Análisis teórico para la construcción de sistemas de información

Autor: Jorge Humberto Reynaga Peña

Tesina presentada para obtener el título de:
Lic. En Sistemas computarizados [sic]

Nombre del asesor:
Sergio Francisco Barraza Ibarra

UNIVERSIDAD
VASCO DE QUIROGA
M. R.

Licenciatura en Sistemas Computarizados

"Análisis Teórico para la Construcción de Sistemas de Información"

T E S I N A

Que para Obtener el Título de

Licenciado en Sistemas Computarizados

P R E S E N T A

JORGE HUMBERTO REYNAGA PEÑA

A S E S O R

M. en A. Sergio Francisco Barraza Ibarra

Clave: 16PSU0014Q Acuerdo: 952006

MORELIA, MICH. OCTUBRE DE 1999.

INDICE

AGRADECIMIENTOS

A mi Madre:
Por el logro del
objetivo que siempre deseó.

A mi Padre:
Con respeto y cariño.

A mi Esposa Angélica:
Sin su ayuda y apoyo esta Tesina no
hubiera llegado a ser lo que hoy es.

A mis hermanas:
Cristina y Dora

Al M.en A. Sergio Fco. Barraza Ibarra:
Por su valiosa ayuda para la
elaboración de este trabajo.

A mis Maestros:
Mi agradecimiento por su
enseñanza.

A la Universidad Vasco de Quiroga:
Por haberme dado la oportunidad
de forjarme un futuro.

INTRODUCCION

En la actualidad los sistemas de información basados en computadora resultan ser un elemento fundamental en las actividades cotidianas de una organización y objeto de gran consideración en la toma de decisiones. El desarrollo de sistemas es una actividad participe de esta incorporación de los sistemas de información en las organizaciones.

Para el desarrollo de sistemas, se cuenta con dos grandes disciplinas, Ingeniería de Software y Análisis y Diseño de Sistemas. La primera de ellas proporciona un enfoque general orientado al desarrollo de sistemas de gran escala y para cualquier tipo de sistemas que puedan automatizarse, desde sistemas operativos, sistemas expertos, hasta sistemas de información.

Por otro lado el Análisis y Diseño de Sistemas según James Senn se enfoca al proceso de examinar la situación de una empresa con el propósito de mejorarla con métodos y procedimientos mas adecuados.

La persona que se inicia en el desarrollo de sistemas de información no percibe las posibles áreas de estudio que rodean esta actividad; dando como resultado remarcadas deficiencias en el desarrollo de software, así basado en lo anterior el objetivo de este trabajo es realizar un análisis teórico para las diferentes propuestas que existen desde el punto de vista de ambas disciplinas, que oriente al estudiante de sistemas y sirva de guía en el desarrollo de sistemas de información.

El presente trabajo esta diseñado en cuatro capítulos, en el primer capítulo se establece una base conceptual sobre los distintos modelos de desarrollo de software existentes, como es el modelo de ciclo de vida clásico, de prototipos, y modelos en espiral.

El segundo capítulo ofrece un tratamiento detallado sobre la fase de análisis, proporcionando desde su concepto hasta las distintas herramientas de recopilación de información, de descripción y una breve descripción de los métodos disponibles de análisis.

En el capítulo tres se presenta los fundamentos que se consideran necesarios para el diseño de un sistema como es la modularidad, abstracción, acoplamiento de la información, independencia funcional, cohesión y acoplamiento, diseño de datos, diseño arquitectónico y procedimental, diseño de la interfaz de usuario y los métodos de diseño.

La intención del cuarto capítulo es dar una descripción sobre la implementación del sistema, integrado por codificación, prueba del sistema, depuración y documentación; y por último las conclusiones del presente trabajo.

OBJETIVOS

Objetivo General:

Realizar un análisis teórico de las diferentes propuestas que existen desde el punto de vista de Ingeniería de Software y Análisis y Diseño de Sistemas, revisando los distintos modelos, conceptos y teorías para el desarrollo de sistemas.

Objetivos Particulares:

- Realizar una recopilación de los distintos modelos de desarrollo de sistemas.
- Identificar las distintas fases en el desarrollo de sistemas.
- Concretar en los puntos esenciales de los modelos de desarrollo de software.
- Identificar las herramientas existentes para análisis de sistemas.
- Comprender los conceptos relevantes en el diseño de sistemas.
- Proporcionar un conjunto de guías para la implementación de sistemas.

CAPITULO I

MODELOS DE DESARROLLO DE SOFTWARE

El primer paso a considerar en el desarrollo de un sistema consiste en elegir un modelo que servirá de guía al desarrollador de Software, para obtener un producto de calidad. Estos modelos incluyen todas las actividades necesarias para lograr este propósito. Actualmente existe una diversidad de modelos que se han propuesto, cada uno de estos con características muy definidas.

Definiremos desarrollador como aquella persona sobre la cual recae la responsabilidad de conformar los elementos y realizar las actividades necesarias para lograr el desarrollo de un sistema de información capaz de automatizar una tarea o actividad. Por otro lado un usuario nos representará aquella persona u organización que tiene la necesidad de mejorar una tarea o actividad participe de su haber cotidiano.

Lo importante a considerar a la hora de elegir un modelo, es que este se adapte a las necesidades de las partes involucradas, es decir al desarrollador y al usuario, así como también al tamaño y complejidad del producto.

A continuación se describen tres de los modelos más utilizados en el desarrollo de software:

1.1. MODELO DEL CICLO DE VIDA CLASICO

El modelo de ciclo de vida clásico puede considerarse el más antiguo y el más utilizado en el desarrollo de sistemas, en ocasiones denominado modelo de cascada. Este modelo proporciona un enfoque tradicional en el desarrollo de sistemas, James Senn sugiere las siguientes actividades representadas en la figura 1.1. que constituyen el modelo de ciclo de vida clásico([1])

Figura 1.1 Ciclo de vida clásico

1. Investigación preliminar. Esta actividad tiene su punto de partida en una solicitud por la parte que requiere del sistema. Se lleva a cabo un estudio de factibilidad y a partir de este último se aprueba o no el desarrollo del sistema.

2. Determinación de Requisitos. Una vez aprobada la solicitud, se deben de realizar las investigaciones correspondientes para determinar los requisitos que deberá cumplir el sistema. Esto puede llevarse a cabo utilizando técnicas de recopilación de

[1] Senn, James A. *Análisis y Diseño de Sistemas de Información*. Segunda Edición. Ed. Mc Graw Hill. México. 1992. Pág.33

información como son cuestionarios, entrevistas, análisis de grupos, observación, muestreo y recopilación de documentos, etc.; es decir en esta parte del modelo se lleva a cabo el análisis del sistema.

3. Diseño del sistema. El diseño del sistema se encuentra orientado a la solución del problema, representando mediante gráficas, tabulaciones u otros elementos, los atributos con los que deberá contar el sistema en proyecto. Estos atributos parten desde la estructura de los datos que encuentran involucrados hasta el proporcionar una caracterización de las interfaces con las que contará el sistema, que cumplan y faciliten su uso.

4. Desarrollo del software. Durante esta fase se llevará a cabo la codificación del programa.

5. Prueba de los sistemas. Una vez que se ha realizado la generación de código, el siguiente paso consiste en realizarle una prueba al sistema generado, para asegurar que éste no tenga fallas y produzca los resultados esperados a partir de las entradas definidas. En esta fase del sistema se llega a involucrar al usuario.

6. Implantación y Evaluación. El proceso de implantación y evaluación consiste desde proporcionar la capacitación necesaria al usuario para el manejo del sistema hasta llevar a cabo una evaluación del sistema para identificar la calidad del producto. También durante este proceso se llevan a cabo tareas de planificación de mantenimiento, ya que posiblemente el sistema sufrirá de cambios a lo largo de su vida útil.

Las fases anteriores solamente representan uno de los múltiples enfoques que se le pueden aplicar a este modelo. Algunos autores pueden ampliar el campo de acción de alguna de las fases o inclusive eliminarla como se demuestra en la figura 1.2 donde

Alfredo Rodríguez y Antonio Márquez nos presentan variaciones sobre la composición de estas.([2])

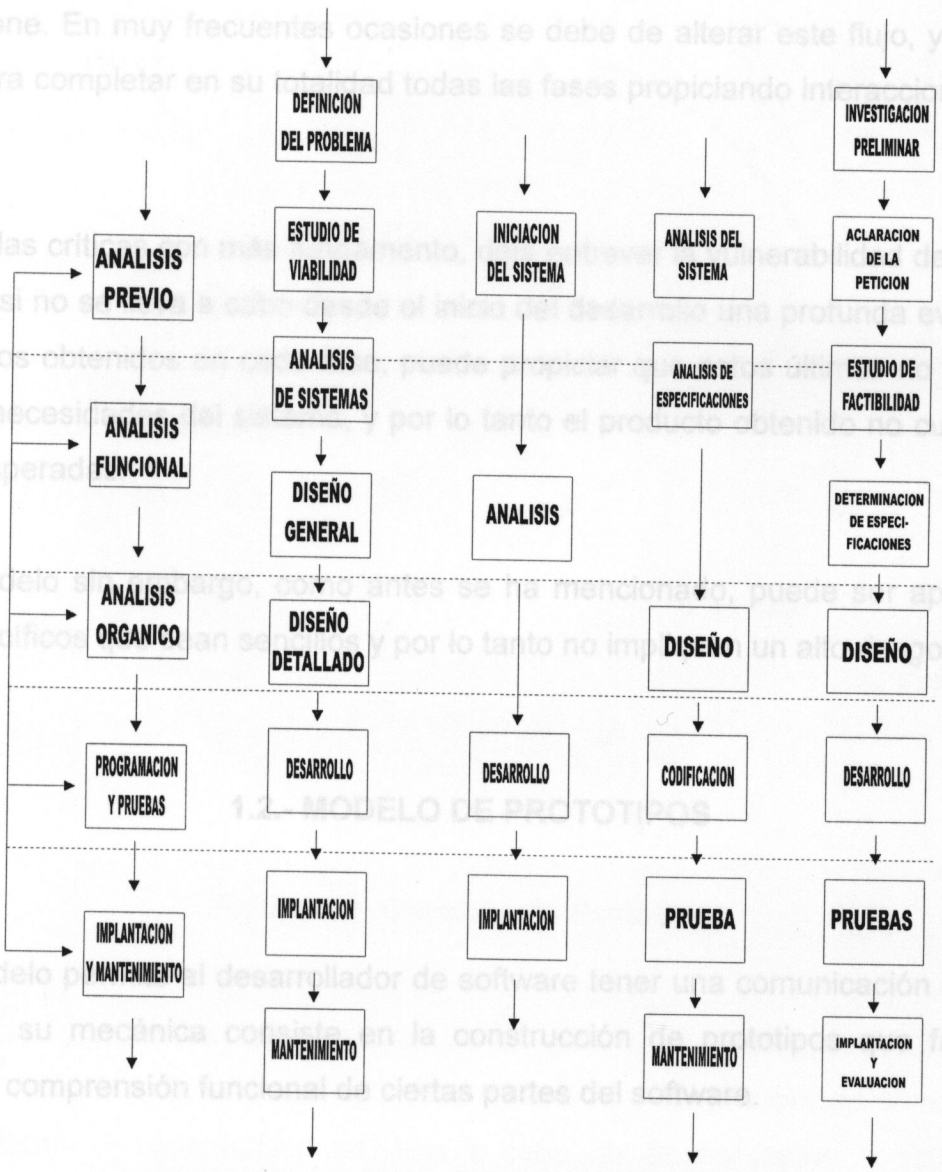


Figura 1.2 Variación del Ciclo de vida clásico.

[2] Rodríguez Cuadrado, Alfredo; Márquez Serrano, Antonio. *Técnicas de Organización y Análisis de Sistemas*. Ed. Mc Graw Hill. España. 1993. Pág. 45

Actualmente este modelo ha recibido severas críticas, inclusive por sus propios seguidores, algunas de las críticas más frecuentes se basan en la afirmación de que el proyecto al desarrollarse bajo este modelo, casi nunca siguen el flujo sistemático y lógico que este propone. En muy frecuentes ocasiones se debe de alterar este flujo, ya que no siempre se logra completar en su totalidad todas las fases propiciando interacciones entre fases.

Otra de las críticas con más fundamento, deja entrever la vulnerabilidad del modelo al afirmar que, si no se lleva a cabo desde el inicio del desarrollo una profunda evaluación de los resultados obtenidos en cada fase, puede propiciar que estos últimos no estén de acuerdo a las necesidades del sistema, y por lo tanto el producto obtenido no cumpla las expectativas esperadas.

Este modelo sin embargo, como antes se ha mencionado, puede ser aplicable a proyectos específicos que sean sencillos y por lo tanto no impliquen un alto riesgo.

1.2.- MODELO DE PROTOTIPOS

Este modelo permite al desarrollador de software tener una comunicación continua con el usuario, su mecánica consiste en la construcción de prototipos que facilite al desarrollador la comprensión funcional de ciertas partes del software.

El prototipo puede considerarse en esencia como la representación funcional de una parte del sistema. Para lograr esta representación se puede comenzar con un prototipo diseñado en papel y posteriormente desarrollarlo de forma tal que represente una "realidad" de la función para la cual fue diseñado.

La figura 1.3 representa las fases que se deberán seguir para la construcción de prototipos de Roger S. Pressman([3]).

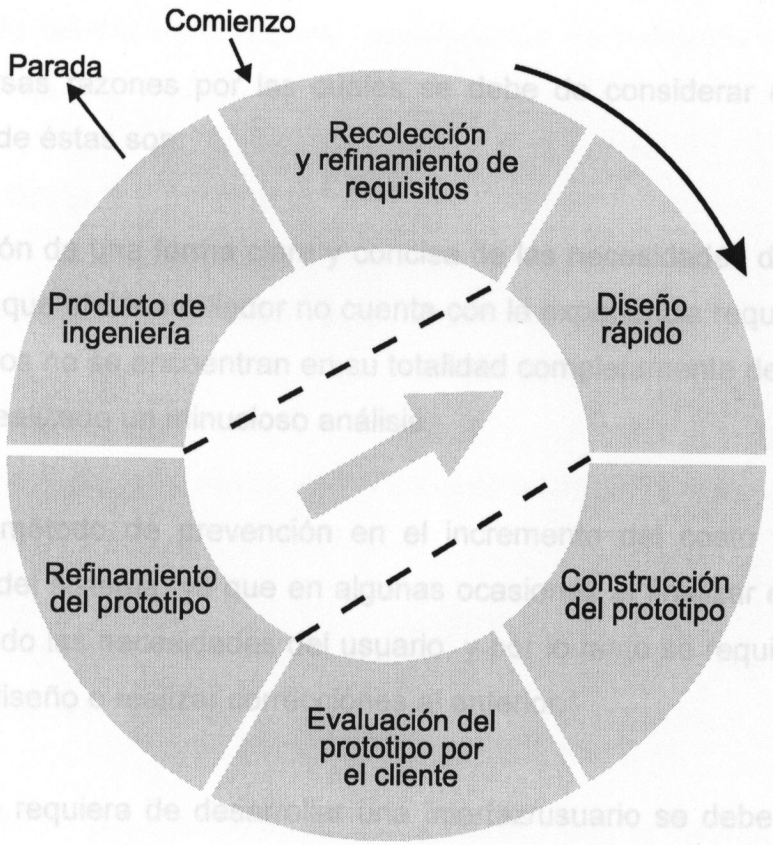


Figura 1.3 Creación de Prototipos

La primera fase para la construcción de un prototipo la define la recolección y refinamiento de requisitos de la función que no se encuentra en su totalidad comprendida por el desarrollador. A continuación se lleva a cabo un diseño rápido a partir de la información recolectada y se construye el prototipo; después de construido, el prototipo debe ser evaluado por el usuario para que a partir de las consideraciones hechas por él mismo se refine y se lleve a cabo un nuevo diseño; construcción y así sucesivamente. El

[3] Pressman, Roger S. *Ingeniería del Software: Un enfoque práctico.* Tercera Edición. Ed. Mc Graw Hill. España. 1993.pag. 29.

número de interacciones que se realizarán se encuentra definido por la capacidad de producir cada vez un prototipo más refinado, hasta que cumpla en su totalidad con las necesidades del usuario.

Existen diversas razones por las cuales se debe de considerar el desarrollo de prototipos, algunas de éstas son:

1. La obtención de una forma clara y concisa de las necesidades del usuario. Esto a partir de que el desarrollador no cuenta con la experiencia requerida, o cuando los requisitos no se encuentran en su totalidad completamente definidos, a pesar de haber realizado un minucioso análisis.

2. Como un método de prevención en el incremento del costo y tiempo en el desarrollo del sistema, ya que en algunas ocasiones al finalizar el diseño, no se han cumplido las necesidades del usuario, y por lo tanto se requiere de elaborar un nuevo diseño o realizar correcciones al anterior.

3. Cuando se requiera de desarrollar una interfaz/usuario se debe buscar la más apropiada, para el usuario.

4. Si la naturaleza del sistema implica una alta seguridad en el funcionamiento de éste, y por lo tanto minimizar el tiempo y esfuerzo reduciendo el número de correcciones.

Es importante señalar que al desarrollar prototipos, éstos deben ser sujetos a algunos lineamentos, para que su desarrollo sea lo más eficaz posible, éstos son:

- El prototipo deberá partir de una especificación de requisitos lo más completa posible.
- Debe ser diseñado con rapidez.
- Deberá ser evaluado en cada interacción por el usuario con el más minucioso análisis.
- Debe de evolucionar a través de un proceso interactivo, hasta satisfacer en su totalidad las necesidades requeridas.
- Debe ser aplicado solo a una función a la vez y no como un producto semifuncional del sistema.
- Deberá realizar todo o parte de la función para la cual se diseña.
- Deberá ser desarrollado, partiendo de que el sistema se basará en el diseño modularizado.

1.2.1.- TIPOS DE PROTOTIPOS

Se pueden identificar claramente tres tipos de prototipos que se pueden desarrollar:

- Prototipo para entradas.
- Prototipo para procesos.
- Prototipo para salidas.

El primero de ellos busca enfatizar en el logro de una interfaz de entrada perfecta para el usuario, definiendo características tales como: El método de entrada (lectura de barras, lápiz óptico, ratón, etc.), los requisitos de entrada, las validaciones y controles necesarios, la facilidad de navegación, etc.

El segundo de ellos, busca completar los algoritmos necesarios para llevar a cabo los procesos que se aplicarán a los datos, como son: almacenamiento, respaldo y recuperación de información, cálculos, validación de transacciones, etc.([4])

Por último, el tercer tipo estriba su importancia en la obtención de los resultados esperados por la aplicación, como son reportes generados en pantalla u otro medio, formato de ellos, etc.

Puede existir una mezcla de estos tres tipos conformando un cuarto tipo, esto se puede dar ya que en algunos casos la función comprenderá todos los procesos.

---

[4] Kendall y Kendall. *Análisis y Diseño de Sistemas.* Ed. Prentice Hall. Tercera Edición. México. 1991.Pág. 254.

1.2.2.- VENTAJAS Y DESVENTAJAS DE LOS PROTOTIPOS

El modelo de prototipos, debe ser una alternativa que deberá considerarse por los siguientes motivos:

En primer lugar proporciona un medio por el cual se podrán concretar los requisitos del sistema, por la gran participación de los usuarios. En segundo lugar, incorpora un medio por el cual podrán identificarse posibles errores en una etapa temprana, por consiguiente las posibilidades de que se obtenga un producto de calidad y libre de errores es muy alto.

También se presenta como una herramienta invaluable para aquellos desarrolladores que cuentan con poca experiencia o que desconocen en su totalidad la información necesaria que involucra el producto que se desea generar.

Por último, al ser el prototipo una entidad flexible y que se encuentra en constante evolución en base a la retroalimentación del usuario, se llegarán a cumplir las expectativas de este.

Sin embargo al igual que los diversos modelos que existen, los prototipos cuentan con algunas desventajas, resaltando dentro de ellas la posibilidad de que el prototipo llegue a aceptarse como un sistema terminado y se implemente sin llevar a cabo una refinación.

1.3.- MODELO EN ESPIRAL

Dentro de los modelos mas recientes que se han propuesto se encuentra el modelo en espiral de Barry Boehm, uno de los mas conocidos críticos al modelo en cascada o clásico, su trabajo se encuentra presentado en "A Spiral Model for Software Development and Enhacement", La figura 1.4 ilustra el modelo.([5])

La principal característica de este modelo, es que añade un elemento no considerado en los modelos anteriores, el análisis de riesgos; sin embargo como puede observarse hace uso del planteamiento consecutivo del modelo de cascada e incorpora prototipos. Este modelo permite generar un prototipo cada vez más sofisticado.

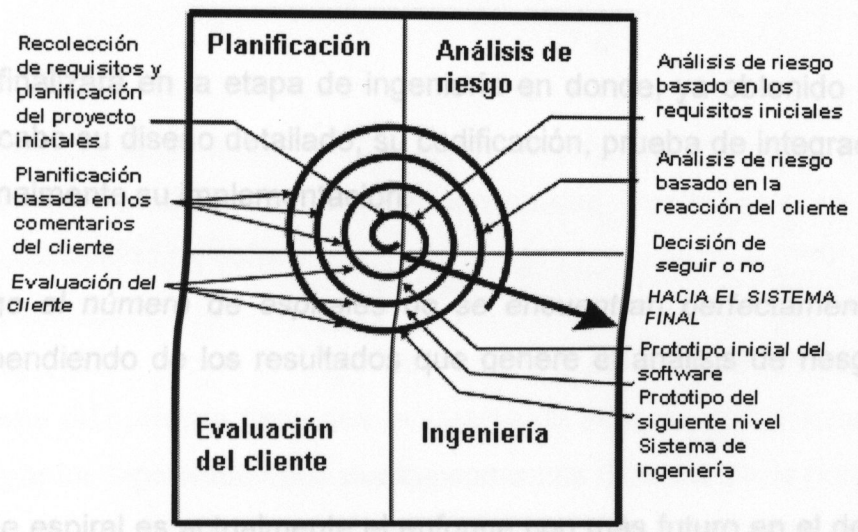


Figura 1.4 Modelo en espiral.

En este modelo el proceso tiene su punto de partida en el centro, este inicio se sustenta en los requisitos iniciales y un plan de desarrollo procede con un ciclo en espiral.

---

[5] Tutorial de "*Análisis y Diseño de Sistemas*". Instituto Tecnológico La Paz Baja California. www.itlp.edu.mx/publica/tutoriales/analisis/index.htm

Las espirales se encuentran divididas en cuatro partes fundamentales; una que permite determinar objetivos, alternativas y limitantes, el segundo segmento permite realizar un análisis de riesgo, es decir evaluar las alternativas e identificarlas.

A partir del análisis de riesgo se genera el prototipo 1, el cual al seguir el curso de la espiral, se le aplican procedimientos, herramientas y métodos de ingeniería de software, para verificar y validar los requisitos.

La cuarta fase está determinada por una evaluación del usuario, a partir de la cual se realizará una nueva planificación, basada en los comentarios y sugerencias de éste. El desarrollo seguirá su curso, generando una serie de interaciones que poco a poco producirán un mejor modelo.

La espiral finalizará en la etapa de ingeniería en donde, ya obtenido un producto final, se llevará a cabo su diseño detallado, su codificación, prueba de integración, prueba de aceptación y finalmente su implementación.

Sin embargo el número de espirales no se encuentran perfectamente definido, éstas variarán dependiendo de los resultados que genere el análisis de riesgo en cada espiral.

El modelo de espiral es actualmente el enfoque con más futuro en el desarrollo de sistemas, sin embargo debe de considerarse que no se encuentra totalmente probado, como es el caso de modelos de cascada. Su principal ventaja por considerar es la incorporación del análisis de riesgo, aunque debe tomarse en cuenta que si este no se lleva a cabo de una forma adecuada, puede suceder que alguna omisión produzca resultados indeseables.

1.4. DESARROLLO DE LAS FASES

Cualquiera de los modelos anteriores representan las fases necesarias que se requieren para el desarrollo de software, sin embargo, una de las actividades del desarrollador de sistemas consiste en proponer o adaptar un modelo si el proyecto lo requiere. A partir de los modelos anteriores se pueden establecer las siguientes fases generales:

1. DEFINICION
2. DESARROLLO
3. IMPLEMENTACION

Estas fases se pueden considerar suficientes y necesarias que permitirán al desarrollador de software llevar a cabo su trabajo de una manera sistemática y ordenada, para llegar a obtener un producto de software con calidad. Por lo tanto es conveniente definir el concepto de calidad de software.

Según Meyer (1988), Sommerville (1989) y otras autoridades en la materia, la calidad de un sistema de software tiene que evaluarse de acuerdo con diversos criterios. Podríamos considerar los siguientes doce puntos como una lista completa para evaluar la calidad([6])

- **Corrección**. Los programas deben satisfacer sus especificaciones correctamente.

[6] Ian, Graham. *Métodos Orientados a Objetos*. Segunda Edición. Ed. Adison-Wesley. E.U. 1994. Págs. 43-44.

- **Flexibilidad y fiabilidad (solidez).** Los programas deben ser sólidos, aún en condiciones anormales.

- **Facilidad de mantenimiento**. Los programas deben ser fáciles de modificar y extender cuando se produzcan cambios en los requisitos.

- **Reutilizabilidad y generalidad**. Los programas deben construirse con módulos reutilizables, con la finalidad de poder ser usados en su totalidad o en parte por nuevas aplicaciones.

- **Interoperabilidad**. Los programas deben ser fácilmente compatibles con otros sistemas; deben ser "sistemas abiertos".

- **Eficiencia**. La eficiencia es la característica de usar adecuadamente los recursos de hardware, tales como procesadores, memorias internas y externas, así como dispositivos de comunicación, tanto en espacio como en tiempo.

- **Transportabilidad**. Los programas deben ser transportables de un equipo a otro (hardware), y de un sistema operativo a otro (software).

- **Verificabilidad**. Es la facilidad para preparar procedimientos de aceptación, particularmente datos de prueba y procedimientos para detectar fallas y localizar errores durante la fase de prueba y depuración.

- **Seguridad**. Los datos, los conocimientos e incluso, las funciones pueden requerir un ocultamiento selectivo y efectivo.

- **Integridad**. Los sistemas requieren protección contra actualizaciones inconsistentes.

- **Amabilidad**. Los programas deben ser fáciles de usar para la mayoría de los usuarios, sin llegar a pecar de excesos en palabras.

- **Facilidad de descripción**. Debe ser posible crear y mantener documentación de los programas.

Los factores anteriores pueden ser interpretados como lo demuestra la tabla 1.1.

FACTOR	INTERPRETACION
Corrección	¿Hace lo que quiero?
Flexibilidad	¿Puedo cambiarlo?
Fiabilidad	¿Lo hace de forma fiable todo el tiempo?
Facilidad de mantenimiento	¿Puedo corregirlo?
Reusabilidad	¿Podré reusar alguna parte del software?
Interoperabilidad	¿Podré hacerlo interactuar con otro sistema?
Eficiencia	¿Se ejecutará en mi hardware lo mejor que pueda?
Transportabilidad	¿Podré usarlo en otra máquina?
Verificabilidad	¿Puedo probarlo?
Integridad	¿Es seguro?
Facilidad de uso	¿Está diseñado para ser usado?

Tabla 1.1 Interpretación de los factores de calidad de software.

[1] Pressman, Roger S. *Ingeniería del Software: Un enfoque práctico*. Tercera Edición. Ed. Mc Graw Hill. España. 1993. Pág. 181.

CAPITULO II

ANALISIS DE REQUISITOS

La fase de análisis de requisitos, puede considerarse como el pilar donde se cimentarán todos los esfuerzos que realice el desarrollador de software. Esta fase deberá ser llevada a cabo con sumo cuidado por parte del desarrollador, de ello depende en mucho el éxito que se pueda tener.

El objetivo principal de la fase es la de lograr determinar con la mayor exactitud posible qué es lo que desea el usuario, y por lo tanto una de las fases en donde la participación de este es fundamental ya que proveerá al desarrollador de bases firmes para fases posteriores.

El análisis de requisitos como lo define Pressman, es un proceso de descubrimiento, refinamiento, modelización y especificación de las características que deberán integrar al sistema en estudio.([7])

Estos procesos los podemos generalizar en dos actividades que se llevan a cabo durante esta fase, las cuales llamaremos el análisis previo y la descripción. La primera consistirá en los procesos de descubrimiento y refinamiento, estos involucran la comprensión del problema, factores asociados y organización de la información que se recolectará. La descripción está constituida por los procesos de modelización y especificación de requisitos, que permitirán documentar y comunicar el resultado del

[7] Pressman, Roger S. *Ingeniería del Software: Un enfoque práctico*. Tercera Edición. Ed. Mc Graw Hill. España. 1993. Pág. 181.

análisis, presentándolos de una forma organizada y estructurada, auxiliándose de significativos complementos en forma de texto o gráficos.

El descubrimiento tiene como objetivo definir apropiadamente el problema, es decir, tener un entendimiento cabal de éste y conocer su entorno. Para lograr esto se definirán sus características como son el funcionamiento o los procesos que deberá llevar a cabo el sistema (si éste no existiese en forma manual), los datos que procesará, etc.; por lo tanto para lograr este objetivo el desarrollador buscará obtener la información con el o los usuarios involucrados.

El refinamiento de las características es un proceso donde el desarrollador realizará el análisis completo de la información recabada, de manera tal que permita a éste sintetizar el problema evitando duplicados de procesos y/o funciones mediante una evaluación del flujo y estructura de la información, para descubrir los posibles factores que afectarán al sistema.

Durante este proceso el desarrollador deberá ir contemplando las posibles soluciones, es importante señalar que esta solución deberá basarse en lo que requiere el sistema y no en el como lo realizará; es decir, qué datos, funciones e interfaces se requieren.

Una vez lograda la comprensión de las características que formarán parte del sistema, el desarrollador deberá hacer uso de herramientas y/o técnicas que le permitan representar un modelo que satisfaga las necesidades del sistema. Este proceso es llamado modelado.

Este modelado deberá ser llevado a cabo en forma conjunta con la especificación de requisitos, actividad que representará de una manera formal los datos y funciones involucradas, además de una indicación de las restricciones de diseño.

La figura 2.1 ilustra la forma en que estas actividades pueden incorporarse al modelo de ciclo de vida clásico, así como los procesos que ellas involucran. Se puede observar que existe la retroalimentación entre el bloque que nos representa el análisis previo y la descripción, ésta se presenta sólo cuando no son comprendidos en su totalidad los hechos investigados y se requiere mayor información o clarificar algunos aspectos del problema.

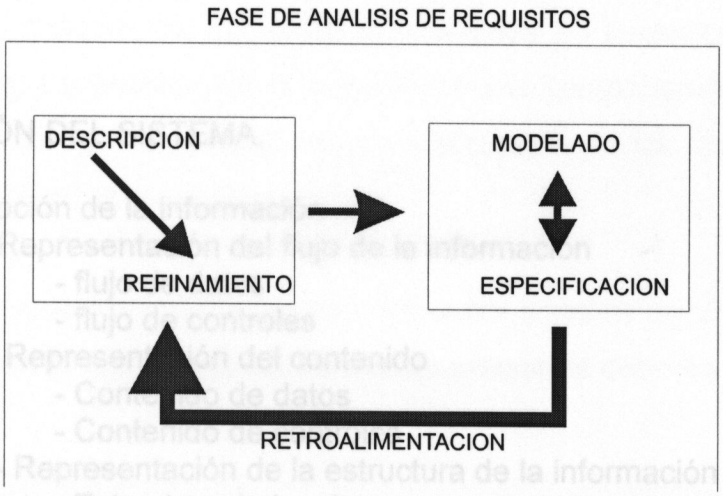


Figura 2.1 Retroalimentación de las actividades en la Fase de Análisis de Requisitos.

Es importante señalar que las actividades recaen en los siguientes dos puntos: El sistema en estudio y la información involucrada, desglosándose de la siguiente manera:

- Sistema
  - Comportamiento
  - Funciones

- Información
  - Flujo (cómo cambian los datos y controles)
  - Contenido (que comprenden los datos y controles)
  - Estructura (cómo se almacenan los datos y controles)

Al finalizar la fase de análisis se obtendrá un documento con la especificación de requisitos. Para la elaboración de este documento se han establecido un sin fin de formatos, en base a la propuesta de la asociación IEEE en su estándar 830-1984 se puede formular la siguiente propuesta como se muestra en la tabla 2:

1. INTRODUCCION

2. OBJETIVOS

3.- DESCRIPCIÓN DEL SISTEMA.

A) Descripción de la información
- I.- Representación del flujo de la información
  - flujo de datos
  - flujo de controles
- II.- Representación del contenido
  - Contenido de datos
  - Contenido de controles
- III.- Representación de la estructura de la información
  - Estructura de los datos
  - Estructura de controles

B) Descripción funcional del sistema
- I.- Descripción de las funciones que realiza
- II.- Restricción y limitaciones
- III.- Restricción de diseño

C) Descripción del comportamiento del sistema

4. ANEXOS CONVENIENTES

Tabla 2. Propuesta de documentación de la fase de análisis.

La introducción deberá contemplar un panorama general del sistema y los factores que han llevado a sugerir su desarrollo. En la sección de Objetivos se explicará claramente los objetivos a cubrir por el sistema en cuestión.

En la sección correspondiente a la descripción de la Información se representará el flujo de los datos y control, así como su contenido y una representación de la estructura de éstos.

La sección de Descripción funcional proporcionará una descripción para cada función requerida en el sistema, así como sus restricciones y/o limitaciones encontradas en cuanto a su diseño. La sección de la descripción del comportamiento del sistema examina a este a partir de sus reacciones como consecuencia de los factores externos y características de control generados internamente.

Cabe mencionar que los incisos de este esquema pueden cambiar dependiendo del tamaño del sistema propuesto, pero los elementos anteriores deberán ser incluidos.

Es importante destacar el cuidado excesivo que deberá considerar el desarrollador, ya que la especificación de requisitos puede caer en dos grandes riesgos: PRIMERO, puede no contener la totalidad de los verdaderos requisitos, y SEGUNDO, puede contener requisitos falsos. Un requisito falso es aquel que puede contemplarse su propósito sin necesidad de implementarlo o puede ser un requisito que su propósito se base en una actividad que sea totalmente innecesaria ([8]).

[8] De Marco T. *Analysis and System Specification*, Prentice Hall, U.S.A. 1979. Págs. 3 y 4

2.1. HERRAMIENTAS DE RECOPILACION DE INFORMACION

Para llevar a cabo la actividad de análisis es necesario contar con herramientas y/o técnicas que nos permitan recopilar hechos de estudio. Entre éstas se encuentran la entrevista, el cuestionario, la observación y la revisión de registros y documentos. El desarrollador podrá elegir una o más de estas técnicas según exista la disposición y lo que sea más conveniente utilizar.

2.1.1. LA ENTREVISTA

La entrevista es sin duda la técnica más significativa y productiva para encontrar hechos, objetivos del sistema, necesidades de información, operaciones, etc., la entrevista puede darse a todos los niveles de una organización desde el responsable del sistema, usuarios del sistema existente, usuarios en potencia del sistema propuesto. En términos sencillos, una entrevista es un intercambio cara-cara de información; es un careo de información entre el desarrollador y el usuario.

Como características significativas puede decirse que la entrevista puede emplearse para obtener apoyo comprensivo por parte del usuario acerca de una nueva idea o método que pueda aplicarse, además de proporcionar una oportunidad para establecer una relación de armonía con el usuario.

Para que la entrevista tenga éxito, es necesario que se establezca una línea guía antes, durante y después de ella, por ejemplo, para antes de una entrevista puede el desarrollador hacerse las siguientes preguntas:

¿A quién preguntar, y qué puesto ocupa?
¿En qué orden preguntar (si existiese mas de una persona)?
¿En qué fecha, hora y lugar se llevará a cabo la entrevista?
¿Qué temas se tratarán?

A partir de estas preguntas (y otras que se consideren necesarias) el desarrollador puede preparar un bosquejo de la entrevista considerando que no se interrogará a los entrevistados, si no que se conversará con ellos.

Además de lo anterior, el desarrollador deberá consultar y obtener la cooperación de todos los involucrados en el proyecto del sistema, tomando en cuenta los siguientes puntos:

- Convenir una cita por adelantado, no se deberá "caer de sorpresa".

- Identificar la posición del entrevistado dentro de la organización y las responsabilidades y/o actividades de su trabajo.

Durante la entrevista deberán tomarse en cuenta las siguientes consideraciones por parte del desarrollador:

- Explicar quién es, cuál es el propósito de la entrevista, de qué se trata el proyecto de sistemas, y de qué manera el entrevistado contribuirá al desarrollo de un nuevo sistema.

- Asegurarse de conocer correctamente las responsabilidades y deberes del entrevistado.

- Tratar de hacer preguntas específicas que permitan respuestas cuantitativas y cualitativas.

- Utilizar un vocabulario adecuado, dependiendo de quien sea el entrevistado.

- Aclarar completamente dudas sobre respuestas vagas.

- Determinar si el entrevistado tiene algunas ideas o sugerencias adicionales.

- Hacer un resumen de los principales puntos, al finalizar la entrevista.

LIMITANTES DE LA ENTREVISTA

Por ser una técnica de interacción directa con los involucrados en el proyecto, no siempre procede según lo planeado. Normalmente las personas reaccionan a una entrevista en formas diferentes, algunas favorables y otras desfavorables.

2.1.2. CUESTIONARIO

El cuestionario es otra técnica de mucha importancia para el desarrollador, ya que se puede utilizar en varios momentos durante el proceso de desarrollo de sistemas. Esta técnica es de mucha utilidad cuando se requiere de información concreta que involucra a un gran número de personas. En este caso se deben utilizar formatos estandarizados y preguntas concretas.

Su éxito se basa en gran manera en la comprensión por parte de los encuestados y de la honestidad de sus respuestas.([9])

Sin embargo esta técnica también tiene sus limitaciones, ya que no permite al desarrollador observar las reacciones o expresiones de los encuestados, por lo que las respuestas pueden ser limitadas y además se corre el riesgo de que no tenga mucha importancia para el encuestado el llenado del cuestionario.

Otras de las limitaciones radica en que es extremadamente difícil estructurar preguntas significativas sin anticipar una cierta respuesta y la capacidad de un seguimiento inmediato tiende a limitar el valor real de este tipo de comunicación.

Algunos de los puntos que se deben tomar en cuenta al elaborar un cuestionario son los siguientes; hacer preguntas claras y concisas; explicar el propósito, el uso y el destino de las respuestas; proporcionar instrucciones detalladas sobre la forma en que se desean contestar las preguntas; indicar una fecha límite o un plazo para la devolución del cuestionario; dar forma a la pregunta de manera que las respuestas puedan tabularse mecánica o manualmente; proporcionar un espacio suficiente para una respuesta

[9] Rodríguez Cuadrado, Alfredo; Márquez Serrano, Antonio. *Técnicas de Organización y Análisis de Sistemas*. Ed. Mc Graw Hill. España. 1993. Pág. 58.

completa; expresar las preguntas claramente; identificar cada cuestionario por nombre, puesto, departamento, etc., de la persona que lo contesta y por último incluir una sección en dónde las personas que contestan puedan expresar opiniones críticas.

2.1.3. LA OBSERVACION

La técnica de la observación puede emplearse para verificar lo que se reveló en una entrevista o como paso preliminar para ésta.

Ya que con esta técnica podemos descubrir detalles imposibles de conseguir con otras, y podemos observar las condiciones en que se desarrolla el trabajo, distribución del personal, el uso del teléfono, movimiento de personas entre mesas, manejo de documentos de documentos, etc.

Como resultado de lo anterior el desarrollador obtiene información de primera mano sobre la forma en que se efectúan las actividades.

Esta técnica es más útil cuando el desarrollador necesita observar, por un lado, la forma en que se manejan los documentos y se llevan a cabo los procesos y por otro, si se siguen todos los pasos especificados([10]).

[10] Burch, John G. y Gary, Grudnitski. *Diseño de Sistemas de Información*. Ed. Megabyte. México. 1994.Págs. 637, 642.

Por último, para maximizar los resultados que se obtienen de las observaciones, el desarrollador deberá seguir las siguientes guías:

- ☐ Preparación para la Observación
- ☐ Realización de la Observación
- ☐ Seguimiento de la Observación

2.1.4. REVISION DE REGISTROS Y DOCUMENTOS

Esta técnica nos sirve para realizar una evaluación tanto cualitativa como cuantitativa con respecto a las organizaciones y a sus operaciones.

Al revisar los registros el desarrollador examina la información asentada en ellos relacionados con el sistema y los usuarios, como son los manuales de procedimientos y normas, estándares de operación, reglamentos internos, publicaciones sobre políticas de la organización, etc.

Esta revisión se puede efectuar al comienzo del estudio, o después y sirve como base para comparar las operaciones actuales, por lo que los registros pueden indicar qué está sucediendo.

Sin embargo, estos registros no indican la forma en la que se desarrollan las actividades en la realidad.

2.2. HERRAMIENTAS DE DESCRIPCION

La descripción gráfica juega un papel muy importante en el análisis de sistemas complejos y la descripción de éste. Se dice que un dibujo vale más que mil palabras, sin embargo este dibujo deberá ser conciso, preciso y claro. En el desarrollo de un sistema cuando se conjugan factores como la falta de experiencia del desarrollador, la incapacidad del usuario por describir adecuadamente las necesidades del sistema o la complejidad de éste es grande, pueden estos factores conducirnos a un análisis deficiente o incorrecto.

Al igual que las herramientas de recopilación de información, las herramientas de descripción auxilian al desarrollador a obtener una comprensión cabal del problema, además de ser una fuerte base para el modelado y especificación de requisitos. La funciones que realizan son las siguientes:

2.2.1. ARBOLES Y TABLAS DE DECISION

Un árbol de decisión es un diagrama que nos facilita la representación de una forma secuencial condiciones y acciones; esta técnica también nos permite mostrar la relación que existe entre cada condición y el grupo de acciones asociados con ella.

El desarrollo de árboles de decisión beneficia al desarrollador en dos formas, por una parte la necesidad de descubrir condiciones y acciones llevan a los desarrolladores a identificar de manera formal las decisiones que actualmente deben tomarse. De esta forma será difícil pasar por alto cualquier etapa del proceso de definición, sin importar que esta dependa de variables cuantitativas y cualitativas.

Por otro lado, los árboles de decisión también obligan a los desarrolladores a considerar las secuencias de las decisiones. Por ejemplo, considérese que se desea realizar un descuento al cliente a partir de la forma de pago y la cantidad consumida, dadas las siguientes condiciones: si el cliente paga a crédito no se hace descuento, si paga al contado se aplican descuentos de: 10%, 5% y 2% si las cantidades corresponden a 100,000, entre 100,000 y 50,000 y 50,000 pesos respectivamente. La figura 2.2 describe el uso de árboles bajo este proceso.

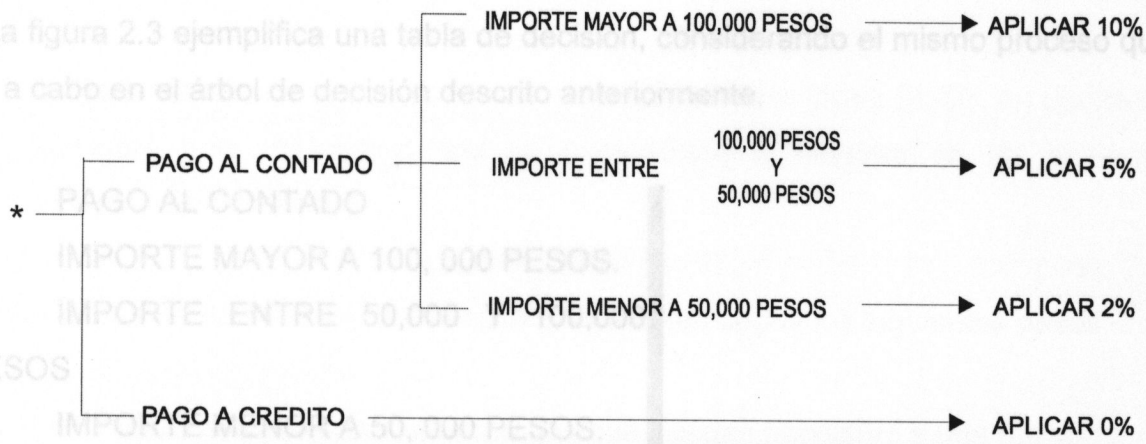


Figura 2.2 Descripción de árbol de decisión.

Una tabla de decisión es una herramienta alternativa para la descripción de decisiones lógicas complejas. Una tabla de decisión está constituida por las siguientes cuatro secciones:

- **Una zona de condiciones**. Esta zona contiene una lista de todas las condiciones o variables que deberán ser evaluadas.

- **Una zona de acciones**. Enlista el conjunto de acciones que se pueden producir basándose en combinaciones de las condiciones.

- **Entrada o especificación de acciones**. Muestran las acciones específicas de un conjunto, cuando ciertas condiciones o combinaciones de éstas son verdaderas.

- **Entrada o especificación de condiciones**. Indica qué valor le corresponde para una determinada combinación. El valor dependerá del tipo de entrada en la tabla.

La figura 2.3 ejemplifica una tabla de decisión, considerando el mismo proceso que se lleva a cabo en el árbol de decisión descrito anteriormente.

PAGO AL CONTADO	
IMPORTE MAYOR A 100, 000 PESOS.	
IMPORTE ENTRE 50,000 Y 100,000 PESOS	
IMPORTE MENOR A 50, 000 PESOS.	
	APLICAR 10% DE DESCUENTO
	APLICAR 5% DE DESCUENTO
	APLICAR 2% DE DESCUENTO
	NO APLICAR DESCUENTO

Figura 2.3 Descripción de una tabla de decisión.

Las formas de entrada son las siguientes:

- **Forma de entrada limitada**. Consiste en utilizar S ó N (Si ó No), y entrada en blanco, es uno de los formatos más comunes.

- **Forma de entrada extendida**. Esta forma reemplaza la S y N con acción que le indica al lector como decidir, tiene la ventaja de ser más explícito para señalar las acciones que el formato anterior.

- **Forma de entrada mixta**. Es la combinación de características de los dos formatos anteriores en la misma tabla, sólo debe utilizarse una forma en cada sección de la tabla; pero entre las zonas de condiciones y acciones se puede utilizar cualquier forma.

- **Entrada ELSE**. Es la cuarta variación conocida como forma ELSE, en donde se agregará una columna donde se especificará si ninguna de las acciones anteriores es verdadera.

Para el desarrollo de tablas de decisión, se pueden seguir los siguientes pasos:

1. Identificar y enlistar todas las acciones que puedan asociarse a una función.
2. Identificar y listar todas las acciones factibles durante la generación de la función.
3. Estudiar las distintas posibilidades de combinaciones y acciones, y asociarlas en conjuntos específicos, eliminando las combinaciones imposibles, desarrollar alternativamente permutaciones de condiciones.
4. Definir las reglas indicando qué acciones ocurren para un conjunto de condiciones.

A partir de las características de cada uno de ellos, podemos elegir su uso bajo los siguientes criterios; los árboles de decisión son más fáciles de usar y comprender, cuando

el número de condiciones es pequeño; cuando este número tiende a ser grande es preferible utilizar las tablas de decisión, ya que éstas nos presentan una visión más amplia de la situación y permiten separar procedimientos a seguir cuando una condición en particular se presenta.

Para un modelo enfocado a los pequeños y medianos desarrollos, independientemente considerando la metodología utilizada y que esta propone su propia notación como antes se mencionaba, las siguientes herramientas son suficientes para llevar a cabo la fase de análisis de requisitos.

2.2.2. DIAGRAMAS DE FLUJO DE DATOS

Los diagramas de flujo de datos, son uno de los elementos básicos del análisis estructurado, sin embargo, esta herramienta ha sido incorporada por otras metodologías, su importancia recae en que el diagrama de flujo de datos nos representa en forma gráfica el flujo de la información y las transformaciones que se aplican a los datos cuando éstos se mueven desde la entrada hasta la salida.

Este tipo de diagramas se derivan de los trabajos elaborados por Constantine y Yourdon (1979), fueron popularizados por Yourdon y De Marco, posteriormente han surgido ampliaciones que permiten tener una mejor descripción del análisis.([11])

Los símbolos básicos para el diseño de un diagrama de flujo de datos se pueden representar en dos formas, la primera propuesta por Yourdon y la segunda por Gane y Sarson. La figura 2.4 describe estos símbolos.

---

11 Sommerville, Ian. *Ingeniería de Software.* Ed. Adisson-Wesley Iberoamericana, Versión en Español. U.S.A. 1988.Pág. 78.

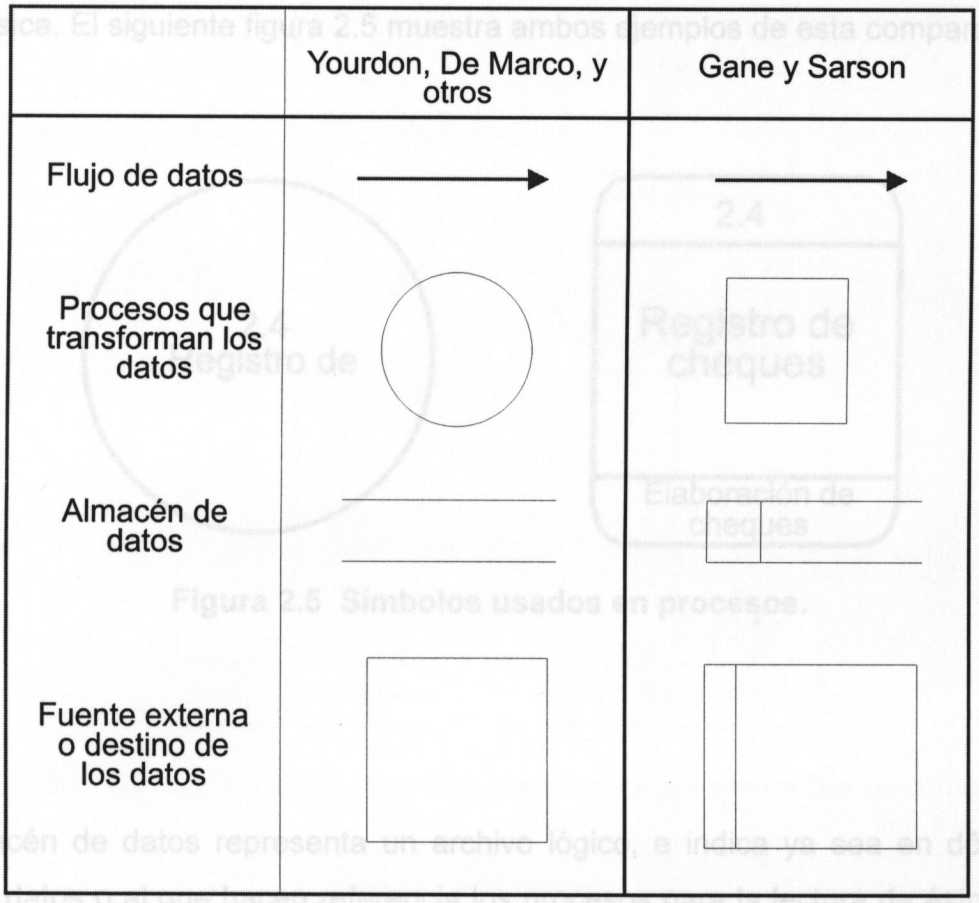


Figura 2.4 Símbolos Usados en los diagramas de flujo.

El flujo de datos nos representa el movimiento de datos, la flecha nos indica dirección desde su origen hacia un destino, el flujo de datos puede considerarse un "paquete" de datos, éste puede ser identificado por su nombre escrito a lo largo de la correspondiente flecha que conectará los procesos.

Los procesos es un componente procedural en el sistema y opera en los datos transformándolos. Por ejemplo puede representar operaciones aritméticas o lógicas en los datos produciendo sus respectivos resultados. Cada proceso puede ser identificado por un nombre que describa la acción y un número. El símbolo propuesto por Gane y Sarson

puede además de contener lo anterior utilizar otra descripción opcional para indicar su localización física. El siguiente figura 2.5 muestra ambos ejemplos de esta comparación.

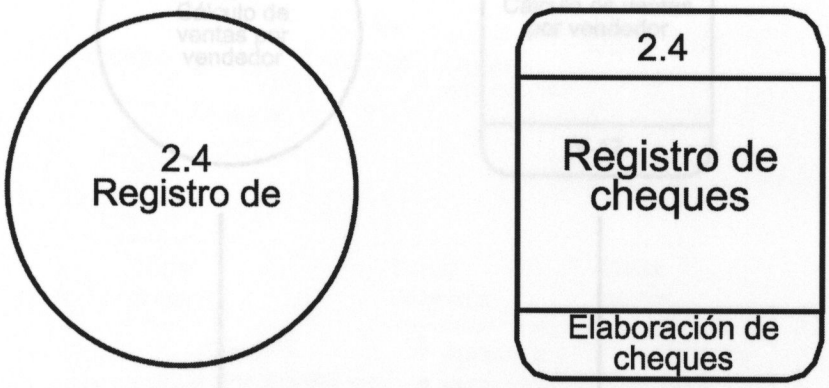


Figura 2.5 Símbolos usados en procesos.

El almacén de datos representa un archivo lógico, e indica ya sea en dónde se guardarán los datos o al que hacen referencia los procesos para la lectura de éstos. Este símbolo en el caso de Gane y Sarson contiene una identificación del almacén. En la siguiente figura 2.6 se presenta un ejemplo con un proceso para la generación de pagos de comisiones al vendedor que almacenará el resultado en un archivo de vendedores, con ambas notaciones.

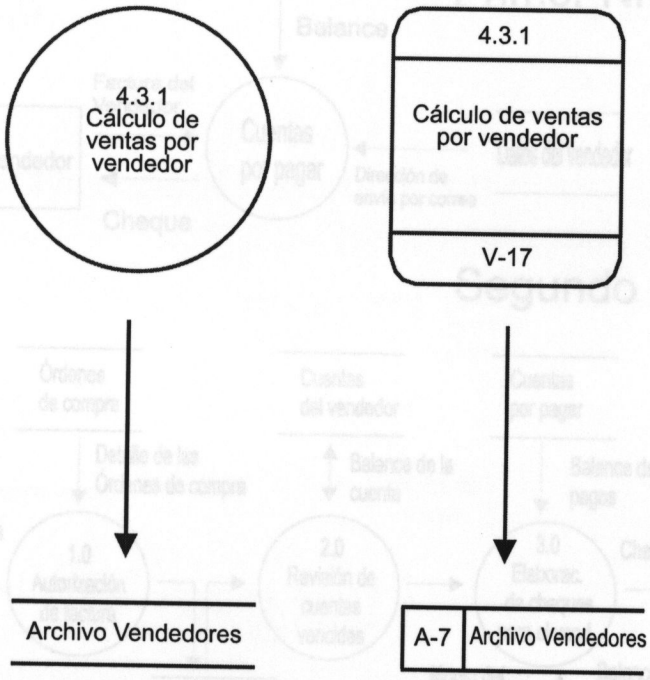


Figura 2.6 Notación que ejemplifica y compara los símbolos de almacén.

La fuente o destino de los datos nos representa entidades, tales como personas o programas ú otras entidades que interactúan con el sistema pero que se encuentran fuera de su frontera.

Este conjunto de elementos son suficientes para construir una representación en red de un sistema, mostrando sus procesos y las interfaces de datos entre ellos; la figura 2.7a y 2.7b ejemplifican el uso utilizando una notación de Yourdon, De Marco y otros.

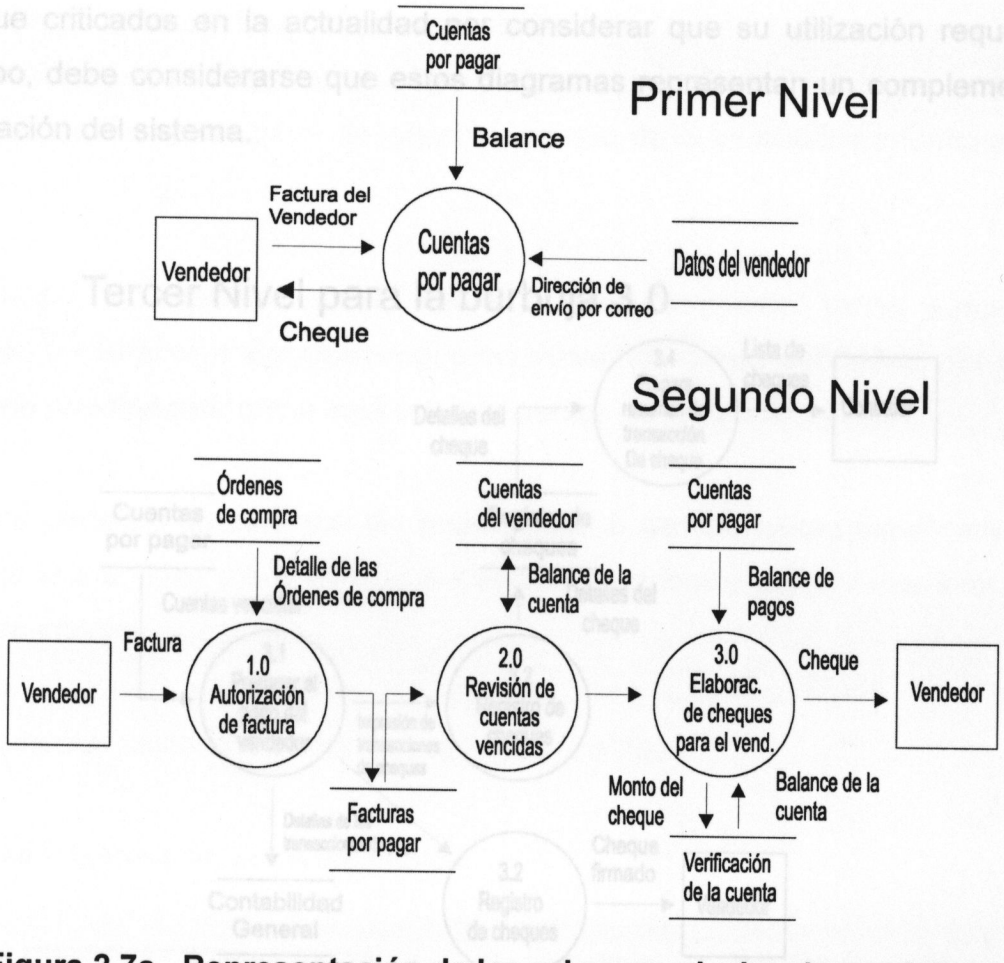


Figura 2.7a. Representación de los primeros niveles de un sistema.

Como puede observarse en este primer nivel se identifica el proceso de cuentas por pagar y dos almacenes los cuales son: cuentas por pagar y datos del vendedor, por otro lado la fuente o destino vendedor.

En el segundo nivel se lleva a cabo una descomposición del proceso de cuentas por pagar. En la figura 2.7b se presenta la descomposición del proceso 3.0

Aunque criticados en la actualidad por considerar que su utilización requiere de mucho tiempo, debe considerarse que estos diagramas representan un complemento en la documentación del sistema.

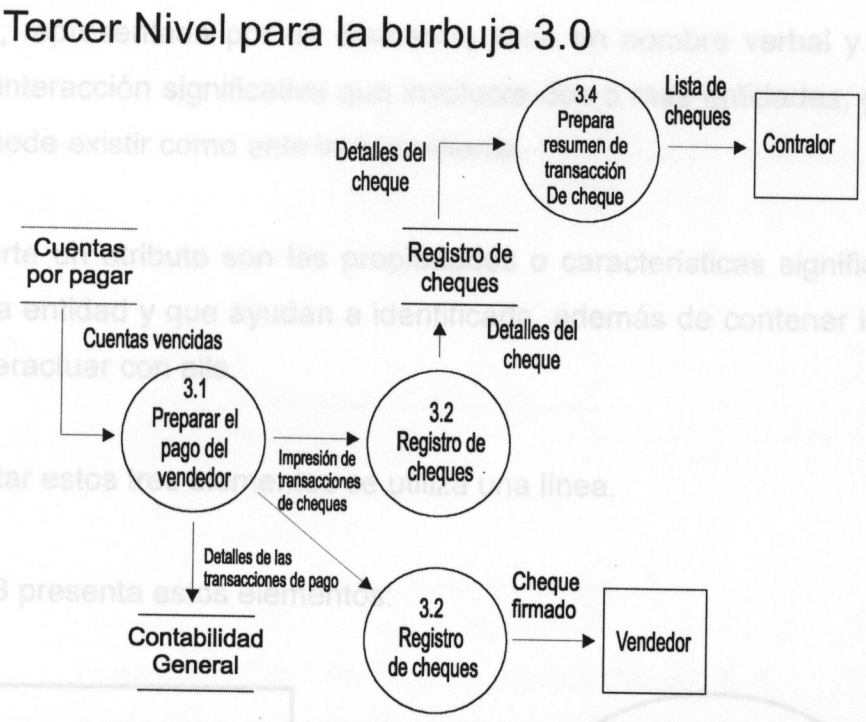


Figura 2.7b. Representación de un tercer nivel de un sistema.

2.2.3. DIAGRAMAS DE ENTIDAD-RELACION

Una de las herramientas gráficas principales para la modelización de datos son los diagramas de Entidad-Relación. Distintos autores han contribuido para el desarrollo de una notación Entidad-Relación, todos ellos identifican un conjunto de componentes primarios como son: objetos de datos, los atributos, relaciones y varios indicadores de tipo. Su principal propósito es el de representar los objetos de datos y sus relaciones.

Los objetos de datos (Entidad) es representada con un rectángulo, tiene un nombre único y su finalidad es representar un objeto o grupo de objetos (personas, conceptos, cosas) que pasarán a formar parte del sistema, ya que de él se obtiene información o la almacena.

La relación, representada por un diamante, tiene un nombre verbal y representa una asociación o interacción significativa que involucra dos o más entidades, por lo tanto una relación no puede existir como ente independiente.

Por otra parte un atributo son las propiedades o características significativas con las que consta una entidad y que ayudan a identificarla, además de contener información necesaria para interactuar con ella.

Para conectar estos tres elementos se utiliza una línea.

La figura 2.8 presenta estos elementos:

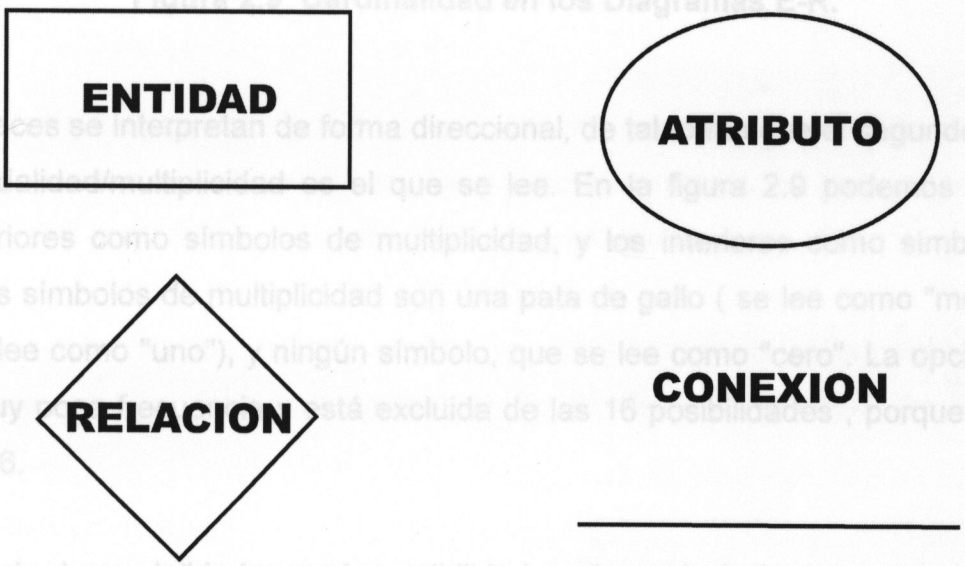


Figura 2.8 Figuras representativas de los diagramas de E-R.

MULTIPLICIDAD Y MODALIDAD

Los enlaces entre las entidades tienen dos clases de propiedades, la cardinalidad (también llamada multiplicidad) y la modalidad. La cardinalidad de un enlace puede ser uno-a-uno o muchos-a-uno y la modalidad puede ser necesaria o posible.

Existen 16 tipos de enlaces posibles que conectan dos entidades o relaciones. La figura 2.9 nos ejemplifican los elementos que nos permitirán representar ambas propiedades.

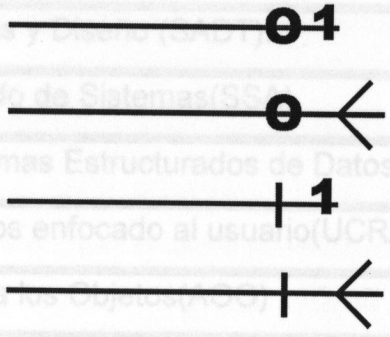


Figura 2.9 Cardinalidad en los Diagramas E-R.

Los enlaces se interpretan de forma direccional, de tal forma que el segundo par de iconos de modalidad/multiplicidad es el que se lee. En la figura 2.9 podemos leer los símbolos exteriores como símbolos de multiplicidad, y los interiores como símbolos de modalidad. Los símbolos de multiplicidad son una pata de gallo (se lee como "muchos"), una barra (se lee como "uno"), y ningún símbolo, que se lee como "cero". La opción cero se usa con muy poca frecuencia y está excluida de las 16 posibilidades , porque de otro modo habría 36.

El símbolo de modalidad, para la posibilidad es 0, es decir tiene una participación opcional. El símbolo de la necesidad es una barra, | es decir obligatoria.

2.3.- METODOS DISPONIBLES DE ANALISIS

Para llevar a cabo la actividad de descripción se cuenta con una serie de métodos que incorporan sus propias herramientas gráficas y/o técnicas que facilitarán al desarrollador la aplicación de los principios fundamentales de los procesos de una manera sistemática y generar las especificaciones correspondientes. Dentro de los métodos más sobresalientes podemos citar los representados en la tabla 3:

Técnicas de Análisis y Diseño (SADT).
Análisis Estructurado de Sistemas(SSA)
Desarrollo de Sistemas Estructurados de Datos(DSED)
Análisis de requisitos enfocado al usuario(UCRA)
Análisis Orientado a los Objetos(AOO)

Tabla 3 Métodos disponibles de análisis.

Aunque cada método introduce su propia notación y heurística de análisis, excepto el AOO, todos los métodos concuerdan en algunas guías comunes. Se enfocan (directa o indirectamente) al flujo de datos y al contexto o estructura de los datos.

En la mayoría de los casos, el flujo se caracteriza en el contexto de las transformaciones que se aplican en la entrada de datos y que repercuten en una salida de información.

Es importante señalar también, que en todos los métodos se encuentran incluidos los conceptos de partición del problema y la abstracción del mismo. Elementos

importantes que facilitarán al desarrollador la representación del dominio de la información y el dominio funcional en los diferentes conceptos de abstracción.

Por ejemplo, todos los métodos facilitan al desarrollador la representación funcional de un proceso mediante un gráfico, sin embargo puede subdividirse a un nivel más bajo de abstracción, usando una notación descriptiva de función tal como un lenguaje de procedimientos.

La partición permite descomponer un problema en sus partes constituyentes, al exponer cada vez con más detalles cada uno de los elementos, éstos llegarán a una descomposición funcional. Siempre se deberá tener en cuenta una jerarquía horizontal.

Otra guía con la que cuentan en común los métodos, se caracteriza por permitir una evaluación física del problema, para después derivar en una solución lógica. Los métodos más usados en la actualidad son el SSA y SADT, sin embargo el método AOO cada vez se extiende más su popularidad, aunque todavía no lo suficiente como para desplazar en su totalidad a los anteriores.

A continuación se presenta una breve descripción de algunos métodos que tienen aplicación en el desarrollo de un sistema de información.

2.3.1. TECNICAS DE ANALISIS Y DISEÑO (SADT)

SADT es una de las técnicas de análisis y diseño más utilizadas para la obtención y análisis de requisitos. SADT es el nombre registrado de una notación desarrollada por D. T. Ross y colegas de SoftTech, Inc., sin embargo, aunque fué en 1977 cuando SoftTech se adjudicó los derechos, sus inicios de remontan desde 1960 y se considera que se aplicó por primera vez en 1974.

Consiste básicamente en un conjunto de procedimientos que permiten descomponer las funciones del sistema; una notación gráfica compuesta por actigramas y datagramas que representan las relaciones de información y las funciones con el software, además de un conjunto de directrices para aplicar el método. Como puede observarse parte de un enfoque orientado al flujo de datos, para esto utiliza la descomposición funcional, así como prácticas de programación como es el refinamiento sucesivo, la abstracción de procedimientos y los tipos de datos abstractos.

Cuando se utiliza SADT se desarrolla un modelo compuesto por muchos actigramas y datagramas definidos en forma jerárquica. Los símbolos básicos son el rectángulo y las flechas. El actigrama nos permite representar las actividades y el datagrama los datos.

En un actigrama los nodos se denotan actividades y los arcos especifican flujos de datos entre las actividades. Los datagramas especifican datos en los nodos y actividades en los arcos. Por esto, los actigramas y los datagramas son duales. Estas estructuras jerárquicas nos permiten trabajar a distintos niveles de detalle cuando es necesario.

La figura 2.10 ilustra los símbolos básicos, así como el diseño de un actigrama y un datagrama.

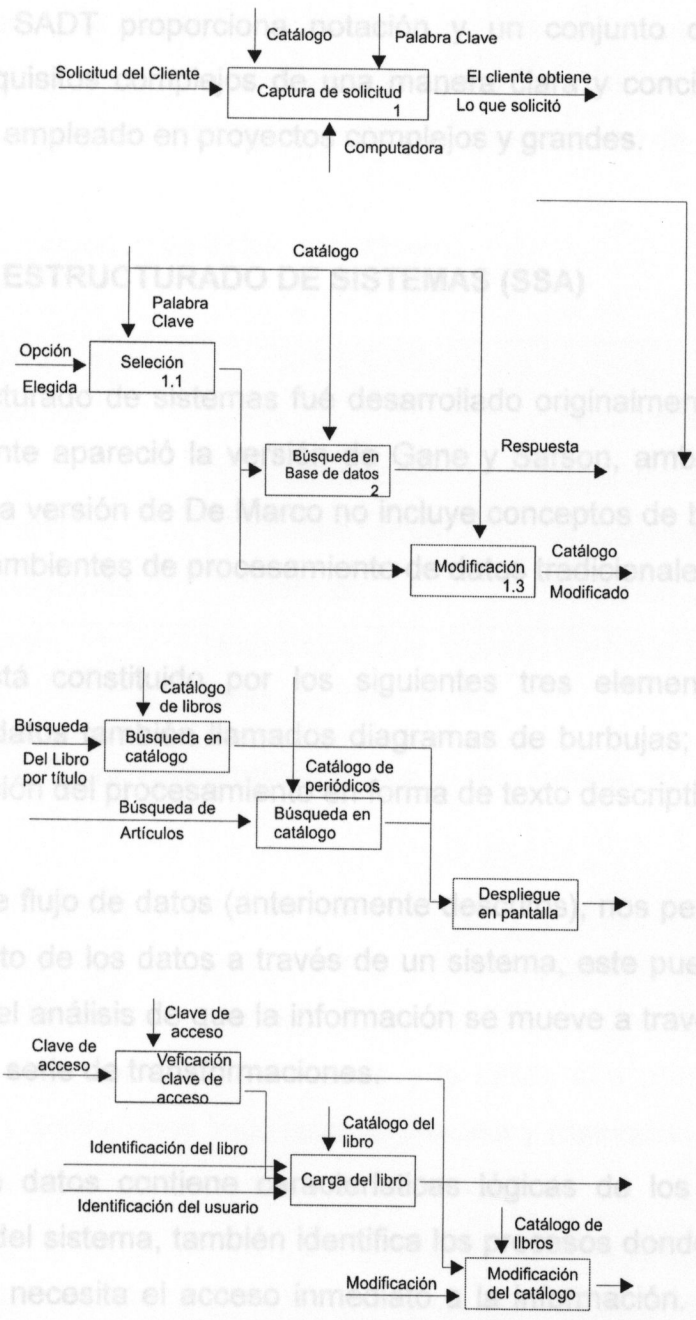


Figura 2.10 Diagramas representativos del método SADT.

Uno de los principales objetivos del método es proporcionar un medio donde una gran variedad de situaciones complejas se puedan especificar empleando descomposición y relaciones estructurales.

La metodología SADT proporciona notación y un conjunto de técnicas para entender y registrar requisitos complejos de una manera clara y concisa. Por otro lado SADT, sólo debería ser empleado en proyectos complejos y grandes.

2.3.2. ANALISIS ESTRUCTURADO DE SISTEMAS (SSA)

El análisis estructurado de sistemas fué desarrollado originalmente por Yourdon y De Marco, posteriormente apareció la versión de Gane y Sarson, ambas versiones son similares, sin embargo la versión de De Marco no incluye conceptos de base de datos, su aplicación concierne a ambientes de procesamiento de datos tradicionales.

Este método está constituido por los siguientes tres elementos básicos, los diagramas de flujo de datos también llamados diagramas de burbujas; el diccionario de datos y una representación del procesamiento en forma de texto descriptivo.

Los diagramas de flujo de datos (anteriormente descritos), nos permiten describir y analizar el procesamiento de los datos a través de un sistema, este puede ser manual o automático, partiendo del análisis de que la información se mueve a través de un sistema y es modificada por una serie de transformaciones.

El diccionario de datos contiene características lógicas de los sitios donde se almacenarán los datos del sistema, también identifica los procesos donde se emplean los datos y sitios donde se necesita el acceso inmediato a la información. Su utilización es significativa para identificar los requisitos de la base de datos durante el diseño del sistema.

Este diccionario de datos consiste de un conjunto de definiciones de datos declarados en los diagramas de flujo y los almacenes de datos, especificando el dominio

de los elementos de datos por medio de limitantes y especificando qué elementos, si los datos son discretos o continuos. El diccionario de datos es conciso y no redundante, utiliza un conjunto de operadores lógicos para producir la descripción de datos.

La tabla 4 describe algunos de estos operadores.

SIMBOLO	SIGNIFICADO	EXPLICACIÓN	USO
=	Equivalente	Alias	Sinónimos
+	Y	Concatenación	Relación de secuencia
[ ]	Uno u otro	Opciones "entre"	Relación de selección
{ }	Interaciones de	Repetición	Relación de interación
( )	Opcional	Interación que ocurre sólo cero o una vez	Relación opcional

Tabla 4 Operadores utilizados en el diccionario de datos.

Para complementar las descripciones que hacen los diagramas de flujo se puede anexar un texto descriptivo, que nos permitirá especificar los detalles del procesamiento que implica una burbuja del diagrama del flujo de datos. El texto describe la entrada a la burbuja, el algoritmo que se aplica a esa entrada y la salida que produce; además de indicar las restricciones y limitaciones impuestas al proceso y restricciones de diseño que puedan tener influencia en la forma de representar el proceso.

La principal ventaja de este método es que quizá ha sido el más probado para el desarrollo de sistemas, actualmente ha provocado muchas críticas y controversias, sin embargo sigue vigente a tal punto que se han desarrollado ampliaciones de este método por varios estudiosos de la materia.

2.3.3. DESARROLLO DE SISTEMAS ESTRUCTURADOS DE DATOS (DSED)

El Desarrollo de Sistemas Estructurados de Datos, es también conocido como Metodología de Warnier-Orr, y se enfoca a la estructura de los datos.

DSED, examina primero el contexto de la aplicación, esto es, cómo se mueven los datos entre productores y consumidores de la información desde la perspectiva de uno de los productores o consumidores. A continuación se establecen las funciones de aplicación con una representación similar a la de Warnier, que describe los elementos de la información y el procesamiento que debe ejecutarse sobre ellos (similar al concepto de diagramas de flujo).

Finalmente, se modelan los resultados de la aplicación usando el diagrama de Warnier. Usando este método, DSED comprende todos los atributos del dominio de información: flujo, contenido y estructura de datos.

DSED propone un diagrama de entidades, este diagrama utiliza una notación que es, lamentablemente muy parecida al diagrama de flujo de datos, sin embargo, símbolos similares tienen diferentes significados. El círculo en un diagrama de entidades corresponde a un productor o a un consumidor de información. Después de revisar cada diagrama de entidades para comprobar que es correcto, se crea un diagrama combinado de entidades con todos los productores y consumidores de información. Las entidades que se encuentran dentro de los límites del sistema propuesto se identifican estableciendo un límite de la aplicación. Se pueden ocultar los detalles internos del límite de la aplicación. El sistema de procesamiento automático a analizar debe procesar la información que se mueve a través del límite de la aplicación.

Mediante una notación de tipo Warnier denominada diagrama de línea de ensamblaje (DLE), DSED proporciona un mecanismo para acoplar la información y los procesos que se le aplican. Conceptualmente el DLE desempeña el mismo papel que el diagrama de flujo de datos.

Se refina cada proceso del DLE desarrollando una narrativa de procesamiento que describa su salida, acción, frecuencia de la acción y entrada. Para representar los detalles procedimentales de cada proceso se puede usar un diagrama de Warnier-Orr.

DSED requiere que el analista construya un prototipo en papel de la salida deseada para el sistema, el prototipo identifica la salida primaria del sistema y la organización de los elementos de información que componen la salida. Una vez que ha sido creado un prototipo, se puede modelizar la jerarquía de la información usando un diagrama de Warnier-Orr.

2.3.4. ANALISIS DE REQUISITOS ENFOCADO AL USUARIO (UCRA)

UCRA es un método para el desarrollo de requisitos a un nivel de gran detalle, este método combina conceptos introducidos por Yourdon/De Marco y Gane y Sarson para el proceso de modelado basado en los diagramas de flujo de datos, utilizado para la verificación de requisitos en los estándares militares su uso se remota a los años de 1984 y 1985 en proyectos del Departamento de Defensa de los Estados Unidos.

También incorpora técnicas de modelado de Bachman, Chen y Martin, los cuales utilizan diagramas de entidad-relación. UCRA utiliza el proceso de refinamiento por pasos para definir los requisitos de tal manera que puedan ser fácilmente comprendidos y criticados por los usuarios finales, en vez de una vaga especificación de requisitos desde el punto de vista usuario, seguido de una especificación de requisitos, técnica del sistema

orientada al desarrollador. La especificación resultante es guiada por el usuario permitiendo al desarrollador saltarse el tradicional paso de especificación del sistema.

Los modelos UCRA de proceso funcional y modelos de bases de datos, se realizan en forma paralela iniciándose con diagramas que son complementados con texto para detallarlos.

2.3.5. ANALISIS ORIENTADO A LOS OBJETOS (AOO)

Aunque la programación orientada a objetos constituye actualmente una alternativa para el desarrollo de sistemas, hasta la fecha no se ha logrado estandarizar un método único para el análisis orientado a objetos.

Esto, ya que algunos aseguran que los métodos de desarrollo de sistemas orientados a objetos no pueden descomponerse en las fases tradicionales como son análisis y diseño, por lo contrario aseguran que durante el desarrollo de sistemas los métodos orientados a los objetos integran ambas fases y no es necesaria identificarlas por separado.

Sin embargo, Graham proporciona una clasificación de los métodos de desarrollo orientados a objetos, considerando que los sistemas poseen tres aspectos de especial importancia concernientes a: a) Los datos, objetos o conceptos y su estructura, b) arquitectura y proceso no temporal, c) la dinámica o comportamiento del sistema; nombrándolos: Datos, Procesos y Dinámica o Controles respectivamente.

A partir de esta consideración Graham desglosa los métodos orientados a objetos en dos tipos básicos a los cuales llama aproximación ternaria y unaria. Los primeros considera que son aquellos que reproducen métodos estructurados existentes, utilizando

tres notaciones distintas para datos, dinámica y procesos. La segunda son aquéllos que afirman que dado que los objetos combinan de modo nato los procesos (métodos) y los datos, solamente es necesaria una notación.([12])

Los trabajos sobre métodos orientados a los objetos actualmente rebasan los 50, sin embargo, se mencionarán las más renombradas clasificándolas bajo las consideraciones anteriores, la Tabla 5 muestra esta clasificación.([13])

TRILATERALES
OOSA De Shlaer/Mellor
OMT- Rambaugh
Ptech-Martin/Odell
OOSE
UNARIOS
Coad/Yourdon
CRC-Wirfs-Brock et al.

Tabla 5 Operadores utilizados en el diccionario de datos.

Sin embargo, para comprender la tendencia del AOO es necesario explicar algunos conceptos que se aplican en casi todos los métodos orientados a objetos, estos conceptos son los siguientes: objetos, los cuales son representaciones del mundo real; se pueden asociar a un conjunto de atributos los cuales son características genéricas de cada objeto

[12] Ian, Graham. *Métodos Orientados a Objetos*. Segunda Edición. Ed. Adison-Wesley. E.U. 1994. Pág.290.

[13] Ibid. Pág. 405

de una clase, una clase es un conjunto de objetos. Un miembro de la clase hereda todos los atributos definidos para la clase. Una vez que se ha definido la clase, se pueden reutilizar los atributos creando nuevas instancias de la clase, cada una de esas operaciones modifican uno o más atributos del objeto.

Encapsulamiento, los objetos encapsulan datos, operaciones, otros objetos, constantes y otra información relacionada. La encapsulación significa que toda esa información está empaquetada bajo un sólo nombre y puede ser reutilizada como especificación o como componente de programa.

También es importante darse cuenta de los que no son objeto, en general, un objeto no debe tener nunca un "nombre procedimental imperativo".

Para desarrollar un conjunto significativo de atributos para un objeto, el desarrollador debe estudiar de nuevo la narrativa de procesamiento para el problema concreto y seleccionar aquello de "pertenezca" de forma razonable al objeto.

Una operación cambia un objeto de alguna forma, más concretamente, cambia valores de uno o más atributos que están contenidos en el objeto.

Por tanto, una operación debe tener "conocimiento" de la naturaleza de los atributos del objeto y deben ser implementadas de forma que se les permita manipular las estructuras de datos que hayan sido derivadas de los atributos.

Aunque existen muchos tipos distintos de operaciones, generalmente se pueden dividir en tres grandes categorías:

1.- Operaciones que manipulan los datos.
2.- Operaciones que realizan algún cálculo.
3.- Operaciones que monitorizan un objeto.

Para realizar un "primer intento" de obtener un conjunto de operaciones para los objetos del modelo de análisis, el desarrollador ha de volver a estudiar la narrativa de procesamiento del problema y seleccionar aquellas operaciones que "correspondan" razonablemente a cada objeto. Para hacerlo, se lleva a cabo un nuevo análisis gramatical y se aíslan los verbos. Algunos de los verbos serán operaciones legítimas y se podrán conectar fácilmente con un objeto específico.

Tras una modificación más detallada, es probable que haya que dividir la operación, programar en varias suboperaciones más especificas requeridas para configurar el sistema.

Además con el análisis gramatical, podemos comprender mejor las operaciones considerando la comunicación que ocurre entre los objetos. Los objetos se comunican mediante el paso de mensajes a otros objetos.

Para que se pueda construir el sistema se debe establecer un mecanismo para la comunicación entre los objetos; este mecanismo se le denomina mensaje.

Los análisis anteriores corresponden a un intento de unificar las distintas perspectivas que surgen de los distintos trabajos conocidos sobre orientación a objetos. Por otra parte la notación que se utiliza para la modelización sí difieren considerablemente.

CAPITULO III

DISEÑO DEL SISTEMA

Diseñar tiene como significado trazar, describir o llevar a cabo un bosquejo de algo ya sea mediante palabras o gráficas. El diseño, en el desarrollo de software puede considerarse un arte ya que el desarrollador de sistemas deberá hacer uso de sus cualidades creativas para llevar a cabo este proceso.

La fase de diseño de sistemas puede definirse como el proceso de aplicar técnicas y principios con la finalidad de definir un modelo con los suficientes detalles que represente la mejor solución para satisfacer las necesidades de un sistema.

El diseño determina el éxito del sistema, ya que constituye la base para las fases consecuentes. El diseño inicia cuando el análisis del sistema a producido un conjunto de requisitos funcionales para un sistema y finaliza cuando el desarrollador ha logrado especificar los componentes del sistema y su relación entre ellos.

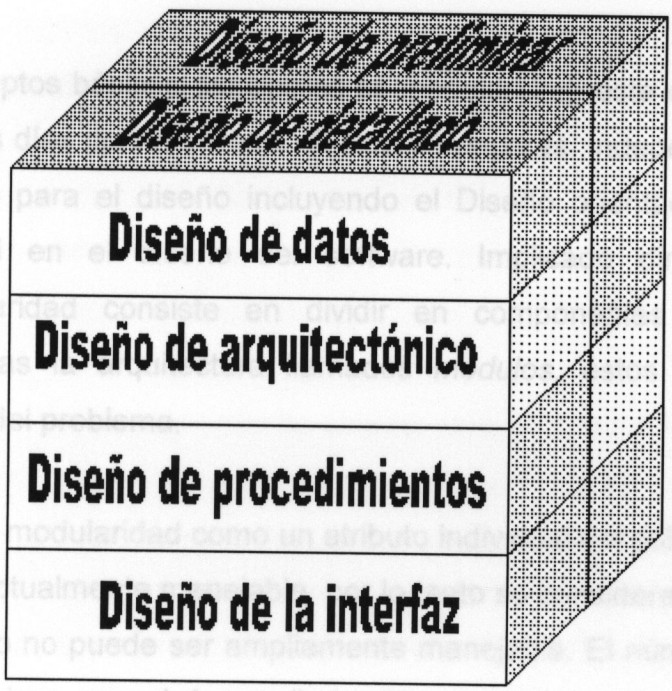
Un buen diseño representa calidad en el software, ésta no ha sido tomada en cuenta por algunos desarrolladores en los términos tan amplios que se requiere, como hemos visto en el capítulo I, sin embargo erróneamente algunos asocian durante esta fase la calidad con un incremento en la velocidad de ejecución y poco requisito de memoria.

Esta fase diseño, se realiza en dos pasos: El diseño preliminar también conocido como diseño de alto nivel o externo y el diseño detallado o interno; el primero tiene como propósito servir como fundamento para el segundo, se centra en el proceso de la

transformación de los requisitos de los datos, la arquitectura del software y los esquemas generales que establecerán la disposición de los mecanismos para la interacción del usuario con el software.

El diseño detallado se encarga del refinamiento de la representación arquitectónica, la estructura de los datos, representaciones algorítmicas del software y de la interfaz.

Los niveles anteriores pueden aplicarse a los datos, procedimientos, arquitectura e interfaz del usuario. Pressman con la siguiente figura, describe como los niveles de diseño se presentan sobre estos elementos ([14]).



3.1 Relación entre las actividades de Diseño.

[14] Pressman, Roger S. *Ingeniería del Software: Un enfoque práctico.* Tercera Edición. Ed. Mc Graw Hill. España. 1993.pag. 332.

Para realizar un buen diseño deben tomarse algunos conceptos básicos que darán una sólida base al desarrollador de software para aplicar alguna de las metodologías existentes más recientes.

Algunos de estos conceptos llevan más de 4 décadas en uso y siguen siendo fundamentales para las nuevas metodologías que se están desarrollando en los noventas y a principios del siglo XXI.

3.1. MODULARIDAD

Uno de los conceptos básicos principales es modularidad, este concepto parte del principio conocido en los días de Julio Cesar: Divide y Vencerás. Independientemente de la metodología utilizada para el diseño incluyendo el Diseño orientado a objetos este concepto es primordial en el diseño del software. Implicado directamente en la arquitectura, la modularidad consiste en dividir en componentes con nombres y ubicaciones determinadas la arquitectura llamados *módulos,* éstos se integran para satisfacer los requisitos del problema.

Myers presenta la modularidad como un atributo individual del software que permite a un programa ser intelectualmente manejable, por lo tanto se considera que un programa compuesto de un módulo no puede ser ampliamente manejable. El número de entidades que lo comprenden ocasiona que el desarrollador lleve a cabo un menor esfuerzo en la fase principalmente de codificación.

Cuando se considera la modularidad en el diseño de un sistema, deberá ser cuidadosamente controlada, ya que se puede llegar a una excesiva modularidad provocando un gasto de esfuerzo innecesario por consiguiente incrementar el costo del

producto. El tamaño y número de módulos necesarios dependerán de su función y aplicación de éstos.

Cuando se modulariza un sistema, se puede diseñar para cada módulo su solución de manera tal que si se requiere modificación, ésta no afectará el resto del diseño.

3.2. ABSTRACCION

Cuando se considera la modularización de un problema, se debe considerar la abstracción. Abstracción significa que debemos concentrar las características esenciales e ignorar detalles que no son relevantes al nivel que actualmente trabajamos.

La complejidad de un problema hace necesaria la aplicación de la abstracción; se pueden distinguir básicamente dos tipos de abstracción: La abstracción de procedimientos y la abstracción de datos.

La primera de ellas tiene como interés obtener una estructura jerárquica. Para su ejemplo tómese en cuenta lo siguiente: considerando que un lenguaje de programación se encuentra compuesto por instrucciones de asignación, bucles e instrucciones de decisión, la transición de un problema para poder ser resuelto mediante estos elementos primitivos, recorre un camino bastante largo en algunos casos. Se inicia descomponiendo el problema en subproblemas, cada uno identificado bajo un orden; estos subproblemas son nuevamente descompuestos a otro nivel y así sucesivamente hasta lograr el objetivo. Entonces se dice que se trabaja en una abstracción de procedimientos.

Sin embargo, también considerando que buscaremos como objetivo solucionar el problema mediante las primitivas antes mencionadas, se debe tomar en cuenta que

mientras nos movemos por los distintos niveles de abstracción, trabajaremos en crear también abstracciones de datos, ésta se puede considerar como una determinada colección de datos que describen un objeto.

3.3. OCULTAMIENTO DE LA INFORMACION

El concepto de ocultamiento de la información parte del trabajo realizado por D. L. Parnas quien dice qué cada módulo deberá tener un secreto que esconder, este principio sugiere que los módulos se han de caracterizar por decisiones de diseño que los oculten unos a otros. Es decir un módulo deberá en lo posible diseñarse para que su información no pueda ser accesible a otros que no necesiten tal información.

Cuando se lleva a cabo la aplicación de este principio en forma conjunta con otros tales como abstracción, tendremos grandes posibilidades de obtener una *modularización efectiva,* esto permitirá por lo tanto evitar de gran manera que cuando se realicen las pruebas o el mantenimiento se afecte con menor grado la arquitectura del software.

3.4. INDEPENDENCIA FUNCIONAL, COHESION Y ACOPLAMIENTO

Cuando se lleva a cabo la modularización, aparece otro concepto derivado de éste: Independencia funcional. Para ejemplificar este concepto se pueden tomar dos módulos e identificar su relación funcional entre ellos, se dice que los dos módulos son totalmente independientes si cada uno de ellos puede funcionar completamente sin la presencia del otro.

Esta definición implica que no existen interconexiones entre los módulos directa o indirectamente, explícita o implícitamente. La independencia funcional nos permitirá lograr un modularidad efectiva, ésta puede medirse a partir de 2 criterios cualitativos: La cohesión y el acoplamiento.

La cohesión puede ser vista como una medida de afinidad mutua de los componentes de un módulo, es decir su fortaleza de asociación de los elementos dentro del módulo.

Para explicar el concepto, podríamos tomar un elemento de procedimiento de un módulo e identificar que éste se encuentra funcionalmente relacionado en distinto grado a cualquier número de otros elementos. Como resultado, diferentes diseñadores pueden ver a través de este ejercicio distintas interpretaciones de la estructura de un problema.

Yourdon identifica los siguientes siete niveles de cohesión:

1.- **COHESION COINCIDENTAL**.- Esta cohesión ocurre cuando se encuentra poca o nada de relación entre los elementos de un módulo. Esta cohesión algunas veces resulta de modularizar un programa de una forma arbitraria segmentando su planteamiento.

2.- **COHESION LOGICA**.- La cohesión lógica modulariza un programa agrupando lógicamente sus actividades. Por ejemplo, un módulo que genere todas las salidas independientemente de su tipo. Se puede identificar fácilmente un módulo con cohesión lógica cuando identificamos que éste requiere un elemento de control de datos, que es pasado para determinar qué tipo de procesamiento ha de realizarse.

3.- **COHESION TEMPORAL**.- Un módulo con cohesión temporal se identifica por el hecho de que contiene tareas que se ejecutan en el mismo momento; es decir sus elementos se encuentran relacionados en base al tiempo, tal como una inicialización o finalización de una rutina.

4.- **COHESION PROCEDIMENTAL**.- Un módulo exhibe cohesión procedimental si éste consiste de un número de componentes que tienen que ser ejecutados en determinado orden. Por ejemplo, un módulo que debe leer primero datos, buscar en una tabla y posteriormente imprimir un resultado.

5.- **COHESION COMUNICATIVA**.- Esta ocurre cuando los elementos de un módulo operan sobre una misma área de una estructura de datos; por ejemplo, un módulo que realice funciones de entrada y salida en un archivo específico.

6.- **COHESION SECUENCIAL**.- Este tipo se presenta si en el módulo se presenta una secuencia de componentes, donde la salida de uno de los componentes sirve como entrada para el siguiente componente, tal como la edición de transacciones o actualización de un archivo maestro.

7.- **COHESION FUNCIONAL**.- Se dice que un módulo es funcionalmente cohesivo, cuando todos los componentes de este contribuyen a una sola función; es decir el módulo realiza solamente una tarea.

Steves y colaboradores proporcionan un conjunto de criterios sencillos que pueden ayudar a establecer el grado de cohesión de un módulo. Ellos sugieren tomando en cuenta una forma descendente del módulo, escribir una frase que describa la función (propósito) del módulo y después examinarla tomando en cuenta lo siguiente:

1.- Si la frase resulta ser una sentencia compuesta, tiene una coma, la palabra 'y' o tiene más de un verbo, entonces el módulo probablemente realiza más de una función. Por lo tanto, es probable que tenga vinculación con la cohesión secuencial o de comunicación.

2.- Si la frase contiene palabras relativas al tiempo, tales como "primero" ,"a continuación" ,"después", "entonces", "cuando", el módulo probablemente tiene vinculación con la cohesión secuencial o temporal.

3.- Si la frase contiene palabras como "inicializar", limpiar", etc., implican que el módulo tiene vinculación con la cohesión temporal.

Los niveles de cohesión que se identifican reflejan la cohesión entre las funciones que un módulo provee; sin embargo los tipos de datos abstractos pueden no ser fácilmente acomodados en este esquema. Buxtin por lo tanto propone se agregue un nivel extra de cohesión llamado **COHESIÓN DE DATOS**, para identificar módulos que encapsulen un tipo de datos abstracto.

Es importante notar que no es una tarea fácil obtener la mayor cohesión posible entre los componentes de un módulo.

El segundo criterio para lograr una modularización efectiva es el acoplamiento; el acoplamiento es una medida de la fortaleza de interconexión entre los módulos y es usada para evaluar distintas organizaciones del programa. Stevens, Myers y Constantine establecen, que el acoplamiento es la medida de fortaleza de asociación establecida por la conexión desde un módulo a otro. Un fuerte acoplamiento complica un sistema ya que un módulo es difícil de comprender, cambiar o corregir si se encuentra altamente

interrelacionado con los otros módulos. La complejidad puede ser reducida mediante el diseño de módulos con el más débil acoplamiento posible entre ellos.([15])

En el diseño buscamos el más bajo nivel de acoplamiento entre los módulos, esto produce como consecuencia que sea más fácil de comprender un software y se reduzcan los caminos a través de los cuales se puedan propagar los cambios y errores.

Desde el de menor grado de acoplamiento hasta el de mayor grado de acoplamiento los tipos de acoplamiento pueden ser identificados de la siguiente forma:

1.- **ACOPLAMIENTO DE DATOS**.- Cuando se presenta este tipo de acoplamiento, los elementos de datos requieren solamente ser pasados como entrada a un módulo así como sus parámetros explícitos. La lista de parámetros contiene solamente tipos de datos simples.

2.- **ACOPLAMIENTO POR ESTAMPADO**.- Se presenta cuando se pasa una porción de una estructura de datos en lugar de datos simples a través de una interfaz de módulo.

3.- **ACOPLAMIENTO DE CONTROL**.- Este acoplamiento implica el pasar banderas de control, ya sea como parámetros o en una forma global entre módulos de tal forma que un módulo controla la forma en que se ejecutara otro.

4.- **ACOPLAMIENTO COMUN**.- En este tipo de acoplamiento, los módulos son atados a un nivel donde los datos requeridos son una parte de una estructura global, el uso de estructuras de datos comunes son ejemplo de este nivel de interfaz.

---

[15] Pressman, Roger S. *Ingeniería del Software: Un enfoque práctico.* Tercera Edición. Ed. Mc Graw Hill. España. 1993.pag. 349.

5.- **ACOPLAMIENTO DEL CONTENIDO**.- Este ocurre cuando un módulo modifica los valores locales o las secuencia de las instrucciones de algún otro módulo, se presenta generalmente en programas en lenguaje ensamblador. Un ejemplo de este tipo de acoplamiento se presenta también si un módulo interfiere a la mitad de otro, o un módulo tiene múltiples puntos de entrada.

3.5. DISEÑO DE DATOS, DISEÑO ARQUITECTONICO Y PROCEDIMENTAL

Si partimos de la consideración de que un sistema de información tiene como finalidad transformar los datos, debemos enfocarnos en primer lugar en el diseño de estos, éste tendrá un gran impacto sobre el diseño arquitectónico y procedimental.

Retomando las áreas de estudio presentadas en el análisis, podemos identificar que durante el análisis se determina como cambian los datos, que comprenden y como se almacenan, sin embargo, el análisis solo identifica pero sus resultados no nos proporcionan directamente la posibilidad de como incorporarlos al sistema. Frecuentemente el diseño de datos se encuentra solapado con el análisis, sin embargo dentro del diseño de datos podemos establecer como identificativas de éste las siguientes directrices que son necesarias y de gran importancia para lograr un buen diseño:

- Identificar estructuras de datos tanto internas como externas.
- Identificar las abstracciones de datos registrándolas mediante una notación.
- Identificar las operaciones que han de realizarse sobre las estructuras con la finalidad de simplificar el diseño mediante la utilización de tipos de datos abstractos.
- Definir las interfaces visibles para cada abstracción de datos.
- Refinar las estructuras de datos hasta lograr la arquitectura definitiva de ellos.
- Evaluar el impacto de la modelización de los datos en el diseño del software.

El diseño arquitectónico por otro lado tiene como finalidad para el desarrollador obtener o generar una vista conceptual del sistema, la cual mediante un respectivo refinamiento se podrán identificar funciones internas, descomponer las funciones de alto nivel en subfunciones e identificar datos con un respectivo almacenamiento, además establecer las interconexiones entre funciones, datos y almacenamiento.

Por las características propias del diseño arquitectónico, algunos autores tal como Richard Fairley establecen que este diseño inicia durante el proceso de análisis hasta la revisión del diseño para posteriormente codificarlo.

Durante el diseño procedimental también llamado interno o detallado se realizan especificaciones a mayor nivel que durante el arquitectónico, es esta fase donde se establecen los algoritmos necesarios para que el sistema realice lo esperado. En esta fase se pueden utilizar técnicas de notación tales como diagramas de flujo de datos, pseudocódigo, diagramas HIPO, etc.

Es importante señalar que este diseño permite exponer defectos en la estructura arquitectónica y las modificaciones que resultan se verán facilitadas por contener menos detalles por manipular, ya que se tiene un enfoque más objetivo del problema encontrado.

3.6. DISEÑO DE LA INTERFAZ

La interfaz de usuario es la parte del programa que permite a éste interactuar con el usuario. La interfaz de usuario puede adoptar muchas formas, que van desde la simple línea de comandos hasta las interfaces gráficas que proporcionan las aplicaciones más modernas.

La interacción del usuario con la máquina es una actividad de diseño muy importante al igual que el diseño de datos, diseño arquitectónico y procedimental que no se puede relegar a segundo plano.

Existen tres estilos de interacción básicos:

1. Mediante interfaz de preguntas y respuestas (línea de comandos).
2. Mediante interfaz de menús.
3. Mediante interfaces orientadas a ventanas con opción de señalar y elegir (gráfica también llamada WIMP).

La primera de ellas es el resultado de los primeras formas de comunicación con la computadora, consistían en proporcionarle a la computadora una orden conjuntamente con una serie de argumentos para especificar con exactitud lo que se deseaba realizar, el sistema debería de responder con una serie de respuestas que complementarían la acción que se deseaba llevar a cabo.

Este tipo de interfaz se ha extinguido casi en su totalidad ya que son muy propensos a cometer errores y con alto grado de dificultad para aprender.

El segundo estilo es una variante del tipo anterior, el cual ofrece al usuario un contexto global y es menos propenso a errores, ya que presenta una lista de opciones de las cuáles solo el usuario puede elegir una que lo llevará a otro nivel de petición y así sucesivamente hasta encontrar la acción requerida por el usuario.

El tercer estilo de interfaz es el resultado del avance tecnológico y de la experiencia de los desarrolladores de software, respecto a los factores humanos y su impacto en el diseño de la interfaz, este estilo ofrece las ventajas de permitir visualizar diferentes capas de información simultáneamente, mediante iconos gráficos menús desplegables, botones y técnicas de presentación que reducen el número de pulsaciones en el teclado, esto ha favorecido la reducción de errores.

Por último es importante considerar los siguientes aspectos al momento de diseñar una interfaz de usuario:

- Utilizar un formato consistente para menús, entrada de órdenes y ayuda.
- Solicitar confirmación cuando se realice alguna acción destructiva.
- Permitir deshacer la última acción en el número de órdenes posibles.
- Minimizar el número de pulsaciones que realice el usuario.
- Reducir la cantidad de información que el usuario necesite para realizar alguna acción.
- Uso de un lenguaje fácil y comprensible.
- Clasificar las acciones por su función.
- Mostrar solo la información relevante del contexto actual.
- Generar errores significativos.
- Uso de colores adecuados, así como tipos de letras, tabulaciones y agrupaciones de texto para facilitar la comprensión en forma consistente.
- Manejo de sonido y gráficos en forma moderada.

PARA ENTRADA DE DATOS

- Permitir al usuario controlar el flujo interactivo.
- Proporcionar ayuda para todas las entradas.
- Eliminar entradas innecesarias.
- Minimizar el número de pulsaciones del usuario.

3.7. METODOS DE DISEÑO

El diseño efectivo del software se logra mejor utilizando una metodología consistente de diseño. Existe una gran diversidad de metodologías de diseño al igual que las metodologías presentadas en la fase de análisis de requisitos y de la misma forma están conformadas por un conjunto de herramientas gráficas y/o técnicas para expresar el resultado del diseño, se utilizan heurísticas de diseño y procedimientos de como debemos abordar el diseño de un sistema. En forma conjunta proporcionan principios sistemáticos para organizar y estructurar el proceso de diseño y el producto.

Algunos de estas metodologías a partir de las expuestas se presentan en la tabla 6.

Técnicas de Análisis y Diseño (SADT).
Diseño Estructurado (DS)
Desarrollo de Sistemas Estructurados en Datos (DSED)
Diseño Orientado a los Objetos(DOO)

Tabla 6 Metodologías disponibles para el diseño.

CAPITULO IV

IMPLEMENTACION DEL SISTEMA

4.1 CODIFICACION

La etapa de implementación tiene que ver con la traducción de las especificaciones obtenidas durante el diseño a código fuente. La programación es una actividad que solo es posible aprenderla mediante la experiencia, es frecuente que cuando se cuenta con poca experiencia se generan programas "espagueti" que son difíciles de mantener.

Para Sommerville, Pressman y Fairley el estilo de programación resulta fundamental para el mantenimiento del producto que se obtenga. En la actualidad se pueden considerar algunos puntos importantes que permiten mejorar la práctica de la programación. A continuación se mencionan algunos:

1. **Distribución del programa o módulo e identificación clara de sus zonas**.- La mayoría de los lenguajes de programación son flexibles en cuanto a la distribución del código en la página. Se deberá por lo tanto identificar sus respectivas zonas señalando: encabezado, zona de declaración de variables, cuerpo del procedimiento, etc.

Se pueden utilizar líneas en blanco para establecer sus límites de cada parte, sin excederse de estas.

Además, la utilización de una adecuada sangría incrementa considerablemente la comprensión del programa, Miara y otros, determinan el valor de la tabulación entre 2 y 4, un número mayor o menor consideran podría resultar perjudicial.

2. **Documentación del código**. La incorporación de comentarios a lo largo del código constituye una herramienta invaluable para la corrección y mantenimiento del producto, de ser posible se sugiere el uso de un prólogo donde se señale los siguientes puntos:

- Nombre y descripción del módulo o programa.
- Descripción de las variables, constantes, parámetros, etc., más significativos.
- Programas o módulos relacionados.
- Fechas de modificación y autor.

Para incluir comentarios a lo largo del código se sugieren tomar en cuenta los siguientes puntos:

- Utilizar comentarios cuando los bloques de código realicen manipulaciones de datos importantes o manejo de excepciones.
- Los comentarios deberán usar terminología adecuada al dominio del problema.
- Se deberán limitar los bloques de comentarios.

- No usar comentarios cuando:

  a) Estos sean largos y confusos a cambio de no reescribir un código que se presente complejo y confuso.

  b) Exista redundancia con los prólogos.

3. **Uso de nombres adecuados**. Los objetos que se incorporan en un programa son entidades del mundo real, por lo tanto las variables, constantes, funciones, etc., que los representen deberá procurarse se encuentren lo mas estrechamente relacionados con los nombres de estos objetos.

Por ejemplo, si se utiliza una variable para almacenar el nombre de un cliente sería conveniente utilizar un nombre como **NombreCliente** en lugar de un nombre como **N** o **NC**, el desarrollador podrá hacer uso de la técnica conocida como mezcla de palabras (WordMixing) para diferenciar las distintas partes de un nombre como en el ejemplo anterior. Algunos autores inclusive recomiendan la ordenación alfabética de las variables durante su declaración y evitar emplear los mismos nombres de identificadores para distintos propósitos aunque el lenguaje lo permita.

Además de lo anterior, podrá utilizarse una notación constante tal como la húngara, originalmente concebida para el leguaje C por el húngaro Charles Simonyi perteneciente al equipo de programación de Microsoft Corporation. Actualmente es ampliamente utilizado por muchos desarrolladores de software.

La notación sugiere los siguientes puntos:

A - Los nombres de las variables, deberán de ir precedidas en minúsculas por un identificador de su tipo.

Por ejemplo:

a Arreglo o Matriz
b Doble Precisión
c Carácter.
f Punto Flotante
n Numérica
l Lógica
o Objeto (genérico)
u Tipo indeterminado (desconocido a priori).

Opcionalmente, se puede preceder al tipo de una constante que se refiera al alcance o ámbito de la variable, como por ejemplo:

l Local
g Pública o global

Ejemplo: **In**ContadorPagina

B - A todos los efectos, los nombres que demos a nuestras funciones y métodos seguirán las mismas reglas de codificación. Al fin de cuentas tanto el nombre de una variable y el nombre de una función son símbolos que contienen un valor (unos lo tienen y otros lo generan).

C - A la hora de dar nombre a una función ó variable se sugiere la siguiente regla:

Tipo Categoría Acción [SubAcción]

Ejemplo: **l**PrtStRdy ()

- Tipo --> **I**, pues devuelve un valor tipo lógico.

- Categoría --> Prt, del inglés Printer (impresora)

  Acción --> St = Status o Estado

  SubAcción --> Rdy (ready) Lista

A simple vista, al examinar el código, se ve que se trata de una función que retorna un valor lógico (verdadero o falso, T. o F.) diciéndonos si la impresora está lista para operar.

D - En la medida de lo posible, no usar guiones bajos, es decir no usar nombres como **NOMBRE_CLIENTE**.

Sólo podrán usarse en sentencias del procesador cuando el lenguaje lo soporte pero nunca en el nombre de una variable o función.

E - Las palabras que definen procedimientos y funciones deben ir siempre con mayúsculas, y preferiblemente al mismo nivel de identación. Lo mismo los comandos, palabras reservadas, acrónimos, nombres de dispositivos, nombres de archivos y constantes. Los nombres de campos, tablas y bases de datos van en minúsculas.

4. **Uso adecuado de construcciones de control y proposiciones**. Bajo este último punto se propone contemplar las siguientes restricciones:

   - Evitar múltiples condiciones profundas que impliquen dificultad para detectar cual condición se está ejecutando.

- Evitar proposiciones nulas o negativas.
- Hacer uso de paréntesis para clarificar las condiciones lógicas o aritméticas, así como espacios en blanco cuando se considere necesario.

Por último es necesario hacer hincapié en la necesidad de utilizar la convención que se elija durante todo el programa.

4.2 PRUEBA DEL SISTEMA

La confirmación de un sistema de software es un proceso continuo en cada etapa del ciclo de vida del software; sin embargo, la prueba de los programas sigue siendo la técnica de confirmación de sistemas más utilizada. ([16])

La prueba de los programas es un proceso destructivo, se diseña para hacer que el comportamiento de un programa sea distinto del que intentaba el desarrollador. Los niveles de prueba que se presentan son los siguientes:

1.- **Prueba de funciones**. Las funciones y procedimientos que componen un módulo se prueban para asegurar que su operación sea la correcta.

2.- **Prueba de módulos**. Las funciones se conforman para construir los módulos. Este módulo se prueba para asegurar que su desempeño cumpla con las especificaciones.

[16] Sommerville, Ian. *Ingeniería de Software.* Ed. Adisson Wesley Iberoamericana. Versión en Español. U.S.A. 1988. Pág. 191.

3.- **Prueba de subsistemas o integración**. En este nivel se integran los módulos para formar subsistemas, esta prueba se centra en las interfaces de los módulos ya que se considera que éstos son correctos.

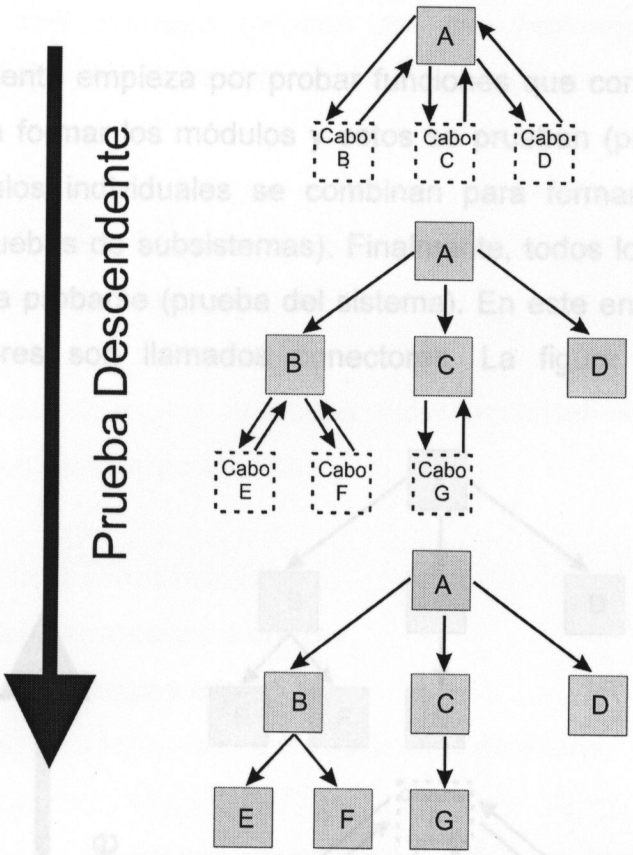
4.- **Prueba del sistema**. Este nivel tiene como finalidad combinar todos los subsistemas para conformar el sistema completo, su finalidad es la de detectar los errores en el diseño y si cumple los requisitos especificados durante el análisis.

5.- **Prueba de aceptación**. Esta etapa se enfoca en trabajar con datos reales en el ambiente de operación, su conformidad con las especificaciones de diseño y en la capacidad de satisfacer los requisitos de los usuarios. Podrá revelar por lo tanto errores en la definición de requisitos y en su capacidad de rendimiento real con respecto a los esperados por los usuarios. Si el sistema supera esta prueba, entonces será posible su operación plena.

Existen distintas estrategias de prueba, las dos más comunes se conocen como prueba ascendente y descendente.

El enfoque de prueba descendente empieza con un esqueleto del sistema; es decir, la estrategia de prueba supone que se han desarrollado los módulos ejecutivos de alto nivel del sistema, pero que los de bajo nivel existen sólo como módulos vacíos; un ejemplo de módulo vacío es uno que no procesa nada, sino que simplemente termina luego de ser llamado.

Para llevar a cabo esta integración el enfoque descendente utiliza cabos, es decir al iniciar desde el primer nivel de módulos en lugar de los módulos subordinados deberán construirse cabos, los cuales simularan de alguna forma la existencia de éstos. La figura 4.1 describe este procedimiento.

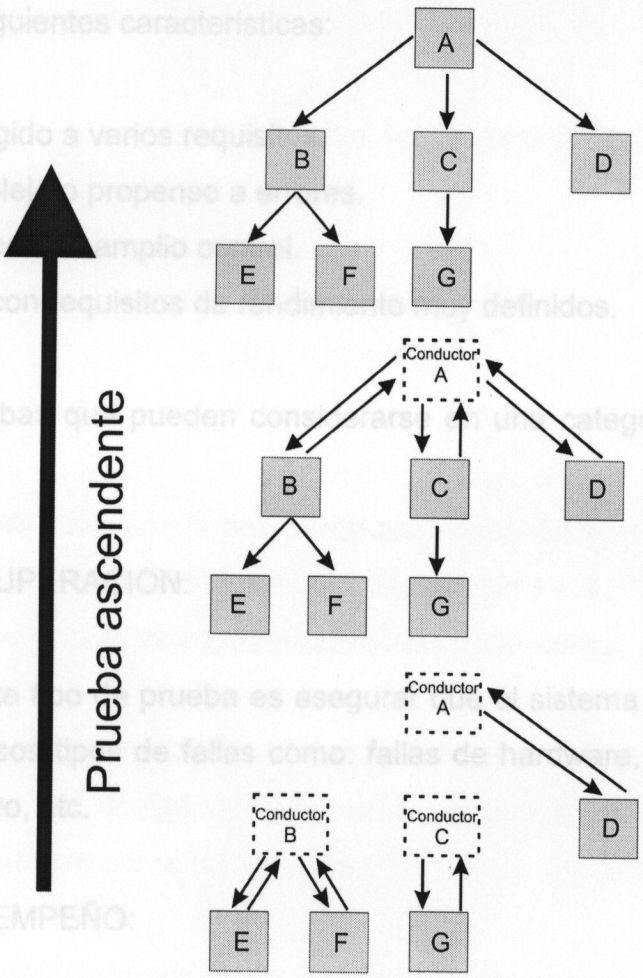


4.1 Prueba descendente.

Las ventajas del enfoque descendente consisten en la alta probabilidad de detectar en las primeras etapas de la prueba los errores de diseño no vistos (no errores del programa) que se pudieran haber cometido, estos errores de diseño suelen cometerse en los niveles más altos del sistema. También se cuenta con la ventaja de disponer de un sistema que funciona, aunque sea de forma limitada.

Las desventajas principales de este enfoque consisten en la necesidad de construir resguardos que simulen de alguna forma los niveles inferiores, dificultándose la construcción de estos, si se encuentran funciones complejas. Otro inconveniente, es que el resultado puede ser difícil de observar, ya que en muchos casos los niveles superiores no generan resultados directamente.

El enfoque ascendente empieza por probar funciones que conforman un módulo, después se integran para formar los módulos y estos se prueban (prueba de módulos), posteriormente los módulos individuales se combinan para formar unidades que se probarán en conjunto (pruebas de subsistemas). Finalmente, todos los componentes del sistema se combinan para probarse (prueba del sistema). En este enfoque la sustitución de los módulos superiores son llamados conectores. La figura 4.2 describe este procedimiento.



4.2 Prueba ascendente.

Las ventajas de este enfoque resultan ser las desventajas de la prueba descendente y las desventajas del descendente son las ventajas del ascendente.

Una alternativa puede ser un enfoque combinado (prueba sándwich) utilizando la prueba descendente para los niveles superiores de la estructura del programa en forma conjunta con la ascendente para los niveles subordinados.

Es importante a la hora de realizar la prueba poder identificar los módulos críticos el cual se define bajo las siguientes características:

1.- Esta dirigido a varios requisitos.
2.- Es complejo o propenso a errores.
3.- Cuenta con un amplio control.
4.- Cuenta con requisitos de rendimiento muy definidos.

Existen otras pruebas que pueden considerarse en una categoría especial, estas son:

PRUEBA DE RECUPERACION:

El propósito de este tipo de prueba es asegurar que el sistema pueda recuperarse adecuadamente de diversos tipos de fallas como: fallas de hardware, fallas de corriente, fallas del sistema operativo, etc.

PRUEBA DE DESEMPEÑO:

El propósito de este tipo de prueba es asegurar que el sistema pueda manejar el volumen de datos y transacciones de entrada especificados en el módulo de implantación del usuario, además de asegurar que tenga el tiempo de respuesta requerido.

PRUEBA DE LOS FACTORES HUMANOS:

Este tipo de prueba tiene como finalidad identificar o detectar las formas en la que reaccionará el usuario ante al sistema en acciones no previstas, por ejemplo; cuando un proceso se tarde periodos largos, deberá considerarse la necesidad de enviar un mensaje para mantener informado al usuario acerca de la función que realiza el sistema. También permite observar como se introducen los datos por los usuarios a medida de determinar si el sistema cumple o no con las necesidades.

Al llevar a cabo esta etapa es recomendable documentar las actividades llevadas a cabo, así como los resultados obtenidos.

4.3 DEPURACION

La depuración se relaciona con la prueba de programas en que se basa en salidas de prueba para mostrar la presencia de errores. La depuración es el proceso de identificar las áreas del programa que causan errores y modificarlas para corregir el error.

Se pueden identificar dos etapas en el proceso de depuración: Primero, localizar aquellas partes incorrectas del código del programa y segundo, el programa se debe modificar de manera que cumpla con sus requisitos.([17])

Se considera la primera etapa como aquella en la cual se encontrarán más dificultades, a diferencia de la segunda que por lo general resulta más sencilla. De este

[17] Sommerville, Ian. *Ingeniería de Software*. Ed. Adisson Wesley Iberoamericana, Versión en Español. U.S.A. 1988. Pág. 218.

proceso de depuración se generan uno de los dos siguientes resultados: primero, se encuentra la causa, se corrige y se elimina; o no se encuentra la causa.

En el primer caso, cuando se encuentra la causa del error y se emprende la corrección, es importante realizarse las siguientes preguntas que sugiere Van Vleck (1989):

1. ¿Se repite la causa de error en otra parte del programa?
2. ¿Cuál es el "siguiente error" que se podría presentar a raíz de la corrección que se va a realizar?
3. ¿Qué podríamos haber hecho para prevenir este error la primera vez?

4.4 DOCUMENTACION

La documentación es quizá la parte más tediosa en el desarrollo de sistemas. Sin embargo, puede considerarse que un sistema sin documentación es un sistema incompleto. La documentación la podemos clasificar en dos partes:

- Manual del usuario.
- Documentación del sistema.

El manual del usuario reúne la información, normas y documentación necesaria para que el usuario conozca y utilice adecuadamente la aplicación desarrollada. Por lo tanto deberá tenerse sumo cuidado en su elaboración. Sommerville sugiere los siguientes puntos a contemplar para el manual de usuario:

1. Una descripción funcional sobre lo que puede hacer el sistema;
2. un documento que explique cómo instalar el sistema y adecuarlo para configuraciones particulares;
3. un manual introductorio que explique, en términos sencillos, como iniciarse en el sistema;
4. un manual de referencia que describa con detalle las ventajas del sistema disponibles para el usuario y cómo se pueden usar;
5. una guía del operador (si se requiere un operador para el sistema), que explique cómo debe reaccionar éste ante situaciones surgidas mientras el sistema se encuentra en uso.

Un ejemplo del contenido de un manual puede ser el siguiente:

1. Portada-título
2. Indice
3. Presentación
   3.1. Introducción.
   3.2. Cómo usar el manual.
   3.3. Requisitos de Instalación.
4. Conceptos generales en la aplicación
5. Descripción de las unidades.
6. Referencia rápida.
7. Procedimientos en caso de error.
8. Apéndices.

La documentación del sistema o manual técnico, se refiere a todos los documentos que pertenecen a la aplicación del sistema, se conforma de los siguientes puntos:

a) Documentación generada al finalizar la fase de análisis.
b) Documentación generada al finalizar la fase de diseño.
c) Documentación generada al finalizar las pruebas del sistema.

Estos documentos se consideran esenciales si se quiere comprender y dar un mantenimiento al programa. La documentación del sistema inicia con una narración general que describa al sistema y una definición de requisitos de preferencia con una fundamentación asociada, posteriormente y basándose en la documentación generada por la fase de análisis (véase Tabla 1.1 de este trabajo) deberán incluirse los diagramas, tablas de decisión, árboles y narraciones utilizados organizándose adecuadamente.

Deberá incluirse también una descripción por cada programa del sistema, por cada unidad identificando sus interfaces y controles, así como la estructura de archivos y datos globales. Por último se anexarán los procedimientos de prueba efectuados y los resultados obtenidos de una forma breve.

CONCLUSIONES

En relación con lo expuesto en este trabajo y los objetivos trazados para el mismo considero que, la elección de un modelo para el desarrollo de sistemas dependerá en gran parte de la experiencia del desarrollador, la magnitud del proyecto y las herramientas con las que se cuente. Los modelos que se identificaron fueron: Modelo de ciclo de vida clásico, Modelo de Prototipos, Modelo en espiral.

Además, es de relevante importancia que al elegir el desarrollador de sistemas un modelo, debe de considerar que este cumpla con todos o la mayoría de los siguientes puntos:

a) El modelo deberá soportar la comunicación entre las distintas partes involucradas, el desarrollador y el usuario lo mas estrechamente posible y continuar a través del proceso de desarrollo.

b) El modelo deberá ser flexible de manera tal que se permita detectar errores y corregirlos con una mayor rapidez.

c) Considerando la continua evolución y aparición de nuevas herramientas y técnicas en el desarrollo de sistemas deberá ser capaz de integrar al máximo estos elementos.

Las actividades que se encontraron para el desarrollo de un sistema, se pueden englobar en las siguientes fases generales: Definición, Desarrollo y Mantenimiento; por tanto, considero necesario que el modelo que elija el desarrollador deba contemplar estas fases.

Es indudable que la fase de análisis constituye un proceso de descubrimiento, refinación, modelización y especificación de las características que deberán integrar el sistema, motivo por el cuál se requiere la cabal comprensión del problema y la participación del usuario, dentro de esta fase se identificaron las siguientes herramientas:

Para la recopilación de información:

a) Entrevista
b) Cuestionario
c) Observación
d) Revisión de registros y documentos.

Para la descripción

a) Arboles y Tablas de decisión
b) Diagrama de flujo de datos
c) Diagramas de Entidad-Relación

El diseño puede verse desde dos perspectivas. Desde el punto de vista técnico, que engloba las áreas afectadas, los datos, la arquitectura, los procedimientos y la interfaz del usuario; y por etapas que implica desde el diseño preliminar y detallado.

Se analizaron los conceptos fundamentales de diseño como modularidad, abstracción, ocultamiento de la información, independencia funcional y los criterios cualitativos que este último involucra como son cohesión y acoplamiento.

Considero que la modularidad deberá aplicarse independientemente de la metodología elegida, al igual que la abstracción y ocultamiento de la información, por otra parte se buscará en lo posible el mayor grado de cohesión y menor grado de acoplamiento entre módulos. Concluyo que éstos representan calidad en el software.

En lo que se refiere a la implementación, al proporcionar guías importantes tanto en codificación como en las características que conforman la documentación de un sistema se logra el objetivo planteado. Por otra parte se describen las estrategias de prueba y se proporciona una definición del concepto de depuración.

BIBLIOGRAFIA

1. Burch, John G. y Gary, Grudnitski. *Diseño de Sistemas de Información*. Ed. Megabayte. México. 1994.

2. De Marco, T. *Structured Analysis and Systems Specification.* Ed. Prentice Hall. U.S.A. 1979.

3. Fairley, Richard E. *Ingeniería de Software*. Ed. Mc Graw Hill. México. 1994.

4. Gane, C. and T. Sarson. *Structured Systems Analysis*. Ed. Prentice Hall. Englewood Cliffs. N.J. U.S.A. 1979.

5. Graham, Ian. *Metódos Orientado a Objetos.* Segunda Edición. Ed. Adisson-Wesley/Díaz de Santos. U.S.A. 1996.

6. Keller, Marilyn y Shumate, Ken . *Software Specification and Design.* Ed. John Wiley & Sons, Inc. U.S.A. 1992.

7. Kendall and Kendall. *Análisis y Diseño de Sistemas.* Ed. Prentice Hall. Tercera Edición. México. 1991.

8. L. Winblad, Ann y otros. *Software Orientado a Objetos*. Ed. Adisson-Wesley/Díaz de Santos. U.S.A. 1993

9. Martin, James y Maclure, Carma . *Diagramming Techniques for Analysis and Programers.* Ed. Prentice Hall. U.S.A. 1985

10. Pressman, Roger S. *Ingeniería del Software: Un enfoque practico*. Tercera Edición. Ed. Mc Graw Hill. España. 1993.

11. Rodríguez Cuadrado, Alfredo; Márquez Serrano, Antonio. *Técnicas de Organización y Análisis de Sistemas*. Ed. Mc Graw Hill. España. 1993.

12. Senn, James A. *Análisis y Diseño de Sistemas de Información*. Segunda Edición. Ed. Mc Graw Hill. México. 1992.

13. Shodi, Jag . *Software Engineering Methods, Managent and Case Tools*. Ed. Mc Graw Hill. U.S.A. 1991.

14. Sommerville, Ian. *Ingeniería de Software*. Ed. Adisson Wesley Iberoamericana. Versión en Español.U.S.A. 1988.

15. Tutorial de "*Análisis y Diseño de Sistemas*". Instituto Tecnológico La Paz Baja California. www.itelp.edu.mx/publica/tutoriales/analisis /index.htm.

16. Van Vliet, Hans. *Software Engineering Principles and Practice*. Ed. Jhon Wiley & Sons, Inc. U.S.A. 1994.

17. Yourdon, Edward. *Modern Structured Analysis*. Ed. Yourdon Press. Englewood Cliffs. N.J. U.S.A. 1989.