollo y mantenimiento de software llamado **Herramientas CASE.**

Las herramientas CASE se definen como *un conjunto de programas y procesos "guiados", que ayudan a los analistas, desarrolladores, ingenieros de software y diseñadores en una o todas las etapas que comprende un ciclo de vida, con el objetivo de facilitar el desarrollo de software*.

El objetivo general de estas herramientas es acelerar el proceso para el que han sido diseñadas, es decir, para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas. CASE proporciona un conjunto de herramientas semiautomatizadas y automatizadas que están creando una nueva cultura de ingeniería en muchas empresas.

Las herramientas CASE se diseñaron para aumentar la productividad en el desarrollo de software y reducir su costo.

Objetivos de las herramientas CASE:

Automatizar:

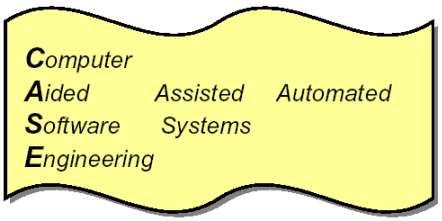
- El desarrollo del software
- La documentación
- La generación del código
- El la búsqueda y corrección de errores
- La gestión del proyecto

Permitir:

- La reutilización del software
- La portabilidad del software
- La estandarización de la documentación

Las herramientas CASE son la unión de las herramientas automáticas de software y las metodologías de desarrollo de software formales.

Variaciones en el significado de CASE.



(Instituto Nacional de Estadística e Informática, colección Cultura Informática 875-99-OI-OTDETI-INEI)

Aunque no es fácil y no existe una forma única de clasificarlas, las herramientas CASE se pueden clasificar en base a los parámetros siguientes:

- Las plataformas que soportan.
- Las fases del ciclo de vida del desarrollo de sistemas que cubren.
- La arquitectura de las aplicaciones que producen.
- Su funcionalidad
- Las fases del ciclo de vida del desarrollo de sistemas que cubren.

El ciclo de vida de una aplicación o de un sistema de información se compone de varias etapas, que van desde la planificación de su desarrollo hasta su implantación,

mantenimiento y actualización. Aunque el número de fases puede ser variable en función del nivel de detalle que se adopte, pueden de modo simplificado, identificarse las siguientes:

- Planeación.
- Análisis y Diseño.
- Implantación (programación y pruebas).
- Mantenimiento y actualización.

Los sistemas Case pueden cubrir la totalidad de estas fases o bien especializarse en algunas de ellas. En este último caso se pueden distinguir sistemas de "alto nivel" (Upper CASE), orientados a la autonomía y soporte de las actividades correspondientes a las dos primeras fases y, sistemas de "bajo nivel" (Lower CASE), dirigidos hacia las dos últimas. Los sistemas de "alto nivel" pueden soportar un número más o menos amplio de metodologías de desarrollo.

Se distinguen 3 grupos de herramientas CASE en base a las fases del ciclo de vida que cubren en el desarrollo del sistema, y son los siguientes:

- *CASE de alto nivel* (Upper CASE) son aquellas herramientas que automatizan o apoyan las fases finales o superiores del ciclo de vida del desarrollo de sistemas como la planificación de sistemas, el análisis de sistemas y el diseño de sistemas.
- *CASE de bajo nivel* (Lower CASE) son aquellas herramientas que automatizan o apoyan las fases finales o inferiores del ciclo de vida como el diseño detallado de sistemas, la implantación de sistemas y el soporte de sistemas.
- *Herramientas CASE Integradas* (Integrated Case) abarcan todas las fases del ciclo de vida del desarrollo de sistemas. Son llamadas también CASE workbench.

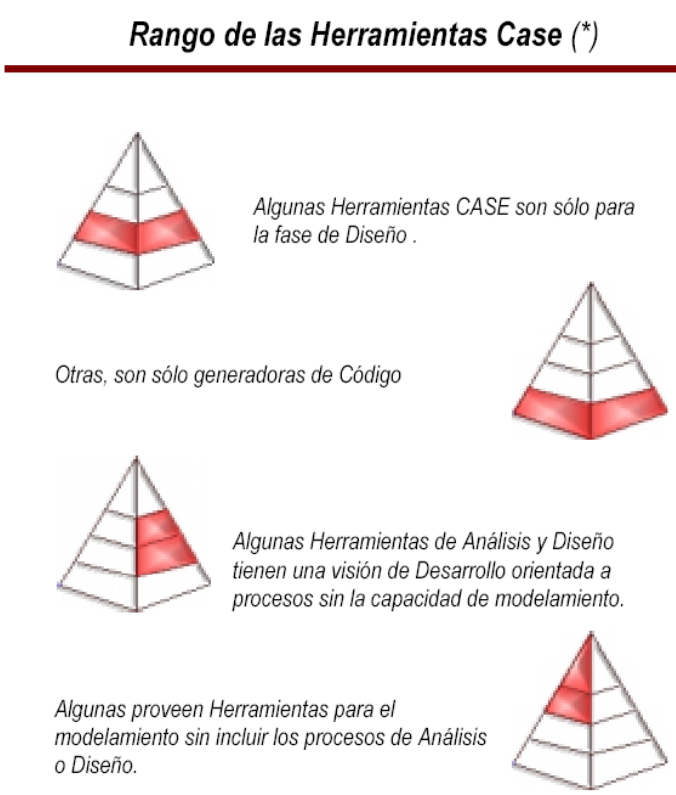


Figura 6.7: Las Herramientas CASE en base a las fases del ciclo de vida que cubren en el desarrollo del sistema.(Instituto Nacional de Estadística e Informática, colección Cultura Informática 875-99-OI-OTDETI-INEI)

Fuera de esta clasificación, es importante mencionar:

- *Juegos de herramientas o Tools-Case*, son el tipo más simple de herramientas CASE. Automatizan una fase dentro del ciclo de vida. Dentro de este grupo se encontrarían las herramientas de reingeniería, orientadas a la fase de mantenimiento

Seleccionar una Herramienta CASE no es una tarea simple. No existe una "mejor" herramienta respecto de otra. Hay numerosas historias respecto al uso de CASE y las fallas que pueden producirse. Las fallas o las respuestas satisfactorias están en relación con las expectativas. Si el proceso de evaluación y selección de las Herramientas CASE falla, entonces la Herramienta no cumplirá con las especificaciones o expectativas del negocio. Esto puede ocurrir durante el proceso de implementación o ejecución del producto.

Tipo de Case	Ventajas	Desventajas
Integrated-Case	-Integra el ciclo de vida. -Permite lograr importantes mejoras de productividad a mediano plazo. - Permite un eficiente soporte al mantenimiento de sistemas. - Mantiene la consistencia de los sistemas a nivel corporativo.	-No es tan eficiente para soluciones simples, sino para soluciones complejas. -Depende del Hardware y Software. -Es costoso.
Upper Case	-Se utiliza en plataforma PC, es aplicable a diferentes entornos. -Menor Costo.	-Permite mejorar la calidad de los sistemas pero no la productividad. -No permite la Integración de ciclo de vida
Lower Case	-Permite lograr importantes mejoras de productividad a corto plazo. -Permite un eficiente soporte al mantenimiento de Sistemas.	-No garantiza la consistencia de los resultados a nivel corporativo. -No garantiza la eficiencia de análisis y diseño. -No permite la Integración de ciclo de vida.

Tabla 6.3: Ventajas y desventajas de las distintas herramientas CASE. .(Instituto Nacional de Estadística e Informática, colección Cultura Informática 875-99-OI-OTDETI-INEI)

Componentes de una herramienta case

Una herramienta CASE se compone de los siguientes elementos:

- Diccionario (Repositorio) donde se almacenan los elementos definidos o creados por la herramienta, la creación y mantenimiento del diccionario se realiza mediante el apoyo de un Sistema de Gestión de Base de Datos (SGBD).
- Meta modelo (no siempre visible), que constituye el marco para la definición de las técnicas y metodologías soportadas por la herramienta.
- Carga o descarga de datos, son facilidades que permiten cargar el repertorio de la herramienta CASE con datos provenientes de otros sistemas, o bien generar a partir de la propia herramienta esquemas de base de datos, programas, etc. que pueden, a su vez, alimentar otros sistemas. Este elemento proporciona así un medio de comunicación con otras herramientas.
- Comprobación de errores, facilidades que permiten llevar a cabo un análisis de la exactitud, integridad y consistencia de los esquemas generados por la herramienta.
- Interfaz de usuario, consta de editores de texto y herramientas de diseño gráfico que permitan, mediante la utilización de un sistema de ventanas, iconos y menús, con la ayuda del ratón, definir los diagramas, matrices, etc. que se incluyen en las distintas metodologías.

Ejemplos de Herramientas Case más utilizadas:

- ER win
- ArgoUML
- Easy Case
- Oracle Designer
- Power Designer
- System Architect
- SNAP

En las *figuras 6.8* a *6.11* se muestran varios ejemplos de aplicaciones de las Herramientas CASE a algunas de las metodologías de análisis vistas en este curso.

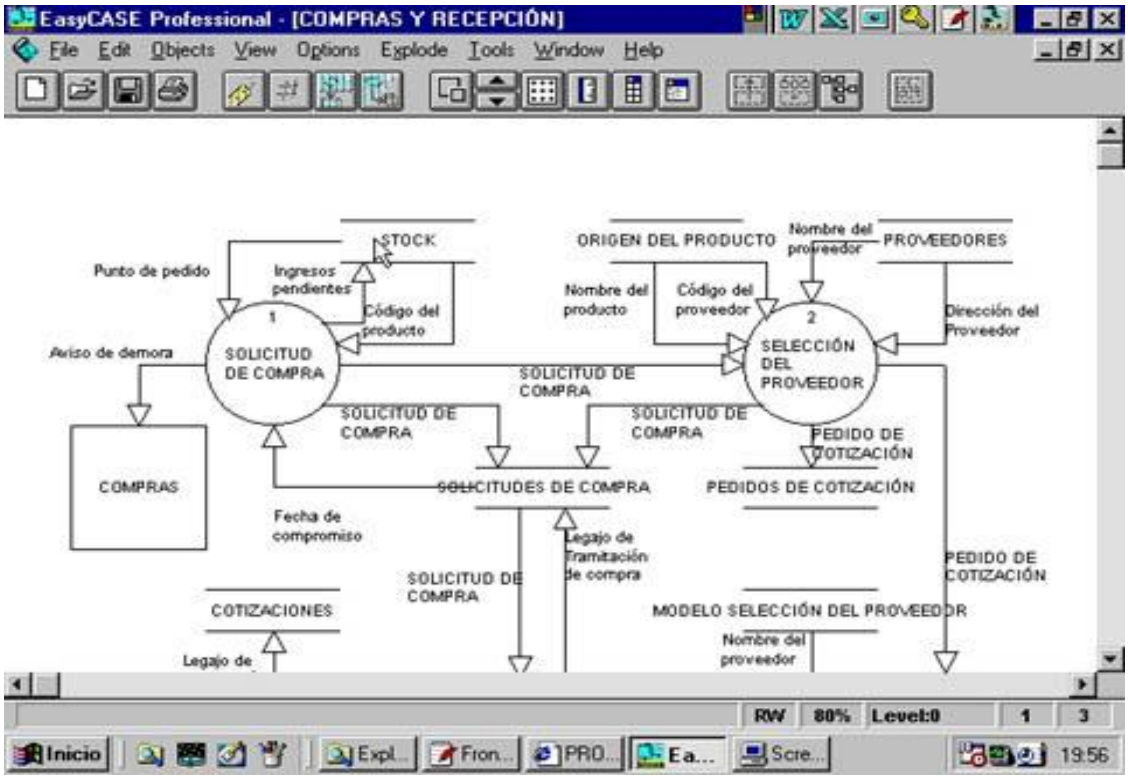


Figura 6.8: Ejemplo de Diagrama de Flujo de Datos hecho con una herramienta CASE (EasyCASE).

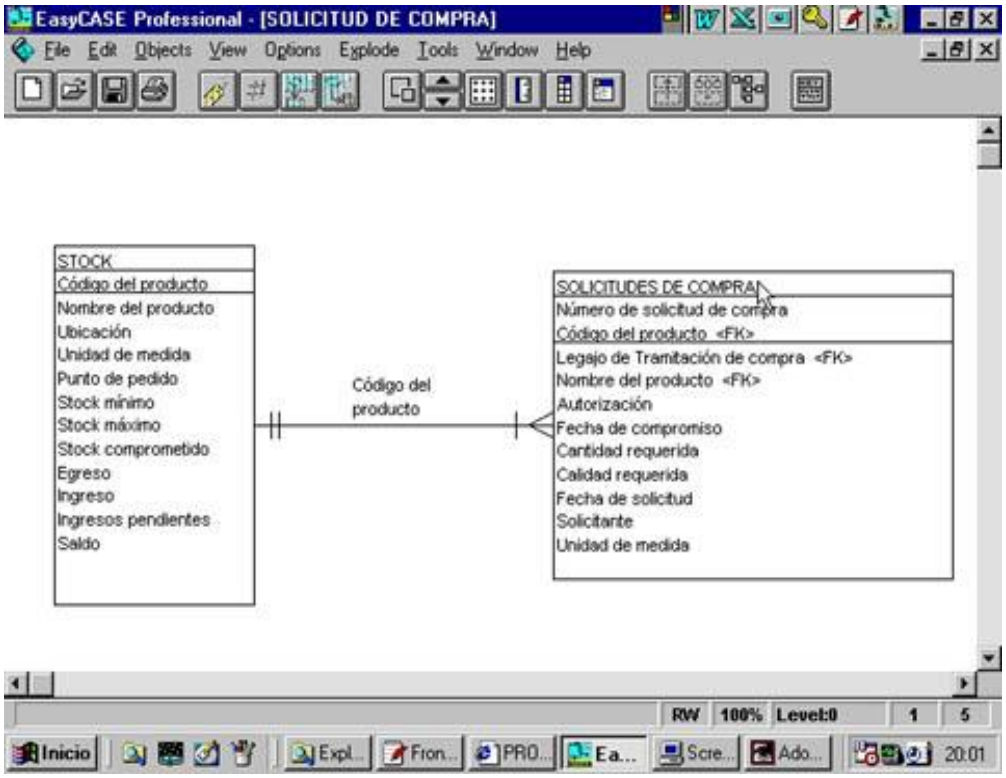


Figura 6.9: Ejemplo de Diagrama Entidad Relación hecho con una herramienta CASE (EasyCASE).

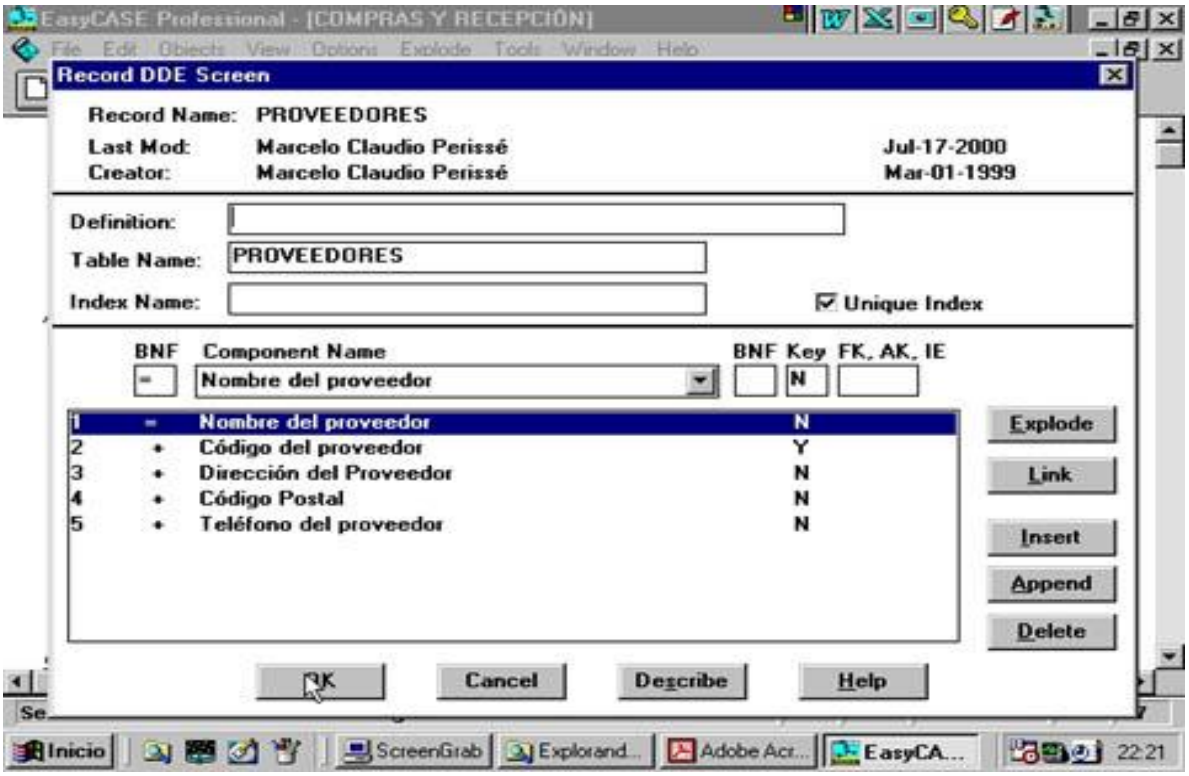


Figura 6.10: Ejemplo de un Diccionario de Datos hecho con una herramienta CASE (EasyCASE).

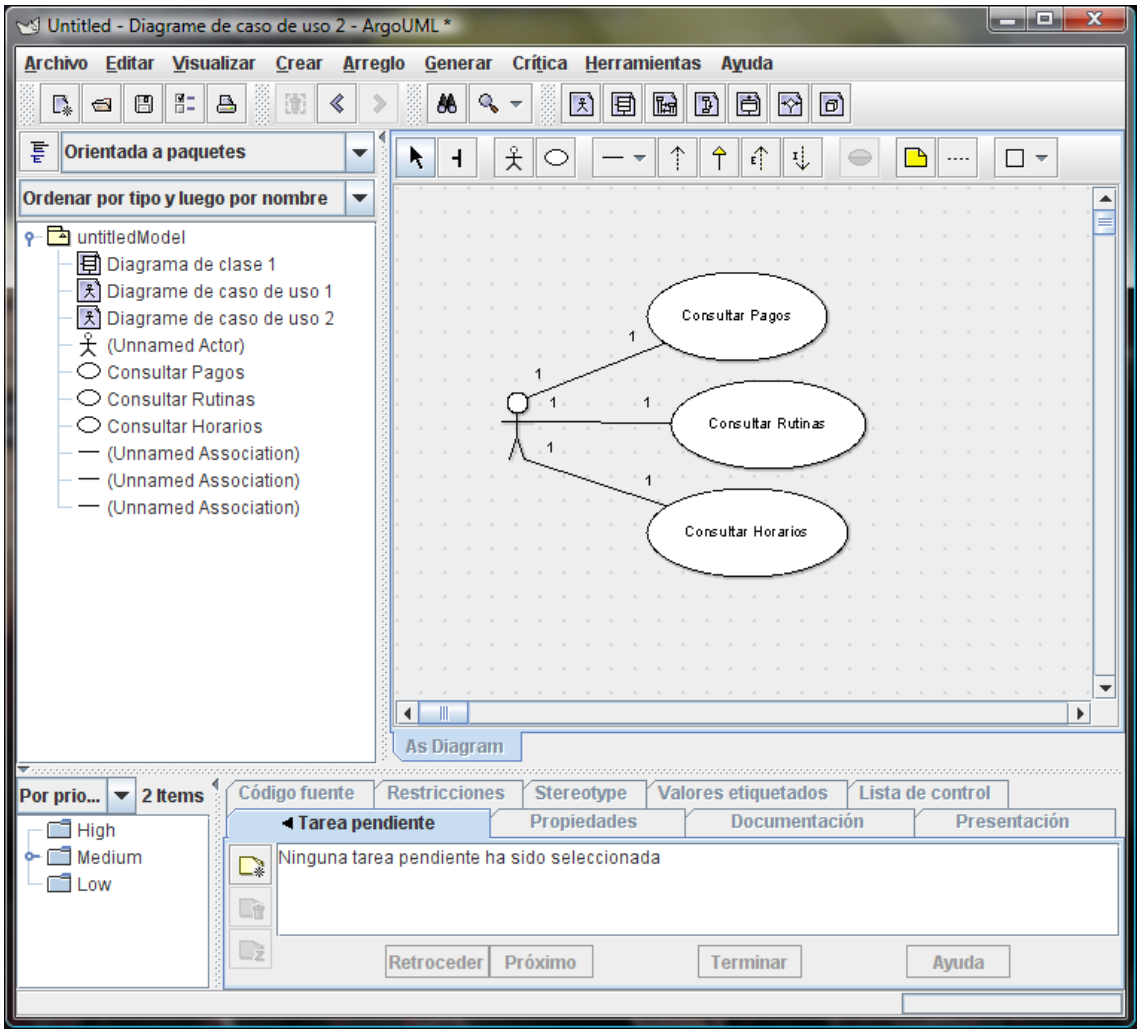


Figura 6.11: Ejemplo de Casos de Uso hecho con una herramienta CASE (ArgoUML).

Capítulo VII: Métodos de comunicación.

Los *métodos de comunicación*, también llamados *Técnicas de recogida de Información*, son procesos mediante los cuales se consigue que los usuarios descubran los requisitos que desean en la aplicación. Son un medio para mejorar la comunicación entre usuarios y desarrolladores del Software.

El proceso de análisis de las técnicas de recogida de la información debe seguir 4 pasos:

- Identificar las fuentes de información relevantes para el proyecto
- Realizar las preguntas apropiadas
- Analizar la información recogida
- Confirmar con los usuarios lo que se ha comprendido de los requisitos.

Las técnicas principales utilizadas son:

- Entrevistas
- Desarrollo conjunto de aplicaciones
- Prototipado
- Observación
- Estudio de la documentación existente en al empresa
- Tormenta de ideas o Brainstorming: Reuniones de usuarios en las que en una primera fase se sugieren toda clase de ideas por muy disparatadas que parezcan,
- Etica: Se busca la satisfacción de los empleados en el trabajo a través de estudios integrales.

VII.1.- Desarrollo Conjunto de Aplicaciones (JAD).

La técnica de Desarrollo Conjunto de Aplicaciones (Joint Application Development: JAD) es una alternativa a las entrevistas. Consiste en una práctica de grupo que se desarrolla durante varios días y en la que participan analistas, usuarios, administradores del sistema y clientes. Está basada en cuatro principios fundamentales: dinámica de grupo, el uso de ayudas visuales para mejorar la comunicación, mantener un proceso organizado y racional y una filosofía de documentación WYSIWYG (What You See Is What You Get, lo que ve es lo que obtiene), es decir, durante la aplicación de la técnica se trabajará sobre lo que se generará. Tras una fase de preparación del JAD para un caso concreto, el equipo de trabajo se reúne en varias sesiones. En cada una de ellas se establecen los requisitos de alto nivel a trabajar, el ámbito del problema y la documentación.

Durante la sesión se discute en grupo sobre estos temas, llegándose a una serie de conclusiones que se documentan. En cada sesión se van concretando más las necesidades

del sistema. Esta técnica presenta una serie de ventajas frente a las entrevistas tradicionales, ya que ahorra tiempo al evitar que las opiniones de los clientes se tengan que contrastar por separado, pero requiere un grupo de participantes bien integrados y organizados.

El Desarrollo conjunto de aplicaciones (JAD) promueve la cooperación y el trabajo en equipo entre usuarios y analistas mediante un conjunto de reuniones de varios días de duración. Esta técnica se utiliza porque:

- Es más difícil cometer errores en la especificación de requisitos cuando la revisa todo el equipo.
- El usuario se siente involucrado por lo que tiene una mayor probabilidad de éxito.
- Se utiliza en lugar de las entrevistas, ya que una entrevista requiere mucho tiempo para su preparación, su realización, y la redacción de un conjunto coherente de requisitos

El Desarrollo Conjunto de Aplicaciones consta de las siguientes fases:

- Adaptación o preparación.
- Sesión JAD: Partiendo de un documento de trabajo se analiza para completar el conjunto de requisitos del sistema.
- Documentación: Consiste en redactar y documentar los detalles, pasarlos en limpio y dar un formato adecuado al texto.

VII.2.- Prototipos.

Un prototipo es una versión preliminar, intencionalmente incompleta o reducida de un sistema. El uso de prototipos es una estrategia que puede aplicarse en casi todas las actividades del proceso de software. Uno de los propósitos de los prototipos es obtener rápidamente la información necesaria para ayudar en la toma de decisiones, otro propósito es ayudar a los desarrolladores a comprender los requerimientos y decidir sobre el diseño definitivo. Las herramientas de prototipado pueden ser programas de dibujo, de presentaciones, hojas de cálculo, etc.

A continuación se mencionan algunos tipos de prototipo que menciona [Pfleeger, 2005]:

- *Prototipo de requisitos*. Permite que los usuarios perciban la funcionalidad del producto final a través del diseño de interfaces o pantallas del sistema. El objetivo es ayudar a aclarar los requisitos y solicitar nuevas ideas.
- *Prototipo de análisis*. Hace posible generar rápidamente una arquitectura general que considere las características principales del sistema de acuerdo a la especificación de requisitos.
- *Prototipo de diseño*. Permite explorar y comprender la arquitectura particular del sistema, para poder evaluar aspectos como cuellos de botella (rendimiento y uso de memoria) o inconsistencias en el diseño.

- *Prototipo vertical*. Ayuda a comprender parte de un problema y a desarrollar su solución completa. Esto se hace generalmente cuando los conceptos básicos no están bien comprendidos; por ejemplo, el seguimiento de cierta metodología.
- *Prototipo de factibilidad*. Demuestra si es posible lograr ciertos objetivos del proyecto; por ejemplo, aplicar una arquitectura particular, conectarse a una base de datos bajo ciertas restricciones de rendimiento, aprender a programar en un lenguaje en un tiempo determinado o predecir los costos de desarrollo de un proyecto.

Existen dos enfoques para la realización de prototipos: el *evolutivo* y el llamado *prototipado desechable*.

El *prototipado evolutivo* se desarrolla para aprender acerca del problema y formar la base de todo o parte del software que se entregará como producto final. El propósito de este prototipo en un ambiente de desarrollo de software es acelerar el proceso de desarrollo al presentar al usuario una versión trabajando del producto final al principio del proceso de desarrollo. A partir de ahí, el esfuerzo de desarrollo se concentra en refinar el proceso acordado. El objetivo es lograr que el usuario acepte que los requerimientos son correctos en una etapa inicial, el prototipo es por lo tanto, interactivo y dirigido por el usuario. El principal peligro del prototipo es el no saber cuando dejar de refinar y determinar que se tiene un producto final.

El *prototipado desechable*.- Es software desarrollado para aprender más sobre un problema o explorar la factibilidad o la conveniencia de las posibles soluciones. Un prototipo desechable es exploratorio y no está pensado para ser utilizado como componente real del software que se entrega al cliente. En prototipo no es un producto de calidad que deba mantenerse a largo plazo. Por el contrario, los prototipos son creados y probados rápidamente, para luego ser desechados. Sin embargo, es común que por presiones de tiempo, se trate de enviar un prototipo al mercado como si éste fuera el producto final. En general, siempre existirá un conflicto entre un desarrollo rápido y un producto de calidad.

Los prototipos tienen éxito cuando:

1. Se tiene claro el propósito del prototipo y se usa de manera adecuada.
2. Se comprende la tecnología a utilizarse y su relación con el proceso de prototipos.
3. Se involucra a tiempo en el proceso a los usuarios finales.
4. Se está dispuesto a repetir el prototipo para comprender mejor la arquitectura básica.

Los prototipos fallan cuando:

1. No se entiende que es un prototipo y cómo debe usarse.
2. No se sabe hasta cuando dejar de evolucionar el prototipo y comenzar de cero.(Puede extenderse demasiado el proceso o terminarse prematuramente).
3. Se cree que un prototipo razonable es un producto aceptable.

4. Los prototipos nunca terminan.

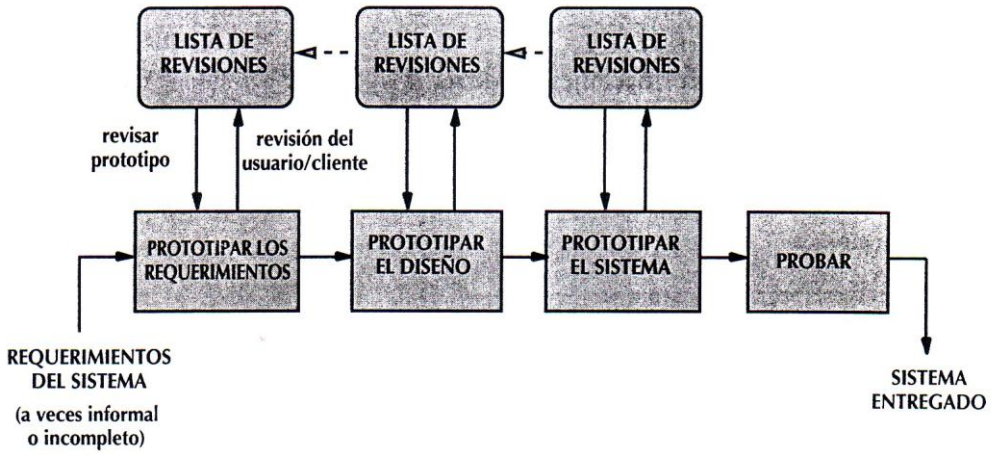


Figura 7.1: El modelo de prototipado (Pleeger 2002)

Para construir un prototipo del software se aplican los siguientes pasos:

PASO 1. Evaluar la petición del software y determinar si el programa a desarrollar es un buen candidato para construir un prototipo.

Debido a que el cliente debe interaccionar con el prototipo en los últimos pasos, es esencial que el cliente participe en la evaluación y refinamiento del prototipo, y que el cliente sea capaz de tomar decisiones de requerimientos de una forma oportuna. Finalmente, la naturaleza del proyecto de desarrollo tendrá una fuerte influencia en la eficacia del prototipo.

PASO 2. Dado un proyecto candidato aceptable, el analista desarrolla una representación abreviada de los requerimientos.

Antes de que pueda comenzar la construcción de un prototipo, el analista debe representar los dominios funcionales y de información del programa y desarrollar un método razonable de partición. La aplicación de estos principios de análisis fundamentales, pueden realizarse mediante los métodos de análisis de requerimientos.

PASO 3. Después de que se haya revisado la representación de los requerimientos, se crea un conjunto de especificaciones de diseño abreviadas para el prototipo.

El diseño debe ocurrir antes de que comience la construcción del prototipo. Sin embargo, el diseño de un prototipo se enfoca normalmente hacia la arquitectura a nivel superior y a los aspectos de diseño de datos, en vez de hacia el diseño detallado.

PASO 4. El software del prototipo se crea, prueba y refina.

Idealmente, los bloques de construcción de software que ya existen se utilizan para crear el prototipo de una forma rápida. Desafortunadamente, tales bloques construidos raramente existen.

PASO 5. Una vez que el prototipo ha sido probado, se presenta al cliente, el cual "conduce la prueba" de la aplicación y sugiere modificaciones.

Este paso es el núcleo del método de construcción de prototipo. Es aquí donde el cliente puede examinar una representación implantada de los requerimientos del programa, sugerir modificaciones que harán al programa cumplir mejor las necesidades reales.

PASO 6. Los pasos 4 y 5 se repiten iterativamente hasta que todos los requerimientos estén formalizados o hasta que el prototipo haya evolucionado hacia un sistema de producción.

Para los casos en los que se desarrolle un prototipo se realiza un manual de usuario preliminar. Puede parecer innecesario realizar un manual de usuario en una etapa tan temprana del proceso de desarrollo, Pero de hecho, este borrador del manual de usuario fuerza al analista a tomar el punto de vista del usuario del software. El manual permite al usuario/cliente revisar el software desde una perspectiva de ingeniería humana y frecuentemente produce el comentario: "La idea es correcta pero esta no es la forma en que pensé que se podría hacer esto". Es mejor descubrir tales comentarios lo más tempranamente posible en el proceso.

VII.2.1.- Prototipos de la Interfaz de Usuario.

Las descripciones textuales y los diagramas, no son adecuados para expresar los requerimientos de las interfaces de usuario. El propósito del prototipado es permitir a los usuarios adquirir una experiencia directa con la interfaz. La mayoría de nosotros encuentra difícil pensar de forma abstracta sobre una interfaz de usuario y explicar directamente lo que deseamos. Sin embargo, cuando se nos presentan ejemplos, es fácil identificar las características que nos gustan y las que no.

La interfaz de usuario consiste en el formato de las pantallas, algunas veces también incluye esquemas de los informes y datos de entrada.

Idealmente, cuando se está construyendo el prototipo de una interfaz de usuario, se debe adoptar un proceso de prototipado en dos etapas:

1. Al principio del proceso, hay que desarrollar prototipos en papel, maquetas de los diseños de las pantallas, y mostrárselos a los usuarios finales.
2. Después, se perfecciona el diseño u se desarrollan prototipos automatizados cada vez más sofisticados y se ponen a disposición de los usuarios para realizar pruebas y simulación de actividades.

La construcción de un prototipo en papel es poco costosa y sorprendentemente efectiva para el desarrollo de prototipos. No se necesita desarrollar ningún software ejecutable y los diseños no tienen por qué hacerse conforme a estándares profesionales.

Hay tres maneras en las que puede realizarse el prototipado de las interfaces de usuario:

1. *Enfoque dirigido por secuencias y comandos*. Si solamente se necesita estudiar ideas con los usuarios, se pueden crear pantallas con elementos visuales como botones y menús, y se asocia una secuencia de comandos con estos elementos. Cuando el usuario interactúa con estas pantallas, se ejecuta la secuencia de comandos y se presenta la siguiente pantalla, que les muestra los resultados de sus acciones.
2. *Lenguajes de programación visuales*. Este tipo de lenguajes, como Visual Basic, incorporan un potente entrono de desarrollo, acceden a una gran variedad de objetos reutilizables y a un sistema de desarrollo de interfaces de usuario que permite crear interfaces de forma rápida, con componentes y secuencias de comandos asociados con los objetos de la interfaz.
3. *Prototipado basado en internet*. Estas soluciones, basadas en navegadores web y en lenguajes como Java, ofrecen una interfaz de usuario hecha. Se añade funcionalidad asociando segmentos de programas en Java con la información a visualizar. Estos segmentos (llamados applets) se ejecutan automáticamente cuando se carga la página en el navegador. Esta es una manera rápida de desarrollar prototipos de interfaces de usuario, sin embargo, existen restricciones inherentes impuestas por el navegador y por el modelo de seguridad de Java.

Evaluación de la interfaz.

Conforme el prototipo se hace más completo, se pueden utilizar técnicas de evaluación sistemática. La evaluación de la interfaz es el proceso de evaluar la forma en que se utiliza una interfaz y verificar que cumple los requerimientos del usuario. Por lo tanto, debe ser parte del proceso de verificación y validación de los sistemas de software.

La evaluación sistemática del diseño de la interfaz de usuario puede ser un proceso caro que implica científicos cognoscitivos y diseñadores gráficos. Es posible que se tenga que diseñar y realizar un número estadísticamente importante de experimentos con los usuarios típicos. Se puede necesitar el uso de laboratorios construidos especialmente con equipos de supervisión. Una evaluación de este tipo no es económicamente viable para sistemas desarrollados por pequeñas organizaciones con recursos limitados.

Existen varias técnicas menos costosas y sencillas en la evaluación de interfaces que pueden identificar deficiencias específicas en el diseño de interfaces:

1. *Cuestionarios que recopilan información de lo que opinan los usuarios de la interfaz*. Las preguntas deben ser precisas en vez de generales. Un ejemplo de una pregunta precisa: “Por favor, indique el valor en una escala del 1 al 5 de cuál es la comprensión de los mensajes de error. Un valor 1 significa muy claro y 5 significa incomprensible”.

2. *La inclusión de código en el software que recopila información de los recursos más utilizados y de los errores más comunes*. Se pueden detectar las operaciones más comunes y reorganizar para que sean más rápidas de seleccionar. Por ejemplo, si se utilizan menús descendentes, las operaciones más frecuentes se deben ubicar en la parte superior del menú y las operaciones destructivas en la parte inferior. Con el código también se pueden detectar los comandos más propensos a errores y modificarlos.
   En este punto es bueno proporcionar a los usuarios un comando que puedan utilizar para enviar mensajes al diseñador de la herramienta. Esto hace que los usuarios sientan que sus opiniones son tenidas en cuenta. Además el diseñador de la interfaz y otros ingenieros pueden obtener una rápida retroalimentación de los problemas particulares.
3. *Videos del uso típico del sistema*. Se pueden grabar las sesiones de usuario para el análisis posterior. Un análisis completo por medio del video es caro y requiere un equipo de evaluación especializado con varias cámaras enfocadas al usuario y a la pantalla. Sin embargo, la grabación en video de algunas operaciones específicas puede ayudar a detectar los problemas. El análisis de las grabaciones permite al diseñador descubrir si la interfaz requiere demasiado movimiento de las manos (un problema con algunos sistemas es que los usuarios deben mover frecuentemente sus manos del teclado al ratón) y ver si son necesarios los movimientos forzados del ojo. Una interfaz que requiera muchos cambios teclado-ratón-teclado puede implicar que los usuarios cometan más errores.
4. *La observación de los usuarios cuando trabajan con el sistema* y "piensan en voz alta" al tratar de utilizar el sistema para llevar a cabo alguna tarea. Esto es, ver los recursos que utilizan, los errores cometidos, etcétera.

A continuación se presenta una tabla con los atributos de uso que debe tener una interfaz, estos atributos son útiles para evaluarla.

ATRIBUTO	DESCRIPCIÓN
Aprendizaje	¿Cuánto tiempo tarde un usuario nuevo en ser productivo con el sistema?
Velocidad de funcionamiento	¿Cómo responde el sistema a las operaciones de trabajo del usuario?
Robustez	¿Qué tolerancia tiene el sistema a los errores del usuario?
Recuperación	¿Cómo se recupera el sistema a los errores del usuario?
Adaptación	¿Está muy atado el sistema a un único modelo de trabajo?

Tabla 7.1: Atributos de uso para una interfaz de usuario.

Ninguna de estas técnicas relativamente simples para la evaluación de la interfaz de usuario es infalible y probablemente no detectan todos los problemas de las interfaces de usuario, sin embargo estas técnicas se pueden utilizar con un grupo de voluntarios si un gran desembolso de recursos antes de que se entregue el sistema. Así se pueden descubrir y corregir muchos de los peores problemas del diseño de las interfaces de usuario.

Ejemplo: En la *figura 7.2* se muestra un ejemplo de tres diferentes interfaces de usuario para solicitar una fecha.

Figura 4.19. Primer prototipo de interfaz de usuario.

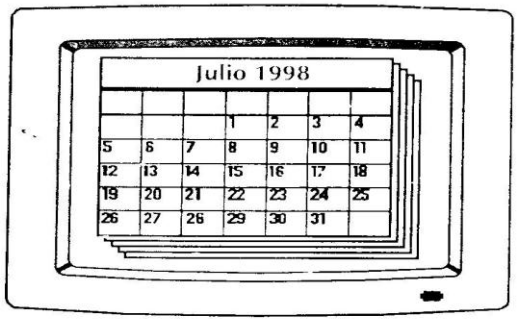


Figura 4.20. Segundo prototipo de interfaz de usuario.

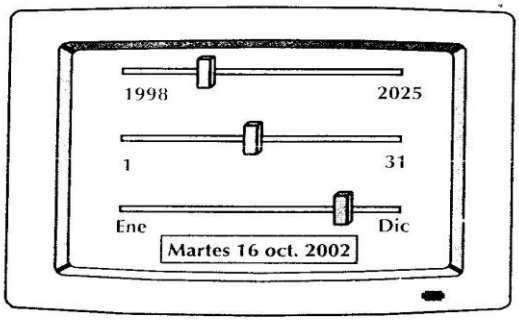


Figura 4.21. Tercer prototipo de interfaz de usuario.

Figura 7.2: Prototipos de interfaz de usuario para ingresar una fecha. (Pleeger 2002).

MOTIVACIÓN FINAL.

"Cualquier actividad se vuelve creativa si el autor se preocupa de hacerlo bien o de hacerlo mejor " John Updike.

"Las personas obtienen tanta satisfacción (o más) del proceso creativo que del producto final. Un artista disfruta con las pinceladas del resultado enmarcado. Un escritor disfruta con la búsqueda de la metáfora adecuada al igual que con el libro final. Un profesional creativo del software debe también obtener tanta satisfacción de la programación como del producto final." Roger Pressman.

Apéndice A: Metodología propuesta para el laboratorio de la UEA "Análisis de Requerimientos"

Introducción

La enseñanza del Proceso de Análisis de Requerimientos a los estudiantes de licenciaturas relacionadas con la Ingeniería de Software es, aparentemente, algo sencillo. Por lo general se imparten los principios básicos de la Ingeniería de Requerimientos en clases teóricas y, se solicita a los alumnos que elaboren los requerimientos de un proyecto en particular para que apliquen los conocimientos adquiridos. Si bien lo anterior es una metodología bastante razonable, en la práctica se presenta un problema fundamental, hay muy pocos ejemplos de requerimientos para sistemas de software en los libros especializados. Este trabajo parte de la hipótesis de que las clases teóricas y los ejemplos de los libros no son suficientes para que los alumnos puedan elaborar correctamente un documento de Especificación y sugiere una metodología que pretende contrarrestar este problema.

Descripción de la Metodología propuesta

La metodología expuesta en este trabajo tiene dos propósitos, el primero, es generar una dinámica en la que los alumnos se vean en la necesidad de investigar lo que el cliente requiere; y el segundo es proveerlos con ejemplos de Especificaciones de Requerimientos de tal forma que tengan la oportunidad de practicar y que lo visto en la clase teórica no quede tan abstracto. La metodología consta de tres fases, la primera fase, llamada **Entrevistas**, consiste en que los alumnos tengan una entrevista con el profesor para que de esta manera extraigan la mayor cantidad de requerimientos posibles de un sistema en particular. Se propone utilizar proyectos de software ya elaborados por alumnos de generaciones anteriores para trabajar únicamente en sus requisitos. Se forman equipos, cada uno tiene una cita con el profesor a una hora diferente durante el horario de clase. Los alumnos deben preparar su entrevista basándose en los conocimientos adquiridos en clase teórica. Se puede generar expectación si no se les menciona cual será el sistema del que deberán extraer los requerimientos. La idea es que el profesor no dé más información de la que le solicite cada equipo.

El objetivo que se persigue con esta entrevista es lograr que los alumnos se den cuenta del grado de dificultad que representa la extracción de requerimientos en la vida real, ya que a lo largo de su vida académica han estado condicionados a hacer tareas bien definidas por un profesor y la tendencia es a seguir instrucciones en lugar de descubrir necesidades. Normalmente este tipo de entrevistas motiva a los alumnos a poner más atención a las clases teóricas. Es una situación común el que los alumnos den poca importancia a los requerimientos, ya que muchos piensan que esta fase debe ser superada rápidamente para comenzar cuanto antes con la implementación del sistema. El inicio temprano de las entrevistas durante el curso permite a los alumnos visualizar que en un futuro cercano se tendrán que

enfrentar a este tipo de situaciones y, darse cuenta de que la extracción de requerimientos no es un paso que deba apresurarse o tomarse a la ligera. Las entrevistas también hacen salir a la luz ciertos aspectos que si solo se mencionan en clase no son tan evidentes, por ejemplo: seguir el protocolo de presentación y despedida, pedir permiso para grabar la entrevista, controlar los nervios cuando la personalidad del entrevistador lo requiera, preparar las preguntas lo mejor posible para obtener la mayor cantidad de información, etc.

Después de la entrevista, se solicita a los alumnos una especificación de los requerimientos obtenidos durante ésta y se procede a la segunda fase llamada **Revisión de los requerimientos**. En esta etapa los equipos intercambian sus trabajos durante la clase y anotan sus observaciones.

La lectura de las especificaciones de los diferentes equipos es interesante tanto para los alumnos como para el profesor, ya que todos aprenden de los aspectos que unos tuvieron en cuenta y que a otros les faltó contemplar. En las carreras universitarias relacionadas con la ingeniería de software es muy común que existan algunos alumnos con algo de experiencia profesional, ellos pueden aportar mucho al grupo. Si se repiten estas entrevistas a lo largo del curso, los alumnos podrán ir poniendo en práctica lo aprendido en sus experiencias anteriores y constatar su avance.

Para cerrar la segunda fase, se regresa a los alumnos sus trabajos con los comentarios tanto de los demás equipos como los del profesor. Puede ser de gran ayuda proporcionar a los alumnos un resultado global del grupo, indicando cuales fueron las fallas generales y también sus aciertos.

En la tercera fase, llamada **Uso del Estándar**, el profesor proporciona a los alumnos un documento de Especificación incompleto elaborado con el Estándar IEEE-830 [IEEE std, 1998], dicho documento corresponde al sistema que previamente trabajaron en la fase de Entrevistas y de Revisión de Requerimientos. Esta tercera fase tiene dos propósitos, el primero es la familiarización con el Estándar IEEE-830 y el segundo es proveer ejemplos concretos de un documento de Especificación. Al comenzar esta fase, los alumnos ya tienen bastante idea de lo que requiere el sistema que se está trabajando, sin embargo, se recomienda dejar las secciones 1. Introducción y 2. Descripción Global de la Especificación con la mayor parte de su contenido final para que sirvan de una mayor referencia. Se recomienda que la fase de Uso del Estándar se lleve a cabo en el laboratorio, de esta manera se garantiza que cada equipo esta haciendo su trabajo sin copiar. La versión del documento de especificación que reciben los equipos contiene los requerimientos específicos únicamente listados pero sin indicador de la sección que les corresponde. Cada equipo trabaja colocando los requerimientos en la sección donde ellos creen más conveniente. Para las primeras prácticas se define una de las 8 plantillas de la IEEE-830 y los alumnos tendrán que identificar cual fue la plantilla que se usó, en prácticas posteriores ellos mismos deberán elegir la plantilla que mejor describa al sistema en particular. Al final de la práctica, una vez que los alumnos ya enviaron su trabajo al profesor, éste les envía sus observaciones y les proporciona el documento de Especificación completo en el Estándar IEEE-830 para que verifiquen su resultado. Si bien la bibliografía donde se puede consultar el proceso de análisis de requerimientos es excelente [Pleeger, 2002], [Pressman, 2002], [Sommerville, 2005], hay una carencia de ejemplos de documentos completos de Especificación de Requerimientos. Al enviar el documento de Especificación se

provee a los alumnos con un ejemplo de requerimientos redactados de tal forma que cumplan con las características deseables de un requerimiento, es decir, que sea correcto, completo, consistente, realista, verificable, modificable y rastreable [IEEE-STD-830, 1998; Pfleeger, 2002].

Apéndice B: "Especificación de Requerimientos para un juego de ajedrez"

Se utilizó la plantilla A7 del IEE-830 (Organizada por jerarquía funcional).

1 Introducción

1.1 Propósito

Desarrollar un juego de ajedrez para que los miembros de un club puedan entrenar. Se deberá aprovechar el "motor" para jugar ajedrez ya existente llamado GNUChess. Para poder interactuar con la parte gráfica es necesario implementar el Protocolo de Comunicación de Motores de Ajedrez.

1.2 Alcance

- El sistema a desarrollar se llama "Cuaji-Ajedrez" y tiene una interfaz gráfica en forma de tablero con las piezas de ajedrez las cuales se mueven según las reglas del juego.
- "Cuaji-Ajedrez" deberá contar con una interfaz gráfica donde se muestre el tablero de ajedrez junto con las 32 piezas y las opciones del sistema (guardar partida, cargar partida…). El usuario en cualquier momento puede solicitar guardar una partida, cargar una partida, cerrar el sistema y realizar un movimiento en base a las reglas del juego.
- "Cuaji-Ajedrez" debe interactuar con el motor de ajedrez GNUChess el cual contiene la lógica y las reglas del juego.

1.3 Definiciones, Siglas y Abreviaturas

Escaque: el nombre de cada una de las 64 casillas del tablero de ajedrez.

GNUChess : Es un motor de ajedrez ya implementado que contiene la lógica y las reglas del juego. GNUChess es uno de los programas de ajedrez más viejos, fue hecho para computadoras UNIX y ha sido llevado a otras plataformas.

PGN: Portable Game Notation: la Notación Portable para Juego es un formato de computadora para guardar y cargar partidas de ajedrez, esto incluye tanto los movimientos como la información relacionada. La mayoría de los programas de ajedrez para computadora reconocen este popular formato porque su uso es fácil.

Reglas de Movimiento: Manera en la que deben moverse las piezas dentro del tablero según su identidad**.**

Motor de ajedrez: Parte "pensante" del sistema, razona los movimientos que realiza el programa.

Protocolo de Comunicación de Motores de Ajedrez: Protocolo creado por Tim Mann que define las reglas para interactuar con un motor de ajedrez.

1.4 Referencias

La descripción del movimiento de cada una de las piezas de ajedrez se incluye en el apéndice A: "Reglas de movimiento".

En el apéndice B: "Archivo PGN" esta la descripción detallada del formato de cada uno de los campos que debe contener en el orden correspondiente. También se incluye un ejemplo.

2 Descripción Global

2.1 Perspectiva del Producto

El Sistema "Cuaji-Ajedrez" no es totalmente autónomo, consta de una interfaz gráfica y de una interfaz que interactúa con motores de ajedrez siguiendo un protocolo preestablecido. El sistema deberá ser capaz de recibir una jugada de parte del usuario, y responder a ese movimiento siguiendo las reglas del juego de Ajedrez. Perspectivas de "Cuaji-Ajedrez":

a) Interfaces del Sistema: el Sistema debe contener una Interfaz Gráfica en forma de tablero y piezas de ajedrez. Además debe contener la Interfaz con el Motor de Ajedrez siguiendo el "Protocolo de Comunicación con Motores de Ajedrez". Se espera que el sistema pueda sugerir jugadas al usuario, validar sus movimientos y guardar o cargar partidas.
b) Interfaces con el usuario: el usuario se comunicará con el sistema moviendo las piezas del tablero, se requiere que el usuario conozca las reglas de Ajedrez.
c) Interfaces del Software: se elaborará una interfaz con el motor de Ajedrez GNUChess, el cual cuenta con las reglas de Ajedrez para contestar, sugerir y validar una jugada. La información de este motor se encuentra en:
htt://www.gnu.org/software/chess/
d) Funcionamientos:
    a. El usuario podrá elegir las piezas blancas o las negras en una partida de Ajedrez, por lo tanto, el segundo movimiento vendrá de parte del motor, en caso de que elija las blancas.
    b. También, se le dará la posibilidad al usuario de cargar una partida para poder continuar el juego dada una configuración.
    c. Cuando un usuario requiera guardar o cargar una partida, debe realizarse en formato PGN, el cual es un formato que la mayoría de los programas de ajedrez para computadora reconocen.

d. Adicionalmente, si un usuario quiere retomar una jugada realizada anteriormente podrá hacerlo en la misma partida.
e. Cuando el peón del usuario llegue al final del tablero, el sistema le dará opción de elegir la pieza por la que cambiará su peón.
f. También será posible que en lugar de la computadora y el usuario, sean dos usuarios los que jueguen ajedrez.
g. Además se podrá elegir el nivel de dificultad del juego: "principiante, intermedio o avanzado".

2.2 Funciones del Producto

"Cuaji-Ajedrez" podrá realizar lo siguiente:

- Cargar y Guardar partidas.
- Validar si el movimiento que lleva a cabo el usuario es posible y si es así, realizarlo.
- Permitir que el usuario consulte las jugadas posibles.
- Cancelar la partida actual

2.3 Características del Usuario

Los usuarios de Cuaji-Ajedrez deberán conocer las reglas de Ajedrez. Sin embargo, no saberlas no impide que el usuario pueda interactuar con el sistema, ya que éste cuenta con ayuda en todo momento que muestra una jugada sugerida por el motor.

2.6.- Prorratear los requisitos

Los requerimientos de tener "dos jugadores" y "3 niveles de dificultad" podrán ser prorrateados para una versión futura del sistema. Para la primera versión solo se requiere un jugador, sin niveles de dificultad.

3. Los Requisitos Específicos

3.1 Requisitos de la Interface Externa

El sistema consta de 5 tareas básicas:

1. Guardar una partida.
2. Cargar una partida.
3. Validar el movimiento del usuario.
4. Sugerir jugadas al usuario.
5. Cancelar la partida actual.

3.1.1 Interfaz con el Usuario

Interfaz del usuario con el sistema: Muestra el tablero de ajedrez. La Interfaz Grafica debe interactuar con el Motor de Ajedrez. Debe tener un buen contraste de colores, las piezas deben estar bien diseñadas y en proporción al tamaño del tablero y los botones deben localizarse en una posición que sea cómoda para el usuario. Deben estar activas

las opciones del sistema que son: guardar partida, cargar partida, cerrar el sistema y realizar un movimiento en base a las reglas del juego y cuando sea su turno.
Estructura de la interfaz: En la parte superior izquierda, estará el tablero. De su lado derecho, estará una pequeña ventana con las características del juego relacionadas al archivo PGN. Debajo de ésta debe haber un panel de eventos, el cual mostrará la coordenada de las jugadas realizadas. Debajo del tablero y de los paneles de la derecha, estará un panel largo, que servirá como un "log" (bitácora de las jugadas).

3.1.2 Interfaz con el Hardware

Mediante el uso del ratón el usuario podrá interactuar con "Cuaji-Ajedrez" para realizar los movimientos de las piezas en el juego. También podrá usar el teclado para llenar los campos requeridos del archivo PGN. Para acceder a las diferentes opciones y características que ofrece el sistema podrá usar tanto el ratón como el teclado. La jugada no se podrá retroceder una vez que se suelte el Mouse.

3.1.3 Interfaz con el Software

No es necesario saber como funciona el motor de ajedrez, lo que se requiere es manejar el protocolo de comunicación para poder interactuar con él. El motor de ajedrez GNUChess contiene la lógica y las reglas del juego. Las reglas del juego son los movimientos válidos de cada pieza, y la lógica consiste en jugar con los mejores movimientos dependiendo del estado del contrincante para tratar de conseguir la victoria.

3.1.4 Interfaces de Comunicaciones

Implementar el protocolo de comunicación de motores de ajedrez. El protocolo de comunicación de motores de ajedrez consta de una serie de pasos a seguir para realizar la conexión con el motor de ajedrez.

3.2 Requisitos Funcionales

1.- Construir una Interfaz Gráfica
2.- Codificar el estado de la partida en un archivo PGN.
3.- Interpretar la información de los archivos guardados en formato PGN.
4.- Interacción del usuario con la interfaz gráfica.
5.- Implantar la conexión con el motor mediante el protocolo de comunicación de motores de ajedrez.
6.- Interacción del motor con el tablero.

3.2.1 Fluido de la información

3.2.1.1 Guardar y cargar una partida

3.2.1.1.1 Entidades de los datos: Información de los archivos guardados
Evento: el nombre del torneo o de la competencia.

Lugar: el lugar donde el evento se llevó a cabo. Esto debe ser en formato "Ciudad, Región PAÍS", donde PAÍS es el código del mismo en tres letras de acuerdo a l código del Comité Olímpico Internacional. Cómo ejemplo: "México, D.F. MEX".
Fecha: la fecha de inicio de la partida en formato AAAA.MM.DD. Cuando se desconocen los valores se utilizan los signos: "??".
Ronda: La ronda original de la partida
Blancas: El jugador de las piezas blancas, en formato "apellido, nombre".
Negras: El jugador de las negras en el mismo formato.
Resultado: El resultado del juego. Sólo puede tener cuatro posibles valores: "1-0" (las blancas ganaron), "0-1" (Las negras ganaron), "1/2-1/2" (Tablas), o "*" (para otro, ejemplos: el juego está actualmente en disputa o un jugador falleció durante la partida).
Partida: La partida debe estar escrita en inglés, sin símbolos ni comentarios. Es muy importante que después de cada número venga un punto, la jugada blanca puede tener o no un espacio. También se debe poner el + en los jaques y debe haber saltos de línea.

3.2.2 Descripciones del proceso

3.2.2.1 Guardar una partida. El usuario podrá almacenar el estado de algún juego. Se utilizará un archivo con el formato PGN que contiene los datos de la jugada guardada. Para indicar que se proseguirá el mismo juego con los datos actuales, se guardará el partido con el estado "jugando".

3.2.2.1.1 Entidades de los datos de entrada: el archivo a guardar tendrá extensión PGN y contendrá los campos tal y como se especifica en **3.2.1.1.1.**

3.2.2.1.2 Algoritmo del proceso: Solicitar al usuario los datos pertinentes para llenar los campos mencionados.

3.2.2.1.3 Entidades de datos afectadas por el proceso: el archivo que se guardará.

3.2.2.2 Cargar una partida. Cuando un usuario haya guardado una partida con anterioridad, ésta podrá ser retomada utilizando como entrada un archivo PGN que contenga el juego guardado. El sistema deberá leer este archivo y organizar las piezas de acuerdo al archivo cargado.

3.2.2.2.1 Entidades de los datos de entrada: el archivo que se leerá tendrá extensión PGN y contendrá los campos tal y como se especifica en **3.2.1.1.1.**

3.2.2.2.2 Algoritmo del proceso: extraer la información del archivo PGN y desplegar en el tablero las piezas conforme a los datos del archivo.

3.2.2.2.3 Entidades de datos afectadas por el proceso: La posición de las piezas en el tablero.

3.2.2.3 Validar el movimiento del usuario. El sistema debe ser capaz de verificar si el movimiento que el usuario desea realizar es posible según las reglas de ajedrez. Como entrada tendremos las piezas en el tablero organizadas de alguna manera en el juego. Cada que el usuario decide mover una pieza, el sistema valida el movimiento. Como salida, tendremos una distribución de piezas diferente en el tablero, dando oportunidad al siguiente jugador de realizar su turno.

3.2.2.3.1 Entidades de los datos de entrada Como entrada, tenemos la organización de las 32 piezas en el tablero. El sistema hará una validación de los movimientos de las piezas para evitar un movimiento erróneo. Las piezas no se podrán salir del tablero.

3.2.2.3.2 Algoritmo del proceso:

Paso 1.- Implementar la conexión con el motor GNUChess usando el protocolo de comunicación de motores de ajedrez.

Paso 2.- La entrada es la codificación en formato PGN del tablero con las piezas organizadas de manera correcta.

Paso 3.- Por medio del protocolo de comunicación, enviar al motor la jugada del usuario (ejemplo: e3, Nc3…) para que éste la valide, si la jugada es válida, desplegar en el tablero la jugada que contestó el motor, en caso contrario indicar al usuario que la jugada no es válida.

3.2.2.3.3 Entidades de datos afectadas por el proceso: el estado de la partida.

3.2.2.4 Sugerir jugadas al usuario. El sistema debe proporcionar opciones al usuario que así lo requiera. Además, de entre todo el conjunto de movimientos posibles, la aplicación podrá sugerirle un movimiento óptimo al usuario.

3.2.2.4.1 Entidades de los datos de entrada Como entrada, tendremos la manera en la que están organizadas las piezas en el juego.

3.2.2.4.2 Algoritmo del proceso:

Una vez que el motor contesta con el movimiento de una pieza negra, "Cuaji-ajedrez" solicitará al motor por medio del protocolo de comunicación una lista de jugadas sugeridas, recibirá e interpretará la contestación del motor. El sistema señalará en el tablero, los posibles movimientos que el usuario puede hacer.

Ejercicio: Elaborar un diagrama con los diferentes *casos de uso* del sistema. ¿Qué otros artefactos usarías para especificar mejor el sistema?

Apéndice C: "Especificación de Requerimientos para un Sistema de Información Geográfica"

Se utilizó la plantilla A7 del IEE-830 (Organizada por jerarquía funcional).

1.- Introducción

1.1 Propósito

Desarrollar un sistema de información geográfica (SIG). Dicho sistema debe ser capaz de leer archivos en el formato ShapeFile y dibujarlos en pantalla, además se desea que la aplicación pueda cargar varios gráficos simultáneamente y mostrarlos en diferentes "capas", donde el último archivo cargado sea la capa superior.

1.2 Alcance

- El sistema a desarrollar se llama "Sistema de Información Geográfica (SIG)" y deberá ser capaz de interpretar archivos en formato ShapeFile.
- SIG permitirá al usuario elegir un archivo ShapeFile y visualizarlo en la pantalla en forma de mapa.
- SIG contendrá una interfaz gráfica en la que se podrán observar diferentes gráficos superpuestos para una misma región. Cada gráfico muestra la ubicación de algún sitio de interés.
- SIG es un sistema únicamente de visualización de los datos, el usuario no podrá modificarlos.
- SIG es una aplicación web capaz de funcionar en cualquier plataforma.

1.3.- Definiciones, siglas y abreviaturas:

- **ShapeFile**: es un formato vectorial de almacenamiento digital donde se guarda la localización de los elementos geográficos y los atributos asociados a ellos. El formato carece de capacidad para almacenar información topológica**.** Para obtener un shapefile se requieren varios archivos, el número mínimo requerido es tres, y tienen las extensiones mencionadas a continuación:
  - **.shp** - es el archivo que almacena las entidades geométricas de los objetos.
  - **.shx** - es el archivo que almacena el índice de las entidades geométricas.
  - **.dbf** - el dBASE, o base de datos, es el archivo que almacena la información de los atributos de los objetos.

  La definición completa de este formato se puede encontrar en la dirección: http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf)

- **Capa:** es un gráfico que contiene la información de algún tipo de sitio de interés, por ejemplo, una capa con la información de los hoteles, otra con la información de los restaurantes, etc.

2 Descripción Global

2.1 Perspectiva del Producto

El "Sistema de Información Geográfica" (GIS) es un sistema autónomo que se basa en la lectura y visualización de archivos en formato ShapeFile. Los archivos con este formato contienen información acerca de ríos, lagos, hoteles, restaurantes y demás sitios de interés en una región. Cada uno de los lugares mencionados anteriormente se representa en forma de figuras geométricas.

e) Interfaces del Sistema: el Sistema debe contener una Interfaz Gráfica que permita hacer zoom a cierta parte de la imagen y el desplazamiento por todo el grafico visualizado el usuario podrá desplazarse a lo largo del mapa por medio del ratón, además podrá solicitar que se muestren varias "capas" del sistema de manera simultánea. Estas capas se podrán agregar o quitar según decida el usuario. Además GIS contará con ventanas de ayuda para el usuario.

f) Interfaces con el usuario: Existirá una interfaz principal del sistema la cual solo permitirá abrir un archivo ShapeFile o Salir. Cuando el archivo es interpretado correctamente aparecerá la otra interfaz llamada "Visor" en la que se muestra el mapa y las opciones del sistema.

2.2 Funciones del Producto

El "Sistema de Información Geográfica (GIS)" debe ser capaz de:

1.- Visualizar un mapa: Interpretar un archivo en formato ShapeFile para mostrar en pantalla un mapa.

2.- Desplazarse a lo largo de la imagen del mapa: Una vez que el mapa se dibuja en la pantalla deberá ser posible manipular la imagen de tal forma que el usuario se pueda mover con el mouse de un lado a otro.

3.- Hacer zoom en alguna región del mapa: El sistema debe permitir al usuario aumentar la visualización de una imagen poderse desplazar dentro de ella y posteriormente restaurarla.

4.- Agregar y quitar capas al mapa: Se requiere que el usuario pueda visualizar un mapa con varias capas simultáneamente. Podrá agregar y quitar una por una.

5.- Ventana de ayuda: Proporcionar ayuda al usuario por medio de ventanas de ayuda que expliquen el funcionamiento del sistema.

6.- Obtener información del objeto.- Al hacer click en un objeto, el sistema deberá desplegar la información contenida en el archivo .dbf, tal como nombre del establecimiento y su dirección.

2.3.- Características del Usuario

El usuario que interactúe con el sistema deberá tener conocimiento sobre los archivos shapefile y lo que se puede realizar con éstos, ya que "GIS" se basa en la lectura e interacción con los de archivos en este formato.

2.4. Limitaciones generales

1) El límite de capas que se puede cargar simultáneamente es 4.
2) El usuario puede copiar y consultar los archivos y la información del sistema, pero no podrá modificarla.
3) GIS podrá abrir y mostrar en pantalla solamente archivos de tipo ShapeFile.

2.6.- Prorratear los requisitos

El requerimiento de "obtener información del objeto" podrá ser prorrateado para una versión futura del sistema.

3.- Requerimientos Específicos:

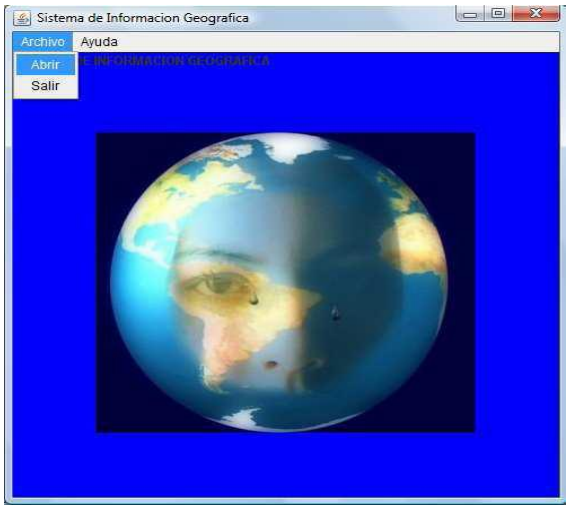
3.1 Requisitos de la Interfaz Externa:

El sistema consta de las siguientes tareas básicas:

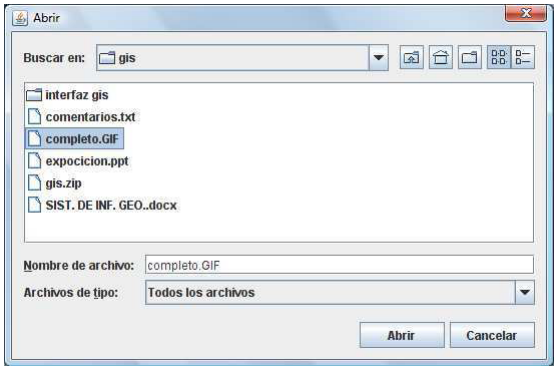
- Visualización de un archivo de tipo ShapFile en pantalla.
- Desplazamiento a través del grafico visualizado.
- Hacer zoom en alguna región del mapa mostrado en pantalla.
- Restaurar la imagen a la que previamente se le hizo el zoom.
- Agregar alguna capa al gráfico visualizado.
- Quitar una capa de las que ya están mostradas en pantalla.
- Mostrar una ventana con ayuda para el usuario.
- Al hacer click en un objeto se mostrarán sus atributos.

3.1.1 Interfaz con el Usuario

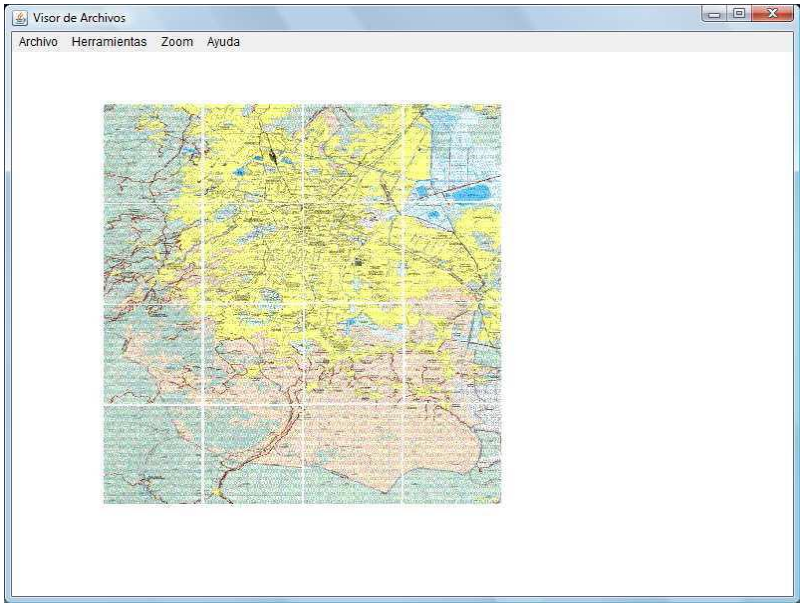
A continuación se muestra la interfaz de la página principal de "GIS" la cual solo permite abrir un archivo ShapeFile o Salir.



Una vez seleccionada la opción "Archivo >> Abrir" deberá aparecer la interfaz de "explorador de archivos" clásica, como se muestra a continuación:



Cuando el archivo ShapeFile seleccionado se pudo interpretar exitosamente, aparece la interfaz del Visor el cual muestra las opciones del sistema y la imagen correspondiente al archivo abierto. Como se ilustra a continuación:



3.1.2 Requisitos de la interfaz con el hardware

No aplica

3.1.3 Requisitos de la interfaz con el software

GIS será una aplicación web implantada en Java que podrá ejecutarse en cualquier explorador de internet (Explorer, Mozila Firefox,…) siempre y cuanto la plataforma sobre la que esté instalado (Windows XP, Windows Vista, Linux,…) contenga la máquina virtual de Java.

3.2 Requisitos funcionales

3.2.1. Fluido de la información.

3.2.1.1 El usuario selecciona exitosamente un abrir archivo ShapeFile,
Se lee el archivo y se interpreta para poder dibujarlo en pantalla. Aparece el visor de GIS, el cual muestra las opciones del sistema.

3.2.1.2 El usuario elige *zoom-aumentar* a la imagen
Se visualiza la imagen aumentada.

3.2.1.3 El usuario elige *zoom-restaurar* la imagen
Se visualiza la imagen en su tamaño original.

3.2.1.4 El usuario elige *Herramientas-arrastrar* la imagen
Se visualiza la imagen en otra posición.

3.2.1.5 El usuario elige *Herramientas- agregar capa*.
Se visualiza la imagen con el gráfico adicional seleccionado-

3.2.1.6 El usuario elige *Herramientas- quitar capa*.
Se visualiza imagen sin el último gráfico que se le había agregado.

3.2.1.7 El usuario elige abrir ventana de *ayuda*
Se abre la ventana de ayuda del sistema.

3.2.1.8 El usuario elige *salir*
Se cierra el sistema.

Ejercicio: Elaborar *diagramas de secuencia* que ilustren el funcionamiento del sistema.
. ¿Qué otros artefactos usarías para especificar mejor el sistema?

Apéndice D: "Especificación de Requerimientos para un Visualizador Molecular."

Se utilizó la plantilla A1 del IEE-830 (Organizada por el modo).

1.- Introducción

1.1 Propósito

Una importante empresa biotecnológica le ha solicitado al grupo de ingenieros de software de la UAM Cuajimalpa desarrollar un visualizador molecular similar a Ras Mol (http://www.openrasmol.org/) para que sus científicos puedan trabajar en el diseño de moléculas. Se espera que la aplicación desarrolladora pueda leer archivos en el formato XYZ (http://openbabel.sourceforge.net/wiki/XYZ) y muestre la molécula representada, permitiendo cambiar la vista, es decir, que permita observarla desde distintas direcciones, se tiene contemplado que en versiones futuras también sea posible dibujar una molécula en tercera dimensión.

1.2 Alcance

- El sistema a desarrollar se llama "Visualizador Molecular" y tiene una interfaz gráfica que permite observar moléculas tridimensionales desde diferentes ángulos y también dibujarlas.
- Será posible convertir una vista de la imagen tridimensional a una imagen plana.
- También deberá ser posible observar una imagen bidimensional.
- En su primera fase, el "Visualizador Molecular" solo servirá para visualizar las moléculas.
- Solamente se podrá abrir un archivo en formato XYZ a la vez.

1.3.- Definiciones, siglas y abreviaturas:

- **Archivo XYZ:** Es un archivo que se forma de cuatro columnas en la primer columna representa el nombre del átomo y las otras tres columnas son las coordenadas las cuales especifican la posición del átomo en el espacio. Para representar una molécula se usan varios átomos.

- **Renderización:** En términos de visualización en una computadora en 3D, es un proceso de cálculo complejo, desarrollado por la computadora para generar una imagen 2D a partir de una escena 3D. En otras palabras, la computadora interpreta la escena en 3 dimensiones y la plasma en una imagen bidimensional.

- **Vista seleccionada:** es la forma en la que se ve la molécula una vez que el usuario la rotó y/o la trasladó.

2 Descripción Global

2.1 Perspectiva del Producto

El "Visualizador Molecular" es un sistema autónomo que consta de una interfaz gráfica en la que se pueden hacer diversas operaciones con la imagen de una molécula.
Perspectivas del "Visualizador molecular":
Interfaces del Sistema: la Interfaz Gráfica debe contener cuatro pestañas que indican los 4 modos de operación del sistema: (i) visualizar una molécula en 3D, (ii) visualizar una molécula en 2D, (iii) ver un archivo XYZ (iv) dibujar molécula.

2.2 Funciones del Producto

El "Visualizador Molecular" debe ser capaz de:

1.- Visualizar una molécula en 3D: Interpretar un archivo XYZ para mostrar en pantalla un modelo 3D de una molécula.

2.- Rotar y trasladar la imagen de la molécula: Una vez que la molécula se dibuja en la pantalla deberá ser posible manipular la imagen de tal forma que esta se pueda rotar y mover de un lado a otro.

3.- Hacer zoom en la molécula presentada en 3D Se requiere desarrollar una herramienta que permita hacer más grande la imagen en 3D y restaurarla.

4.- Guardar la imagen 2D de la molécula a partir de la vista seleccionada por el usuario: se requiere de una herramienta que convierta a 2D la vista de la imagen 3D que seleccionó el usuario.

5.- Visualizar una molécula en 2D: mostrar una imagen plana, que fue previamente renderizada, esto puede hacerse cuando se elige el modo (ii).

6.- Ver un archivo XYZ: ver el contenido de un archivo que esta en este formato, esto puede hacerse cuando se elige el modo (iii).

7.- Dibujar molécula: Hacer el dibujo de una molécula en 3D a partir de un dibujo anterior o bien comenzando desde el principio y guardarlo en un archivo.

2.3.- Características del Usuario

Este producto estará destinado a científicos con conocimientos de química, en específico, con conocimientos sobre el estudio y desarrollo de moléculas.

2.6.- Prorratear los requisitos

El requerimiento de "dibujar una molécula" podrá ser pospuesto para una versión futura del sistema. Para la primera versión solo se requiere poder visualizar y renderizar imágenes 3D a 2D, así como poder ver un archivo en formato XYZ.

3.- Requerimientos Específicos:

3.1 Requisitos de la Interfaz Externa:

El sistema consta de 4 tareas básicas:

- Visualización de una molécula en 3D en pantalla.
- Visualización de una molécula en 2D en pantalla.
- Mostrar un archivo en formato XYZ.
- Dibujar molécula en 3D.

3.1.1 Interfaz con el Usuario

El usuario podrá elegir las siguientes opciones dentro de cada uno de los modos:

a. Visualizar una molécula en 3D

- Cargar archivo XYZ
- Renderizar la vista
- Zoom

b. Visualizar una molécula en 2D,

- Cargar archivo renderizado

c. Ver un archivo XYZ

- Cargar archivo XYZ

(iv) Dibujar molécula

- Cargar dibujo.
- Guardar dibujo.

3.1.2 Requisitos de la interfaz con el hardware

Mediante el uso del ratón el usuario podrá interactuar con el "Visualizador Molecular" para rotar y trasladar la imagen. También podrá usar el teclado para nombrar el archivo JPG con la imagen renderizada. Para acceder a las diferentes opciones y características que ofrece el sistema podrá usar tanto el ratón como el teclado.

3.1.3 Requisitos de la interfaz con el software

El "Visualizador Molecular" trabajará sobre un sistema operativo Windows XP, un Windows Vista, o un Windows 7. Para desarrollarlo se utilizará Java y la librería 3D de java.

3.2 Requisitos funcionales

3.2.1 Modo de visualización 3D

3.2.1.1 Debe existir por lo menos un archivo XYZ en la maquina donde opera el sistema.

3.2.1.2 El "Visualizador Molecular" debe poder interpretar un archivo XYZ y desplegar la molécula en pantalla.

3.2.1.3 La imagen debe poder ser renderizada en una posición elegida por el usuario.

3.2.1.4 Una vez que la imagen ha sido renderizada deberá guardarse en un archivo como imagen JPG.

3.2.2 Modo de Visualización 2D

3.2.2.1 Mostrar en la pantalla la imagen de un archivo JPG que debe ser la imagen renderizada de una figura 3D.

3.2.3 Ver un archivo XYZ

3.2.3.1 Mostrar en la pantalla el formato de un archivo XYZ seleccionado por el usuario.

3.2.4 Dibujar molécula en 3D

3.2.4.1 Se debe poder cargar la imagen de un dibujo previo.

3.2.4.2 También se puede comenzar a hacer un dibujo desde el principio.

3.2.4.3 El dibujo de la molécula se traducirá a formato XYZ para poder guardarlo en un archivo.

Ejercicio: Elaborar un diagrama con los diferentes *casos de uso* y *diagramas de secuencia* que ilustren el funcionamiento del sistema. ¿Qué otros artefactos usarías para especificarlo mejor?

Bibliografía

[1] Arboleda H.. "Modelos de ciclo de vida en desarrollo de Software", Sistemas (ACIS: Asociación Colombiana de Ingenieros en Sistemas), No. 93, Julio 2005.

[2] Booch, G. "Análisis y Diseño Orientado a Objetos con aplicaciones". 2ª edición, Addison Wesley Longman, México 1998.

[3] Braude. "Ingeniería de Software, una perspectiva Orientada a Objetos", Alfaomega, México, 2003.

[4] Castro Gil R., "Estructura básica del proceso unificado de desarrollo de software". Revista Sistemas y Telemática, Universidad ICESI, Colombia, 2004.

[5] De Amescua S. Antonio, García S. Luis, Martínez F. Paloma, Díaz P. Paloma. "Ingeniería de Software de Gestión, Análisis y Diseño de Aplicaciones". Ed. Parainfo, Madrid 1995.

[6] De Miguel A., Piattini M., "Fundamentos y Modelos de Bases de Datos". 2ª ed., Alfaomega-Ra-ma, México, 2004.

[7] IEEE-STD-830-1998: Recommended Practice for Software Requirements Specifications.

[8] IBM, Rational Unified Process, Best Practices for Software Development Teams. Rational Software white paper TP026B, Rev 11/01.

[9] Kendall & Kendall , Análisis Y Diseño De Sistemas 3ª Ed., Prentice Hall, México,1997.

[10] Maglione H. y Placentino V. "Modelo Esencial para un Sistema Integrado de Gestión Universitaria". Departamento de Ingeniería. Universidad del CEMA, 2001.

[11] McConnell, S., *Desarrollo y gestión de proyectos informáticos,* McGraw Hill y Microsoft Press, España, 1997.

[12] Norris & Rigby. “Ingeniería de software explicada”, 1ª edición. Editorial Megabyte-Noriega editores, México, 1994.

[13] Piattini M., Calvo-Manzano J., Cervera J., Fernandez L. “Análisis y diseño de Aplicaciones Informáticas de Gestión”. Una perspectiva de Ingeniería de Software. Alfaomega-Rama, México, 2004.

[14] Pleeger, Shari L. “Ingeniería de software. Teoría y práctica” 1ª edición. Editorial Pearson Education, Buenos Aires, 2002.

[15] Pressman, Roger S. "Ingeniería del Software: Un enfoque práctico", 5a edición. Editorial McGraw Hill, España, 2002.

[16] Rumbaugh, J., “Modelado y Diseño Orientado a Objetos: Metodología OMT”. Prentice Hall, 1996.

[17] Robertson & Robertson. “Mastering the Requirements Process”, Addison-Wesley, 1999.

[18] Soto, I. & González, P., “Propuesta de un plan para la recuperación de proyectos de software”. Teorías, Modelos y Aplicaciones de Matemáticas y Computación, Memorias de la 1ª y 2ª Semana de Computación y Matemáticas Aplicadas SCMA 08 y SCMA 09. UAM Cuajimalpa, México D.F., 2010.

[19] Sommerville, Ian. “Ingeniería del Software”, 7ª Ed., Pearson Addison Wesley, Madrid, 2005.

[20] Weitzenfeld, Alfredo. “Ingeniería de Software Orientada a Objetos con UML, Java e Internet”. Editorial Thomson, 2004.

[21] Yourdon, Edward, “Análisis estructurado moderno”, Ed. Prentice Hall/Pearson, 1993.

Notas del Curso: Análisis de requerimientos
Se terminó de imprimir en Mayo de 2011 en
Publidisa Mexicana S. A. de C.V.
Calz. Chabacano No. 69, Planta Alta
Col. Asturias C.P. 06850.