UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

AUTOR: Gabriel Guevara Gutiérrez

INFORMÁTICA IV (Análisis y Diseño de Sistemas Estructurados)		Clave: 1465
Plan: 2005		Créditos: 8
Licenciatura: Informática		Semestre: 4º
Área: Informática		Hrs. Asesoría: 4
Requisitos: Informática II. (Estructura de datos y dinámicas en memoria principal)		Hrs. Por semana: 4
Tipo de asignatura:	Obligatoria (X)	Optativa ( )

Objetivo general de la asignatura

Al finalizar el curso, el alumno aprenderá a desarrollar sistemas utilizando el análisis y diseño según el enfoque estructurado.

Temario oficial (64 horas)

1. Introducción (10 horas)
2. Análisis de sistemas (26 horas)
3. Diseño de sistemas (28 horas)

Introducción

Este Apunte tiene como propósito el que identifiques los conceptos más relevantes que le dan sustento al análisis y diseño de sistemas como etapas del ciclo de vida de sistemas. La estructura está conformada por tres temas:

1. Introducción. Se explican los conceptos que conforman la Teoría General de Sistemas que hoy en día siguen estando vigentes y que tienen gran relevancia en el desarrollo de sistemas. Se tratan los modelos del ciclo de vida de sistemas que conforman las bases de lo que son los procesos de desarrollo de software.

2. Análisis Estructurado. Se explican los conceptos que intervienen para la administración de requerimientos así como los del análisis de sistemas.

3. Diseño Estructurado. El contenido de la asignatura es fundamental para tu formación como Licenciado(a) en Informática. La asignatura de análisis y diseño estructurado de sistemas (Informática IV) conforma las bases para asignaturas que tomas en el mismo semestre y futuros semestres. Por tal razón, necesitas poner especial atención a los conceptos y procedimientos que aquí se explican.

TEMA 1. INTRODUCCIÓN

Objetivo particular

Al finalizar el tema identificarás:

- Los conceptos involucrados en el análisis y diseño de sistemas en el campo de la Informática.
- Los modelos y procesos de desarrollo que existen para crear un sistema.
- Los roles requeridos para realizar un análisis y diseño de sistemas.
- Los enfoques que existen para el desarrollo de sistemas.

Temario Detallado

1.1. Definición de sistema

1.2. La información

- 1.2.1. Definición de Información
- 1.2.2. Propiedades o atributos de la Información
- 1.2.3. Fuentes de la Información

1.3. Teoría General de Sistemas

1.4. Sistemas de Información

- 1.4.1. Definición de Sistema de Información
- 1.4.2. Componentes de un Sistema de Información
- 1.4.3. Clasificación de los Sistemas de Información
- 1.4.4. Sistemas de Información y Sistemas Informáticos

1.5. Ciclo de Vida de Sistemas de Información

- 1.5.1. Análisis
- 1.5.2. Diseño
- 1.5.3. Implementación
- 1.5.4. Pruebas
- 1.5.5. Implantación
- 1.5.6. Mantenimiento

1.6. Entorno de Desarrollo de los Sistemas de Información
   1.6.1. Estructura Organizacional del Área de Sistemas
   1.6.2. Departamentos Involucrados
   1.6.3. Funciones y Responsabilidades de cada Departamento
1.7. Modelos, Metodologías y Procesos de Desarrollo
   1.7.1. Proyectos y Sistemas
   1.7.2. ¿Qué son los Modelos? (Concepto de Abstracción)
   1.7.3. Modelos de Ciclo de Vida de Sistemas más Importantes
   1.7.4. Procesos de Desarrollo más Importantes
1.8. Enfoques del Desarrollo de Sistemas
   1.8.1. Enfoque Estructurado
   1.8.2. Enfoque Orientado a Objetos

Introducción

Los sistemas y la información son los elementos básicos que conforman los sistemas de información, por tal razón, su definición es de suma importancia para que se pueda entender en qué consiste un sistema de información.

Existen varios tipos de sistemas de información cuya clasificación está en función de la forma en que procesan la información y el nivel jerárquico de la organización que hace uso de ella.

Para desarrollar un sistema de información se hace uso de las etapas que conforman el ciclo de vida de sistemas (análisis, diseño, implementación, pruebas, implantación, mantenimiento) pero existen diversas formas de aplicar estas etapas que son denominados modelos.

Es así, que en este tema revisarás esta información para que tengas los elementos conceptuales que se necesitan para comprender el área de desarrollo de sistemas.

1.1. Definición de sistema

Sin lugar a dudas, para el siglo XXI el concepto de sistema y su comprensión tiene una gran importancia para cualquier ciencia o disciplina. Al entender el concepto de sistema podemos analizar las situaciones con una perspectiva integradora donde cada elemento o componente que interviene modifica el comportamiento del todo, es decir, del sistema.

Un sistema en su concepción más sencilla lo podemos definir como una caja negra que recibe una entrada, la cual es procesada para generar una salida. Bajo esta concepción los elementos más básicos de un sistema son:

- Entrada.
- Proceso.
- Salida.

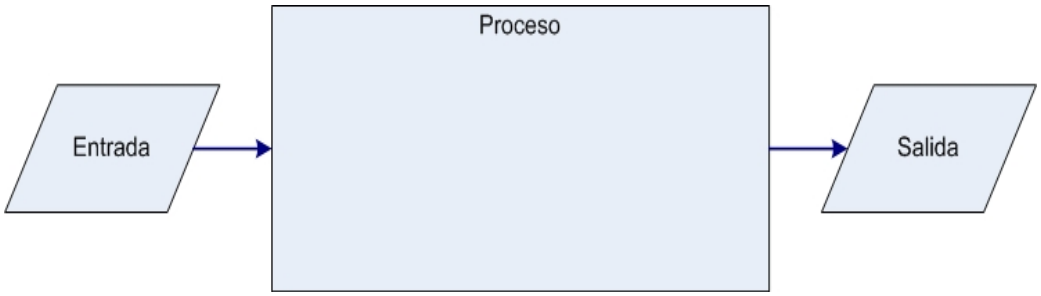


Figura 1.1. Vista de caja negra de un sistema

Otra concepción de sistema es definirlo como un conjunto de elementos[1] interrelacionados e interdependientes entre sí que buscan un objetivo común. El hecho de que los elementos busquen un objetivo común no excluye la capacidad de que cada elemento tenga un objetivo en particular.

[1] Otras palabras que se pueden ocupar en lugar de elementos son: ‘parte’, ‘componente’, ’módulo’.

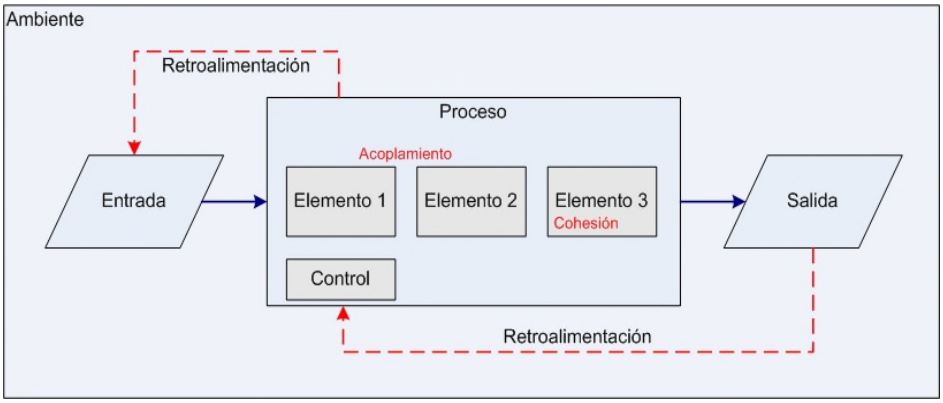


Figura 1.2. Vista de caja blanca de un sistema

Esta definición obliga a visualizar al sistema como una caja blanca o transparente donde se puede analizar el comportamiento de cada elemento donde se podrán observar dos características: **acoplamiento** y **cohesión**.

El **acoplamiento** es nivel de interdependencia que existe entre los elementos. Pressman lo define como "…una medida de interconexión entre módulos dentro de una estructura de software. El acoplamiento depende de la complejidad de interconexión entre los módulos, el punto donde se realiza una entrada o referencia a un módulo, y los datos que pasan a través de una interfaz."[2]

La **cohesión** es la especificidad de la función que realiza un elemento. Pressman lo define como

> […] una extensión natural del concepto de ocultación de información […] Un módulo cohesivo lleva a cabo una sola tarea dentro de un procedimiento de software, lo cual requiere poca interacción con los procedimientos que se llevan a cabo en otras partes de un programa. Dicho de otra manera sencilla, un módulo cohesivo deberá (idealmente) hacer una sola cosa.[3]

Los conceptos de cohesión y acoplamiento forman parte de otro concepto llamado **modularidad**, el cual tiene como objetivo descomponer el sistema en "módulos" a fin de reutilizarlos en ese sistema u otro.

---

[2] Roger S. Pressman, *Ingeniería del Software. Un Enfoque Práctico*, 5ª ed., Madrid, McGraw-Hill, 2002, p 231.
[3] Loc. cit.

Aparte del **acoplamiento** y de la **cohesión**, se presentan otros dos conceptos: **ambiente** y la **retroalimentación**.

El **ambiente** es el entorno que rodea al sistema y que lo afecta directamente con algún tipo de estímulo. Teóricamente cada sistema cuenta con una **frontera** o **límite** que lo distingue del ambiente en el que se desenvuelve. Por ejemplo, la frontera del cuerpo humano es la piel.

La **retroalimentación** es una actividad de autorregulación, la cual supervisa que las salidas que genera el sistema cumplan con ciertas características con base en las entradas y su proceso de transformación.

1.2. La información

1.2.1. Definición de Información

La información es un conjunto de datos cuya relación entre sí da un significado a un ente (persona u organización) para tomar una decisión.

El valor de la información es totalmente subjetivo ya que depende de la persona u organización y de acuerdo con las características que tiene la información.

1.2.2. Propiedades o atributos de la información

La información debe de cumplir con ciertas propiedades, también llamadas características, para que se le pueda considerar información. Las características son:[4]

Propiedad	Descripción
Exacta	La información debe ser precisa y libre de errores.
Completa	La información debe contener todos aquellos hechos que pudieran ser importantes.
Económica	El costo que se debe incurrir para obtener la información debe ser menor que el beneficio proporcionado a la organización.
Confiable	La información debe garantizar tanto la calidad de los datos utilizados, como la de las fuentes de información.
Relevante	La información debe ser útil para la toma de decisiones. Evitar todos aquellos hechos que sean superfluos o que no aporte algún valor.
Detallada	La información debe presentar el nivel de detalle indicado para la decisión a la que se destina. Se debe proporcionar con la presentación y el formato adecuados, para que resulte sencilla y fácil de manejar.
Oportuna	La información debe ser entregada a la persona y en el momento que la necesita para poder tomar la decisión.
Verificable	La información debe poder ser contrastada y comprobada.

Cuadro 1.1. Características de la información

[4] Cf., Álvaro Gómez Vieites y Carlos Suárez Rey, *Sistemas de Información: Herramientas prácticas para la gestión empresarial*, 2ª ed., México, Alfaomega, 2007, pp. 5 y 6.

1.2.3. Fuentes de la información

La información puede tener diferentes orígenes y formatos a la vez, en el caso de desarrollo de sistemas de información, que es el que nos compete, la información tiene sus fuentes en:

Fuente	Descripción
Personas	Las personas son la fuente principal para el analista de sistemas; de ellas obtiene la información necesaria para la construcción del sistema.
Documentos	Los documentos se refieren a la información contenida en un medio físico o electrónico dentro o fuera de la organización y que describen en forma parcial o completa como funciona. Ejemplos: Manual de Procedimientos, Manual de Organización, Políticas, Normas, Informes, Reportes, Expedientes, etc.
Reglamentos	Los reglamentos son un tipo especial de documentos, la diferencia consiste en que existe una alta probabilidad de que no se puedan cambiar y establecen lineamientos de carácter obligatorio. Ejemplos: Constitución Política de los Estados Unidos Mexicanos, Ley Federal del Trabajo, Ley de Impuesto Sobre la Renta, etc.
Sistemas	Los sistemas, en este caso refiriéndonos a los sistemas informáticos, son actualmente, para las organizaciones, sus principales fuentes de información (Ver punto 1.4.1.).

Cuadro 1.2. Fuentes de la información

1.3. Teoría General de Sistemas

La Teoría General de Sistemas, por sus siglas TGS, fue desarrollada en 1925 por Ludwing von Bertalanffy. Una de las ideas más sobresalientes de esta teoría indica que no es suficiente analizar un elemento en forma aislada, sino que es indispensable analizar al elemento junto con los demás con los que se

encuentra relacionado. Con esto, se establece que el comportamiento de un elemento del sistema afectará, en menor o mayor grado, a aquellos elementos con los que se encuentra relacionado.

La TGS es un conjunto de modelos, principios y leyes válidos para cualquier tipo de sistema no importando la naturaleza de sus elementos y las relaciones entre ellos. Como la TGS es general, su espectro es amplio, así que no es de extrañarse que veamos su estudio en la Biología, la Administración, la Informática, etc.

La importancia que tiene la TGS en el Análisis y Diseño de Sistemas está dada porque en cualquier tipo de proyecto que tenga como objetivo la creación de un sistema de información se hace indispensable la integración de conocimientos, así como la especialización a fin de reducir tiempos y costos, pero sobre todo, se hace indispensable la comprensión del funcionamiento del todo, entiéndase el sistema. Si ocupamos un enfoque reduccionista al analizar solo una parte del sistema tendríamos una visión limitada del problema y como consecuencia no identificaríamos las causas que lo originan.

Dentro de los conceptos que más destacan en esta teoría tenemos:

Sinergia. El conjunto de elementos tiene más valor que la simple suma de las partes.
Entropía. Es el desgaste natural que sufre el sistema por los cambios que se presentan en el medio ambiente. No existe más que en modelos teóricos, ambientes estáticos. Todo ambiente provoca cambios en los sistemas que rodea, esto trae como consecuencia que el sistema tenga que responder a esos cambios de alguna manera.
Isomorfismo. Define un conjunto de similitudes estructurales que pueden compartir algún conjunto de sistemas.

Cibernética. Teoría de los sistemas de control basada en la comunicación entre el sistema y el medio ambiente.

Equifinalidad. Se refiere al hecho que un sistema vivo a partir de distintas condiciones iniciales y por distintos caminos llega a un mismo estado final. El fin se refiere a la mantención de un estado de equilibrio fluyente.

1.4. Sistemas de Información

1.4.1. Definición de Sistema de Información

Para definir lo que es un sistema de información podemos ocupar la concepción que se dio de sistema en el punto 1.1., pero su objetivo es más preciso, el cual es generar información para la toma de decisiones.

1.4.2. Componentes de un Sistema de Información

En este tipo de sistema se presentan tres componentes más:

Proceso. Es el conjunto de actividades por realizar para generar un producto (bien y/o servicio), para el caso de los sistemas de información consiste en algún tipo de información para la toma de decisiones. También es conocido como procedimiento.

Persona. Es el individuo que tiene la responsabilidad para crear, modificar, destruir y/o verificar un insumo que entra al sistema o que es procesado por el sistema.

Base de Datos. Es un conjunto de datos almacenados en cierto orden para que posteriormente sean recuperados.

1.4.3. Clasificación de los Sistemas de Información

Básicamente en cualquier organización se distinguen tres niveles para la toma de decisiones (de nivel inferior a nivel superior): operativo, táctico y estratégico, de manera similar podemos clasificar la información. Dependiendo del nivel de que se trate, las personas necesitarán de información operativa, táctica o estratégica.

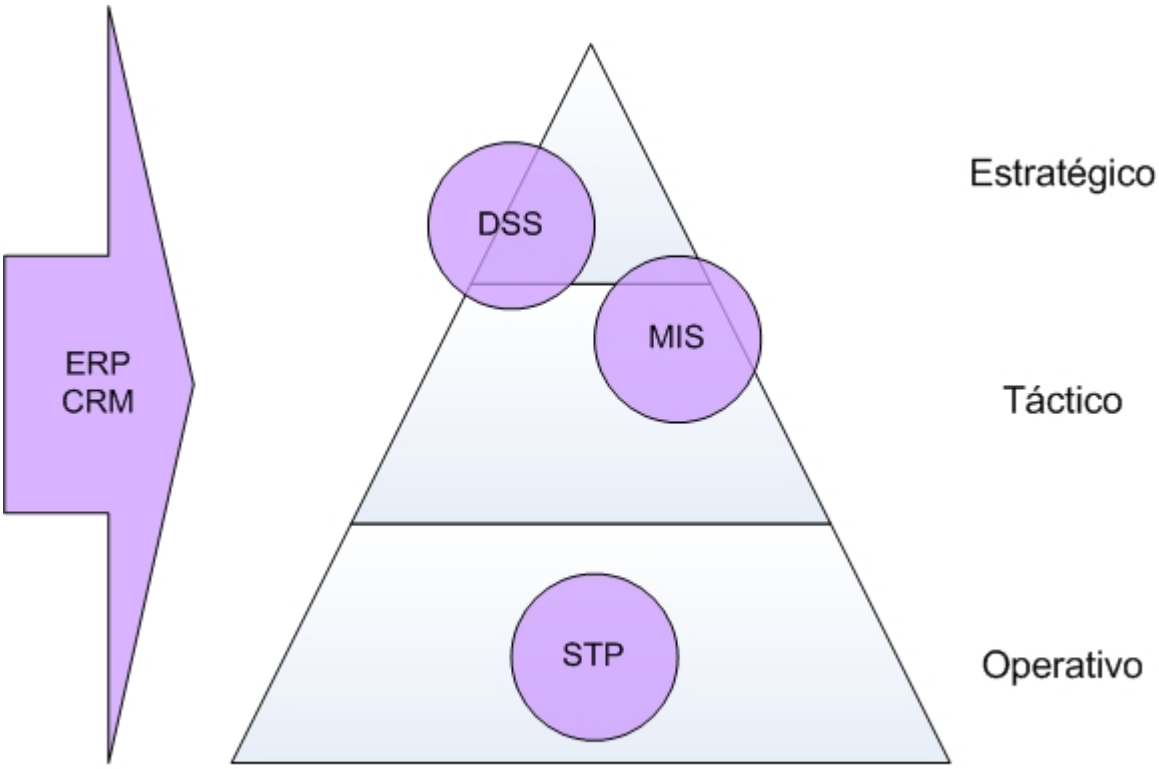


Figura 1.3. **Pirámide de nivel de tipo de decisiones y sistemas de información**

Operativo. Información muy detallada.
Táctico. Información resumida.
Estratégico. Información sintetizada.

Los diferentes tipos de sistemas de información se ubican en alguno de los niveles o en varios de ellos:

TPS (*Transaction Proccessing System* – Sistema de Procesamiento de Transacciones).- Este sistema tiene como objetivo recoger datos básicos en las operaciones organizacionales enfocándose principalmente en las áreas de contabilidad, ventas, compras, nómina.

MIS (*Management Information System* – Sistema de Información Administrativa).- Este sistema tiene como objetivo generar informes que permitan a los directivos a mejorar el control de la ejecución de las actividades de las áreas funcionales de la organización.

DSS (*Decision Support System* – Sistema de Apoyo a las Decisiones).- Este sistema tiene como objetivo apoyar y asistir al nivel estratégico al proceso de toma de decisiones creando escenario y/o generando alternativas.

ERP (*Enterprise Resource Planning* – Planeación de Recursos Empresariales).- Este sistema tiene como objetivo agilizar la administración de los recursos materiales y humanos de la organización por medio de la integración de la información de las diferentes áreas y con un enfoque orientado a procesos.

CRM (*Customer Relationship Managment* – Administración de Relaciones con el Cliente).- Este sistema tiene como objetivo proporcionar información de los clientes lo más completamente posible sobre qué compra, cuándo compra, qué busca, sus comentarios sobre la organización y los productos que adquiere.

1.4.4. Sistemas de Información y Sistemas Informáticos

El propósito de un sistema de información y uno informático es el mismo, la diferencia radica en que el sistema informático hace uso de ciertos elementos característicos para lograr el objetivo:

Hardware. Es el equipo de cómputo y telecomunicaciones utilizado para procesar los datos.

Software. Son los programas de computadora, donde encontramos sistemas operativos, lenguajes de programación, sistemas manejadores de base datos, procesadores de textos, generador de presentaciones, etc.

Con lo anterior podemos apreciar que los sistemas informáticos son un subtipo de los sistemas de información, por lo que no se incurre en error al hablar de la utilización de algún tipo de hardware y/o software en el contexto de sistemas de información.

Figura 1.4. Conjuntos: Sistema de Información y Sistema Informático

1.5. Ciclo de Vida de Sistemas de Información

La concepción de sistemas viene de las ciencias naturales al tratar de describir un ser vivo como un conjunto de partes, el cual tiene un ciclo de vida (*nace, crece, se reproduce y muere*). Los sistemas de información tienen su propio ciclo: *análisis, diseño, implementación, pruebas, implantación y mantenimiento*. Dependiendo del autor que se consulte, las etapas o fases pueden ser más o menos, también pueden integrarse en una o cambiar de nombre, aún así, en esencia tienen el mismo propósito.

Es necesario recalcar que el ciclo de vida de sistemas es un modelo teórico que tiene como propósito agrupar las actividades en fases que se tienen que

realizar para construir un sistema. Por otro lado el modelo del Ciclo de Vida de Sistemas no indica la forma o secuencia en que se tienen que aplicar las fases.

1.5.1. Análisis

Esta fase contesta a la pregunta ¿*qué*?, en otras palabras es una descripción de las causas de las necesidades y/o problemas. Además, se observa y modela cómo funciona la organización y/o sistema actual. Sus actividades principales son:

- Diagnosticar la necesidad y/o problema por resolver.
- Identificar a los participantes así como sus necesidades.
- Establecer los objetivos que persigue el sistema.
- Establecer los alcances y exclusiones.
- Crear un modelo de dominio o negocio con su respectiva especificación describiendo la situación actual.

Otro nombre que recibe está fase es: *Requerimientos*.

1.5.2. Diseño

Esta fase contesta a la pregunta ¿*cómo*?, no hay que confundir el cómo de la fase de análisis, donde se investiga cómo se hacen las cosas; éste se refiere a cómo voy a dar solución a las necesidades y/o problemas. Sus actividades principales son:

- Diseñar procesos y procedimientos.
- Diseñar algoritmos.
- Diseñar la base de datos.
- Diseñar la distribución de los componentes.

Regularmente los puntos anteriores reciben el nombre de arquitectura de software o simplemente arquitectura. La arquitectura describe con diferentes modelos cómo se va a construir la solución. “Una arquitectura de software

define la estructura general de un sistema y varía de acuerdo con el tipo de sistema a desarrollarse".[5]

1.5.3. Implementación

Esta fase se encarga de construir los componentes de software que le darán funcionalidad al sistema. Sus actividades principales son:

- Codificar algoritmos a programas.
- Construir la base de datos en el DBMS (*Database Managment System*).

Otros nombres que recibe esta fase son: *Desarrollo, Programación, Construcción*.

1.5.4. Pruebas

Esta fase se encarga de asegurar que la calidad del software sea la correcta. Por un lado se verifica que el software se construya de manera correcta, es decir, que se apegue a los estándares establecidos, y por otro lado se valida que el software que se construya sea el correcto, es decir, que satisfaga las necesidades por las cuales se concibió el sistema.

Otros nombres que recibe está fase son: *Verificación y/o Validación, Evaluación*.

[5] Alfredo Weitzenfeld, *Ingeniería de Software: Orientada a Objetos con UML, JAVA e Internet*, México, Thomson, 2004, p 36.

1.5.5. Implantación

Esta fase se encarga de poner en marcha el software construido y capacitar a las personas involucradas para su uso.

1.5.6. Mantenimiento

Esta fase se encarga de darle mantenimiento al sistema en caso de que surja una falla no detectada, se requiera extender, adecuar, o mejorar el sistema. Estas formas de mantenimiento reciben los nombres de:

- Mantenimiento Correctivo.
- Mantenimiento Preventivo.
- Mantenimiento Adaptativo.

1.6. Entorno de desarrollo de los Sistemas de Información

El ambiente en el que se desenvuelve el equipo de desarrollo que va a construir el sistema es altamente cambiante ya sea que se trate del ambiente interno o del ambiente externo de la organización. Una de las constantes en las organizaciones que se dedican al desarrollo de sistemas es que están orientadas a los procesos y por consecuencia están orientadas a sus clientes.

Las empresas no se deben preocupar por crear en primer lugar una estructura organizacional, en lugar de eso se deben preocupar por diseñar y establecer un proceso que permita construir las características de su producto (bien o servicio) para que dichas características satisfagan las necesidades de sus clientes. Una vez definido el proceso, ahora sí, se preocupa por crear la estructura organizacional que permita ejecutar el proceso.

Tomando en cuenta lo anterior, se hace indispensable que la persona encargada del área de sistemas esté al tanto de los procesos críticos de la organización para la que está trabajando. Si lo hace, tendrá definidas

claramente las metas que debe perseguir el área de sistemas, que al fin de cuentas es buscar una agilización y/o optimización de esos procesos.

1.6.1. Estructura organizacional del Área de Sistemas

Por lo regular se tiende a crear un departamento por cada una de las fases del ciclo de vida de sistemas. Sin embargo, la definición de la estructura organizacional depende de factores como:

- Tamaño de la empresa.
- Estructura organizacional general de la empresa.
- Personal con el que se cuenta para el área de sistemas.
- Infraestructura en tecnologías de información y comunicación con las que se dispone.
- Características del proyecto con el que se esté trabajando.

Con base en lo anterior, lo importante es identificar las actividades críticas que nos ayudan a construir el sistema. Así, identificamos la cantidad y el perfil de las personas que necesitamos para ejecutar las actividades. Y con eso, se establece la estructura organizacional más adecuada para no perder el control del proyecto y de las personas.

1.6.2. Departamentos Involucrados

El área o departamento de sistemas se ha convertido en uno de motores críticos de las organizaciones, por ello, no es de extrañarse que el área tenga relación con cada uno de las demás áreas existentes. Tampoco es extraño, que el responsable de esta área sea convocado en las decisiones estratégicas que se toman, al fin de cuentas la empresa es un sistema social.

1.6.3. Funciones y responsabilidades de cada departamento

Analista de Sistemas. Es una persona experta en la identificación y comprensión de los problemas y oportunidades. Esto incluye la habilidad para asociar las necesidades con el problema principal por resolver o la oportunidad por conseguir.

- Habilidad para asociar las necesidades con el problema principal a resolver o la oportunidad a conseguir.
- Es un facilitador y con gran capacidad de comunicación.
- Tiene conocimiento sobre el negocio y la tecnología.

Diseñador. Es una persona con madurez, visión y una amplia experiencia para conjugar sus conocimientos y juicio en crear soluciones:

- Experiencia en el dominio del problema, comprender los requerimientos y el dominio de la ingeniería de software.
- Líder para dar orden y dirección a los esfuerzos de diferentes equipos, y tomar decisiones críticas bajo presión y hacer estas decisiones regla.
- Comunicación para escuchar, persuadir, motivar y guiar.

1.7. Modelos, metodologías y procesos de desarrollo

Al revisar diversas fuentes se presenta una gran confusión entre los conceptos de: modelo, metodología, método, ciclo de vida y proceso de desarrollo, ocupándose indistintamente uno del otro. En la siguiente figura se muestra las relaciones que existen entre los conceptos mencionados y evitar confusiones.

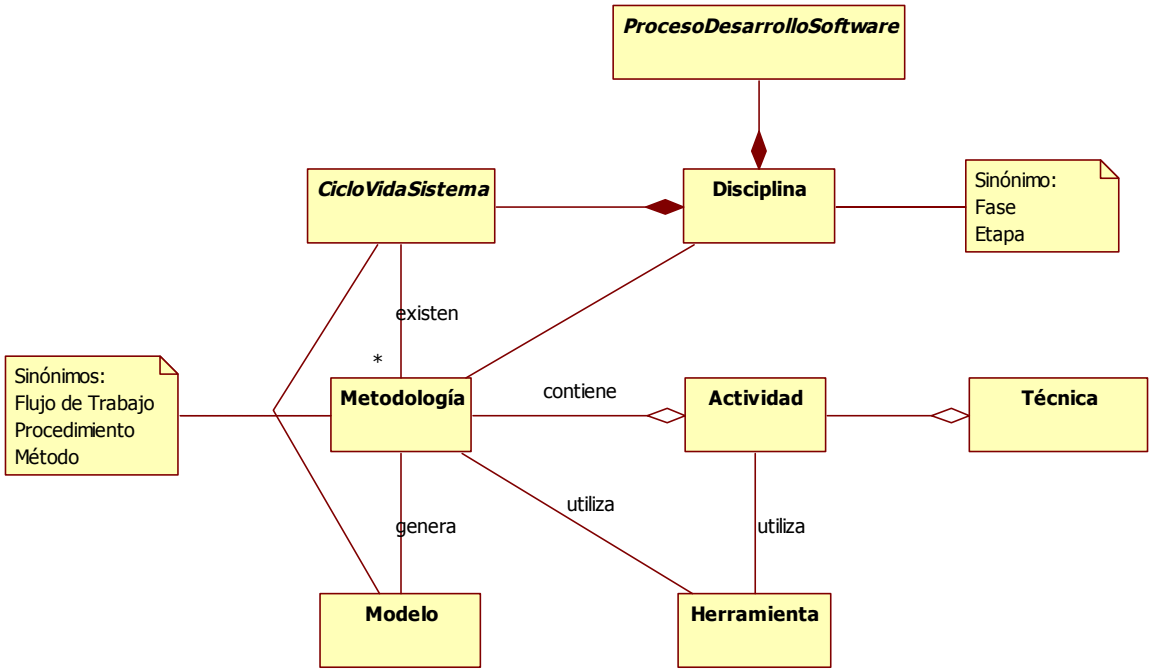


Figura 1.5. Asociación de conceptos: Ciclo de Vida, Proceso de Desarrollo, Metodología, Modelo

Conceptos Asociaciones	Descripción
Ciclo de vida de sistema	Es conjunto de fases que se tienen que llevar a cabo para construir un sistema de información.
Proceso de desarrollo de software	En un proceso que indica **qué** se tiene que hacer, **cómo** se tiene que hacer, **cuándo** se tiene que hacer y **quién** lo tiene hacer para construir un software con calidad.
Disciplina	Se le conoce como **disciplina** en un proceso de desarrollo de software, y como **fase** en el ciclo de vida de sistemas. Como tal es un área de conocimientos y actividades por realizar para construir una parte del sistema de información.
Metodología	No hay un consenso respecto al concepto de **metodología**. Maddison (1983, pág. 45), define **metodología** como: "Conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas de información". Una **metodología** especifica: ▪ Cómo se debe dividir un proyecto en etapas. ▪ Qué actividades se llevan a cabo en cada etapa. ▪ Qué salidas se producen y cuándo se deben producir. ▪ Qué restricciones se deben aplicar. ▪ Qué herramientas se van a utilizar. ▪ Cómo administra el proyecto.
Actividad	Es una acción por realizar para generar, actualizar, revisar algún producto de trabajo como documentos, modelos, software.
Técnica	Dice cómo realizar una actividad, soportándose en herramientas específicas para cada caso.
Herramienta	Indican con qué contamos para resolver una actividad específica. Hoy día estas herramientas reciben el nombre de CASE (*Computer Assisted Software Engenering*).
Ciclo de Vida de Sistema ⇔ Disciplina	Un **ciclo de vida de sistemas** está compuesto de una serie de **disciplinas** o fases. Las más comunes son: Análisis, Diseño, Implementación, Pruebas, Implantación, Mantenimiento.
Proceso de Desarrollo de Software ⇔	Un proceso de desarrollo de software está compuesto de una serie de disciplinas o fases. Además de las fases disciplinas clásicas del ciclo de

Disciplina	vida de sistemas están: Administración de Proyectos, Administración de la Configuración y Cambios, Ambiente.
Ciclo de Vida de Sistema ⇔ Metodología	Un ciclo de vida de sistemas tiene una o más metodologías. Ver concepto de metodología.
Disciplina ⇔ Metodología	Una **disciplina** puede ejecutarse bajo diferentes **metodologías**. Las metodologías van a depender del autor y pueden tener dos enfoques: estructurado u orientado a objetos.
Ciclo de Vida de Sistema ⇔ Modelo	Un **ciclo de vida de sistemas** tiene uno o más **modelos**. Es decir, existen diversas formas de aplicar o secuenciar cada una de las disciplinas o fases. Ver punto 1.7.3.
Metodología ⇔ Modelo	Una **metodología** genera gran cantidad de **modelos** que representan el aspecto estático y dinámico del sistema.
Metodología ⇔ Actividad	Una **metodología** está compuesta de varias **actividades**.
Actividad ⇔ Técnica	Una **actividad** está compuesta de varias **técnicas**.

Cuadro 1.3. Asociación de conceptos: Ciclo de Vida, Proceso de Desarrollo, Metodología, Modelo

1.7.1. Proyectos y Sistemas

Es muy común confundir el concepto de proyecto con el de sistema, sin embargo, existe una diferencia. El PMBoK (*Project Managment Body of Knowledge*) del PMI (*Project Managment Institute*) define un *proyecto* como un esfuerzo temporal emprendido para crear un producto, servicio o resultado único. Esto significa que se puede emprender un proyecto para crear un nuevo coche, construir una casa, o en este caso un sistema.

Si recordamos la definición de sistema, encontramos que es un conjunto de elementos interrelacionados entre sí para conseguir un objetivo común. Este sistema se va a construir a través de la administración de un proyecto.

Aquí es necesario destacar que la confusión entre proyecto y sistema surge porque algunas de las actividades que se llevan la administración de proyectos y el ciclo de vida de sistemas son iguales o similares, por ejemplo, la obtención de requerimientos es una constante en ambos ciclos.

1.7.2. ¿Qué son los Modelos? (concepto de abstracción)

La abstracción es una actividad mental que consiste en resaltar, disminuir y/o ignorar la importancia de los hechos y/o objetos de la realidad para analizar una situación dada. Cuando construimos un modelo resalta el concepto de abstracción, dado que los modelos están orientados a representar la realidad bajo determinada perspectiva.

Los modelos, como resultado de un proceso mental, buscan comunicar una idea-concepto a otra persona. Esto se presenta porque cada persona cuenta con un marco referencial (ideas, estudios, experiencias, creencias) distinto respecto a las otras personas. Eso hace necesario tener una herramienta, en este caso los modelos, para poder **comunicar** nuestras ideas.

Dentro del análisis y diseño de sistemas se construyen una diversidad de modelos, principalmente bajo dos perspectivas: **estática** y **dinámica**. La perspectiva estática se enfoca principalmente en los datos ⇨ información. Y la perspectiva dinámica se enfoca principalmente en las funciones ⇨ procesos.

1.7.3. Modelos[6] de Ciclo de Vida de Sistemas más importantes

La importancia de los modelos que a continuación se mencionan se sustenta bajo el hecho de ser las más utilizadas en la industria del software.

[6] También conocidos como metodologías.

Cascada *(Ciclo de Vida clásico, secuencial, lineal)*.

Descripción	Aplicación	Ventajas	Desventajas
Las fases se aplican de manera secuencial donde no se puede avanzar a la siguiente fase hasta haber terminado la anterior.	Cuando la complejidad de los sistemas es baja.	▪ Adecuado para sistemas pequeños y de complejidad baja.	▪ Si se comete un error en alguna fase repercute en la fase subsecuente, pero no se descubre sino hasta que se tiene el sistema. ▪ El sistema solo se ve funcionando hasta el final por lo que el usuario se desespera por no ver resultados. ▪ Las personas responsables de las fases subsecuentes no pueden empezar hasta que terminen las personas de la fase antecedente.

Cuadro 1.4. Modelo de cascada

Prototipos.

Descripción	Aplicación	Ventajas	Desventajas
Se crean prototipos (con o sin funcionamiento) con el objeto de que el usuario aclare los requerimientos específicos.	Cuando el usuario tiene claro el objetivo o necesidad general, pero no tiene claro los requerimientos específicos.	▪ Existe un acercamiento con el usuario en etapas tempranas del proyecto. ▪ Se puede aplicar y/o adaptar a otro ciclo de vida de sistemas.	▪ El usuario puede creer que es el sistema final. ▪ No todos los prototipos funcionan. Pueden ser una interfaz gráfica sin comportamiento alguno. ▪ Por lo regular los prototipos no consideran la calidad del sistema.

Cuadro 1.5. Modelo de prototipos

Incremental.

Descripción	Aplicación	Ventajas	Desventajas
Su nombre deriva de que su objetivo es crear el sistema a través de una serie de incrementos. Cada incremento tiene su propio ciclo de vida de sistemas. Cuando se termina un incremento se integra al sistema. Los incrementos se hacen en paralelo.	Cuando no se cuenta con el personal suficiente para desarrollar todo el sistema.	▪ El usuario puede ver "resultados" desde el primer incremento. ▪ Permite realizar una planeación para maneja los riesgos técnicos.	▪ Para lograr la aplicación de este modelo es necesario tener todos los requerimientos para poder planear cada uno de los incrementos.

Cuadro 1.6. Modelo incremental

Evolutivo *(Desarrollo Rápido de Aplicaciones)*.

Descripción	Aplicación	Ventajas	Desventajas
Es una extensión del modelo incremental, pero los incrementos se hacen de manera secuencial no en paralelo. Los incrementos que se van desarrollando son aquellos que están claros. Conforme se va avanzando los requerimientos confusos y/o ambiguos se van clarificando.	Cuando la complejidad del sistema es alta.	▪ Se entrega funcionalidad del sistema desde el principio.	▪ No es apropiado cuando los riesgos técnicos son muy grandes. ▪ Se requiere de varias personas para desarrollar los incrementos si es un sistema grande. ▪ Si el sistema no se modularías apropiadamente la delegación de trabajo se vuelve complicada.

Cuadro 1.7. Modelo evolutivo

Espiral.

Descripción	Aplicación	Ventajas	Desventajas
Creado por Bohem en los años 80, integra el modelo de prototipos y el modelo en cascada. Dirige su aplicación mitigando los riesgos técnicos con mayor probabilidad e impacto. Se aplica una serie de ciclos en cascada en espiral, de ahí viene su nombre. Al finalizar cada ciclo o un poco antes, se realiza la planeación del siguiente ciclo.	Cuando la complejidad del sistema es alta.	▪ Se ajusta el plan de trabajo de cada ciclo, por lo que siempre está actualizado. ▪ Se afrontan los riesgos más peligrosos desde un principio por lo que si uno no se puede resolver se minimizan las pérdidas económicas.	▪ Se requiere de personas que puedan detectar los riesgos y generar los planes de contingencia y/o contención, lo cual implica una gran experiencia y conocimiento por lo que se ve reflejado en los costos.

Cuadro 1.8. Modelo en espiral

1.7.4. Procesos de Desarrollo más Importantes

Hay que tomar en cuenta que el ciclo de vida de sistemas difiere de lo que es un *proceso de desarrollo de software* o también conocido como *modelo de proceso de software*, al respecto Weintzenfeld dice:

> Un **modelo de proceso de software** define cómo solucionar la problemática del desarrollo de sistemas de software. Para desarrollar software se requiere resolver ciertas fases de su proceso, las cuales se conocen en su conjunto como el **ciclo de vida** del desarrollo de software. Un modelo de proceso debe considerar una variedad de aspectos, como el *conjunto de personas, estructuras organizacionales, reglas, políticas, actividades, componentes de software, metodologías y herramientas utilizadas*.[7]

La ingeniería de software es una tecnología multicapa. Cualquier enfoque de ingeniería (incluida ingeniería del software) debe apoyarse sobre un compromiso de organización de calidad.

El fundamento de la ingeniería del software es la capa de **proceso**. El proceso de la ingeniería de software es la unión que mantiene juntas las capas de

[7] Alfredo Weitzenfeld, op. cit., p 36.

tecnología y que permite un desarrollo racional y oportuno de la ingeniería de software. El proceso define un marco de trabajo para un conjunto de áreas clave de proceso que se deben de establecer para la entrega efectiva de la tecnología de la ingeniería de software. Las áreas clave del proceso forman la base de control de gestión de proyectos del software y establecen el contexto en el que se aplican los métodos técnicos, se obtienen productos del trabajo (modelos, documentos, datos, informes, formularios, etc.), se establecen hitos, se asegura la calidad y el cambio se gestiona adecuadamente.

Los **métodos** de la ingeniería del software indican *cómo* construir técnicamente el software. Los métodos abarcan una gran gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento. Los métodos de la ingeniería de software dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.

De los procesos de desarrollo de software más utilizados en la industria del software tenemos: *Rational Unified Process* y *Extreme Programming*, aunque hay otros procesos, analizaremos sólo los mencionados.

RUP (Rational *Unified Process - Proceso Unificado de Rational*).- RUP es un proceso de ingeniería de software, como tal proporciona una estrategia para asignar tareas y responsabilidades en el equipo de desarrollo. Su meta es asegurar la producción de software con calidad de acuerdo con las necesidades de los usuarios finales en calendario y presupuesto planeado.

RUP es un proceso y a la vez un producto que es desarrollado y mantenido por *Rational Software*, y está integrado con una suite de herramientas de desarrollo.

RUP, también, es el marco de trabajo de un proceso que puede ser adaptado y extendido de acuerdo con las necesidades de la organización que lo adopta. RUP captura 6 buenas prácticas del desarrollo de software moderno

Buena Práctica	Descripción
Desarrollo iterativo de software.	Uno de los problemas que tiene el modelo de cascada al aplicarse las etapas en forma secuencial sin retroalimentación es que se van acumulando los errores y sólo saldrán a flote dichos problemas reflejándose en la inconformidad y disgusto de los clientes y/o usuarios. Para evitar este problema, RUP utiliza un modelo iterativo incremental que se basa en el modelo en espiral que propone Bohem, este modelo utiliza la estrategia de mitigar los riesgos principales desde las iteraciones iniciales del proyecto. Las ventajas que presenta el enfoque iterativo incremental son: evitar mal entendidos; retroalimentación constante con el usuario; identificación pronta de entre los requerimientos, diseño e implementación; se ejecutan las pruebas desde la etapas iniciales; se obtienen lecciones aprendidas que se aplican en la mejora del proceso; durante la vida del proyecto se puede encontrar evidencia del estado del proyecto.
Administración de requerimientos.	El desarrollo de un sistema se desenvuelve en un ambiente altamente dinámico, como tal las necesidades y requerimientos se tienen que adecuar a estos cambios. Por tal razón se necesita tener un proceso que permita identificar estos cambios cuando se presenten y hacer los ajustes necesarios en la planeación. Las ventajas que tenemos al contar con una administración de requerimientos son: la comunicación se basa en la definición de requerimientos; los requerimientos se priorizan, se filtran y se rastrean; las inconsistencia son detectadas más fácilmente; con alguna herramienta de apoyo, es posible proveer un repositorio de los requerimientos, atributos, y fuente de los requerimientos, con enlaces a documentos externos.

Buena Práctica	Descripción
Arquitectura basada en el uso de componentes.	Para visualizar, especificar, construir y documentar un sistema se necesita de diferentes perspectivas. La arquitectura de un sistema se utiliza para manejar los diferentes puntos de vista de los *stakeholders*. La arquitectura define: la organización; los elementos estructurales y sus interfaces entre ellos; el comportamiento de los elementos. El desarrollo basado en componentes es una estrategia importante para la arquitectura del software porque permite reutilizar y configurar miles de componentes de diferentes fuentes comerciales. Las ventajas que tenemos al contar con el uso de componentes basados en la arquitectura son: facilita la creación de arquitecturas resistentes; permite modularización clara; los componente proveen una base natural para la administración del a configuración; herramientas de modelado proveen la automatización para el desarrollo de componentes.
Modelado visual del software.	Un modelo es una simplificación de la realidad que describe un sistema desde una perspectiva particular. Se construyen modelo para entender mejor el sistema y porque no podemos comprender al sistema en su totalidad. La actividad de modelado es importante porque ayuda al equipo de desarrollo a visualizar, especificar, construir y documentar la estructura y comportamiento de la arquitectura de un sistema. Las herramientas de modelado visual facilitan la administración de los modelos, permitiendo ocultar o exponer detalles como sea necesario. El modelado visual también ayuda a mantener la consistencia en la gran cantidad de documentos que se generan en: requerimientos, diseño, e implementación. En breve, el modelado visual ayuda al equipo en administrar la complejidad del software.
Verificación continua de la calidad del software.	Verificar la funcionalidad del sistema involucra crear pruebas para cada escenario principal, cada uno representa algún aspecto del comportamiento esperado del sistema. Si se ocupa un desarrollo iterativo, se debe probar cada iteración, un proceso continuo, asegura la calidad. Con esta forma de trabajo tenemos las siguientes ventajas: se detectan las inconsistencias en los requerimientos, diseño e implementación; las pruebas y verificación están enfocadas en las áreas de alto riesgo; los defectos que son identificados prontamente, reducen el costo de reparación.

Buena Práctica	Descripción
Control de cambios del software.	Un desafió clave en el desarrollo de sistemas es que se afronta con múltiples desarrolladores organizados en equipos, posiblemente en diferentes sitios, trabajando juntos en múltiples iteraciones, versiones, productos y plataformas. En la ausencia de una disciplina de control, el proceso de desarrollo degenera rápidamente en el caos. Coordinar las actividades y los artefactos de los desarrolladores y equipos implica establecer un flujo de trabajo repetible para administrar los cambios al software y otros productos de trabajo de desarrollo. Esta coordinación permite una mejor ubicación de los recursos basado en las prioridades y riesgos del proyecto, y se administra activamente el trabajo de estos cambios a través de las iteraciones. Los cambios de peticiones facilitan una comunicación clara; espacios de trabajo aislados reducen la interferencia entre los miembros del equipo que trabajan en paralelo.

Tabla 1.9. **Buenas prácticas de RUP**

El proceso de RUP está definido en dos ejes:

- El eje horizontal representa el tiempo y muestra el ciclo de vida del proceso de desarrollo en fases divididas en iteraciones. Esta dimensión representa el aspecto dinámico del proceso y está definido en términos de ciclos, fases, iteraciones e hitos.
- El eje vertical representa los procesos de las disciplinas principales, las cuales agrupan lógicamente las actividades por su naturaleza. La segunda dimensión representa el aspecto estático del proceso y está definido en términos de componentes del proceso, actividades, disciplinas, productos de trabajo, y roles.

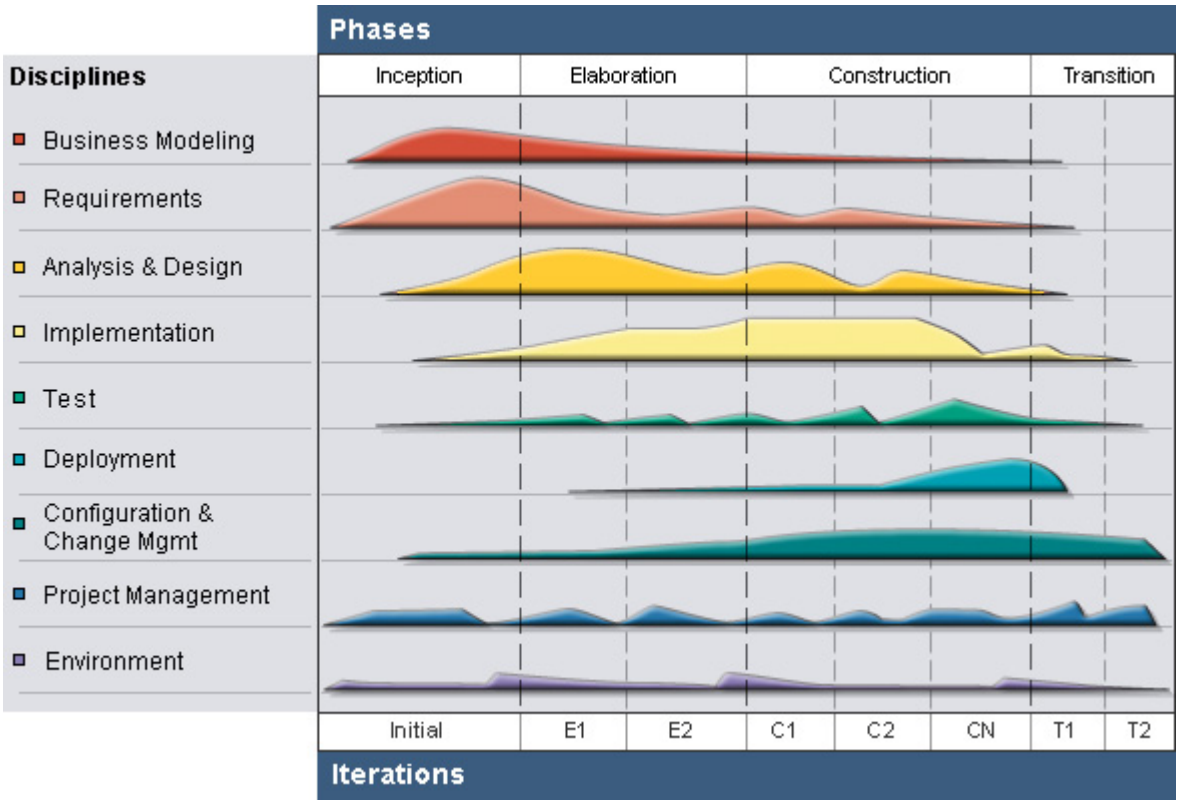


Figura 1.5. *Rational Unified Process***. Fuente: www.therationaledge.com**

Fases

Concepción. La meta de esta fase es alcanzar un común acuerdo entre todos los *stakeholders* en los objetivos del proyecto. Los objetivos principales de esta fase son:

- Establecer el alcance del proyecto de software y las condiciones del ambiente, incluyendo la concepción operacional, criterios de aceptación, y descripciones de qué es y qué no es parte del producto.
- Discriminar los casos de uso críticos del sistema, que son, los escenarios primarios del comportamiento que debe manejar la funcionalidad del sistema.
- Exhibir y demostrar, al menos una arquitectura candidata de algunos de los escenarios primarios.

- Estimar el costo total y el calendario de todo el proyecto y proveer un estimado detallado para la fase de elaboración.
- Identificar los riesgos.

Elaboración. El propósito de la fase de elaboración es analizar el domino del problema, estableciendo los cimientos de la arquitectura, desarrollar el plan del proyecto, y eliminar los elementos de alto riesgo del proyecto. Para complementar estos objetivos, se debe tener una vista del sistema “en amplitud y profundidad”. Las decisiones arquitectónicas se deben hacer con un entendimiento amplio del sistema: su alcance, funcionalidad, y requerimientos no funcionales. Los objetivos de la fase son:

- Definir, validar y controlar la arquitectura tan rápido y prácticamente sea posible.
- Demostrar que la arquitectura soporta la visión con costo y tiempo razonable.

Construcción. Durante la fase de construcción, todos los componentes y características de la aplicación restantes son desarrollados e integrados en el producto, y todas las características son probadas. En cierto sentido, esta fase establece un proceso de manufactura que se enfoca en la administración de los recursos y controlar las operaciones para optimizar costos, calendarios, y calidad. Los objetivos son:

- Minimizar los costos de desarrollo para optimizar recursos y evitar el retrabado.
- Alcanzar una calidad adecuada.
- Alcanzar versiones útiles.

Disciplinas

Administración de Proyectos. La administración de proyectos de software es el arte de balancear los objetivos, los riesgos y las restricciones para entregar

un producto que satisfaga las necesidades del cliente y los usuarios finales. Sus propósitos son:

- Proveer de un marco de trabajo para administrar proyectos de software.
- Proveer líneas guía para planear, organizar, ejecutar y controlar proyectos.
- Proveer un marco de trabajo para administrar el riesgo.

Modelado Negocio. Sus propósitos son:

- Entender la estructura y la dinámica de la organización en que el sistema se desarrollará.
- Entender el problema correcto en la organización destino e identificar mejoras potenciales.
- Asegurar que los clientes, usuarios finales, y desarrolladores tengan un común entendimiento en la organización destino.
- Identificar los requerimientos del sistema para apoyar la organización destino.

Requerimientos. Describe cómo definir la visión del sistema y traducir la visión a un modelo de casos de uso que, con el apoyo de especificaciones suplementarias, defina los requerimientos detallados de software. Además, describe cómo los atributos de los requerimientos ayudan a administrar el alcance y cambiar los requerimientos en el sistema. Las metas son:

- Establecer y mantener el acuerdo con los clientes y los demás *stakeholders* de lo que debe hacer el sistema y por qué.
- Provee a los desarrolladores del sistema un mejor entendimiento de los requerimientos del sistema.
- Define los límites del sistema.
- Provee una base para la planeación de cada iteración.
- Provee una base para estimar el costo y tiempo de desarrollo del sistema.

Análisis y Diseño. Su propósito es traducir los requerimientos a especificación que describan cómo implementar el sistema. Para hacer esta traducción, se debe entender los requerimientos y transformarlos a un diseño de sistema para seleccionar la mejor estrategia de implementación.

Implementación. Sus propósitos son:

- Definir la organización del código en términos de implementación de subsistemas organizados en capas.
- Implementar clases y objetos en términos de componentes (archivos fuente, binarios, ejecutables, y otros).
- Probar los componentes desarrollados como unidades.
- Integrar en un sistema ejecutable los resultados producidos por implementadores individuales o por equipos.

Pruebas. Esta disciplina es un proveedor de servicios a otras disciplinas. Las pruebas se enfocan principalmente en evaluar y asegurar la calidad del producto, usando ciertas prácticas principales:

- Encontrar y documentar la fallas en el producto de software: defectos y problemas.
- Aconsejar a la administración desde una perspectiva de calidad del software.
- Validar que el producto de software trabaje como se diseño.
- Validar que los requerimientos están implementados apropiadamente.

Administración de Cambios y la Configuración. Su propósito es rastrear y mantener la integridad de la evolución de los activos del proyecto. Durante el desarrollo muchos productos de trabajo son creados, los cuales son activos importantes que deben ser salvaguardados y estar disponibles para volver a usarlos. Estos productos de trabajo evolucionan, especialmente en un desarrollo iterativo, son actualizados una y otra vez. Al menos que una persona

sea usualmente responsable de un artefacto, no podemos confiar en la memoria individual para guardar los activos de cada proyecto.

Ambiente. Su propósito es apoyar el desarrollo de la organización con procesos y herramientas. Esto incluye lo siguiente:

- Seleccionar y adquirir herramientas.
- Configurar las herramientas en la organización.
- Configurar el proceso.
- Mejorar el proceso.
- Proporcionar apoyo técnico a los procesos de: infraestructura de tecnologías de información, administración de contabilidad, recuperación.

Despliegue. Su propósito es llevar el producto de software al usuario, lo que involucra las siguientes actividades:

- Probar el software en ambiente final de operación.
- Empaquetar el software para su entrega.
- Distribuir el software.
- Instalar el software.
- Capacitar a los usuarios finales y a los vendedores.
- Migrar software existente o convertir bases de datos.

XP (*Extreme Programming* - Programación Extrema).
XP es un método que tiene como objetivo principal disminuir el costo del cambio en los requerimientos, a través de valores, principios, prácticas, etc. XP intenta mantener relativamente fijo el costo del cambio en cada una de las etapas del proyecto.

XP cuenta con 4 áreas y cada una de ellas define un conjunto de prácticas:

Planeación.

- Redacción de historias de usuario.
- Creación de calendarios de planeación.
- Creación frecuente de versiones pequeñas
- División del proyecto en iteraciones.
- La iteración de planeación inicia cada iteración.
- Movimiento alrededor de la gente.

Diseño.

- Simplicidad.
- Usar tarjetas CRC para las sesiones de diseño.
- Crear soluciones para reducir el riesgo.
- Re-factorizar siempre que sea posible

Codificación.

- El cliente siempre está disponible.
- El código debe ser escrito de acuerdo con los estándares.
- Codificar la primera prueba de unidad.
- Todo código es programado por pares.
- Únicamente un par integra el código a la vez.
- Usar código colaborativo.
- Dejar la optimización hasta el último.

Pruebas.

- Todo código debe tener pruebas unitarias.
- Todo código debe pasar todas las unidades antes de que pueda ponerse en marcha.
- Cuando se encuentra un *bug*, deben crearse las pruebas.
- Las pruebas de aceptación se ejecutan y las estadísticas se publican.

1.8. Enfoques del desarrollo de sistemas

1.8.1. Enfoque estructurado

En el enfoque estructurado, el concepto principal es el de **función**. Bajo esta perspectiva el enfoque estructurado analiza y diseña los sistemas con un enfoque de descomposición funcional.

1.8.2. Enfoque Orientado a Objetos

En el enfoque orientado a objetos, el concepto principal es el de **objeto**. Bajo está perspectiva el enfoque orientado objetos analiza y diseña los sistemas con un enfoque de responsabilidades a través de objetos que se encuentran presentes en la situación real, ya sea de carácter físico o abstracto.

Bibliografía de la tema 1

Bertalanffy, Ludwig von, *Teoría General de Sistemas: Fundamentos, desarrollo, aplicaciones*, Traducción: Juan Almela, México, Fondo de Cultura Económica, 1968.

Gómez Vieites, Álvaro y Carlos Suárez Rey, *Sistemas de Información: Herramientas prácticas para la gestión empresarial*, 2ª ed., México, Alfaomega, 2007.

Pressman, S. Roger, *Ingeniería del Software. Un Enfoque Práctico*, 5ª ed., Adaptación de Darle Ince, Traducción de Rafael Ojeda Martín (et al.), Madrid, McGraw-Hill, 2002.

----------, *Ingeniería del Software. Un Enfoque Práctico*, 6ª ed., Traducción: Jesús Elmer Murrieta Murrieta y otros, México, McGraw Hill 2005.

Weitzenfeld, Alfredo, *Ingeniería de Software: Orientada a Objetos con UML, JAVA e Internet*, México, Thomson, 2004.

Actividades de aprendizaje

A.1.1. Investiga en Internet por medio del buscador www.google.com tres definiciones de sistemas. Identifica en cada definición las palabras que consideres más importantes. Con las palabras seleccionadas crea tu definición de sistema.

A.1.2. Selecciona un modelo de ciclo de vida de sistemas con base en el que sea el más adecuado para sistemas grandes. Investiga en las bibliotecas en trabajos de tesis o similares si se ha ocupado el modelo seleccionado. En las tesis ubicadas busca las razones por las cuales seleccionaron el modelo (mínimo 3). Con las razones encontradas enriquece la tabla de este trabajo correspondiente al modelo seleccionado. Por último, crea un diagrama que esquematice la secuencia en que se aplique las etapas del ciclo de vida de sistemas de acuerdo con el modelo seleccionado.

A.1.3. En el análisis de sistemas se busca modelar los aspectos dinámicos y estáticos del sistema. Los diagramas que se utilizan en el enfoque estructurado y en el enfoque orientado a objetos son diferentes. Investiga en Internet por medio del buscador www.google.com qué diagramas se ocupan en ambos enfoques para representar el aspecto dinámico y estático de un sistema. Crea una tabla en donde pongas como columnas los aspectos y en las filas los enfoques. En las intersecciones entre ellas pon el nombre de los diagramas.

Cuestionario de autoevaluación

1. Da la definición de sistema.
2. Menciona cuáles son los atributos de la información.
3. Explica la diferencia entre acoplamiento y cohesión.
4. Menciona las etapas del ciclo de vida de sistemas.
5. Cuáles son las ventajas del modelo de ciclo de vida de sistemas en cascada.
6. Explica en qué consiste el modelo de ciclo de vida de sistemas en espiral.
7. ¿Cuáles son los dos ejes en los que trabaja *Rational Unified Procces*?

8. Menciona tres metas que persiga la disciplina de requerimientos de *Rational Unified Procces*.
9. Menciona cuáles son las fases de *Rational Unified Process*.
10. Menciona la diferencia que existe entre el análisis y diseño estructurado con el orientado a objetos.

Examen de autoevaluación

1. Esta cualidad de la información indica que se debe contemplar aquellos hechos que pudieran ser importantes:

A. Relevante
B. Económica
C. Detallada
D. Completa

2. Creado por Bohem en los años 80, integra el modelo de prototipos y el modelo en cascada:

A. Modelo tradicional
B. Modelo espiral
C. Modelo incremental
D. Modelo evolutivo

3. Es un conjunto de personas, software, hardware, y procesos que tienen como propósito generar información para la toma de decisiones:

A. Sistema de información
B. Sistema informático
C. Base de datos
D. Sistema

4. En esta etapa del ciclo de vida de sistemas se busca definir la solución del problema que se identificó previamente:

A. Implementación

B. Implantación

C. Diseño

D. Análisis

5. Es una fuente de información formal que permite conocer la manera en que trabaja la organización:

A. Personas

B. Sistemas

C. Documentos

D. Reglamentos

6. Es el desgaste natural que sufre el sistema por los cambios que se presentan en el medio ambiente.

A. Sinergia

B. Entropía

C. Isomorfismo

D. Equifinalidad

7. Este sistema tiene como objetivo generar informes que permitan a los directivos a mejorar el control de la ejecución de las actividades de las áreas funcionales de la organización.

A. *Transaction Proccessing System* – Sistema de Procesamiento de Transacciones

B. MIS (*Management Information System* – Sistema de Información Administrativa)

C. *Decision Support System* – Sistema de Apoyo a las Decisiones

D. *Customer Relationship Managment* – Administración de Relaciones con el Cliente

8. En esta etapa del ciclo de vida de sistemas se transforman los algoritmos en programas y los modelos de datos en base de datos:

A. Análisis

B. Diseño

C. Implantación

D. Implementación

9. En esta disciplina de *Rational Unified Process* se definen y ejecutan los mecanismos para garantizar que el software se haya construido correctamente y que sea el correcto.

A. Análisis y Diseño.

B. Implementación.

C. Pruebas.

D. Despliegue.

10. En esta fase de *Rational Unified Process* uno de los objetivos es identificar los riesgos del proyecto.

A. Transición.

B. Construcción.

C. Elaboración.

D. Concepción.

TEMA 2. ANÁLISIS ESTRUCTURADO

Objetivo particular

Al finalizar el tema, el alumno identificará las actividades y técnicas más relevantes que conforman la administración de requerimientos y el análisis de sistemas utilizados en el enfoque estructurado.

Temario Detallado

2.1. ¿Qué es el Análisis Estructurado?
2.2. Principios del Análisis Estructurado
2.3. Definición de los Requerimientos
2.4. Identificación de los Requerimientos
2.5. Técnicas de Recopilación de Información
    2.5.1. Entrevistas
    2.5.2. Cuestionarios
    2.5.3. Revisión de Registros, Reportes, Formularios, etc.
    2.5.4. Observación
2.6. Análisis de los Requerimientos
2.7. Clasificación de los Requerimientos
2.8. Negociación de los Requerimientos
2.9. Refinamiento de los Requerimientos
2.10. Cualidades del Software como Requerimientos
2.11. Técnicas para la Definición de Requerimientos
    2.11.1. Árboles y Tablas de Decisión
    2.11.2. Español Estructurado
2.12. Diagramas de Flujo
2.13. Administración de los Requerimientos
2.14. Productos (artefactos) de la Definición de Requerimientos
2.15. Glosario

2.16. Modelo de Casos de Uso
2.17. Documento de Especificaciones Técnicas
2.18. Modelado del Análisis
2.19. Elementos del Modelado de Análisis
2.20. Modelado Ambiental con Diagramas de Contexto (DC)
2.21. Modelado de Datos
    2.21.1. Datos, Atributos Y Relaciones
    2.21.2. Cardinalidad Y Modalidad
    2.21.3. Diagrama Entidad-Relación (DER)
2.22. Modelado de Comportamiento
    2.22.1. Diagrama de Flujo de Datos (DFD)
    2.22.2. Diagrama de Flujo de Control (DFC)
    2.22.3. Diagrama de Transición de Estados (DTE)
2.23. Diccionario de datos
2.24. Prototipos del software
2.25. Participantes en el análisis estructurado
2.26. Perfil del analista de sistemas
2.27. Funciones y Responsabilidades del Analista de Sistemas
2.28. Planes y estrategias cuando no se aplica la administración de proyectos
2.29. Estudio de factibilidad y análisis costo-beneficio

Introducción

En este tema profundizarás en el análisis de sistemas con el paradigma estructurado. En todo análisis buscamos representar la realidad de la organización desde la perspectiva dinámica y estática, para lograrlo hacemos uso de diferentes diagramas.

2.1. ¿Qué es el Análisis Estructurado?

En el ciclo de vida de sistemas la primera etapa es la de **análisis**. Esta etapa tiene como propósito identificar cuál es la problemática por resolver y cuál es el comportamiento del sistema actual esté o no automatizado. En esta etapa es muy importante identificar las causas del problema, no es suficiente con conocer el problema, las causas deben ser detectadas para poder atacarlas y dar una solución adecuada. También, se tiene como objetivo identificar las necesidades y requerimientos de los participantes que deben ser contemplados para construir el sistema. El análisis es una actividad muy importante ya que si se hace de manera incorrecta las demás etapas se harán mal, trayendo como consecuencia que el sistema no cumpla con las necesidades por las cuales fue creado.

Existen diferentes metodologías para el análisis estructurado, y por ende anotaciones y simbologías. Los autores más representativos son:
Tom de Marco,
Farley,
Saen a Garzon,
Yourdon.

2.2. Principios del Análisis Estructurado

Antes de que inicies con el análisis estructurado de un sistema tienes que tener en mente los siguientes principios:

- Enfocar el modelo de análisis solamente a los requisitos que ayudan a resolver el problema o satisfacer la necesidad.
- Posponer la toma de decisiones que conciernen a la infraestructura hasta la etapa de diseño.
- Tratar de minimizar el acoplamiento de los módulos del sistema.
- Asegurar que la documentación que se genere aporte valor a la mayoría de los interesados.

Tratar de mantener el análisis lo más simple posible.

Los principios anteriormente mencionados no son absolutos, pueden ser adaptados de acuerdo a las características de la organización o del sistema.

2.3. Definición de los Requerimientos

Un requerimiento es una capacidad del software[8] que solicita de manera implícita o explícita el usuario, esta capacidad busca solucionar un problema o conseguir un objetivo.

Estas capacidades del software, también conocidas como características, son solicitadas por los usuarios o son detectadas por los analistas para satisfacer un contrato, una especificación, un estándar u otra documentación impuesta.

2.4. Identificación de los Requerimientos

La identificación de los requerimientos no es una tarea fácil, de hecho es una de las actividades más complejas de la fase de análisis. En esta actividad interviene mucho la percepción del analista así como de su conocimiento sobre el negocio donde se quiere implantar el sistema. También, intervienen los intereses organizacionales y particulares que tienen el cliente y/o los usuarios. Además, el vocabulario es otro factor que aumenta la complejidad debido a que si no se toma en cuenta el contexto, se podrían estar mal interpretando las ideas de los participantes.

Los requerimientos no solamente son proporcionados por los usuarios también tienen otras fuentes como lo son los reglamentos, los manuales, la competencia, el gobierno, los proveedores, los clientes o cualquier otra fuente que delimite el comportamiento que debe tener el sistema.

[8] En este caso 'software' tiene la misma connotación que sistema.

Para identificar los requerimientos se hace uso principalmente de dos técnicas: las entrevistas y los cuestionaros que se explican en la sección 2.5.1.

2.5. Técnicas de Recopilación de Información

Las técnicas más utilizadas para la recopilación de requerimientos son la entrevista y el cuestionario, aunque también tenemos la revisión de documentación como manuales, reportes, estadísticas, reglamentos, políticas.

2.5.1. Entrevistas

La entrevista es un conjunto de preguntas que están dirigidas a una persona o un grupo muy pequeño de personas. Las preguntas son generalmente abiertas, es decir, no hay una respuesta exacta. El objetivo de la entrevista es obtener datos de carácter cualitativo porque busca conocer el punto de vista de las personas de acuerdo con la realidad que viven, por lo que estos datos no se podrán tabular o manipular estadísticamente.

Su uso en la recopilación de requerimientos se enfoca en buscar los motivos que originan la creación del sistema, las causas que origina el problema, y especificar los procesos.

2.5.2. Cuestionarios

Al igual que la entrevista, es un conjunto de preguntas pero que están dirigidas a una cantidad considerable de personas. Las preguntas son generalmente cerradas: verdadero o falso, opción múltiples, etc.

2.5.3. Revisión de registros, reportes, formularios, etc.

En las organizaciones existe una gran cantidad de documentación que debe ser analizada: manuales, reportes, oficios, memorándums, etc.

Estos documentos son de gran utilidad por que describen en menor o mayor sentido los procesos que se llevan a cabo. Los datos que contienen nos permiten modelar los aspectos de comportamiento y datos. Si se ve desde el punto de vista del cliente el proceso de revisar este tipo de documentación agilizar el proceso de desarrollo del sistema porque el analista al realizar esta actividad se presenta con los usuarios y clientes con una idea más clara de lo que hace y cómo se hace en la organización.

2.5.4. Observación

Esta es una técnica que siempre se utiliza por los analistas de sistemas. Se debe prestar atención en las relaciones interpersonales porque nos ayudan a identificar la forma en que trabajan así como para identificar la organización informal existente.

Está técnica puede ser un tanto incómoda para las personas que están siendo observadas, por lo que el analista debe tener tacto en la forma en que se presenta en el lugar de trabajo de las personas.

2.6. Análisis de los Requerimientos

Conforme se desarrolla el sistema se irán detectando varios requerimientos por lo que se hace necesario analizarlos, es decir, hay que clasificarlos, evaluarlos, y refinarlos.

2.7. Clasificación de los Requerimientos

Básicamente los requerimientos se clasifican en funcionales y no funcionales. A su vez los funcionales se dividen en: Usabilidad, Confiabilidad, Desempeño, Soporte y otros.

Funcionales. Son acciones que el sistema debe desarrollar sin tomar en cuenta restricciones físicas. Este tipo de requerimientos se describen en tablas de decisión, español estructurado, diagramas de flujo pero sobre todo con casos de uso. Los requerimientos funcionales especifican el comportamiento interno y externo del sistema.

Usabilidad. Se refieren al manejo y percepción visual que tiene el usuario con el sistema. Se debe considerar la estética, consistencia de la interfaz del usuario, ayuda en línea, documentación del usuario, materiales de capacitación.

Confiabilidad. Se refieren a la capacidad del sistema para soportar las transacciones que se desea que el sistema ejecute así como la capacidad de recuperación del sistema y la información en caso de falla.

Desempeño. Está muy relacionado con los requerimientos funcionales ya que es cuando un participante solicita: rapidez, eficiencia, disponibilidad, certeza, tiempo de respuesta, tiempo de recuperación.

Soporte. Se refieren a características como: facilidad de prueba, extensibilidad, adaptación, mantenimiento, compatibilidad, configurabilidad, etc.

Los requerimientos no funcionales describen únicamente atributos del sistema o atributos del ambiente del sistema que no tienen que ver directamente con el comportamiento interno y externo del sistema.

2.8. Negociación de los Requerimientos

La constante de todo proyecto es la escasez de recursos, motivo por el cual no se podrán atender todas las peticiones de los participantes. Es tarea del analista entender la misión y objetivos de la organización con el objeto de identificar aquellos requerimientos del sistema que serán de ayuda para lograrlos.

Por la razón anterior el momento más indicado para negociar los requerimientos que se van a desarrollar es en las primeras reuniones. Las personas que deben asistir a esta reunión son los miembros de equipo de desarrollo, los usuarios y los clientes. El equipo de desarrollo debe dar una estimación del tiempo requerido para cada requerimiento, los usuarios deben definir cuáles de los requerimientos son los más importantes; y el cliente confirmar si tiene los recursos necesarios para los requerimientos. La habilidad del analista para cruzar la factibilidad, prioridad y costos es determinante, se espera que pueda dar alternativas que sean de la satisfacción del cliente y que se acople a la misión de la organización.

2.9. Refinamiento de los requerimientos

Un error común en el análisis de sistemas es que con una declaración inicial de los requerimientos es más que suficiente para comprender el problema que se busca resolver. Por un lado tenemos que el analista tiene que seguir un proceso en donde tiene que entender el qué, para qué y para quién de la organización, este proceso consume más tiempo si es la primera vez del analista en trabajar en el giro al que se dedica la organización. Y por el otro lado, los usuarios tienden a pensar cosas que no expresan y que dan por hecho que el analista conoce y entiende.

Estos dos factores enfatizan el cuidado que debe tener al analista al momento de registrar los requerimientos. El analista no puedo dar por hecho nada y debe

estar consciente que conforme se avance en el proyecto deberá especificar con mayor detalle aquellos requerimientos con una alta complejidad o que sean susceptibles a constantes cambios.

Independientemente de la técnica que se haya elegido para registrar los requerimientos, el analista tendrá que actualizar el registro conforme haya nuevos hallazgos o cambios en los requerimientos.

2.10. Cualidades del software como requerimientos

Casi siempre, los usuarios y/o clientes solo expresarán requerimientos de carácter funcional. Sin embargo, esto no es el único tipo de requerimiento ya que tenemos los no funcionales. Estos requerimientos no son sencillos de identificar, sobre todo cuando se empieza. Una ayuda que proporciona la ingeniería de software para identificar requerimientos no funcionales son las llamadas cualidades de software que se listan a continuación:

Cualidad	Descripción
Correcto	Un sistema es correcto si se comporta de acuerdo con la especificación de los requerimientos funcionales que debería proveer. Esta definición de correcto asume que existe una especificación de requerimientos funcionales del sistema y que es posible determinar en forma no ambigua si las cumple o no. Es de notarse que esta definición de correcto no toma en consideración el que la especificación en sí -misma puede ser incorrecta por contener inconsistencias internas, o no corresponder de forma adecuada a las necesidades para las que fue concebido el programa.

Cualidad	Descripción
Confiabilidad	La confiabilidad se define en términos del comportamiento estadístico: la probabilidad de que el sistema opere como se espera en un intervalo de tiempo especificado. Contrariamente a la cualidad de correcto que es una cualidad absoluta, la confiabilidad es relativa. Cualquier desviación de los requerimientos hace que el sistema sea incorrecto, por otro lado, si la consecuencia de un error en el sistema no es seria, el sistema incorrecto aún puede ser confiable.
Robustez	Un sistema es robusto si se comporta en forma razonable aun en circunstancias que no fueron anticipadas en la especificación de requerimientos; por ejemplo cuando encuentra datos de entrada incorrectos o algún malfuncionamiento del hardware. Un sistema que genere un error no recuperable en tiempo de ejecución tan pronto como el usuario ingrese inadvertidamente una instrucción incorrecta no será robusto, aunque podría ser correcto si en la especificación de requerimientos no se establece la acción a tomar si se ingresa una instrucción incorrecta. La robustez es una cualidad difícil de definir, ya que si se pudiera establecer en forma precisa lo que se debiera hacer para obtener un sistema robusto, se podría especificar completamente el comportamiento “razonable”, con lo cual sería equivalente a lo correcto, o a la confiabilidad en el caso ideal mencionado en la definición anterior.

Cualidad	Descripción
Eficiencia o Desempeño	Un sistema es eficiente si utiliza los recursos en forma económica. El desempeño de un sistema es importante porque afecta su usabilidad, por ejemplo, si es muy lento reduce la productividad de los usuarios, si usa demasiado espacio de disco puede ser muy caro de ejecutar, si utiliza demasiada memoria puede afectar al resto de las aplicaciones que se están ejecutando o ejecutarse demasiado lentamente mientras el sistema operativo intenta balancear el uso de la memoria por parte de las distintas aplicaciones. Detrás de estos problemas están los límites cambiantes de la eficiencia según cambia la tecnología, ya que la visión de “demasiado caro” cambia constantemente según los avances tecnológicos, actualmente una computadora cuesta bastante menos que hace unos años y son bastante más poderosas.
Amigabilidad	Un sistema es amigable si un usuario humano lo encuentra fácil de utilizar. Esta definición refleja la naturaleza subjetiva de la amigabilidad: un sistema utilizado por usuarios no experimentados califica como amigable por varias propiedades distintas a las de un sistema utilizado por usuarios expertos.
Verificabilidad	Un sistema es verificable si sus propiedades pueden ser verificadas fácilmente. Por ejemplo, la cualidad de correcto o el desempeño de un sistema son propiedades que interesa verificar. El diseño modular, prácticas de codificación disciplinadas, y la utilización de lenguajes de programación adecuados contribuyen la verificabilidad de un sistema. La verificabilidad es en general una cualidad interna pero a veces también puede ser externa.

Cualidad	Descripción
Mantenibilidad	El término mantenimiento es utilizado generalmente para referirse a las modificaciones que se realizan a un sistema luego de su liberación inicial, siendo visto simplemente como “corrección de *bugs*”. Algunos estudios han mostrado que la mayor parte del tiempo utilizado en mantenimiento es para agregarle al producto características que no estaban en las especificaciones originales o estaban definidas incorrectamente. En realidad la palabra “mantenimiento” no es apropiada para el software ya que cubre un amplio rango de actividades que tienen que ver con la modificación de un software existente para lograr una mejora, un término que se aplica mejor a este proceso es “evolución del software”. Hay evidencia de que los costos de mantenimiento exceden entre el 60% y el 80% del total de los costos del software. Para analizar los factores que afectan estos costos, es usual dividir el mantenimiento del software en tres categorías: de corrección, de adaptación y de mejora.

Cualidad	Descripción
Reparabilidad	Un sistema es reparable si permite la corrección de sus defectos con una carga limitada de trabajo. En el software las partes no se deterioran, y aunque el uso de partes estándares puede reducir el costo de producción del sistema, el concepto de partes reemplazables pareciera no aplicar a la reparabilidad del sistema. Otra diferencia es que el costo del software está determinado, no por partes tangibles sino por actividades de diseño. Un producto de software consistente en módulos bien diseñados es más fácil de analizar y reparar que uno monolítico, sin embargo, el solo aumento del número de módulos no hace que un producto sea más reparable. Una modularización adecuada con definición adecuada de interfaces que reduzcan la necesidad de conexiones entre los módulos promueve la reparabilidad ya que permite que los errores estén ubicados en unos pocos módulos, facilitando la localización y eliminación de los mismos.

Cualidad	Descripción
Evolucionabilidad	Un sistema es *evolucionable* si acepta cambios que le permitan satisfacer nuevos requerimientos. En el caso de sistemas, en general la implementación del cambio se comienza sin realizar ningún estudio de factibilidad, dejando únicamente el diseño original y sin actualizar las especificaciones para reflejarlo, lo que hace que cambios futuros sean cada vez más difíciles de aplicar. La *evolucionabilidad* es una cualidad tanto del producto como del proceso, aplicado al segundo éste debe ser capaz de adaptarse a nuevas técnicas de gestión y organización, cambios en la educación en ingeniería, etc. Es una de las cualidades más importantes del software, e involucra otros conceptos como familias de programas cuyo propósito es fomentar la evolucionabilidad.
Reusabilidad	La reusabilidad es similar a la evolucionabilidad: en la segunda se modifica un producto para construir una nueva versión del mismo producto, en la primera se utiliza un producto, posiblemente con modificaciones menores, para construir otro producto. Es difícil lograr reusabilidad terminado el sistema, por el contrario se debe contemplar al momento de desarrollar los componentes del software. Otro nivel de reusabilidad está dado en los requerimientos: al analizar una nueva aplicación se pueden identificar partes que son similares a otras utilizadas en una aplicación previa. También, en el nivel del código se pueden reutilizar componentes desarrollados en una aplicación anterior.

Cualidad	Descripción
Portabilidad	El sistema es portable si puede ser ejecutado en distintos ambientes, refiriéndose este último tanto a plataformas de hardware como a ambientes de software como puede ser determinado sistema operativo. Si bien se ha transformado en un tema importante debido a la proliferación de procesadores y sistemas operativos distintos, puede ser importante incluso en una misma familia de procesadores debido a las variaciones de capacidad de memoria e instrucciones adicionales, por lo que una forma de lograr portabilidad es asumir una configuración mínima y utilizar un subconjunto de las facilidades provistas que se garantiza estarán disponibles en todos los modelos de la arquitectura, como instrucciones de máquina y facilidades del sistema operativo. También es necesario utilizar técnicas que permitan al software determinar las capacidades del hardware y adaptarse a éstas. En general la portabilidad se refiere a la habilidad de un sistema de ser ejecutado en plataformas de hardware distintas, y a medida que la razón de dinero gastado en software contra hardware crece, la portabilidad gana importancia.

Cualidad	Descripción
Comprensibilidad	Algunos sistemas son más fáciles de comprender que otros, algunas tareas son inherentemente más complejas que otras. La comprensibilidad es una cualidad interna del producto y ayuda a lograr muchas de las otras cualidades como evolucionabilidad y verificabilidad. Desde un punto de vista externo, un usuario considera que un sistema es comprensible si su comportamiento es predecible, en este caso la comprensibilidad es un componente de la amigabilidad al usuario.
Interoperabilidad	La interoperabilidad se refiere a la habilidad de un sistema de coexistir y cooperar con otros sistemas. Un concepto relacionado con la interoperabilidad es el de sistema abierto, que es una colección extensible de aplicaciones escritas en forma independiente que cooperan para funcionar como un sistema integrado y permiten la adición de nuevas funcionalidades por parte de organizaciones independientes, luego de ser liberado.
Productividad	La productividad es una cualidad del proceso de producción de software, mide la eficiencia del proceso y es una cualidad de desempeño aplicada al proceso. Un proceso eficiente resulta en una entrega más rápida del producto.

Cualidad	Descripción
Oportunidad	La oportunidad es una cualidad del proceso que se refiere a la habilidad de entregar un producto a tiempo. Históricamente los procesos de producción de software no han tenido esta cualidad lo que llevó a la llamada “crisis del software” que a su vez trajo aparejada la necesidad y el nacimiento de la ingeniería de software. Incluso actualmente muchos procesos fracasan en lograr sus resultados a tiempo. Entregar un producto a tiempo requiere una planeación cuidadosa, con un trabajo de estimación acertado y puntos de revisión especificados claramente y verificables. Todas las demás disciplinas de la ingeniería utilizan técnicas estándares de administración de proyectos.
Visibilidad	Un proceso de desarrollo de software es visible si todos sus pasos y su estado actual son claramente documentados. Otros términos utilizados para visibilidad son transparencia y apertura. La idea es que los pasos y el estado del proyecto están disponibles y fácilmente accesibles para ser examinados externamente.

Cuadro 2.1. Cualidades de software como requerimientos

2.11. Técnicas para la definición de requerimientos

Las técnicas que se presentan en las siguientes secciones no son las únicas que existen.

2.11.1. Árboles y tablas de decisión

Árboles de decisión.- El árbol de decisión es un diagrama que representa en forma secuencial condiciones y acciones; muestra qué condiciones se

consideran en primer lugar, en segundo lugar y así sucesivamente. Este método permite mostrar la relación que existe entre cada condición y el grupo de acciones permitidas asociadas con ella.

Los árboles de decisión son normalmente construidos a partir de la descripción escrita de un problema. Los árboles proporcionan una visión gráfica de la toma de decisión necesaria, especifican las variables que son evaluadas, qué acciones deben ser tomadas y el orden en el cual la toma de decisión será efectuada. Cada vez que se ejecuta un árbol de decisión, solo un camino será seguido dependiendo del valor actual de la variable evaluada. Se recomienda el uso del árbol de decisión cuando el número de acciones es pequeño y no son posibles todas las combinaciones.

Tabla de decisión. La tabla de decisión es una matriz de filas y columnas que indican condiciones y acciones. Las reglas de decisiones, incluidas en una tabla de decisión establecen el procedimiento a seguir cuando existen ciertas condiciones. Este método se emplea desde mediados de la década de los 50, cuando fue desarrollado por General Electric para el análisis de funciones de la empresa como control de inventarios, análisis de ventas, análisis de créditos y control de transporte y rutas. Se utiliza la tabla de decisión cuando existen muchas combinaciones.

2.11.2. Español estructurado

También conocido como lenguaje estructurado, es el más utilizado para realizar especificaciones de procesos. Es un subconjunto del español, como lo es el inglés en los lenguajes de programación.

El propósito del español estructurado es proporcionar los elementos suficientes para construir especificaciones de procesos tales que, sin perder la capacidad de comunicación que tiene el lenguaje natural, se encaminen hacia lo preciso y

lo estructurado de los lenguajes de programación. Es decir, se busca especificar el proceso sin perder de vista que lo que se está especificando en la fase de análisis es un proceso desde el punto de vista del usuario.

El esquema de especificación del español estructurado está compuesto de: sentencias declarativas simples; estructuras de decisión y estructuras de repetición. También, contempla cualquier combinación que pueda darse entre estos esquemas. El vocabulario del español estructurado está conformado por:

si condición **entonces**
    bloque acciones
sino
    bloque acciones
fin-si

mientras condición hacer
    bloque acciones
fin-mientras

repetir
    bloque acciones
hasta condición

para var **desde** inicio **hasta** fin **hacer**
    bloque acciones
fin-para

Recomendaciones cuando se especifica con español estructurado:

- Considerar sólo aquellas palabras que tienen un significado único, ya que de lo contrario no serán de ayuda para la descripción de procesos al permitir el surgimiento de ambigüedades.
- Excluir sinónimos, adjetivos, adverbios, símbolos de exclamación e interrogación.
- Considerar el uso de modos verbales en infinitivo, como calcular, o en imperativo, como comienza.
- Considerar que los verbos deben ser, principalmente, verbos orientados a la acción, como encontrar, obtener, sumar, reemplazar, etc.
- Evitar, en lo posible, verbos como procesar o manejar, ya que se les considera simplemente no significativos o redundantes.
- Sustituir toda sentencia que pueda reemplazarse por otra más simple.
- Evitar incluir anotaciones fuera de línea, como notas al margen.

Con éstas consideraciones el español estructurado permite configurar las frases de las especificaciones de los proceso. En las frases deben tomarse los nombres dados en los almacenes, flujos de datos, y datos elementales, tanto en el Diccionario de datos como en el Diagrama de flujo de datos, o bien ser términos locales.

2.12. Diagramas de flujo

Un diagrama de flujo es una forma de representar gráficamente los detalles algorítmicos de un proceso. En el análisis se ocupan para especificar procesos que se ejecutan actualmente. Estos diagramas utilizan una serie de símbolos con significados especiales y son la representación gráfica de los pasos de un proceso.

A continuación se lista una serie de acciones que son recomendables hacer antes de elaborar el diagrama de flujo:

- Identificar a los participantes de la reunión donde se desarrollará el diagrama de flujo. Lo recomendable es que esté presente toda aquella persona que tenga que ver o se vea afectada por el proceso.
- Definir qué se espera obtener del diagrama de flujo.
- Identificar quién lo empleará y cómo.
- Establecer el nivel de detalle requerido.
- Determinar los límites del proceso a describir.

Los pasos a seguir para elaborar el diagrama de flujo son:

- Establecer el alcance del proceso a describir. De esta manera quedará fijado el comienzo y el final del diagrama. Frecuentemente el comienzo es la salida del proceso previo y el final la entrada al proceso siguiente.
- Identificar y listar las principales actividades que están incluidos en el proceso a describir y su orden cronológico.
- Identificar y listar los puntos de decisión.
- Construir el diagrama respetando la secuencia cronológica y asignando los correspondientes símbolos.
- Asignar un título al diagrama y verificar que esté completo y describa con exactitud el proceso elegido.

2.13. Administración de los requerimientos

La administración de requerimientos comprende las actividades relacionadas con la identificación, especificación, clasificación, seguimiento y control de los requerimientos durante todo el ciclo de vida de desarrollo del sistema. Es una metodología indispensable para el aseguramiento de la calidad de los productos, así como para el control y seguimiento de los proyectos.

2.14. Productos (artefactos) de la definición de requerimientos

En el desarrollo de sistemas, se denomina producto o artefacto a cualquier tipo de documentación que se genere durante el proceso de construcción del sistema. Para la etapa de análisis los productos más conocidos son:

- Glosario.
- Entrevista.
- Cuestionario.
- Especificación de procesos.
- Diccionario de datos.
- Diagrama de contexto.
- Diagrama de flujo de datos.
- Diagrama de entidad-relación.
- Diagrama de transición de estados.

Los cuatros últimos artefactos son específicos del análisis estructurado, por lo que no es de extrañarse que existan otros.

2.15. Glosario

El glosario es un documento donde se describen las palabras que forman parte de la jerga común de los usuarios y clientes. En algunos casos, se incluyen las palabras que ocupa el equipo de desarrollo, esto se da cuando el usuario necesita tener acceso a este documento.

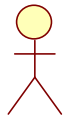
Cuando el equipo de desarrollo no forma parte de la organización donde se va a construir este sistema o los miembros del equipo están en constante rotación es muy importante que se elabore este documento. El beneficio se encuentra en que se ahorra tiempo en estar explicando aquellos conceptos que son de importancia que debe dominar cada miembro del equipo.

El glosario ayuda a identificar entidades y atributos cuando se está realizando el modelo de datos. Con esto se asegura que se están contemplando todos los datos que se requerirán procesar para generar la información que necesitan los usuarios.

2.16. Modelo de casos de uso

En el modelo de casos de uso se utilizan los diagramas de casos de uso en los cuales participan los actores y los casos de uso. La característica principal del modelo de casos de uso es que identifica requerimientos de tipo funcional.

Los actores son personas, organizaciones u otro sistema que interactúa con el sistema que estamos modelando. Para nombrarlos se tiene que poner el nombre del rol que está jugando con el sistema. Su símbolo es:



Los casos de uso representan la funcionalidad que observa directamente los actores, otro objetivo es representar y especificar los requerimientos funcionales. Los casos de uso son procesos que lleva a cabo el sistema para procesar los datos proporcionados por los actores y generar la información que requieren los actores. Su símbolo es una elipse u óvalo con el nombre del caso de uso dentro de él o debajo de él:

Los actores y casos de uso se relacionan entre sí con una línea recta que se llama asociación. Esta asociación indica que hay un intercambio de datos/información entre el actor y el caso de uso. Se recomienda poner los casos de uso dentro de un rectángulo para indicar los límites del sistema este rectángulo va acompañado de una leyenda que dice “system” en la esquina superior derecha.

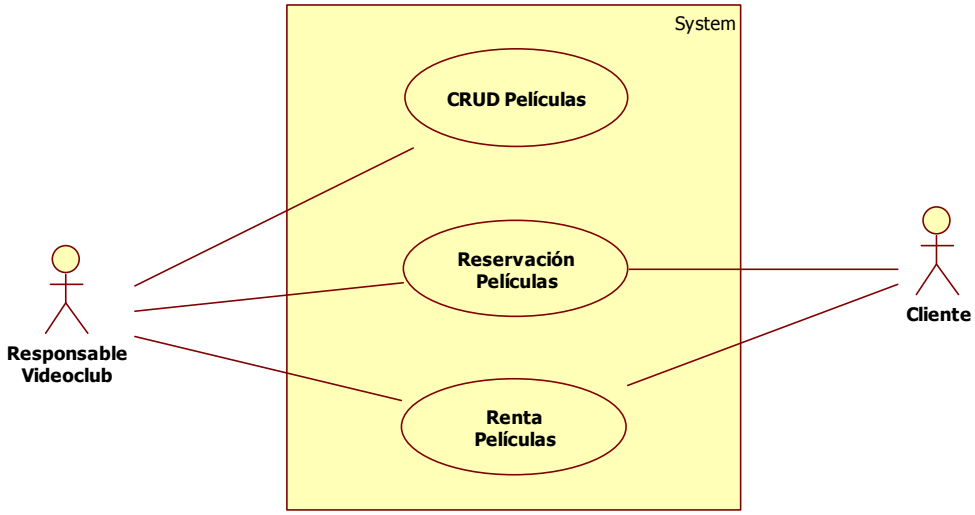


Figura 2.1. Diagrama de casos de un videoclub. Elaboración propia con la notación UML 2.1

2.17. Documento de especificaciones técnicas

El diagrama de casos de uso no es suficiente para entender que es lo que se necesita que haga el sistema, por lo que se hace necesario describir el detalle de cada caso uso, esta descripción también recibe el nombre de especificación de caso de uso o especificación técnica.

Las secciones de una descripción básica de un caso de uso son:

- Nombre.
- Descripción.
- Actores.

- Precondiciones.
- Flujo Principal o Normal.
- Flujos Alternativos.
- Poscondiciones.
- Excepciones.

El **nombre** corresponde al caso de uso que se desea especificar del diagrama de casos de uso. Es necesario recalcar que el nombre del caso de uso empieza con un verbo y después lo acompañan los sustantivos que permitan identificar qué hace el caso de uso.

La **descripción** es texto breve que explica en qué consiste el caso de uso.

Los **actores** son los que intervienen en ese caso de uso, por lo que sólo se debe poner los actores que están asociados con el caso de uso en el diagrama.

Las **precondiciones**, son condiciones que se pueden o deben presentarse antes de que inicie el caso de uso.

La parte más importante es el **flujo principal** y **flujos alternativos**, en algunas ocasiones se juntan o se separan, todo depende de que escritura facilite más su comprensión.

La redacción de los casos de uso debe ser simple ya que es una narrativa que describe las acciones que hay entre el sistema y el actor. Siempre tiene el patrón el actor hace, el sistema hace, el actor hace, el sistema hace. Una ayuda de esto es la numeración de cada una de las actividades que ocurren entre el actor y el sistema.

5.1.- Si tiene adeudos vencidos.
5.1.1.- El sistema le indica al Responsable de Videoclub que el cliente no puede rentar películas hasta que entregue las que debe.
El sistema actualiza el formulario solicitando los números de las películas que debe devolver el cliente.
5.1.2.- El Responsable de Videoclub proporciona los números de películas y acepta registrar la devolución.
5.1.3.- El sistema registra la devolución.
5.1.4.- El sistema presenta el comprobante de de devolución que tiene el número de cliente, el nombre del cliente, las claves y nombres de las películas, la fecha y hora de renta, y la fecha y hora de devolución.
5.1.5.- El Responsable de Videoclub selecciona imprimir el comprobante de devolución (**Excepción 4**).
5.1.6.- El sistema imprime el comprobante de devolución.
5.1.7.- El caso de uso reinicia desde el punto 5.

Pos-Condiciones.- Ninguna.

Excepciones.-
1.- El cliente proporcionó un valor no numérico. El sistema le indica al Responsable de Videoclub que el valor proporcionado no es correcto. Lo corrige o el caso de uso termina.
2.- El número de cliente proporcionado no existe. El sistema le indica al Responsable de Videoclub que el número de cliente proporcionado no corresponde a algún cliente. Corrige o pone otro número o el caso de uso termina.
3.- El número de película proporcionado no existe. El sistema le indica al Responsable de Videoclub que el número de película proporcionado no corresponde a alguna película. Corrige o pone otro número de película o el caso de uso termina.
4.- La impresora no está en línea. El sistema le indica al Responsable de Videoclub que la impresora está inactiva y el caso de uso termina.

Hay que tener cuidado de evitar describir interrelaciones entre actores; lo único que interesa son las interrelaciones entre el actor y el sistema. Ejemplo:

Nombre: Rentar películas

Descripción: Registrar las películas que son rentadas por los clientes.

Actores: Responsable de Videoclub

Pre-Condiciones: El cliente debe traer y presentar su membresía.

Flujo Principal y Alternativos:

El caso de uso inicia cuando el cliente se presenta en el mostrador con las películas que quiere rentar.

1.- El Responsable de Videoclub le solicita su membresía al cliente.
2.- El Responsable de Videoclub selecciona la opción de renta de películas.
3.- El sistema le muestra un formulario al Responsable de Videoclub solicitando el número de cliente.
4.- El Responsable de Videoclub proporciona el número de cliente que se encuentra en la membresía y acepta iniciar la renta (**Excepción 1**) (**Excepción 2**).
5.- El sistema verifica los adeudos de entrega de películas del cliente.

Figura 2.2. Especificación de casos de uso

Nótese que la numeración es un factor importante para la claridad de la redacción ya que nos permite describir situaciones donde el actor tenga que elegir una opción de un conjunto de alternativas.

Las **poscondiciones** son condiciones o acciones que se tienen que hacer después de concluido alguno de los flujos del caso de uso.

Las **excepciones** ayudan a establecer flujos que están fuera de lo normal, es decir, son eventos que no tienen nada que ver con las actividades normales del negocio. Ejemplo de ello tenemos que el usuario puso texto en un campo que se espera ponga números, o que haya fallado la conexión a la base de datos, que se haya ido la luz, que el protocolo http (*hyper text transfer protocol*) no esté disponible, etc.

2.18. Modelado del análisis

Cuando se requiere comunicar los resultados de abstraer la realidad respecto a cierto ambiente no es suficiente que se expresen en forma verbal, para ello se hace uso de los modelos. Un modelo es una representación de la realidad, para el caso del análisis de sistemas y según el paradigma que se esté utilizando, se emplean ciertos diagramas para especificar los modelos.

La comunicación que se da en forma interna y externa respecto al equipo de desarrollo. La comunicación interna requiere de mayor especificidad por tratarse de iguales en el equipo de desarrollo, al contrario de la comunicación externa su formalidad puede ser menor, a menos que sea un requerimiento por parte del cliente. En la comunicación externa se requiere de simplicidad y formalidad, los usuarios y clientes desconocen los tecnicismos del proceso de desarrollo de software por lo que se debe tener especial cuidado de elegir los artefactos que se presentarán al usuario, incluso se tendría que capacitar a los usuarios en el uso de los artefactos.

2.19. Elementos del modelado de análisis

Identificando el problema al cual se le requiere dar solución se procede a documentar la parte dinámica y estática del sistema actual. Los aspectos que se encargan de modelar la parte dinámica del sistema es el Ambiental y el de Comportamiento.

El aspecto que se encarga de modelar la parte estática del sistema es el Aspecto de Información. Cada uno de estos aspectos va a generar un modelo, por lo que tenemos un Modelo Ambiental, un Modelo de Comportamiento y un Modelo de Datos.

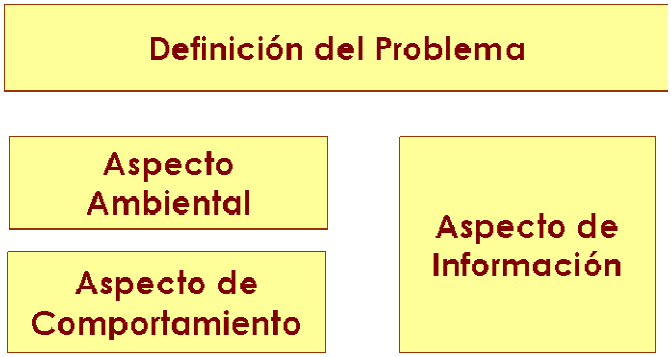


Figura 2.3. Aspectos del modelo de análisis

¿Por cuál aspecto se debe comenzar? La respuesta está en función de los datos que tenemos a nuestra disposición y de la comprensión de esos datos.

Si tenemos un mejor conocimiento de los procesos de la organización es recomendable empezar por el Aspecto Ambiental y el Aspecto de Comportamiento para luego pasar al Aspecto de Información. Si tenemos un mejor conocimiento sobre los datos e información que genera la empresa es recomendable empezar por el Aspecto de Información y posteriormente con el Aspecto Ambiental y el Aspecto de Comportamiento. Estos aspectos se les denominan los elementos del modelo de análisis de sistemas.

2.20. Modelado ambiental con Diagramas de Contexto (DC)

El objetivo del aspecto ambiental es modelar el aspecto dinámico externo del sistema, es decir, establecer el alcance del sistema así como sus límites. Para modelar el aspecto ambiental se utiliza el denominado diagrama de contexto. Cabe destacar que este diagrama se ocupa tanto para el sistema actual como para el sistema propuesto. En este diagrama se ocupan 3 símbolos:

Nombre	Descripción
Burbuja (Sistema) 	Representa al sistema que se está modelando y solo hay una en cada diagrama de contexto. Dentro de la burbuja se debe poner el nombre del sistema que se está representando.
Rectángulo (Entidad) 	Representa un agente externo o entidad que interactúa con el sistema proporcionando datos al sistema o recibiendo información del sistema. Dentro del rectángulo se debe poner el nombre del agente externo con mayúsculas y en singular. Los agentes externos pueden ser personas, organizaciones u otro sistema.

Nombre	Descripción
Flechas (Flujos) 	Representan el sentido en que se envía la información, ya sea del sistema a un agente externo o en sentido contrario. Hay de dos tipos: unidireccionales y bidireccionales. Las unidireccionales solo hay una flecha en un extremo y se ocupan para representar que se está enviando o recibiendo datos de manera asíncrona. Las bidireccionales tienen flecha en ambos extremos y se ocupan para representar que se está enviando y recibiendo datos de manera síncrona. Se pueden nombrar en forma de título o minúsculas. Siempre deben estar nombrados con sustantivos y no con verbos, además se deben omitir las palabras datos e información, ya que de hacerlo, estaríamos siendo redundantes, a fin de cuentas en este diagrama lo que se quiere ver son flujos de información entre el sistema y los agentes externos.

Cuadro 2.2. Símbolos del Diagrama de Contexto

En el cuadro, puedes observar que por el nombre que tiene la burbuja se trata de un sistema de un videoclub el cual se lleva de manera manual. Realmente es un sistema muy sencillo donde el Responsable del Videoclub envía al sistema los datos de la película y del cliente para realizar la renta, como respuesta, el Responsable del Videoclub tiene un comprobante de Renta. También por medio del título de la película o el autor puede buscar las películas y obtendrá un listado de las películas que coincidan con esos datos. Y por último el Responsable del Videoclub proporciona los datos de la película ya sea para ingresar una nueva y, actualizar o eliminar una película existente.

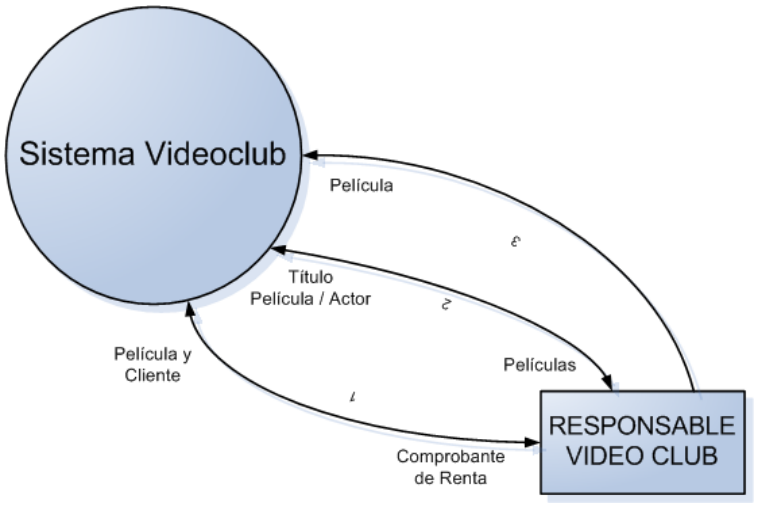


Figura 2.4. Diagrama de Contexto

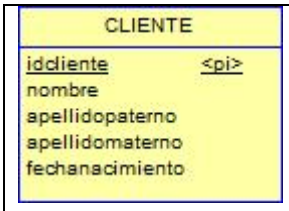
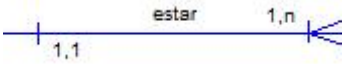
2.21. Modelado de datos

El objetivo del aspecto de información es modelar el aspecto estático del sistema, es decir, las estructuras de persistencia de datos y las relaciones que existen entre ellas. Para modelar el aspecto de información se utiliza el denominado Diagrama de Entidad-Relación (DER). Cabe destacar que este diagrama se ocupa tanto para el sistema actual como para el sistema propuesto.

2.21.1. Datos, atributos y relaciones

Las entidades son los elementos más significativos en el DER, cada entidad está compuesta de atributos. En un futuro, cuando el DER se convierta en una base de datos, cada atributo contendrá algún tipo de datos:

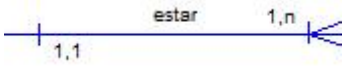
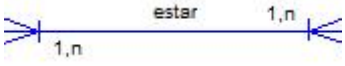
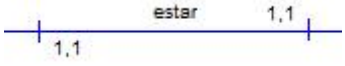
Nombre	Descripción
Rectángulo (Entidad)	Representa una relación, también llamada entidad, la cual almacena datos de un objeto de la vida real. Las entidades son los almacenes que se ocupan en el diagrama de flujo de datos. Dentro del rectángulo se

	debe poner el nombre de la entidad con mayúsculas y en singular. También, se debe poner los atributos de la entidad, es decir, sus características, las cuales se escriben con minúsculas sin espacios. Hay un atributo especial denominado identificador, el cual tiene como propósito distinguir un registro con respecto a los demás registros. En algunos casos el identificador es compuesto, es decir, está conformado por más de un atributo.
Relación 	Representa una relación bidireccional entre dos entidades. Las relaciones se dibujan con una línea continua con su nombre en minúsculas. El nombre de la relación debe estar escrito con un verbo y con letras minúsculas. Otro elemento importante de las relaciones es la “cardinalidad” la cual indica la forma en que se relacionan la entidades.

Cuadro 2.3. Símbolos del Diagrama de Entidad-Relación

2.21.2. Cardinalidad y modalidad

Existen tres tipos de cardinalidad: de uno a uno, de uno a muchos y de muchos a muchos.

Símbolo	Descripción
	Relación de uno a muchos.
	Relación de muchos a muchos.
	Relación de uno a uno.

Cuadro 2.4. Símbolos para representar la cardinalidad

Respecto a la modalidad hay de cuatro tipos:

Modalidad	Descripción
0, 1	Establece una interrelación en donde la entidad puede tener/contener ningún o un elemento de la otra entidad.
1, 1	Establece una interrelación en donde la entidad debe tener/contener un elemento de la otra entidad.
0, n	Establece una interrelación en donde la entidad puede tener/contener ningún o más de un elemento de la otra entidad.
1, n	Establece una interrelación en donde la entidad puede tener/contener uno o más de un elemento de la otra entidad.

Cuadro 2.5. Nomenclatura para representar la modalidad

En la siguiente sección se explican con un ejemplo cómo se ocupa la cardinalidad y la modalidad.

2.21.3. Diagrama Entidad-Relación (DER)

Se hace necesario recalcar que hay dos tipos de diagramas de entidad-relación: el lógico y el físico. El primero se genera en la fase de análisis y el segundo en la fase de diseño.

En este ejemplo se representa un diagrama de entidad-relación lógico del sistema videoclub.

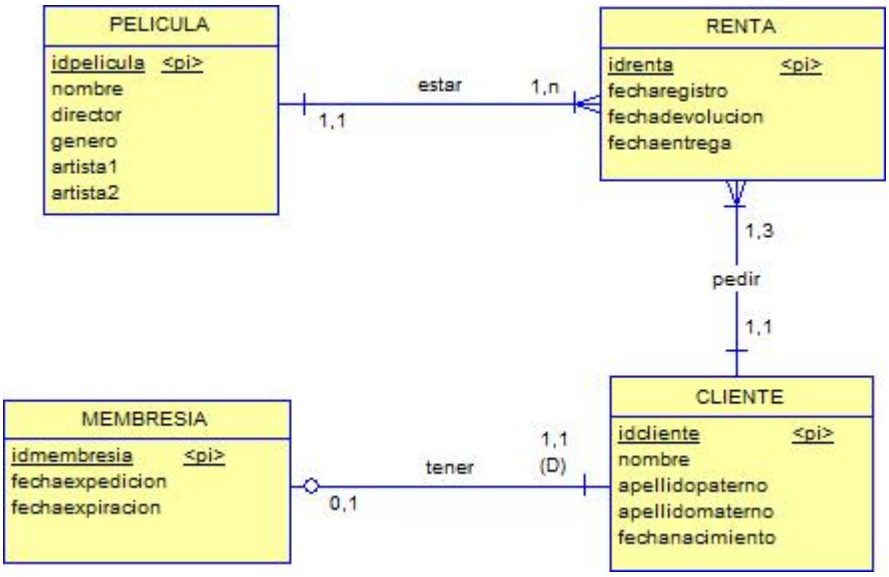


Figura 2.5. Diagrama de Entidad-Relación.

Como primera relación encontramos la de la entidad PELÍCULA y la entidad RENTA, la cual se leería “una película está en una o más rentas, y una renta está o tiene una película”. En esta relación apreciamos que la cardinalidad se pone en ambos extremos, la que está del lado izquierdo se lee “una renta está o tiene una película”, por eso se pone “1 coma 1” el primer número representa el valor mínimo y el segundo número representa el valor máximo, estos números reciben el nombre de modalidad. La cardinalidad del lado derecho se lee “una película está en renta una o más veces”, por eso se pone “1 coma n”, nótese que en este caso en especial tenemos una representación iconográfica denominada “patas de gallo”, que solo se utiliza en las relaciones de uno a muchos o de muchos a muchos.

Otra relación es la que hay entre la entidad MEMBRESÍA y la entidad CLIENTE, la cual se leería “una membresía es / tiene uno y solo un cliente, y un cliente puede tener una membresía”. La cardinalidad del lado izquierdo se lee “un cliente puede tener una membresía”, por eso se pone “0 coma 1”. La cardinalidad del lado derecho se lee “una membresía es / tiene uno y solo un cliente” por eso se pone “1 coma 1”, en este tipo de relación es importante denotar la entidad que es dominante, para ello se ocupa la letra “D” que indica

que la entidad cliente es la fuerte, y para cuando se pase al diagrama de entidad relación física la entidad cliente pasará su llave primaria como foránea a la entidad membresía. Con esto se evita tener referencia circular entre tablas.

Con estos dos ejemplos de relaciones se recalca la importancia de identificar las reglas de negocio en la fase de requerimientos, ya que ellas nos indican cómo construir estos diagramas.

2.22. Modelado de comportamiento

El objetivo del aspecto de comportamiento es modelar el aspecto dinámico interno del sistema. Para modelar el aspecto de comportamiento se utiliza el denominado diagrama de flujo de datos, que al igual que el diagrama de contexto permite modelar el aspecto dinámico de un sistema manual como de un sistema automatizado.

2.22.1. Diagrama de Flujo de Datos (DFD)

De igual forma que en el Diagrama de Contexto, en los Diagramas de Flujo de Datos se ocupan los símbolos de: burbuja, que representa un proceso; rectángulo, que representa un agente externo; y la flecha, que representa un flujo. Estos símbolos tienen la misma semántica. En este diagrama se incluye un nuevo símbolo que está compuesto por dos líneas horizontales paralelas.

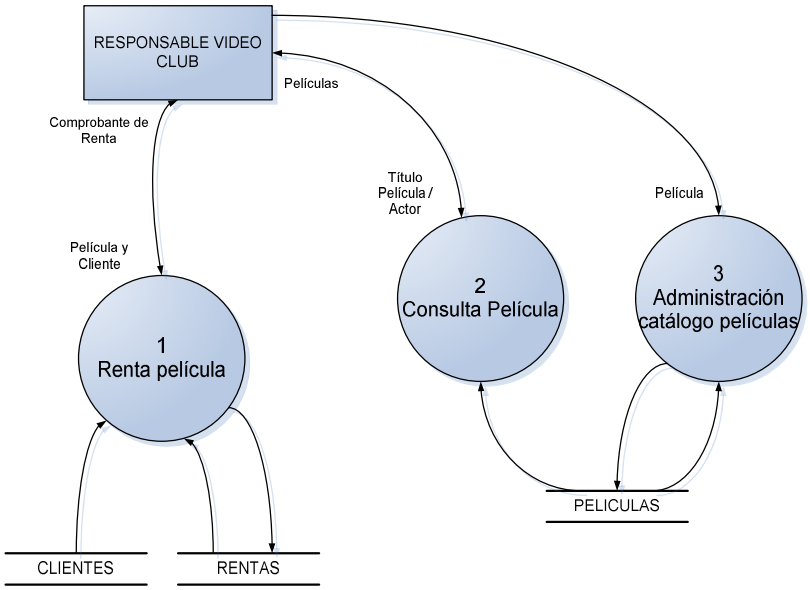


Figura 2.6. Diagrama de Flujo de Datos

Las líneas horizontales paralelas se utilizan para modelar un almacén, los almacenes son repositorios de datos, su objetivo es que se pueda guardar, modificar, eliminar y consultar datos cuando se requiera. En el caso de un sistema manual un almacén es un archivero, agenda, o cualquier lugar donde se almacenen papeles con datos. En el caso de un sistema automatizado los almacenes son archivos electrónicos o tablas en alguna base de datos. Los nombres de los almacenes deben ser sustantivos y, escribirse con mayúsculas y en plural.

Los diagramas de contexto no son suficientes para tener una visión completa del sistema, falta conocer cómo trabaja o trabajará internamente el sistema. Los DFD ayudan en esta parte. Teniendo el diagrama de contexto se procede a crear su DFD correspondiente el cual es conocido como Diagrama de Flujo de Datos de Nivel cero o uno. En este diagrama se descompone el sistema en sus procesos principales, como regla tenemos que solo pueden haber 7 + 2 procesos, es decir, hasta 9 procesos. Cuando se rebasa la cantidad de 9 procesos se está siendo muy detallado por lo que será necesario juntar uno o más procesos. No es correcto tener un diagrama de flujo de datos que solo tenga un burbuja.

En los diagramas de flujo de datos se busca identificar los procesos que reciben datos, los procesan y generan información. En este caso las burbujas no representan sistemas sino procesos los cuales deben de ir numerados, esta numeración NO indica su precedencia solo se utiliza para identificar un proceso de otro. Todo proceso debe iniciar con un verbo. Los procesos solo pueden tener flujos de un agente externo o de un almacén. Solo se puede comunicar un proceso con otro cuando se trata de un sistema en tiempo real. El sentido de la dirección de los flujos indica si el proceso recibe o proporciona datos.

Los flujos que van o vienen de un almacén no llevan nombre porque solo se puede enviar u obtener datos que correspondan al almacén, esto hace necesario que tenga un nombre adecuado para los datos que guarda. El significado del flujo varía según la dirección que tenga:

- Cuando un flujo va de un proceso hacia un almacén significa que se está agregando un nuevo registro, o se está modificando un registro existente, o se está eliminado un registro existente.
- Cuando un flujo va de un almacén a un proceso significa que se está recuperando datos de un registro existente.

Los procesos de los diagramas de flujos de datos pueden dividirse a su vez en otro diagrama de flujo de datos el cual está conformado por los subprocesos que conforman el proceso principal. ¿Hasta qué profundidad se puede hacer esto? Lo más recomendable es no pasar de 7 niveles. En estos diagramas se siguen las mismas reglas que para los DFD de nivel 1. En este ejemplo se puede apreciar que descompusimos el “Proceso de Administración de Catálogo de Películas”, en este diagrama los procesos tienen otra numeración, la cual está formada por el número del proceso padre y número consecutivo de cada proceso.

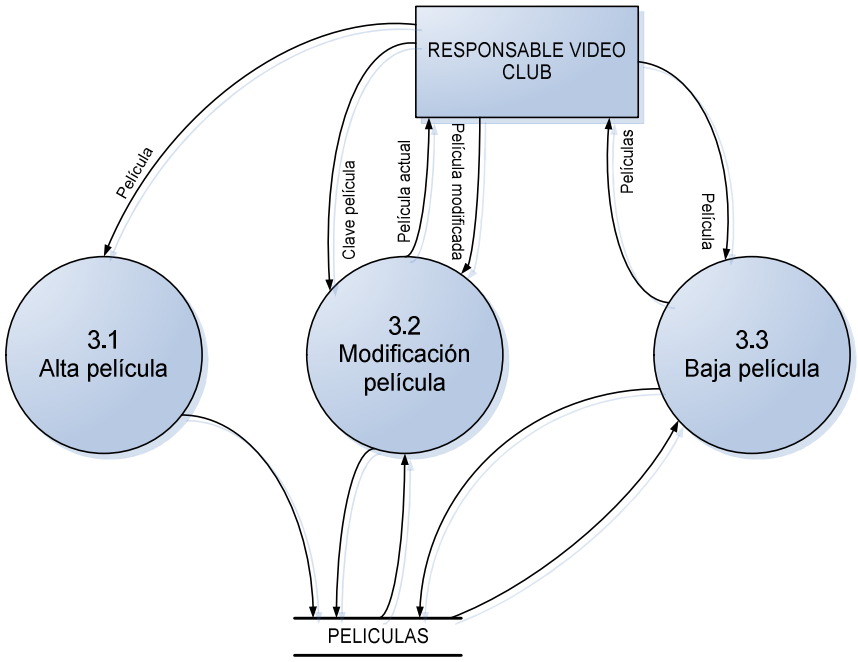
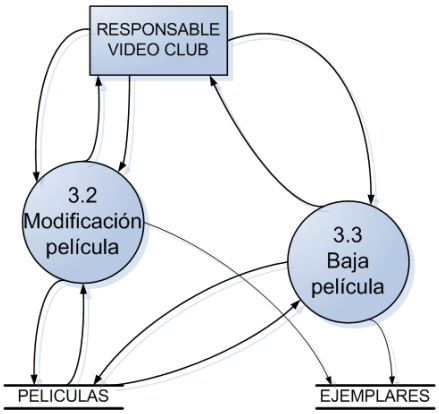


Figura 2.7. Diagrama de Flujo de Datos de Nivel 2

En algunas situaciones se puede presentar que un flujo que atraviesa otros flujos; lo cual no es recomendable porque dificulta la lectura del diagrama y puede llegar a convertirse en una telaraña. Para estos casos lo recomendable es repetir la entidad o almacén que está involucrado agregando una línea inclinada en la parte inferior derecha para indiciar que esa entidad y/o almacén ya fue creado anteriormente y que sólo se trata de un clon de la original.

Incorrecto



Correcto

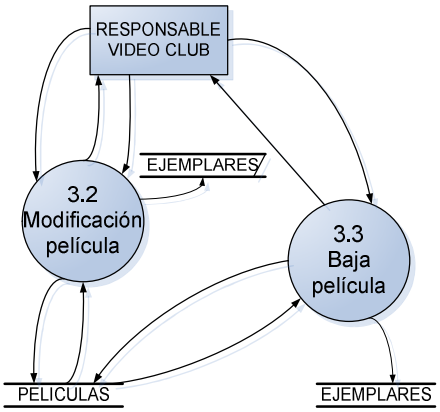


Figura 2.8. Diagramas de Flujo de Datos: Forma Incorrecta y Correcta

2.22.2. Diagrama de Flujo de Control (DFC)

Existe una extensión para los Diagramas de Flujo de Datos denominados Diagramas de Flujo de Control (DFC). Estos diagramas se utilizan cuando se requiere modelar un sistema en tiempo real como por ejemplo el software para controlar el tráfico de un sistema de vuelo o el software para monitorear el ritmo cardiaco.

En los DFC se ocupan dos símbolos adicionales a los DFD: las burbujas punteadas y las flechas punteadas.

Una burbuja punteada representa un proceso de control que tiene como propósito coordinar a otros procesos que no son de control. Lo único que puede entrar o salir de un proceso de control son flujos de control.

Las flechas punteadas representan flujos de control que tienen como propósito indicarle a otro proceso algún estado o situación que requiera conocer. Cabe destacar que son los únicos flujos que van a otro proceso. Las flechas de entrada al proceso de control indican que el proceso ha terminado o que se ha presentado alguna situación excepcional. Las flechas de salida del proceso de control indican otro proceso que debe iniciar.

2.22.3. Diagrama de Transición de Estados (DTE)

Otro diagrama que se utiliza para modelar el aspecto dinámico interno del sistema es el Diagrama de Transición de Estados. En este diagrama se busca representar los diferentes estados que puede tener una entidad del Diagrama de Entidad-Relación. La simbología que se utiliza es:

Nombre	Descripción
Rectángulo (Estado) 	Representa un estado que puede tener la entidad, dado por un evento de información o de tiempo. Dentro del rectángulo se debe poner el nombre del estado, una buena práctica es poner el nombre de la entidad seguida de su estado para una mejor comprensión
Rectángulo vacío (Estado final) 	Representa un estado final, es decir, es cuando la entidad deja de existir. En un ambiente de producción significa que el registro u objeto es borrado físicamente por lo que no podrá ser recuperado.
Flechas (Evento de transición) 	Representan los eventos de información o eventos de tiempo que provocan un cambio de estado en la entidad. Las flechas están acompañadas del nombre del evento que cambia el estado de la entidad. Hay tres clasificadores para los eventos: **C** para cuando el evento *crea* la entidad. **M** para cuando el evento modifica o actualiza el estado de la entidad. **E** para cuando el evento elimina la entidad, lo cual implica que la entidad ya no existirá en la base de datos.

Cuadro 2.6. Símbolos del Diagrama de Transición de Estados

Cabe recalcar que los estados de las entidades están conformados por los valores que tienen las entidades en un momento dado.

En el ejemplo de la figura 2.9 se muestran los diferentes estados que atraviesa la entidad RENTA, al principio, por el evento “El cliente se presenta en el mostrador con las películas que quiere rentar” se crea la entidad RENTA y tiene el estado de “Realizada”. Otro evento “El cliente agrega otra película a su renta” modifica el estado de la RENTA y el evento “El responsable del Videoclub actualiza la renta” lo regresa a su estado original. La entidad RENTA puede

pasar al estado de “Cancelada” por el evento “El cliente cancela su renta” o por el evento “El responsable del Videoclub cancela la renta por sanción del cliente” en este caso vemos que es un estado final. Y por último vemos el evento “Es tiempo de depurar los registros de hace 10 años” que genera un estado especial, que es un rectángulo vacío lo cual significa que la entidad es destruida físicamente de la base de datos.

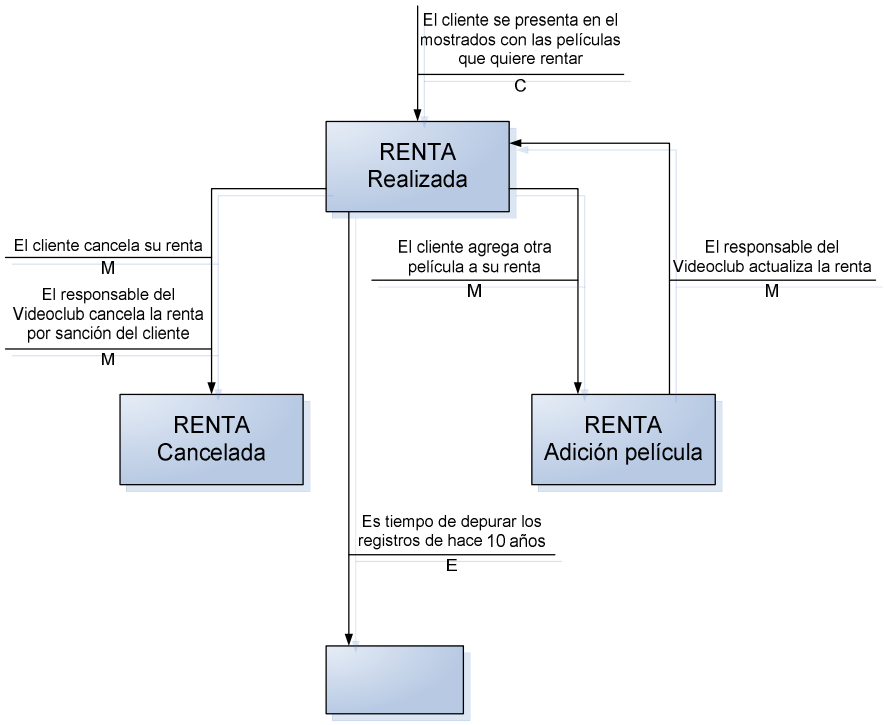


Figura 2.9. Diagrama de Transición de Estados. Elaboración propia

2.23. Diccionario de datos

Los diagramas utilizados para modelar los aspectos dinámicos y estáticos del sistema no son suficientes para que diseñadores y programadores puedan continuar con su tarea. Si el analista solo considera el nombre de los flujos de datos dejaremos a su imaginación lo que contienen, causando problemas en el diseño y las fases subsecuentes. Cada flecha de un DFD representa uno o más elementos de información. Por lo tanto, el analista debe disponer de algún otro

método para representar el contenido de cada flecha de un DFD. El diccionario de datos es una herramienta que define una gramática para describir el contenido de los elementos de información:

Carácter(es)	Descripción
=	está compuesto de
+	y
( )	optativo (puede estar presente o ausente)
{ }	iteración
[ ]	seleccionar una de varias alternativas
* *	comentario
@	identificador (campo clave) para un almacén
|	separa opciones alternativas en la construcción

Cuadro 2.7. Símbolos del Diccionario de Datos

Un ejemplo tomando como base el ejemplo del videoclub:

renta = membresía + nombre + (segundo nombre) + apellido paterno + apellido materno
título de cortesía = [Oro | Plata | Bronce]
nombre = {carácter legal}
apellido paterno = {carácter legal}
apellido materno = {carácter legal}
carácter legal = {A-Za-z}

El diccionario de datos debe contener, en lo medida de lo posible y lo valioso, las definiciones de todos los datos mencionados en el DFD. Los datos compuestos (datos que pueden ser divididos como el nombre completo de una persona) se definen en términos de sus componentes; los datos elementales (datos que no pueden ser divididos) se definen en términos del significado de cada uno de los valores que puede asumir.

El diccionario de datos es un listado organizado de los datos pertinentes al sistema, con definiciones exentas de ambigüedades para que tanto el usuario como el analista tengan un entendimiento común de todas las entradas, salidas, componentes de los almacenes.

El diccionario de datos no solo sirve para especificar los flujos de datos, también sirve para especificar cada almacén contenido en los DFD. Al realizar esta actividad se especifica las entidades que conforman el diagrama de entidad-relación. ¿Por qué? Porque a cada almacén debe corresponder una entidad en el DER.

2.24. Prototipos del software

En análisis debe ser conducido independientemente del paradigma de ingeniería de software aplicado. Sin embargo, la forma que ese análisis tomará puede variar. En algunos casos es posible aplicar los principios de análisis fundamental y derivar a una especificación en papel del software desde el cual pueda desarrollarse un diseño. En otras situaciones, se va a una recolección de los requerimientos, se aplican los principios de análisis y se construye un modelo de software, llamado un prototipo, según las apreciaciones del cliente y del que lo desarrolla. Finalmente, hay circunstancias que requieren la construcción de un prototipo al comienzo del análisis, puesto que el modelo es el único mediante el que los requerimientos pueden ser derivados efectivamente.

2.25. Participantes en el análisis estructurado

En la etapa de análisis se requiere la participación activa de los clientes, usuarios, analistas, y líder de proyecto. El objetivo es delimitar el alcance del sistema, identificar el problema y sus causas, así como identificar los requerimientos de cada participante.

Anteriormente, se creía que no era necesario que participaran diseñadores, programadores, administradores de base de datos o cualquier otro rol que interviene directamente en las etapas subsecuentes a la de análisis. Con el tiempo esto ha cambiado, estos han tomado gran relevancia cuando se hace el análisis del sistema sobre todo en el aspecto del estudio de factibilidad técnica. Sus conocimientos y experiencia aportan elementos valiosos para determinar si se tiene o no la capacidad de desarrollar el sistema.

2.26. Perfil del analista de sistemas

De las habilidades y conocimientos que se espera de un analista tenemos:

- Capacidad de abstracción.
- Capacidad de síntesis
- Capacidad de comunicación.
- Disciplinado.
- Organizado.
- Ser meticuloso, incluso con los pequeños detalles cuando la situación lo amerite.
- Comprensión de la cultura organización y su impacto sobre los sistemas de información.
- Identificación de problemas y sus causas.
- Administración de los requerimientos.
- Modelado del aspecto dinámico y estático del sistema actual.

2.27. Funciones y responsabilidades del analista de sistemas

- Planear las actividades de análisis de sistemas.
- Escoger (o diseñar) y utilizar los métodos, técnicas y herramientas más adecuadas para el análisis del sistema.

- Estudiar el sistema de planeación, organización, dirección y control de la empresa.
- Representar con algún tipo de especificación los procesos que se realizan en la organización que estén relacionados con el sistema por desarrollar.
- Modelar los aspectos dinámicos y estáticos del sistema actual.
- Proponer y aplicar las medidas de carácter organizativo que se requiera para perfeccionar los procesos.
- Revisar los resultados obtenidos por los programas elaborados por los programadores.
- Elaborar los datos de prueba para comprobar la calidad de los programas, individualmente y en su conjunto.
- Capacitar a los usuarios en la operación del sistema.

2.28. Planes y estrategias cuando no se aplica la administración de proyectos

En organizaciones en donde no se tiene la capacidad o el interés para administrar proyectos de desarrollo de sistemas aumenta las probabilidades de fracaso. Por esta razón es necesario tomar en cuenta los siguientes lineamientos que pueden ser considerados como estrategias:

- Identificar a todas las personas que tengan que decir algo sobre el sistema e invitarlas a participar activamente en las decisiones concernientes en la construcción del sistema.
- Construir los artefactos que sólo ayuden a entender los procesos que se pretenden sistematizar.
- Invitar a los usuarios a que revisen las especificaciones de los procesos que realice el equipo el desarrollo.
- Mantener una comunicación constante con los clientes y usuarios ante cualquier situación que requiera su intervención.

- Crear una bitácora con el trabajo realizado por el equipo de desarrollo.

2.29. Estudio de factibilidad y análisis costo-beneficio

Antes de iniciar cualquier desarrollo de sistemas es necesario evaluar si se tiene la capacidad para realizarlo. Por lo regular son tres los aspectos que se deben contemplar: el económico, el operativo, y el técnico. Según las características de la organización se pueden aplicar todas, alguna o ninguna. Por ejemplo, en una organización que contrata a su personal por proyecto, es decir, bajo determinado tiempo le es indispensable hacer su estudio de factibilidad económico. Para una organización que tiene a su personal de planta y que no está dispuesta a invertir más, se enfocará más en los estudios de factibilidad operativo y/o técnico.

Todo proyecto busca algún tipo de beneficio, sobre todo el económico. ¿El software que se va a construir tiene un costo-beneficio aceptable? Es una de las preguntas que siempre se hacen los dueños y/o gerentes de las organizaciones. Los recursos son limitados por lo que no se puede emprender un proyecto sin tener cierta certidumbre que se obtendrá un beneficio con el resultado del proyecto.

¿Quiénes utilizan los sistemas? Las personas, mejor conocidas como usuarios. Ellas son las encargadas de operar el software que se construya, por lo tanto es necesario evaluar sus habilidades en el uso de software de computadoras. Esta evaluación no es sencilla sobre todo porque las personas son resistentes a que se les establezcan indicadores de medición por lo que no hay un método de evaluación único y absoluto. Se hace necesario ser lo suficientemente empáticos y asertivos para llevar a cabo esta evaluación.

Con ello lo que prosigue es preguntar: ¿Las personas ocupan la computadora en sus actividades diarias? ¿Suena absurda la pregunta, en pleno siglo XXI y

pensar que alguien no ocupa una computadora? Pues la respuesta es sí, hay muchas personas que no han tenido la necesidad o han rechazado ese tipo de cambio.

Si la respuesta es no, tenemos un serio problema, y se hace indispensable llevar a cabo un proyecto de capacitación. Si la respuesta es sí es necesario conocer qué hacen con las computadoras para determinar si tienen los conocimientos necesarios para el uso del futuro sistema.

¿Qué tipo de software se va a desarrollar? ¿Un sistema cliente-servidor? ¿Un sistema monousuario? Las habilidades que se requieren para desarrollar un software con Visual basic o Delphi difieren en cuanto a las habilidades que se requieren para desarrollar un software que trabajará sobre Internet por la WWW desarrollada con Java o PHP. Si el equipo de trabajo cuenta con los conocimientos y experiencia sobre las metodologías, herramientas CASE, lenguajes de programación, sistema administrador de base de datos, etcétera que se ocuparán para que el proyecto tenga más probabilidades de éxito.

Sin embargo, en caso contrario de que no se cuente con el conocimiento y experiencia, se tendrá que evaluar si vale la pena capacitar al personal para que adquiera los conocimientos o en su caso contratar a personas que sí cuenten con ese conocimiento. En ambos casos, estas alternativas afectarán la factibilidad económica ya que se tendrá que invertir tiempo, dinero y esfuerzo: en el primer caso para capacitar al personal y en el segundo para contratar al personal.

¿La complejidad del sistema es baja o alta? Ningún software es igual aunque se trate de organizaciones que se encuentren en el mismo ramo, incluso que generen el mismo producto. Hay que evaluar si el equipo de trabajo ha enfrentando problemas similares, esto ayudará mucho a asimilar la problemática y darle la solución adecuada, pero si el equipo de trabajo nunca se

ha enfrentado a este tipo de problemas será necesario más tiempo para entender la complejidad de la situación y darle solución.

¿Se cuenta con la tecnología necesaria para desarrollar el software? Es necesario tener los conocimiento y experiencias pero también es necesario contar con la tecnología necesaria para construir el software y desde luego para implantarlo. Cuando mencionamos tecnología nos estamos refiriendo tanto a los procesos como al equipo. En los procesos encontramos las metodologías, estándares, y técnicas. En el equipo encontramos el hardware y el software.

Bibliografía del Tema 2

Braude J., Eric, Ingeniería de Software: *Una Perspectiva Orientada a Objetos*, México, AlfaOmega, Traducción: Marcia González Osuna, 2003.

Lawrence Pfleeger, Shari, *Ingeniería del Software. Teoría y Práctica*, Prentice Hall / Pearson Educación, Traducción: Elvira Quiroga, 2002.

Piattini Velthuis, Mario G., et al, *Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*, México, Alfa omega / Ra-Ma, 2000.

Pressman, S. Roger, *Ingeniería del Software. Un Enfoque Práctico*, 5ª edición, Adaptación de Darle Ince, Traducción de Rafael Ojeda Martín (et al.), Madrid, McGraw Hill, 2002.

_____, *Ingeniería del Software. Un Enfoque Práctico*, 6ª Edición, México, McGraw Hill, Traducción: Jesús Elmer Murrieta Murrieta y otros, 2005.

Sommerville, Ian, *Ingeniería de Software*, 6ª edición, México, Addison Wesley / Pearson Educación, Traducción: José Alejandro Domínguez Torres, 2002.

Yourdon, Edward, *Análisis Estructurado Moderno*, Traducción: Alexandra Taylor Amitage, México, Prentice Hall / Pearson Educación, 1993.

Actividades de aprendizaje

A.2.1. Investiga en Internet por medio de www.google.com ejemplos de especificaciones de casos de uso (al menos 5). Identifica qué elementos se repiten en las especificaciones. Con los elementos filtrados crea un formato en tu procesador de textos para especificar casos de uso, en cada sección explica qué información se debe poner.

A.2.2. Investiga en tu escuela qué trámites escolares puedes hacer (identifica quiénes participan en los procesos). Con la información obtenida crea un diagrama de contexto que modele el sistema actual.

A.2.3. Cuando te inscribes a tus asignaturas o grupos tu inscripción pasa por diferentes estados. Investiga cuáles son esos estados con el área de trámites escolares y crea un diagrama de estados.

Cuestionario de autoevaluación

1. Explica qué es un requerimiento.
2. Explica qué es un requerimiento de tipo funcional.
3. Explica en qué consiste la cualidad de confiabilidad del software como requerimiento.
4. Menciona y explica las secciones de una especificación de casos de uso.
5. Menciona y explica dos técnicas para la especificación de requerimientos.
6. Dibuja y explica para qué sirve cada uno de los símbolos del diagrama de contexto.

7. Dibuja y explica para qué sirve cada uno de los símbolos del diagrama de flujo de datos.
8. Dibuja y explica para qué sirve cada uno de los símbolos del diagrama de entidad-relación.
9. Menciona cinco habilidades o conocimientos de un analista de sistemas.
10. Explica qué es un prototipo de software.

Examen de autoevaluación

1.- En esta técnica de recopilación de información que hace uso, principalmente, de preguntas cerradas.

A. Entrevista.
B. Método delphi.
C. Cuestionario.
D. Observación.

2.- En este tipo de requerimiento funcional se solicita características como: rapidez, disponibilidad, certeza, tiempo de respuesta, tiempo de recuperación.

A. Usabilidad
B. Confiabilidad
C. Soporte
D. Desempeño

3.- Esta técnica de recopilación de requerimientos consiste en un diagrama que representa en forma secuencial condiciones y acciones.

A. Español estructurado.
B. Lista de verificación.
C. Tabla decisión.
D. Árbol decisión.

4.- Un sistema es _____ si se comporta de acuerdo con la especificación de los requerimientos funcionales que debería proveer.

A. Confiable

B. Correcto

C. Robusto

D. Eficiente

5.- Un sistema es _____ si un usuario humano lo encuentra fácil de utilizar.

A. Amigable.

B. Verificable.

C. Confiable.

D. Portable.

6.- Un sistema es _____ si permite la corrección de sus defectos con una carga limitada de trabajo.

A. Mantenible.

B. Reparable.

C. Evolucionable.

D. Adaptable.

7.- El sistema es _____ si puede ser ejecutado en distintos ambientes, refiriéndose este último tanto a plataformas de hardware.

A. Amigable.

B. Verificable.

C. Confiable.

D. Portable.

8.- En esta sección de caso de uso se ponen las acciones que hacen tanto el sistema como el actor.

A. Nombre.

B. Descripción.

C. Flujo principal.

D. Excepciones.

9.- En esta sección de una especificación de caso de uso se describen aquellos eventos o condiciones que no tienen relación directa con las reglas de negocio que le competen al caso de uso.

A. Nombre.

B. Descripción.

C. Flujo principal.

D. Excepciones.

10.- En este diagrama se modela el ambiente externo del sistema donde se identifican entidades y flujos de información principalmente.

A. Diagrama de casos de usos.

B. Diagrama de flujo de datos.

C. Diagrama de contexto.

D. Diagrama de entidad-relación.

TEMA 3. DISEÑO ESTRUCTURADO

Objetivo particular

Al finalizar el tema identificarás:

- los conceptos del diseño estructurado,
- las arquitecturas para representar el diseño de un sistema,
- las técnicas para especificar el diseño de los procesos, y
- los principios que se deben tomar en cuenta para diseñar una interfaz gráfica.

Temario detallado

3.1. ¿Qué es el diseño estructurado?

3.2. Principios del diseño estructurado

3.3. Conceptos del diseño estructurado

- 3.3.1. Abstracción
- 3.3.2. Refinamiento
- 3.3.3. Modularidad
- 3.3.4. Independencia funcional
- 3.3.5. Cohesión
- 3.3.6. Acoplamiento
- 3.3.7. Arquitectura de software
- 3.3.8. Jerarquía de control
- 3.3.9. División estructural
- 3.3.10. Estructura de datos
- 3.3.11. Procedimiento
- 3.3.12. Ocultamiento de información
- 3.3.13. Concurrencia
- 3.3.14. Verificación

3.4. Diseño arquitectónico

Introducción

Terminado o durante el proceso de análisis se inicia el proceso de diseño del sistema. El objetivo del diseño es crear una especificación del sistema propuesto para dar solución a los problemas y necesidades identificados en el análisis. Si bien es cierto que se puede iniciar con la programación del software y la construcción de la base de datos sin hacer diseño, así se está cayendo en uno de los errores más comunes en el desarrollo de sistemas. Omitir la fase de diseño implica que cada programador tome la decisión de cuál es la mejor manera de resolver el problema ocasionando con ello que se haga mal los programas, se tenga que volver a hacer el trabajo y al final no se pueda integrar las partes del sistema.

En el diseño se toma las decisiones concernientes a la organización de los componentes y la estructura de cada uno de ellos. En este tema identificarás los modelos que se toman para diseñar la arquitectura de los sistemas así como las técnicas para especificar los componentes del sistema.

3.1. ¿Qué es el diseño estructurado?

El diseño es la etapa del ciclo de vida de sistemas que tiene como objetivo especificar la solución con base en los problemas y requerimientos detectados en la fase de análisis. En el diseño se requiere hacer uso de la creatividad para construir la solución y está delimitada por los recursos con los que cuenta la organización para desarrollar el sistema.

3.2. Principios del diseño estructurado

Los principios de diseño estructurado son:

- Descomposición por refinamientos sucesivos.
- Creación de una jerarquía modular.
- Elaboración de módulos Independientes.

3.3. Conceptos del diseño estructurado

El diseño de sistemas se caracteriza por hacer uso de un conjunto de conceptos que deben ser comprendidos a fin de poder especificar una propuesta de solución exitosa.

3.3.1. Abstracción

Es el proceso por el cual se resaltan ciertas características y se descartan o minimizan ciertas características de la realidad. Para el diseño, este proceso está enfocado en construir la solución, a diferencia del análisis en donde se busca especificar la situación actual.

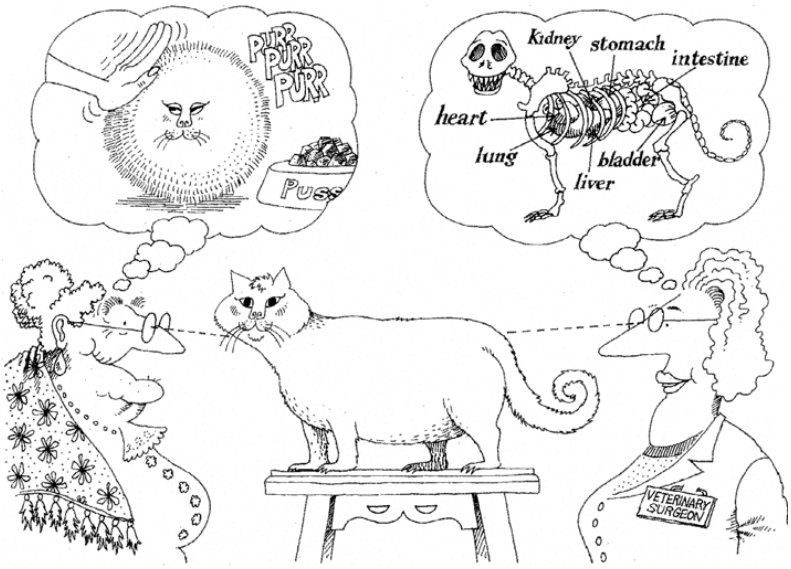


Figura 3.1. Abstracción (Booch, 2007, p. 45)

El concepto de abstracción tiene relevancia en el sentido en que los intereses, gustos, conocimientos, etc. afectan la forma en que percibimos la realidad.

3.3.2. Refinamiento

Es el proceso por el que se refinan los procesos por automatizar en su mínima expresión, que son las actividades. Como en el diseño la materia prima son los

algoritmos, las actividades deben ser transformadas en algoritmos. Estos algoritmos deben ser especificados de acuerdo con el lenguaje de programación que se haya seleccionado debido a las singularidades que tiene cada lenguaje.

No solamente se debe refinar los algoritmos, también se tiene que refinar los datos. Aunque esta tarea se hace en el análisis cuando se crea el Diccionario de Datos se presenta la misma situación que con los algoritmos, es necesario refinarlos en función del Sistema Manejador de Datos seleccionado.

3.3.3. Modularidad

Es el proceso por el que se agrupan y/o se dividen los componentes de un sistema. Los componentes están constituidos por un conjunto de programas. La forma de hacer la agrupación está en función de la arquitectura que se haya seleccionado así como de los estándares que maneje el equipo de desarrollo y la organización a la que se está construyendo el sistema.

La ventaja de la modularización es que permite mantener y reutilizar los componentes del sistema siempre y cuando se haya tenido una alta cohesión y un bajo acoplamiento como se mencionó en el tema 1.

Figura 3.2. Modularidad, (Booch, 2007, p.54)

En el libro *Análisis y Diseño Orientado a Objetos con Aplicaciones* de Booch (2007) se ilustra el concepto de modularidad con la anterior figura 3.2. Observa que cada componente que conforma el gato electrónico puede ser sustituido por otro en caso de que se descomponga.

3.3.4. Independencia funcional

La independencia funcional es el resultado de aplicar la modularidad en el diseño de sistema. Sin embargo, esta afirmación puede resultar un tanto falsa ya que cada elemento del sistema se encuentra interrelacionado con por lo menos otro elemento. Dado un cambio en el comportamiento de un elemento, este afectará en cierto grado directa o indirectamente a otro elemento por lo que la independencia tiende a ser virtual.

3.3.5. Cohesión

La **cohesión** es la especificidad de la función que realiza un elemento. Pressman lo define como:

> [U]na extensión natural del concepto de ocultación de información… Un módulo cohesivo lleva a cabo una sola tarea dentro de un procedimiento de software, lo cual requiere poca interacción con los procedimientos que se llevan a cabo en otras partes de un programa. Dicho de otra manera sencilla, un módulo cohesivo deberá (idealmente) hacer una sola cosa.[9]

3.3.6. Acoplamiento

El acoplamiento es nivel de interdependencia que existe entre los elementos. Pressman lo define como:

> [U]na medida de interconexión entre módulos dentro de una estructura de software. El acoplamiento depende de la complejidad de interconexión entre los módulos, el punto donde se realiza una entrada

[9] Roger S. Pressman, *Ingeniería del Software: Un Enfoque Práctico*, 5ª ed., Madrid, McGraw Hill, 2001, p 231.

o referencia a un módulo, y los datos que pasan a través de una interfaz.[10]

3.3.7. Arquitectura del software

"La arquitectura es la estructura jerárquica de los componentes del programa (módulos), la manera en que los componentes interactúan y la estructura de datos que van a utilizar los componentes."[11] La arquitectura la establece el diseñador, obedeciendo a las características que debe tener el sistema y a la tecnología que se va a emplear, por lo que no es de extrañarse que exista una diversidad de formas de crear la arquitectura de un sistema.

Otra forma de plantear la arquitectura de software es creando un conjunto de modelos que representen la parte estática y la parte dinámica del sistema. Los modelos se representan por medio de diagramas y sus especificaciones correspondientes.

3.3.8. Jerarquía de control

La jerarquía representa la organización de los componentes del sistema instaurando una jerarquía en donde se establece qué componente controla a qué componente. Para este concepto es necesario establecer una regla: un componente solo puede ser controlado por otro componente.

La jerarquía no es otra cosa que la forma en que se agrupan los componentes. La agrupación obedece a las características que comparten cada uno de los componentes buscando con ello una administración eficiente. No hay un límite para establecer cuántos componentes puede controlar un componente pero se recomienda que no sea mayor de 7.

---

[10] loc. cit.
[11] Ibid, p. 226.

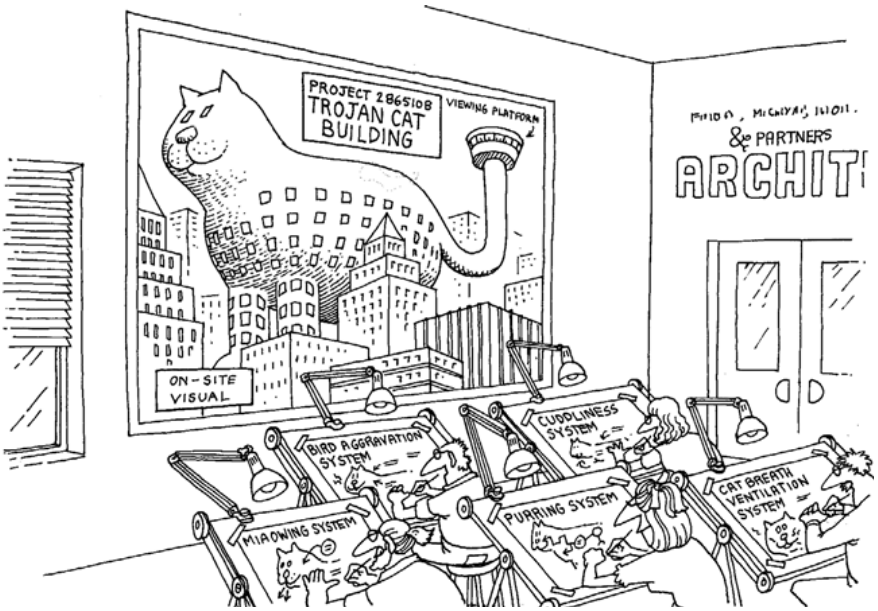


Figura 3.3. Jerarquía de Control (Booch, 2007), fig. 13

Observa en la figura 3.3 cómo el grupo de diseñadores está especificando cada una de las partes del gato electrónico. No es al azar que se encuentren en un mismo cuarto, como los componentes se encuentran interrelacionados entre sí es necesario que el equipo de trabajo tenga una comunicación directa con la finalidad de compartir las interfaces que hay en los componentes.

3.3.9. División estructural

La división estructural es un concepto inherente al concepto de jerarquía de control. Como se mencionó, es en la jerarquía de control como se organizan los componentes agrupándose entre sí. Estas agrupaciones se dan en forma horizontal (ancho) y en forma vertical (profundidad).

3.3.10. Estructura de datos

La estructura de datos es una relación entre datos. Conforme a los requerimientos del usuario, éste establecerá ciertos conceptos que tienden a convertirse en estructuras de datos. Por ejemplo, el usuario menciona que

“renta películas”, la renta es una estructura de datos que está compuesta por una fecha de renta, fecha de devolución, películas rentadas, cliente que renta, entre otros.

3.3.11. Procedimiento

El procedimiento es la organización mínima de un conjunto de actividades. Se especifica por uno o más algoritmos dependiendo de la complejidad que tenga el procedimiento. Cada procedimiento debe definir una secuencia lógica de las actividades así como los diversos caminos que pueden tomar las actividades.

3.3.12. Ocultamiento de información

Cuando se crea un componente es necesario que tenga una cohesión alta, esto quiere decir que sólo debe llevar a cabo una sola tarea en la medida de lo posible. La tarea requiere de manipular datos para llevar a cabo su cometido por lo que el componente tiene la responsabilidad de que estos datos sólo sean visibles para su comportamiento. Esta característica es lo que se conoce como ocultamiento de la información.

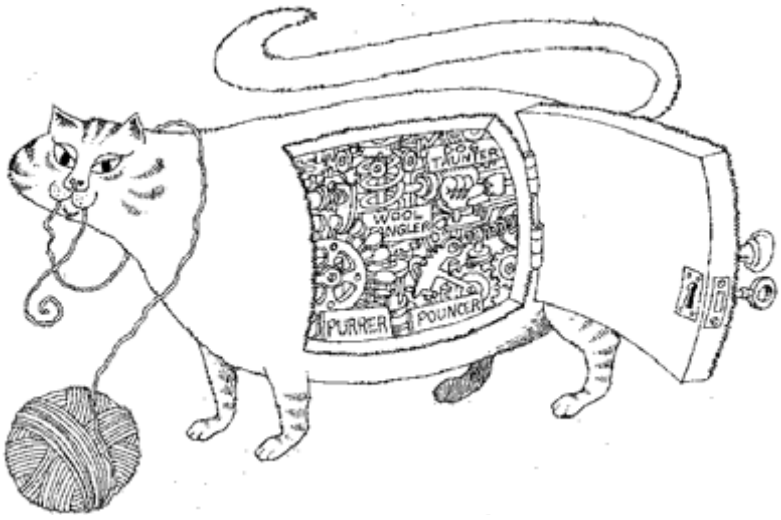


Figura 3.4. Ocultamiento (Booch, 2007, fig. 51)

El ocultamiento de información es una característica muy apreciada por los usuarios y desarrolladores. A los usuarios no les interesa los detalles técnicos de los sistemas pero sí les importa que dé la respuesta esperada. A los desarrolladores les interesa ocultar los detalles técnicos de los componentes con el fin de que se facilite su reutilización.

3.3.13. Concurrencia

Esta propiedad del software se da cuando tiene que responder a las peticiones de los usuarios al mismo tiempo. En entornos multiusuario tiene un peso mayor por lo que será necesario establecer pruebas de rendimiento que permitan validar esta característica.

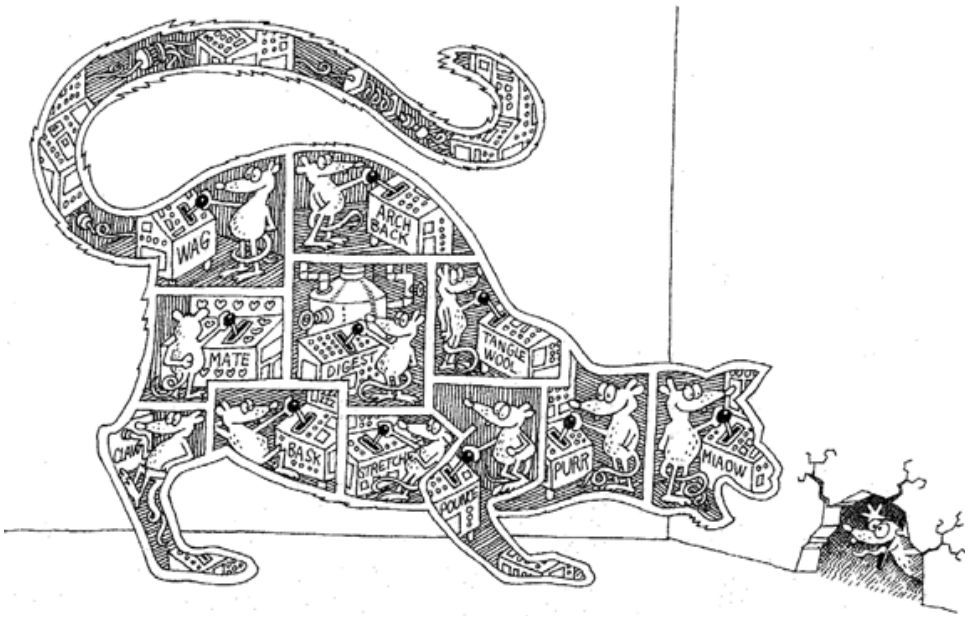


Figura 3.5. Concurrencia (Booch, 2007, p. 67)

3.3.14. Verificación

Es un proceso en que se revisa que el software se construya de la manera correcta. Al iniciar el desarrollo de un sistema es necesario establecer un conjunto de estándares para los documentos, algoritmos, base de datos y

cualquier otro tipo de artefacto. La verificación se encarga de asegurar que conforme a los estándares definidos los productos los cumplan, este proceso busca asegurar la calidad interna del software.

3.4. Diseño arquitectónico

El diseño arquitectónico busca especificar una solución con base en el paradigma que se esté utilizando; en este caso es el estructurado. El diseño arquitectónico tiene como objetivo definir la estructura de los programas y los datos que conformarán el software.

En este sentido, la estructura define la organización de cada uno de los componentes a fin de que respondan a los requerimientos de los usuarios. Esta condición amerita que antes de construir los programas y los datos primero se tiene que especificar los algoritmos, especificar la base de datos pero sobre todo cómo se organizarán y comunicarán los componentes.

3.4.1. Arquitectura del software

La arquitectura de software establece la organización de los componentes a fin de constituir el sistema que se está solicitando. Al igual que un arquitecto, el diseñador tiene que modelar diferentes aspectos que conforman un sistema. El arquitecto no es el responsable de instalar la tubería de un edificio o de instalar la electricidad de una casa pero sí es responsable de crear los modelos para un plomero y un electricista que sí lo hacen.

Con el ejemplo anterior queda claro que el diseñador construirá una diversidad de modelos a fin de que programadores, diseñadores gráficos y administradores de base de datos puedan realizar sus actividades para la construcción del sistema.

También, al igual que el arquitecto, el diseñador tendrá que generar diferentes versiones de los modelos a fin de que satisfagan los requerimientos para que usuarios y clientes estén satisfechos.

3.4.2. Diseño de datos

En el análisis se elaboró un diagrama de entidad-relación considerando los datos que maneja actualmente la organización tenga o no tenga un software. En el diseño se retoma ese diagrama para completarlo y especificarlo de acuerdo con el Sistema Administrador de Base Datos que se seleccionó.

3.4.2.1. Modelo físico de datos (Modelo relacional)

El modelo físico de datos se crea por medio de un diagrama de entidad-relación físico. Este diagrama se basa en el modelo relacional propuesto por Codd en 1970, sustentado en la teoría de conjunto en donde a cada conjunto se le denomina relación. Cada relación agrupa objetos de la realidad que comparten características tal como lo indica la teoría de conjuntos. En el diagrama entidad-relación a las relaciones se les denomina tablas y a la interdependencia entre las tablas interrelaciones.

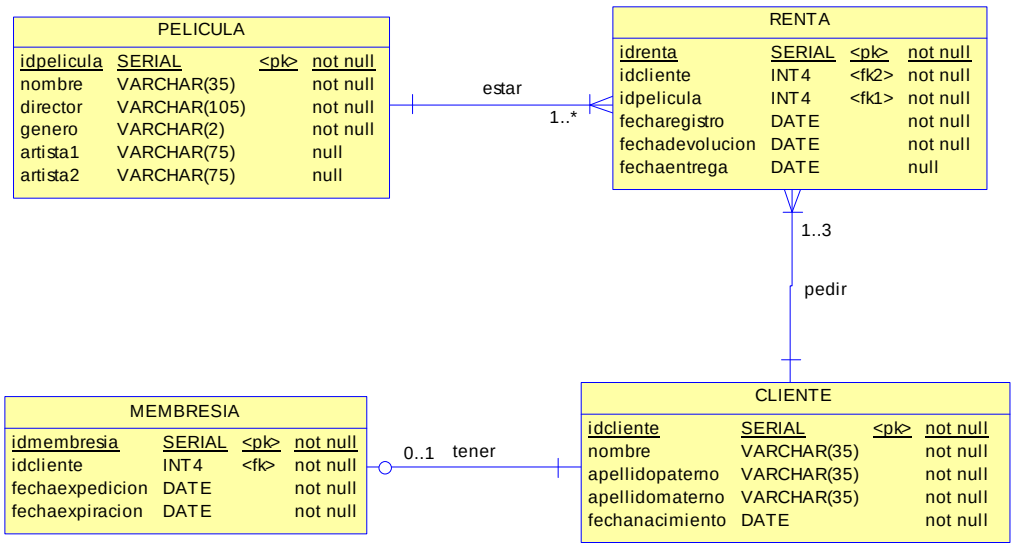


Figura 3.6. Diagrama de entidad-relación físico

3.4.2.2. Estructura de Datos

Las estructuras de datos son abstracciones de bajo nivel que agrupan un conjunto de datos que están interrelacionados entre sí. Adicional a esto, la estructura de datos define un conjunto de operaciones que se pueden realizar con los datos que contiene. Hay diferentes tipos de estructuras: pilas, colas, árboles, listas, etc.

3.4.2.3. Datos a nivel de componentes

Es la representación de las estructuras de datos a las cuales accede uno o más componentes de sistemas. Existe un conjunto de principios definidos por Wasserman que se debe considerar en el momento de diseñar las estructuras de datos:

1. El tiempo que se le dedica al análisis de las funciones y comportamiento del sistema debe ser proporcional al de los datos. Esto significa que en el momento en que se especifican los flujos de datos por medio del diccionario de datos deben ser completos y correctos.
2. Las estructuras de datos así como las operaciones que se ejecutan en ellas deben estar claramente definidas.
3. Se debe crear un diccionario de datos para describir cada dato de la que está compuesto la estructura de datos.
4. La estructura de datos sólo debe ser conocida por los componentes que las usarán.
5. Se debe desarrollar una biblioteca donde se describan las estructuras de datos y sus operaciones relacionadas.
6. El lenguaje de programación seleccionado debe soportar la especificación de la estructura de datos.

Hay que tener cuidado con estos principios porque no todo el software hace uso de estructuras de datos como se conocían en los años ochenta y noventa. Hoy

en día, esas estructuras de datos se crean y manipulan en lo que son las tablas que conforman una base de datos.

3.4.3. Estilos arquitectónicos

Un estilo arquitectónico establece un marco de referencia para crear el diseño. Como tal establece una serie de reglas y recomendaciones que se deben seguir para establecer la organización de los componentes del sistema. El estilo arquitectónico no es una receta de cocina que deba seguirse al pie de la letra por la razón de que la variable Persona crea un alto nivel de volatilidad en la construcción del sistema, en consecuencia se recomienda que el estilo que se elija deba ser flexible y a la vez robusto.

Arquitectura centrada en los datos.- En esta arquitectura, como su nombre lo indica, las decisiones de diseño están orientadas a la centralización de los datos. En este estilo, el software accede a un almacén centralizado de los datos para agregar, eliminar, modificar y/o recuperar alguno de los datos contenidos en él. La ventaja de este modelo consiste en la independencia de los datos, es decir, el software debe estar construido de tal manera que si uno de sus componentes es sustituido no se verá afectado el almacén de datos.

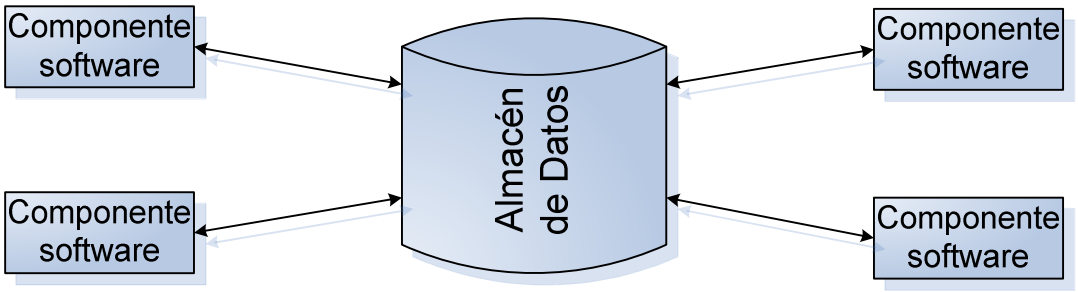


Figura 3.7. Arquitectura centrada en los datos

Arquitectura de flujo de datos. Esta arquitectura se centra en la transformación de los datos de entrada para obtener los datos de salida. Para

lograr esto se utiliza los componentes del software como tuberías, estas tuberías están interconectadas entre sí trabajando en forma independiente. Cada tubería solo está consciente de que debe recibir los datos de cierta manera y generar una salida para la siguiente tubería, esto significa que cada tubería desconoce la forma en que trabajan las demás, sólo conoce que debe recibir la otra tubería si es que están interconectadas.

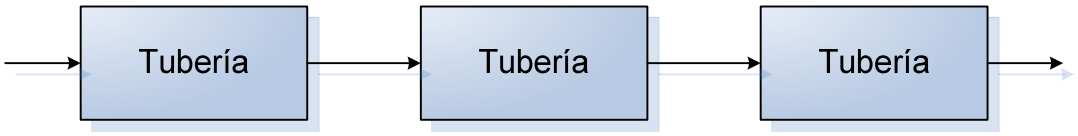


Figura 3.8. Arquitectura de flujo de datos

Arquitectura estratificada. En esta arquitectura se crea una cantidad definida de capas, cada capa tiene una responsabilidad claramente definida. Las capas externas se orientan al uso del sistema y las capas internas se orientan a la manipulación de la computadora. Dependiendo de cómo se aplique el estilo puede tener dos a más capas, de manera tradicional se utilizan tres capas: la primera tiene como responsabilidad la presentación de los datos al usuario, la segunda tiene como responsabilidad ejecutar la lógica de negocio, y la tercera tiene como responsabilidad la manipulación de los datos.

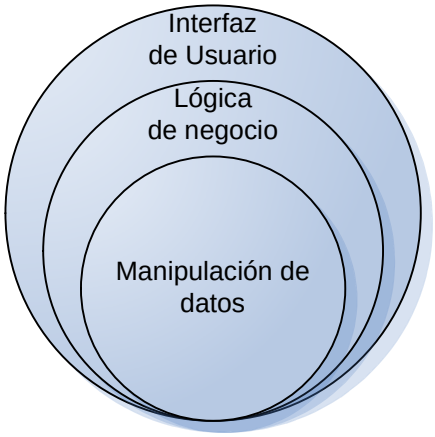


Figura 3.9. Arquitectura estratificada

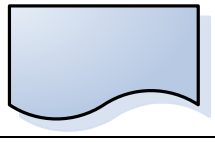
Es necesario recalcar que estos estilos en la realidad son meramente teóricos y que más bien son utilizados en forma combinada, dada las condiciones cambiantes en las que se desenvuelve el desarrollo de sistemas. Incluso, el diseñador se puede encontrar en la situación de que ninguno de ellos le ayude a resolver problema y tenga que elaborar un estilo propio.

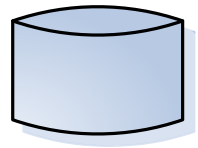
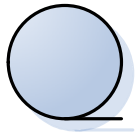
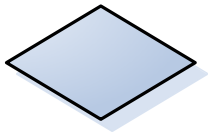
3.5. Modelado de procesos

La forma más común de modelar procesos en el análisis y diseño estructurado son los Diagramas de Flujos de Datos. Sin embargo, en ocasiones es necesario apoyarse de otras herramientas como los diagramas: HIPO, Nassi-Schneiderman, Warnier/Orr. Aunque actualmente ya se encuentran en desuso es necesario conocerlos para cuando se tenga que actualizar un sistema que esté documentado con estos tipos de diagramas.

3.5.1. Diagramas HIPO

El acrónimo HIPO corresponde a *Hierarchy-Input-Process-Output* que podría traducirse como Jerarquía de entrada-proceso-salida. Los diagramas HIPO buscan especificar un proceso con base en su entrada, el proceso para procesar las entradas en la salida, y la salida. Los símbolos que se utilizan en este tipo de diagramas son:

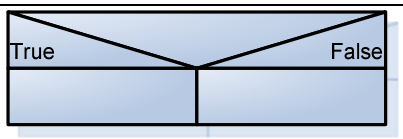
Símbolo	Descripción
	Programa.- Es un actividad que es llevada a cabo por el sistema.
	Listado.- Es un conjunto de registros del mismo tipo.
	Utilidad o rutina.- Es una actividad en su mínima expresión, es decir, ya no se puede fragmentar otras actividades.

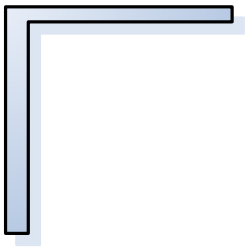
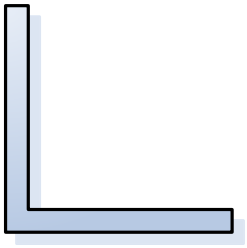
	Disco.- Representa un medio de almacenamiento en donde se guardan datos para su posterior recuperación.
	Cinta magnética.- Representa un medio de almacenamiento electromagnético.
	Utilidad *sort*.- Representa una actividad de ordenamiento.

Cuadro 3.1. Símbolos del diagrama HIPO

3.5.2. Diagramas Nassi-Schneiderman

Los diagramas Nassi-Schneiderman son similares a un diagrama de flujo pero omite las flechas que representan el flujo de control. En lugar de las flechas se ocupan rectángulos colocados en forma sucesiva, dentro de cada rectángulo se pone la instrucción en forma de pseudocódigo u otro conjunto de rectángulos o símbolos para representar una condición o ciclo. Los símbolos que se utilizan en este tipo de diagramas son:

Símbolo	Descripción
	Acción.- Con este símbolo se representa una instrucción, operación u expresión del algoritmo. El rectángulo permite poner dentro de otros símbolos para representar una condición o ciclo.
	Condición.- Con este símbolo se representa una condición lógica. Como se puede observar, en el lado izquierdo está la palabra *true* y debajo de ella un rectángulo donde se pone las instrucciones cuando la condición se cumple, si es necesario se puede poner más rectángulos. Del lado derecho está la palabra *false* y debajo de ella un rectángulo para poner las instrucciones cuando la condición no se cumple.

	Ciclo mientras.- Con este símbolo se representa el ciclo mientras, lo cual quiere decir que todos los rectángulos que estén contenidos en él se repetirán mientras la condición se cumpla. Si la condición es falsa desde un principio los rectángulos contenidos no se ejecutarán.
	Ciclo repetir.- Con este símbolo se representa el ciclo repetir, lo cual quiere decir que todos los rectángulos que estén contenidos en él se repetirán mientras la condición se cumpla. En este ciclo por lo menos se ejecutan una vez los rectángulos contenidos en el.

Cuadro 3.2. Símbolos del diagrama Nassi-Schneiderman

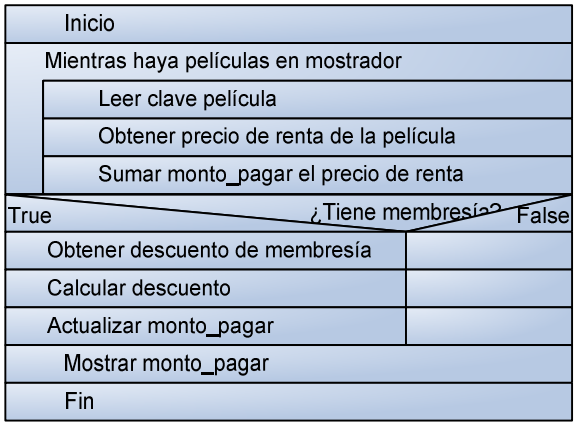


Figura 3.10. Diagrama Nassi-Schneiderman del cobro de una renta de películas

Continuando con el ejemplo de la renta de películas, si se necesitara especificar el algoritmo para especificar el cobro de renta de películas el ejemplo de la figura anterior sería una posible solución para representar dicho algoritmo.

3.5.3. Diagramas Warnier/Orr

Este diagrama permite la descripción *de los procedimientos y los datos* de la organización. Se desarrollaron inicialmente en Francia por Jean-Dominique Warnier y Kenneth Orr. Estos diagramas permiten diseñar un programa identificando primero su salida y entonces trabajar hacia atrás para determinar

las combinaciones de pasos y de entradas para producirlas. La ventaja de estos diagramas consiste en su apariencia simple y sencilla de entender. En ellos se agrupan los procesos y datos de un nivel a otro.

Para hacer estos diagramas el diseñador trabaja de atrás hacia adelante, comienza con las salidas del sistema y en un papel escribe de derecha a izquierda. Primero se ponen salidas o resultados de los procesos. En el siguiente nivel se escribe una inclusión con una llave los pasos necesarios para producir la salida. Las llaves agrupan los procesos requeridos para producir el resultado del siguiente nivel.

Elementos:

Elemento	Descripción
{ Conjunto	Delimita un bloque de información jerarquizada que pueden ser datos o acciones, de derecha a izquierda denota los niveles de abstracción, de arriba abajo, muestra la secuencia y las relaciones lógicas entre las funciones.
(0,1) Condicionalidad	La información entre los paréntesis (variable o cantidad) indica el número de veces que ocurrirá el conjunto. Si se coloca una letra C indica que un ciclo se termina cuando la condición se cumpla.
+ Secuencia de acciones mutuamente excluyentes	Indica que una acción o grupo de acciones son mutuamente excluyentes dadas las condiciones que se establezcan.

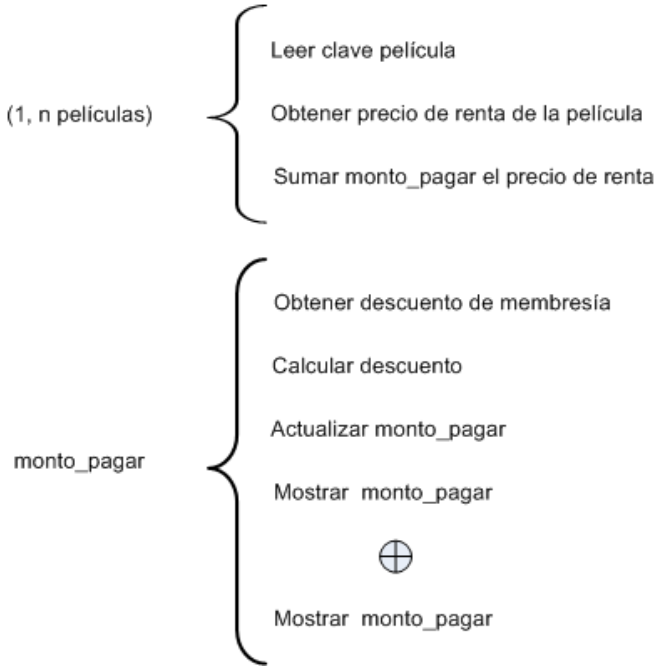


Figura 3.11. Diagrama Warnier/Orr del cobro de una renta de películas

3.6. Diseño de la interfaz de usuario

Uno de los elementos en los que el diseñador debe tener sumo cuidado es en la interfaz de usuario. La percepción que tenga el usuario en cuanto a la facilidad de uso impactará en su percepción de utilidad que tenga del sistema. El diseñador debe conocer cuáles son los conocimientos que tiene el usuario con respecto al uso de software, esto le da un marco de referencia para construir una interfaz que se adecue a las necesidades y costumbres que tenga el usuario.

El diseño de la interfaz de usuario es una actividad que se realiza con el apoyo de diseñadores gráficos por tener conocimientos más especializados en cuanto a la ergonomía, navegabilidad, colores, brillo, nitidez. Sin embargo, no siempre se cuenta con este apoyo por lo que se recomienda profundice en estos conocimientos y establezca una comunicación continua con el usuario respecto a las decisiones a tomar con la interfaz.

3.6.1. Tipos de interfaces de usuario

Las interfaces de usuario se pueden clasificar según la interacción que hay entre el usuario y el sistema, los cuales son:

- Línea de comando.
- Menú de selección.
- Llenado de forma.
- Manipulación directa.
- Antropomórfica.

La selección del estilo de interacción está delimitada por el tipo de sistema a desarrollar y de las características de los dispositivos de entradas y salida que se utilizarán en la interfaz. A continuación se describe cada uno de los tipos de interacción:

Línea de comando.- Es una de las más viejas en la que se requiere que el usuario presione una tecla de función o teclee un comando en un área de la pantalla. Los comandos pueden ser letras, abreviaturas, palabras, o múltiples palabras y funciones. La línea de comandos es un estilo poderoso que ofrece acceso inmediato a funciones del sistema. Además, tiene la flexibilidad y facilidad para incorporar opciones o parámetros. El problema con la línea de comandos es que no siempre es fácil recordar los comandos, más si se encuentran en otro idioma o abreviaturas desconocidas.

Menú de selección.- Un menú es un conjunto de opciones que el usuario puede elegir. En las pantallas, el usuario selecciona una opción con un puntero o presionando una combinación de teclas. La mayoría de las veces, algunos menús retroalimentan al usuario indicando la opción seleccionada. Los menús tienen la ventaja de que los usuarios reconocen las opciones sin necesidad de memorizar comandos con todas sus opciones. Sin embargo, los nombres de las opciones están limitados por el espacio.

Llenado de forma. Este estilo es muy ocupado para recolectar información. Las formas están estructuradas en pantallas que contienen controles o campos en que el usuario proporciona datos o selecciona una opción u opciones. Los antecedentes del llenado de forma están en las formas de papel por lo que trae como consecuencia su familiaridad por parte de los usuarios.

Manipulación directa. Esta se encuentra en los sistemas gráficos, permite al usuario interactuar directamente con los elementos que se presentan en las pantallas. Estos elementos reemplazan la acción de teclear comando o seleccionar menús. Los usuarios seleccionan los objetos de la pantalla y las acciones usando el ratón o joystick. La navegación en la pantalla y la ejecución de los cambios es accediendo a una barra de menú con opciones desplegables.

Antropomórfica. Una interfaz antropomórfica intenta interactuar con la gente tal como la gente interactúa entre sí. Para ello es necesario que la interfaz pueda establecer un diálogo en lenguaje natural, gestos manuales, expresiones faciales, y movimiento de ojo. El desarrollo de estas interfaces requiere un entendimiento del comportamiento humano; como la gente interactúa una con otra, el significado de los gestos y expresiones, etc.

Estos estilos no son mutuamente excluyentes, es decir, se pueden combinar según se necesite así como de la tecnología que se vaya a ocupar para el desarrollo del sistema.

3.6.2. Reglas de oro para el diseño de interfaces de usuario

Theo Mandel[12] estableció tres “reglas de oro” para el diseño de la interfaz:

- **Dar el control al usuario**. Esta regla busca crear una interacción con el usuario donde el usuario no esté obligado a ejecutar acciones no necesarias

[12] Theo Mandel, *The elements of user Interface Design*, USA, John Wiley & sons, 1997.

y/o no deseadas. La interacción debe ser flexible, es decir, llegar a una funcionalidad desde diferentes formas o caminos. Permitir al usuario interrumpir las acciones que ejecuta con el sistema en cualquier momento. Permitir al usuario modificar la forma en que interactúa con el sistema. Ocultar errores, advertencias o cualquier otro elemento técnico que corresponda a la implementación del sistema.

- **Reducir la carga de memoria del usuario**. Esta regla busca minimizar el esfuerzo por parte del usuario para tener que recordar cómo tiene que interactuar con el sistema. La regla está relacionada con la petición “una interfaz amigable” que expresan los usuarios en las reuniones. Para ello se debe tratar de diseñar elementos que sean intuitivos, diseñar elementos lo más aproximado posible a los objetos de la vida real, proporcionar información adicional de cada elemento cuando se requiera.

- **Construir una interfaz consistente**. Esta regla busca la consistencia de la información en cada una de las pantallas, las acciones de cada uno de los elementos sean consistentes evitando sorpresas para el usuario y mantener la secuencia de acciones coherente.

3.6.3. Criterios para el diseño de interfaces de usuario

Lo que se busca con las interfaces es que sean una extensión de las personas, es decir, el sistema debe ser acorde con las capacidades de las personas y que responda en forma específica a sus necesidades. Esto implica que la interfaz busca apoyar en la consecución de los objetivos de la organización en forma eficiente, esto lo logra si es fácil de aprender, y fácil de usar evitando sentimientos de tedio y/o frustración. Para lograr eso se ha generado un conjunto de criterios mínimos que deben ser tomados en cuenta al momento de crear una interfaz que se explican a continuación:

3.6.3.1. Consistencia

La consistencia gravita en la uniformidad en apariencia, colocación, y comportamiento. Esta es una regla que orienta a todas las actividades de diseño. La importancia de este criterio está en que puede reducir el esfuerzo humano para aprender, así como las habilidades requeridas para su aprendizaje.

3.6.3.2. Corrección de errores

El usuario debe tener la posibilidad de retractarse o cancelar una acción haciendo uso de un comando de deshacer. Esta característica ayuda mucho a los usuarios nuevos disminuyendo su estrés cuando hacen algo mal. Se debe tener mucho cuidado cuando una acción no se puede deshacer y sus consecuencias son críticas, hay que asegurar que los usuarios nunca pierdan su trabajo como resultado de sus errores o errores técnicos.

3.6.3.3. Metáforas

Muchas veces, para simplificar descripciones que ejecutan cierta tarea se sustituyen por un elemento visual que representa la acción que se espera que haga el usuario, a esto se le llama metáforas. Las metáforas o analogías deben ser realistas y simples. Además, se debe tratar de que se asemejen lo más posible a la vida real y que tengan significado para el usuario.

3.6.3.4. Ergonomía y estética

Los sistemas son más usables cuando indican claramente su estado, las posibles acciones que se pueden tomar, y los resultados de las acciones que se hagan con él. Esto se logra creando una organización jerárquica colocando la información o controles dentro de categorías lógicas. También, es necesario

presentar y ocultar la información y control según el contexto en el que se encuentre el sistema.

Las características anteriores se logran aplicando los conceptos de ergonomía y de estética. La ergonomía es un proceso que busca encontrar la distribución de los elementos más adecuada a fin de que las personas ejecuten sus actividades eficientemente. La estética busca que los elementos de la interfaz gráfica sea agradables a la vista del usuario. Aunque ambos conceptos pueden ser aplicados por separado su fortaleza radica en su aplicación conjunta.

3.6.3.5. Interfaces dinámicas

Por su naturaleza, la de ser un intermediario entre dos entes (máquina-usuario), todas las interfaces gráficas son de carácter dinámico. El término de interfaces dinámicas en algunas ocasiones es sinónimo de interfaces amigables y se presenta en la fase de análisis cuando algún usuario o cliente quiere dar a entender que espera una interfaz gráfica atractiva y que sea fácil de usar a fin de agilizar su trabajo.

3.6.4. Modelos de diseño de interfaces de usuario

Cuando se está en el proceso de desarrollo de un sistema se puede presentar uno o más de los siguientes modelos respecto a la interfaz de usuario:

- Modelo de diseño.- Es creado por un ingeniero de software.
- Modelo de usuario.- Es creado por cualquier otro ingeniero que no sea ingeniero de software.
- Percepción de usuario.- Es creado por el usuario final.
- Imagen del sistema.- Es creado por los programadores del sistema.

Cabe destacar que estos modelos pueden ser meramente mentales según lo cree cada persona por lo que el diseñador de la interfaz tiene como tarea conciliar estos modelos a fin de considerar los puntos de vista y el sistema cumpla con su propósito.

3.6.5. Problemas de diseño de interfaces de usuario

Los problemas que se presentan en el diseño de interfaces de usuario por lo regular caen en los rubros de: tiempo de respuesta, servicios de ayuda, e información sobre errores.

Tiempo de respuesta. En cuanto el tiempo de respuesta, hay dos características que tomar en cuenta: duración y variabilidad. El tiempo que tarda el sistema en presentar la respuesta (duración) puede crear ideas equivocadas del sistema, un tiempo muy corto le podría indicar al usuario un sentido de urgencia provocándole que se precipite y cometa un error, un tiempo muy largo le podría provocar frustración o que algo hizo mal. La variación de tiempo a un mismo evento (variabilidad) es un factor que también afecta la percepción del usuario respecto al sistema, en entornos Web donde la respuesta está influida por el ancho de banda es algo que muy pocas veces está al alcance del diseñador para controlar.

Servicio de ayuda. Hay dos formas de crear el servicio de ayuda de un sistema, una es durante la construcción del sistema y la otra se hace cuando se finaliza la construcción del sistema. Para ambos casos es necesario considerar los siguientes puntos:

- ¿Es la misma ayuda para todos los usuarios? ¿Según la responsabilidad del usuario con el sistema son los elementos de ayuda?

- ¿Cómo va acceder el usuario a la ayuda: impresa y/o electrónica? ¿Si la ayuda es electrónica será por medio del sistema, un recurso externo al sistema?
- ¿Cómo será estructurada la información en la ayuda? ¿Será la misma estructura para el formato impreso que el electrónico?´

Información sobre errores. Un error con la leyenda “Error 34: Desbordamiento de memoria en tiempo de ejecución en el sector 1846” podrá ser muy útil para el programador. Sin embargo, para el usuario final carece de sentido y puede resultar atemorizante. Por esta razón es necesario tomar en cuenta las siguientes consideraciones en los mensajes de error:

- La redacción y vocabulario del mensaje debe ser acorde con el perfil del usuario.
- El contenido del mensaje debe tener instrucciones claras, si es el caso, del motivo del error así como las acciones viables para corregirlo.
- El contenido del mensaje debe ser claro en la consecuencia negativa del error para que el usuario pueda verificar el efecto.
- De ser posible, el mensaje debe tener un distintivo visual y/o auditivo.

3.6.6. Herramientas para la implementación de interfaces de usuario

El uso de las herramientas de implementación empieza cuando se tiene un prototipo de la interfaz de usuario que haya sido validado por los usuarios. Existen diversas herramientas que atienden a diferentes propósitos según el ambiente en que se va a desarrollar el sistema. No es lo mismo crear una interfaz de usuario para un ambiente web o un ambiente Windows o un ambiente Mac o un ambiente Linux u otro. Cuando se realice la investigación de la herramienta por ocupar se debe considerar que como mínimo facilite la creación de ventanas, menús, mensajes o cualquier otro elemento de la interfaz

de usuario. En algunos casos es preferible que cuente también con las siguientes características:

- Validar los datos de entrada del usuario.
- Gestionar mensajes (avisos, advertencia, error).
- Facilitar la creación de ayuda para el usuario.
- Facilitar la interacción con base de datos para presentar los datos.
- Permitir que el usuario pueda configurar la presentación de la interfaz.

Analizando el listado anterior se encuentra que no siempre las herramientas contarán con estas características. Además, tal pareciera que se está programando parte de la lógica de negocio, por tal razón los líderes del proyecto con el fin de ahorrar tiempo utilizan un lenguaje de programación para construir la interfaz de usuario.

3.6.7. Evaluación de las interfaces de usuario

No hay mejor evaluación de interfaz de usuario que el mismo usuario haciendo uso de ella. Esta actividad se puede hacer de manera informal o formal. En la manera informal el diseñador proporciona al usuario la interfaz y observa su comportamiento al usarla, durante el proceso el diseñador hace anotaciones respecto a los tiempos de uso y comportamiento que presenta el usuario. En la manera formal implica una mayor inversión de tiempo porque es necesario determinar las características que se esperan evaluar, construir el instrumento que las medirá, aplicarlo con el usuario y con los resultados obtenidos actualizar y/o corregir la interfaz.

Las características que se pueden evaluar de una interfaz gráfica son: tiempos de respuesta del sistema, tiempo de aprendizaje del usuario, cantidad de acciones realizadas por el usuario para ejecutar una tarea, claridad y

pertinencia de instrucciones y mensajes de error, satisfacción por parte del usuario en ocupar la interfaz y cualquier otro que ayude a mejorar la interfaz.

3.7. Productos (artefactos) derivados del diseño

Son diversos los productos que se obtienen durante el proceso de diseño, dependiendo mucho del tiempo que se tenga, de la experiencia y conocimiento del equipo de trabajo se pueden distinguir por lo menos los siguientes.

- Diseño de la arquitectura del sistema.
- Diseño de la base de datos.
- Diseño de interfaz de usuario.
- Especificación de procesos (diagramas HIPO. diagramas Warnier/Orr, diagramas Nassi-Schneiderman, etc.)

3.7.1. Diseño conceptual

El diseño conceptual es otro nombre con el que se conoce al diseño de entidad-relación lógico que se menciona en el tema 2.

3.7.2. Especificaciones del diseño

Las especificaciones de diseño tienen como objetivo formalizar los acuerdos que se tomaron para automatizar el sistema. El detalle con que están hechos varía dependiendo de la claridad que se tenga sobre los procesos. Estas especificaciones las ocuparán los programadores y administradores de base de datos para construir respectivamente los programas y base de datos.

Las especificaciones en el diseño estructurado están conformadas por lo que has leído en este tema.

Actividades de aprendizaje

A.3.1. Investiga en Internet por medio de www.google.com los estándares que existen para el diseño de interfaz gráfica. Identifica qué criterios toma en cuenta cada estándar. En tu procesador de textos crea una tabla donde pongas el estándar y las características que menciona.

A.3.2. Investiga en Internet por medio de www.google.com la cuarta, quinta y sexta regla formal de normalización de base de datos. En tu procesador de textos describe y pon un ejemplo de cada una de ellas.

A.3.2. Elabora un mapa conceptual de este tema.

Bibliografía del tema 3

Booch, G. (2007). *Object-Oriented Analysis and Design with Applications* (3º ed., p. 638). Upper Saddle River, NJ: Addison-Wesley Longman.

Galitz, Wilbert O., *The Essential Guide to User Interface Design An Introduction to GUI Design Principles and Techniques*, 3ª edición, Estados Unidos de América, Wiley, 2007, 857 pp.

Mandel, Theo*. The elements of user Interface Design*, USA, John Wiley & sons, 1997

Normalización de bases de datos - Wikipedia, la enciclopedia libre. (n.d.). . Retrieved August 18, 2009, disponible en: http://es.wikipedia.org/wiki/Normalizaci%C3%B3n_de_bases_de_datos

Pressman, S. Roger, *Ingeniería del Software. Un Enfoque Práctico*, 5ª edición, Adaptación de Darle Ince, Traducción de Rafael Ojeda Martín (et al.), Madrid, McGraw Hill, 2002, 601 pp.

Cuestionario de autoevaluación

1. Explica el concepto de abstracción.
2. Explica la diferencia entre acoplamiento y cohesión.
3. Explica qué es la arquitectura del software.
4.- Explica en qué consiste el principio de ocultamiento de la información.
5. Explica en qué consiste la arquitectura estratificada.
6. Menciona los diagramas que se ocupan para el modelado de procesos.
7.- Menciona y explica las reglas de oro para el diseño de interfaces de usuario.
8.- Menciona cuáles son las características deseables de una herramienta para la implementación de interfaces.
9.- Menciona qué es un producto (artefacto) de diseño y menciona cuáles son.
10.- Explicación qué es una especificación de diseño.

Examen de Autoevaluación

1.- Es el proceso por el cual se resaltan ciertas características y se descartan o minimizan ciertas características de la realidad.

A. Refinamiento.
B. Abstracción.
C. Cohesión.
D. Acoplamiento.

2. Es la especificidad de la función que realiza un elemento del software.

A. Refinamiento.
B. Abstracción.
C. Cohesión.
D. Acoplamiento.

3.- Este estilo arquitectónico se centra en la transformación de los datos de entrada para obtener los datos de salida.

A. Arquitectura centrada en los datos.

B. Arquitectura de flujo de datos.

C. Arquitectura estratificada.

D. Arquitectura cliente servidor.

4.- Este tipo de interfaz se encuentra en los sistemas gráficos donde se permite al usuario interactuar directamente con los elementos que se presentan en la pantalla.

A. Línea de comando.

B. Menú de selección.

C. Llenado de forma.

D. Manipulación directa.

5.- Los antecedentes de este tipo de interfaz está en las formas de papel por lo que trae como consecuencia su familiaridad por parte de los usuarios.

A. Línea de comando.

B. Menú de selección.

C. Llenado de forma.

D. Manipulación directa.

6.- Este diseño de interfaz es una de las más viejas en la que se requiere que el usuario presione una tecla de función o teclee un comando en un área de la pantalla.

A. Línea de comando.

B. Menú de selección.

C. Llenado de forma.

D. Manipulación directa.

7.- Simplificar descripciones que ejecutan cierta tarea se sustituyen por un elemento visual que representa la acción que se espera que haga el usuario, es el criterio de interfaz gráfica de:

A. Consistencia.

B. Corrección de errores.

C. Metáfora.

D. Ergonomía.

8.- En este problema de diseño de interfaces de usuario se consideran dos características: duración y variabilidad.

A. Presentación de los datos.

B. Información sobre errores

C. Servicios de ayuda

D. Tiempo de respuesta.

9.- ¿Cuál de los siguientes artefactos es un artefacto de diseño?

A. Glosario.

B. Diagrama HIPO.

C. Diagrama de contexto.

D. Diagrama de casos de uso.

10. La pregunta ¿Cómo va acceder el usuario a la ayuda: impresa y/o electrónica? corresponde al error de interfaz de usuario:

A. Presentación de los datos.

B. Información sobre errores

C. Servicios de ayuda

D. Tiempo de respuesta.

Bibliografía básica

Bertalanffy, Ludwig von, *Teoría General de Sistemas: Fundamentos, desarrollo, aplicaciones*, Traducción: Juan Almela, México, Fondo de Cultura Económica, 1968.

Booch, G. *Object-Oriented Analysis and Design with Applications* 3ª ed. Upper Saddle River, NJ: Addison-Wesley Longman, 2007.

Braude J., Eric, Ingeniería de Software: *Una Perspectiva Orientada a Objetos*, México, AlfaOmega, Traducción: Marcia González Osuna, 2003.

Galitz, Wilbert O., *The Essential Guide to User Interface Design An Introduction to GUI Design Principles and Techniques*, 3ª edición, Estados Unidos de América, Wiley, 2007.

Gómez Vieites, Álvaro y Carlos Suárez Rey, *Sistemas de Información: Herramientas prácticas para la gestión empresarial*, 2ª ed., México, Alfaomega, 2007.

Lawrence Pfleeger, Shari, *Ingeniería del Software. Teoría y Práctica*, Prentice Hall / Pearson Educación, Traducción: Elvira Quiroga, 2002.

Mandel, Theo*. The elements of user Interface Design*, USA, John Wiley & sons, 1997.

Piattini Velthuis, Mario G., et al, *Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*, México, Alfa omega / Ra-Ma, 2000.

Pressman, S. Roger, *Ingeniería del Software. Un Enfoque Práctico*, 5ª ed., Adaptación de Darle Ince, Traducción de Rafael Ojeda Martín (et al.), Madrid, McGraw-Hill, 2002.

----------, *Ingeniería del Software. Un Enfoque Práctico*, 6ª ed., Traducción: Jesús Elmer Murrieta Murrieta y otros, México, McGraw Hill 2005.

Sommerville, Ian, *Ingeniería de Software*, 6ª edición, México, Addison Wesley / Pearson Educación, Traducción: José Alejandro Domínguez Torres, 2002.

Weitzenfeld, Alfredo, *Ingeniería de Software: Orientada a Objetos con UML, JAVA e Internet*, México, Thomson, 2004.

Yourdon, Edward, *Análisis Estructurado Moderno*, Traducción: Alexandra Taylor Amitage, México, Prentice Hall / Pearson Educación, 1993.

Sitios electrónicos

Normalización de bases de datos - Wikipedia, la enciclopedia libre. (n.d.). . Retrieved August 18, 2009, disponible en: http://es.wikipedia.org/wiki/Normalizaci%C3%B3n_de_bases_de_datos

RESPUESTAS A LOS EXÁMENES DE AUTOEVALUACIÓN
Informática IV

Tema 1		Tema 2		Tema 3	
1.	A	1.	C	1.	B
2.	B	2.	D	2.	C
3.	B	3.	D	3.	B
4.	C	4.	C	4.	D
5.	C	5.	A	5.	C
6.	B	6.	B	6.	A
7.	B	7.	D	7.	C
8.	D	8.	C	8.	D
9.	C	9.	D	9.	B
10.	D	10.	C	10.	C