ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

PROPUESTA PARA LA ELABORACIÓN DE LA ETAPA DE DISEÑO DENTRO DEL PROCESO DE DESARROLLO DE SISTEMAS DE SOFTWARE

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

AUTORES:

IVÁN PATRICIO PROAÑO DÍAZ

ivan_patriciop@hotmail.com

RITA MARCELA TORRES SANMARTÍN

ritamtorres22@hotmail.com

DIRECTOR:

MSC.ING. RAÚL CÓRDOVA

raul.cordova@epn.edu.ec

QUITO, JULIO 2011

DECLARACIÓN

Nosotros, Iván Patricio Proaño Díaz y Rita Marcela Torres Sanmartín, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Iván Patricio Proaño Díaz

Rita Marcela Torres Sanmartín

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Iván Patricio Proaño Díaz y Rita Marcela Torres Sanmartín, bajo mi supervisión.

Msc. Ing. Raúl Córdova

DIRECTOR DE PROYECTO

DEDICATORIA

Dedico esta Tesis a Dios, por ser mí luz y mi guía en todos los instantes de mi vida; a mi madre Martha, por ser la persona que me ha apoyado incondicionalmente con su paciencia, sabiduría y amor, y me ha ayudado a alcanzar todos los objetivos en mi vida; a Rita, por estar a mi lado en los momentos felices y en los momentos difíciles y ser la fuente de mi inspiración para ayudarme a conseguir este logro.

Iván Proaño.

Dedico esta Tesis primero a Dios y a la Virgen Dolorosa por bendecir mi vida a cada instante; a mi padre Cristóbal, por el esfuerzo realizado al brindarme una buena educación; a mi madre Chanita, por ser mi mejor amiga, mi fuerza, por su paciencia, consejos y amor sincero; a mis hermanos Rocío y Pablo, por su confianza y esperanza depositadas en mí; a Iván, por permanecer a mi lado con su amor y apoyo incondicionales, con lo cual juntos hemos culminado una más de nuestras metas.

Rita Torres.

AGRADECIMIENTO

Agradecemos a todos nuestros Maestros que durante estos años de estudio han sembrado en nosotros sus conocimientos y nos han inspirado a ser excelentes profesionales y excelentes personas.

Agradecemos especialmente a nuestro apreciado Maestro y Director de Tesis, Msc. Ing. Raúl Córdova, por transmitirnos sus conocimientos y experiencia y darnos el apoyo y las palabras de aliento necesarias para llevar a cabo este Proyecto.

Agradecemos a nuestras familias, quienes con su cariño y apoyo incondicional nos han ayudado en estos años de estudio a alcanzar nuestros objetivos; y que finalmente, todo el esfuerzo y dedicación se ven reflejados en este Proyecto de Titulación.

Finalmente, queremos agradecer a nuestros amig@s por su amistad, compañerismo y estima, ya que juntos nos hemos apoyado en esta importante etapa de nuestras vidas y hemos caminado juntos hacia un objetivo que se ha cumplido.

Iván y Rita.

CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

RESUMEN

El presente proyecto plantea una propuesta para la elaboración de la Etapa de Diseño de Software dentro del Proceso de Desarrollo de Software. La propuesta abarca una gran cantidad de enfoques, métodos, estrategias y otros tópicos relacionados con el diseño, que permitirán obtener un producto de calidad, que sirva como punto de partida, y facilite las etapas posteriores del desarrollo.

Este documento se divide en cuatro capítulos, en los cuales se describe detalladamente el proceso de diseño desde sus etapas iniciales hasta obtener un producto final.

En el Capítulo I se describen las áreas clave relacionadas al Diseño de Software. El punto de partida de este capítulo son los fundamentos y tópicos relacionados al diseño; la definición de arquitectura y estructura de software; los métodos, estrategias y notaciones utilizadas; y se considera el análisis y evaluación de calidad.

El Capítulo II detalla una serie de pasos y tareas a seguir durante el diseño de software, clasificados de acuerdo al enfoque que presente el sistema (Funcional, Orientado a Objetos, Basado en Componentes). Dichas tareas abarcan el diseño arquitectónico, diseño detallado, diseño de datos, diseño de interfaces de usuario, medición y análisis de la calidad y finalmente la elaboración de un documento de salida.

En el Capítulo III se plantea un caso de estudio que certifique la aplicabilidad de la propuesta, siguiendo los pasos indicados en el Capítulo II. El caso de estudio es el desarrollo de la etapa de diseño para el sistema “e-papyrus”, el mismo que se realizó con un enfoque Orientado a Objetos.

Finalmente, en el Capítulo IV se presentan las conclusiones y recomendaciones del Proyecto de Titulación.

INTRODUCCIÓN

En la actualidad la complejidad de los sistemas de software ha ido en aumento. Los requerimientos del usuario son cada vez más complejos y extensos, lo cual ha obligado a que los procesos de desarrollo sean más completos y mejor estructurados, para que puedan asegurar la obtención de un producto que satisfaga los requerimientos en un tiempo de desarrollo aceptable, y que garantice la calidad total del producto.

Dentro del proceso de desarrollo de sistemas, una de las etapas más críticas es la etapa de diseño, ya que en esta etapa se conceptualiza la estructura, organización y propiedades que tendrá la solución a un problema o requerimiento planteado; además esta etapa servirá como punto de partida para etapas futuras, como la construcción e implementación del sistema. Sin embargo, muchas empresas no realizan un proceso de diseño formal, lo cual puede generar un producto mal estructurado, que en el futuro genere costos muy altos en cambios y adecuaciones.

Considerando que al realizar adecuadamente la etapa de diseño se pueden ahorrar costos en tiempo y esfuerzo en etapas posteriores, se ha planteado un procedimiento estándar para la elaboración de esta etapa. Este procedimiento abarca las áreas y tópicos más importantes del diseño. Mediante una serie de pasos detallados se puede obtener la arquitectura del sistema, y refinarla sucesivamente hasta obtener el diseño detallado; también se considera el diseño del almacenamiento de datos y la obtención de interfaces de usuario, en todas estas actividades se evalúa la calidad del producto mediante métricas propias de esta etapa.

Como resultado al proceso de diseño se obtiene un documento detallado que describa el diseño del sistema y permita la construcción e implementación del software.

1 CAPITULO I. ÁREAS CLAVES RELACIONADAS CON LA FASE DE DISEÑO DE SOFTWARE

1.1 FUNDAMENTOS DEL DISEÑO DE SOFTWARE

1.1.1 CONCEPTOS GENERALES DE DISEÑO

Según la definición de la IEEE, el diseño de software es tanto “el proceso de definir la arquitectura, componentes, interfaces y otras características de un sistema o componente”, como “el resultado de este proceso” (IEEE Std 610.12, 1990).

Como resultado, el diseño de software debe especificar la arquitectura del sistema, es decir, cómo se descompone el software y se organiza en componentes, así como, las relaciones entre estos; la descripción de los componentes se la hace de manera detallada para facilitar su construcción.

La etapa de diseño cumple un rol importante dentro del proceso de desarrollo de software, el diseñador produce varios modelos iniciales como solución a un problema; del conjunto de modelos se analiza y evalúa cuáles de estos son útiles para poder construir el sistema.

1.1.2 CONTEXTO DEL DISEÑO

Dado que la etapa de diseño se encuentra entre las fases de análisis de requerimientos y construcción de software, el proceso de diseño lleva a cabo dos actividades relacionadas con dichas fases:

- *Diseño Arquitectónico:* Es el enlace entre la Ingeniería de Requerimientos y la Ingeniería de Diseño. Consiste en elaborar un modelo de la solución con un alto nivel de abstracción, esto implica, identificar los componentes del sistema sin dar mayor detalle.

- *Diseño Detallado:* Este es el nivel más bajo de abstracción, donde se detalla el comportamiento de cada componente, de tal forma que permita su posterior construcción.

La relación que mantienen las fases del ciclo de vida del software con la fase del diseño se visualiza en el modelo formulado por Royce en el año de 1970, dicho modelo se muestra en la Figura1.1.

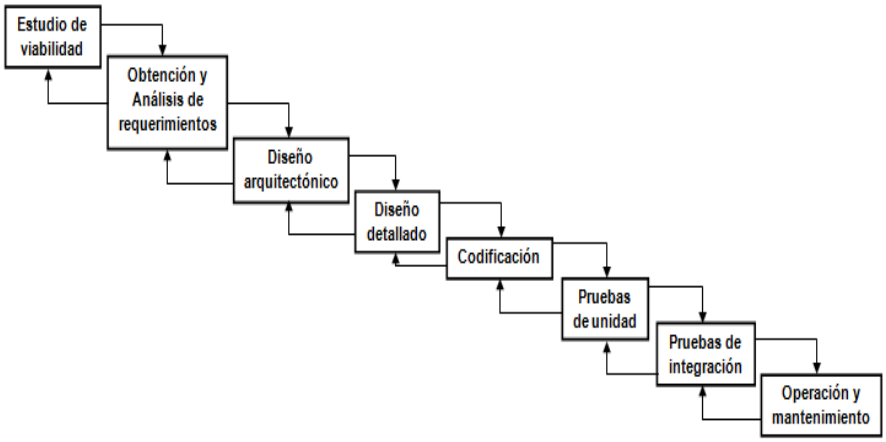


Figura 1.1. Modelo cascada del ciclo de vida de software. Tomado del libro Software Design. Budgen. 2003.

A continuación se explican cada una de las fases:

- *Estudio de viabilidad:* Permite verificar si una solución planteada es correcta con respecto a un conjunto de restricciones. Se relaciona con la fase de diseño ya que analiza la solución más práctica.
- *Obtención y análisis de requerimientos:* En la obtención de requerimientos, se identifican las necesidades de los usuarios finales del sistema, independientemente del diseño y ejecución. En el análisis, se tiene una visión más formal del problema, lo cual constituye una entrada para la fase del diseño y ayuda a plantear la solución.
- *Diseño arquitectónico:* Se refiere a la solución del problema de forma general, con un alto nivel de abstracción.

- *Diseño detallado:* Se realiza una descripción detallada de los elementos identificados en la fase anterior.
- *Codificación: T*ambién conocida como aplicación, traduce los planes de diseño y genera el código fuente del sistema, adicionalmente a la elaboración de datos y estructuras de comunicación.
- *Pruebas de unidad:* Son realizadas a nivel de módulos. Se verifica que el código de cada módulo cumpla con las especificaciones dadas en la etapa de requerimientos, además, se comprueba la funcionalidad de la interfaz.
- *Pruebas de integración:* Tienen como finalidad asegurar el funcionamiento del sistema evaluando la unificación de todos los módulos que lo conforman.
- *Operación y mantenimiento:* La fase de operación brinda retroalimentación a los usuarios, mientras que la fase de mantenimiento, puede implicar repetir ciertas actividades de las anteriores fases, pero siempre manteniendo la integridad del sistema.

Aunque las fases del ciclo de software se presentan como secuenciales, entre ellas puede existir retroalimentación, producto de inconsistencias u omisiones llevadas a cabo en algunas tareas a lo largo del proceso de desarrollo.

1.1.3 TÉCNICAS FACILITADORAS

La construcción del software se basa en varios principios fundamentales, los cuales han sido ampliamente aceptados a través de los años. Dichos principios son conocidos como “técnicas facilitadoras”, y proporcionan un marco de trabajo ideal para lograr que los programas funcionen correctamente. A continuación se detallan en la Tabla 1.1 las principales técnicas.

TÉCNICA	DESCRIPCIÓN	USO
Abstracción	Captar los detalles más importantes del problema. Abstracción funcional y Abstracción de Datos.	Útil para enfrentar la complejidad enfocándose en los detalles importantes.
Acoplamiento	Fuerza de asociación en la conexión entre módulos.	Es recomendable diseñar sistemas con acoplamiento débil para evitar la complejidad.
Cohesión	Grado de conectividad entre elementos y funciones de un módulo simple.	Es recomendable lograr una alta cohesión estructural.
Descomposición/ Modularización	Descomponer un sistema de software en módulos funcionales más pequeños.	Utilizarla con sistemas complejos o de gran tamaño para obtener sistemas manejables.
Encapsulamiento	Ocultar detalles internos de los módulos de un sistema.	Se utiliza como herramienta de la orientación a objetos.
Separación de Interfaces e Implementación	Separar la interfaz pública del componente de su implementación interna.	Esta técnica es utilizada para mantener a los clientes de un módulo alejados de su implementación
Suficiencia, Integridad y Primitividad	Un componente deberá ser lo más completo y simple posible y deberá abarcar todas las características relevantes de su abstracción	Esta técnica es utilizada en el desarrollo basado en componentes.

Tabla 1.1. Principales Técnicas Facilitadoras de Diseño de Software.

1.2 TÓPICOS CLAVE EN EL DISEÑO DE SOFTWARE

A continuación se detallan en la Tabla 1.2 los principales tópicos relacionados con el Diseño de Software.

ELEMENTO	DESCRIPCIÓN	CONSIDERACIONES
Concurrencia	Propiedad de un sistema para ejecutar múltiples procesos simultáneamente.	Al utilizar la concurrencia se deben considerar mecanismos que permitan manejar adecuadamente múltiples procesos.
Control y Manejo de eventos	Un evento se produce ante la ejecución de una acción por parte del usuario.	Útil para el diseño adecuado de interfaces de usuario.
Distribución de Componentes	Un componente satisface una función específica del sistema.	Decidir si el componente debe ser desarrollado o adaptado. En cualquier caso se debe promover la reutilización.
Manejo de Errores y Excepciones/Tolerancia a Fallos	Una excepción es un evento específico que interrumpe la ejecución normal de una rutina.	Una buena práctica de diseño es manejar las excepciones de tal forma que el sistema pueda continuar su operación ante un evento inesperado.
Interacción y Presentación	Los sistemas actuales requieren que la capa de presentación o interfaz tenga una adecuada interacción con el usuario.	Diseñar una arquitectura que permita separar la interfaz de la aplicación, de tal forma que un cambio en una de ellas no afecte significativamente a la otra.
Persistencia	Determinar los datos que deben ser persistentes y fijar mecanismos de almacenamiento y recuperación.	Utilizar un sistema de gestión de bases de datos con todas sus propiedades.

Tabla 1.2. Tópicos clave en el Diseño de Software

1.2.1 CONCURRENCIA

A más de cumplir con los requerimientos funcionales, un programa concurrente debe cumplir con ciertas propiedades adicionales:

- *Propiedad de exclusión mutua*: mientras un proceso acceda a un recurso, otros procesos deberán esperar a que sea liberado, para evitar conflictos.
- *Condición de sincronización:* si un proceso necesita que ocurra un evento para seguirse ejecutando, esta propiedad garantiza que el proceso no continúe hasta que el evento no ocurra.
- *Interbloqueo:* garantiza que no se produzca interbloqueo cuando todos los procesos están esperando a un evento que nunca va a ocurrir.
- *Interbloqueo activo*: el sistema ejecuta varias instrucciones sin obtener algún progreso.
- *Inanición:* los procesos que no están bloqueados no pueden acceder a recursos que ocupa otro proceso.

1.2.2 CONTROL Y MANEJO DE EVENTOS

Cuando se desarrolla una aplicación se deben manejar combinaciones válidas de contextos y eventos y para cada una definir el comando asociado. Un principio de un buen diseño de interfaz de usuario es asegurar que en cualquier estado una sesión de usuario tenga a su disposición tantos comandos como sea posible en lugar de tener que cambiar de estado.

1.2.3 DISTRIBUCIÓN DE COMPONENTES

Según la función que se requiera, los componentes pueden ser reutilizados o construidos. En el caso de la reutilización se sigue un proceso de selección, adaptación, integración y pruebas.

1.2.4 MANEJO DE ERRORES Y EXCEPCIONES / TOLERANCIA A FALLOS

Durante la ejecución de una rutina existen dos posibles respuestas ante una excepción:

- *Reintentar:* se intenta cambiar las condiciones que llevaron a la rutina al estado de excepción, y se ejecuta nuevamente desde el principio. Generalmente es la opción más adecuada que permite que se rectifiquen los parámetros incorrectos y no paraliza la ejecución de la rutina
- *Falla:* No se puede manejar la excepción. La ejecución de la rutina no se logró completar, y termina en un estado de falla.

Cuando se desarrolla un sistema, una característica deseable es la tolerancia a fallos, es decir que permita manejar las excepciones de tal manera que pueda recuperarse de un evento inesperado, y continuar su operación, sin llegar a un estado fallido.

1.2.5 INTERACCIÓN Y PRESENTACIÓN

Los sistemas actuales, mediante el uso de interfaces gráficas, permiten un alto grado de interacción con el usuario. Esto requiere que los sistemas sean usables, es decir que provean un acceso adecuado a sus servicios, además de permitir que los usuarios se familiaricen con la aplicación y por lo tanto ejecuten las tareas más rápidamente.

El diseño de un sistema debe tomar en cuenta estos aspectos, y enfocarse en mantener separado al núcleo funcional del sistema de las interfaces de usuario. El núcleo funcional está basado en los requerimientos funcionales del sistema y generalmente permanece estable, mientras que las interfaces de usuario generalmente están en constante cambio y adaptación.

1.2.6 PERSISTENCIA DE DATOS

Cuando se ejecuta una aplicación se manipula una cierta cantidad de datos. Una vez que la aplicación concluye, y la sesión termina, los datos transitorios desparecen y son desechados. Sin embargo, en muchos de los casos es necesario que los datos sean persistentes, es decir que permanezcan disponibles sesión tras sesión según el usuario lo requiera, entonces surge la necesidad de almacenarlos en bases de datos.

Un conjunto de mecanismos de almacenamiento y recuperación de datos merece ser llamado un sistema de manejo de base de datos si cumple las siguientes características:

- *Persistencia:* los objetos pueden sobrevivir a la terminación de las sesiones de la aplicación, para ser usados posteriormente.
- *Estructura Programable:* el sistema trata a los objetos como datos estructurados conectados por relaciones claramente definidas.
- *Tamaño Arbitrario:* no hay límite en el número de objetos que se pueden construir en una base de datos (exceptuando las limitaciones físicas de almacenamiento de la máquina).
- *Control de Acceso:* los usuarios son propietarios de los objetos y pueden definir derechos de acceso sobre ellos.
- *Restricciones de Integridad:* define restricciones semánticas para los objetos que se almacenan.
- *Administración:* herramientas que permitan monitorear, auditar, archivar y reorganizar la base de datos, así como realizar gestión de usuarios.
- *Compartir:* usuarios y aplicaciones que puedan acceder a los datos simultáneamente.
- *Seguridad:* se deben manejar privilegios (solo lectura, lectura y escritura) sobre los objetos.
- *Transacciones:* son secuencias de operaciones que se definen sobre la base de datos. Se debe garantizar que una transacción se realice por completo y en caso de falla asegurar que la base de datos quede en un estado consistente.

1.3 ARQUITECTURA Y ESTRUCTURA DE SOFTWARE

1.3.1 ESTRUCTURAS ARQUITECTÓNICAS Y PUNTOS DE VISTA

Los sistemas modernos son cada vez más complejos, lo que hace muy difícil su comprensión a primera vista. Es necesario enfocarse sólo en una, o en pequeñas estructuras de software a la vez.

Para enfocarse en un aspecto parcial de la arquitectura de un sistema se utilizan vistas. Una vista es la representación o descripción del sistema desde una perspectiva simple, de tal forma que muestre sus propiedades específicas.

Un elemento fundamental de buenas prácticas en el diseño, es el uso de múltiples vistas para describir una arquitectura (IEEE Std1471, 2000).

Algunos autores difieren en cuáles son las vistas apropiadas para describir una arquitectura, al igual que los métodos utilizados en cada vista. Así por ejemplo, Kruchten propone las siguientes vistas dentro de un enfoque orientado a objetos:

- *Vista Lógica:* determina que los elementos son abstracciones claves, los cuales se manifiestan como objetos o clases de objetos, donde cada uno tiene una funcionalidad o responsabilidad determinada.
- *Vista de Proceso:* tiene que ver con la concurrencia y la distribución de funcionalidad.
- *Vista de Desarrollo:* muestra la organización de los módulos de software, librerías, subsistemas y unidades de implementación.
- *Vista Física:* ubica los elementos dentro de nodos de comunicación y procesamiento, es decir organiza el software dentro del hardware. A esta vista se la conoce como vista de despliegue.

Para otros autores como Budgen (Budgen, 2003) las principales vistas son:

- *Vista Estructural:* se concentra en los aspectos estáticos del sistema.
- *Vista de Comportamiento:* busca describir la interrelación entre eventos y respuestas del sistema durante su ejecución.
- *Vista Funcional:* busca describir lo que el sistema realiza en términos de tareas.
- *Vista de Modelado de Datos:* se enfoca en los objetos de datos usados en el sistema y la relación entre ellos.

Debido a la amplitud de opiniones con respecto a las vistas apropiadas para un sistema, el estándar IEEE 1471-2000 recomienda no considerar un conjunto fijo de

vistas, e introduce el concepto de punto de vista, que permite designar un medio para construir vistas individuales, independientes de un sistema en particular.

Un punto de vista establece las convenciones mediante las cuales una vista es creada, representada y analizada. Una vista se ajusta a un punto de vista, ya que éste proporciona el lenguaje (modelos y notaciones) utilizados para describirla.

La Figura 1.2 muestra los conceptos definidos anteriormente.

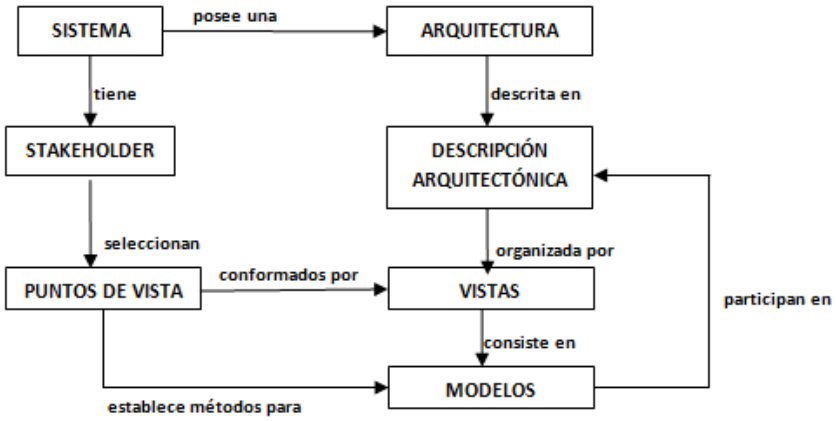


Figura 1.2 Modelo Conceptual de una Descripción Arquitectónica. Basado en el estándar IEEE1471-2000

Dentro de las buenas prácticas de diseño se recomienda que los stakeholders (individuo, equipo u organización con intereses en, o relacionados a un sistema) deban ser quienes definan los puntos de vista.

Así el proceso de realizar una buena descripción de la arquitectura de un sistema consistiría en:

- Seleccionar el punto de vista para una describir una arquitectura en particular.
- Personalizar el punto de vista acorde a los stakeholders que tengan que ver con esa arquitectura.
- Construir un conjunto de vistas correspondientes a cada punto de vista.

1.3.2 ESTILOS ARQUITECTONICOS

Un estilo arquitectónico define una familia de sistemas de software, refiriéndose a su organización estructural. Un estilo arquitectónico expresa componentes y su interrelación, las restricciones de su aplicación, la composición asociada y las reglas de diseño para su construcción (Buschmann, 1998).

Varios autores han identificado algunos estilos arquitectónicos importantes especificados en la Tabla 1.3:

TIPO	ESTILO	USO
Estructura General	Capas	Utilizado en sistemas organizados jerárquicamente, donde se intercambian servicios entre capas.
	Tuberías y Filtros	Utilizado en sistemas donde los datos sufren transformaciones sucesivas a lo largo del proceso a través de tuberías que conectan filtros.
	Llamado y Retorno	Usados con sistemas con una estructura jerárquica de programa principal con llamados a funciones o subprogramas secundarios.
	Pizarras	Se utiliza en sistemas donde se debe dar prioridad al acceso y actualización iterativa de datos desde una fuente especializada del conocimiento.
Sistemas Distribuidos	Cliente – Servidor	Utilizado cuando se requiere repartir la carga de trabajo entre proveedores de servicios y solicitantes de servicio.
	Tres Niveles	Estilo Cliente-Servidor donde se separa la interfaz de usuario, la lógica del proceso y el almacenamiento en diferentes capas.
Sistemas Interactivos	Modelo-Vista-Controlador	Utilizado para mantener cierta independencia entre el procesamiento las entradas y las salidas de un sistema
	Presentación – Abstracción-Controlador	Separación en agentes donde cada uno es responsable de una tarea específica con una interfaz propia.
Sistemas Adaptables	Micro Kernel	Utilizado en sistemas donde existe un núcleo separado de los demás componentes.
	Reflexión	Recomendado en sistemas de estructura dinámica, soporta cambios de tipos de estructura e invocación a funciones.
Otros	Batch	Se utiliza para programas ejecutados de forma secuencial.
	Intérpretes	Utilizados para analizar y ejecutar otros programas de alto nivel
	Basado en Reglas	Usado con sistemas con motores de reglas, motores de inferencia y motores de validación
	Orientado a Objetos	Utilizado cuando lo más importante en un sistema son los datos y las operaciones sobre los datos.

Tabla 1.3. Principales Estilos Arquitectónicos.

1.3.3 PATRONES DE DISEÑO

Según Pressman, un patrón de diseño “expresa una relación entre un determinado contexto, un problema una solución”, donde el contexto permite “comprender el entorno en el que reside el problema y qué solución podría ser apropiada dentro de ese entorno” (Pressman, 2010).

Los patrones poseen varias propiedades relacionadas a la arquitectura de software como se muestra a continuación (Buschmann, 1998):

- Un patrón se dirige a un problema de diseño recurrente que surge en situaciones específicas de diseño, y presenta una solución a ella.
- Los patrones de documentos existentes provienen de la experiencia de diseño probado.
- Los patrones proporcionan un vocabulario común y la comprensión de los principios de diseño
- Los patrones son un medio para documentar las arquitecturas software.
- Los patrones apoyan a la construcción de software con propiedades definidas.
- Los patrones ayudan a construir arquitecturas de software complejas y heterogéneas.
- Los patrones ayudan a gestionar la complejidad del Software.

Además, un patrón se compone de tres partes en cada modelo, las mismas que están estrechamente acopladas (Buschmann, 1998):

- *Contexto:* Una situación que da lugar a un problema de diseño.
- *Problema:* El surgimiento de un problema recurrente en el contexto.
- *Solución:* Una solución probada para un problema recurrente. En la arquitectura de software, la solución incluye dos aspectos: la estructura del modelo y la relación entre componentes, y el comportamiento en el tiempo de ejecución.

1.3.4 FAMILIA DE PROGRAMAS Y MARCOS DE TRABAJO

La familia de programas o línea de productos de Software es un conjunto de sistemas de software que comparten características comunes y que son desarrolladas mediante la reutilización de arquitectura y otros elementos asociados como diseños y documentación, planes y programas de prueba.

Un marco de trabajo es una micro-arquitectura reutilizable que posee un conjunto de "puntos de conexión" que trabajan juntos para resolver un problema común para un dominio común. Estos puntos de conexión constituyen por ejemplo, ganchos y ranuras que permiten integrar clases o funcionalidades específicas del problema.

1.4 EVALUACIÓN Y ANÁLISIS DE LA CALIDAD DEL DISEÑO DE SOFTWARE

1.4.1 CALIDAD

La calidad del software es el cumplimiento de los requisitos de funcionalidad y desempeño explícitamente establecidos, de los estándares de desarrollo explícitamente documentados, y de las características implícitas (por ejemplo la facilidad de uso) que se esperan de todo software desarrollado profesionalmente.

El concepto de calidad se maneja en todas las etapas de desarrollo de software, sin embargo la calidad en el diseño es un aspecto clave para avanzar a las siguientes etapas.

Para determinar la calidad de un producto de software es necesario el uso de medidas y factores de calidad que permitan determinar si el producto final satisface los objetivos para los cuales fue creado. Estas medidas permiten una mejor comprensión de la calidad del software mediante la utilización de valores numéricos que puedan ser ubicados en escalas y rangos con valores bien definidos, y así poder realizar una comparación adecuada.

1.4.1.1 Ilidades

Las ilidades son un grupo de factores de calidad, que deben ser considerados para evaluar la calidad de diseño. Debido a que existe un gran número de ilidades, a continuación se consideran aquellas que son más aplicables (Budgen 2003):

- *Fiabilidad:* hace referencia a la habilidad de un sistema de mantener su funcionalidad ante situaciones de error y ante situaciones inesperadas debido a un uso inadecuado.
- *Eficiencia:* tiene que ver con el uso de los recursos disponibles para la ejecución del sistema y cómo impacta esto en el nivel de desempeño del mismo.
- *Mantenibilidad:* es la capacidad del software de ser corregido, modificado o ampliado con la finalidad de aumentar su tiempo de vida útil.
- *Usabilidad:* tiene que ver con la facilidad de uso de un sistema, es decir el esfuerzo requerido para lograr una tarea u objetivo específico.
- *Portabilidad:* hace referencia a la capacidad de un sistema para ser transferido desde una plataforma a otra.

1.4.1.2 Nesses

Son un grupo de factores de calidad, asociados a la terminación inglesa “nesses”; por ejemplo (correctness). A continuación se detallan las principales (Swebok, 2004):

- *(Correctness and Completeness) Corrección y Completitud:* indica que la arquitectura debe estar acorde a la especificación de requerimientos y debe cumplir con todas las restricciones de ejecución.
- *Robustness (Robustez):* un sistema es robusto si es capaz de hacer frente adecuadamente a las variaciones en su ambiente de ejecución, logrando un impacto mínimo en su funcionalidad.
- *Fitness of purpose (Ajuste al Propósito):* es la medida de calidad definitiva de un sistema, ya que el producto final debe ser capaz de cumplir todas las tareas que le sean asignadas.

1.4.1.3 Calidad de la Arquitectura

Además de los factores de calidad expuestos, cabe mencionar la calidad directamente relacionada con la arquitectura del sistema (Bass, 2003):

- *Integridad Conceptual:* es la visión general que unifica el diseño del sistema en todos sus niveles. La arquitectura debe hacer cosas similares de formas similares.
- *Edificabilidad:* se relaciona con la facilidad de construcción, que permite al sistema ser completado en el tiempo estimado con el equipo disponible y dejar un espacio abierto a cambios durante el desarrollo.

1.4.2 TÉCNICAS DE ANÁLISIS Y EVALUACIÓN DE LA CALIDAD DE DISEÑO

Las técnicas de análisis y evaluación que ayudan a asegurar la calidad del diseño, se las puede categorizar de la siguiente manera (Sweebok, 2004):

- *Revisión del diseño de software:* Informal o semiformal, frecuentemente basados en grupos, técnicas para verificar y asegurar la calidad del diseño de sistemas, por ejemplo, revisar la arquitectura, revisar e inspeccionar el diseño, técnicas basadas en el escenario, revisar requerimientos.
- *Análisis estático:* Es el análisis estático formal o semiformal que puede ser usado para evaluar un diseño, por ejemplo: análisis de árbol de fallas o la comparación automatizada.
- *Simulación y prototipeo:* Son técnicas dinámicas para evaluar un diseño, por ejemplo, simulación del rendimiento o viabilidad del prototipeo.

1.4.3 MEDIDAS

Las medidas ayudan a comprender de mejor forma los atributos de los modelos de diseño, así como evaluar la calidad de los productos de software que son construidos (Pressman, 2010).

La medición es el proceso mediante el cual se asignan números o símbolos a los atributos de entidades reales para definirlas de acuerdo con reglas claramente

establecidas (Fenton, 1998). Sin embargo, debido a la abstracción del software, existen muchos atributos que son casi imposibles de medir, por lo que se recurre a medirlos de forma indirecta.

1.4.3.1 Métricas para el Modelo de Diseño

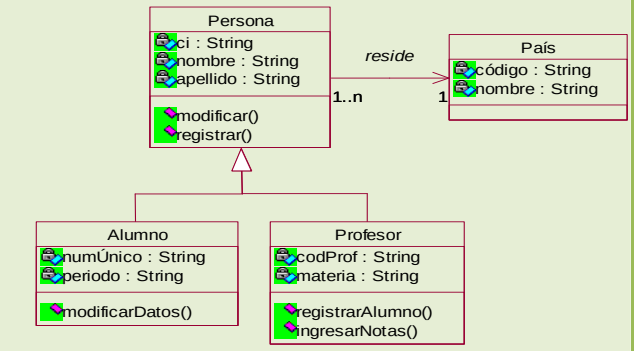
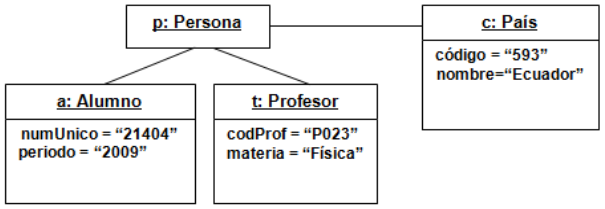
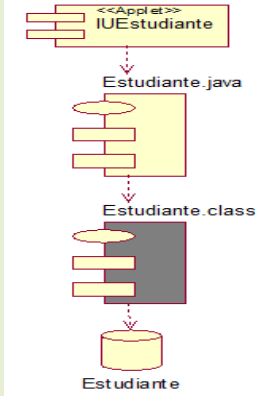
Existen métricas específicas que permiten al diseñador cuantificar la calidad del producto en cuanto a la etapa de diseño (Pressman, 2010).

- *Métricas Arquitectónicas:* proporcionan un indicio de la calidad del diseño arquitectónico.
- *Métricas a Nivel de Componentes:* miden la complejidad de los componentes de software y otras características que impactan en la calidad.
- *Métricas de Diseño de la Interfaz:* tienen que ver principalmente con la facilidad de uso de un sistema.
- *Métricas especializadas de diseño orientado a objetos:* miden las características de clases, además de las de colaboración y comunicación.

1.5 NOTACIONES PARA EL DISEÑO DE SOFTWARE

1.5.1 DESCRIPCIONES ESTRUCTURALES (VISTA ESTÁTICA)

Son notaciones en su mayoría gráficas que ayudan a describir y representar los aspectos estructurales de un diseño de software, es decir, que describen los componentes principales y cómo están interconectados. Como ejemplos de notaciones de vista estática tenemos: ADL (Arquitecture Description Languages), Diagramas de Clase y Objetos, Diagramas de Componentes, Tarjetas CRC (Class-Responsibility-Colaborator), Diagramas de Despliegue, Diagramas Entidad-Relación, IDL (Lenguajes de Descripción de Interface). A continuación en la Tabla 1.4 se enumeran las notaciones más utilizadas en el diseño de software dando un punto de vista estático.

NOTACIÓN	DESCRIPCIÓN	USO	EJEMPLO
Diagrama de Clases	Muestra las relaciones existentes entre clases, interfaces, colaboraciones.	Ayudan a modelar la estructura de la solución del problema desde la perspectiva de clases.	
Diagrama de Objetos	Muestra objetos y sus relaciones en un instante de tiempo.	Modela la vista estática de diseño desde la perspectiva de casos reales o prototípicos	
Diagrama de Componentes	Muestran la interacción y relación entre componentes de un sistema.	Modelan la topología física del sistema; útil para su implementación.	

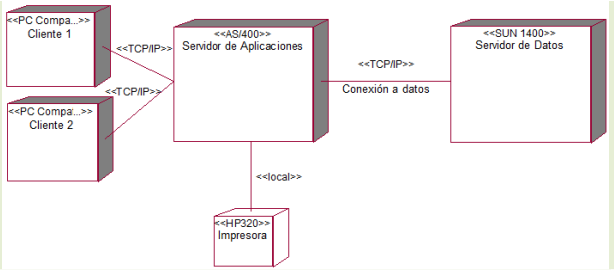
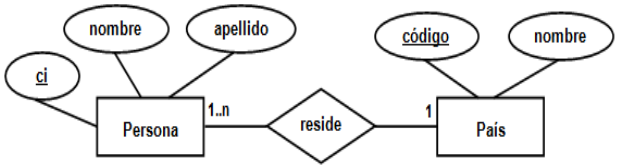
Tarjetas CRC	Identifican Objetos y Componentes de un requisito de manera informal.	Útiles para comprender de mejor manera los requerimientos del usuario.	
Diagrama de Despliegue	Modela aspectos físicos del sistema.	Utilizados para planificar la organización física del sistema en elementos de hardware.	
Diagrama Entidad Relación	Captura los objetos de datos del sistema y sus relaciones.	Útil para modelar la estructura del almacenamiento del sistema.	
Lenguaje de descripción de interface IDL	Lenguaje que especifica los componentes de una interfaz de software.	Utilizado cuando se requiere comunicar componentes de software desarrollados en lenguajes diferentes.	Microsoft Interface Definition Language (MIDL), lenguaje de descripción de servicios web (WSDL).

Tabla 1.4. Descripciones Estructurales del Diseño de Software

1.5.2 DESCRIPCIONES DE COMPORTAMIENTO (VISTA DINÁMICA)

Estas descripciones se relacionan con las cuestiones de causalidad, es decir, conectar un evento con una respuesta en presencia de ciertas condiciones necesarias. A continuación, en la Tabla 1.5 se enumeran algunas notaciones y lenguajes usados para describir el comportamiento dinámico de un sistema de software, muchas de las cuales son utilizadas durante el diseño detallado.

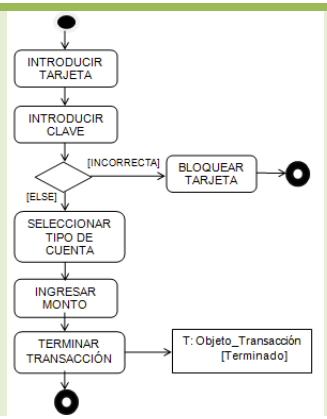
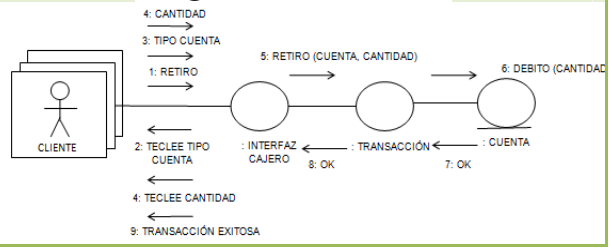
NOTACIÓN	DESCRIPCIÓN	USO	EJEMPLO
Diagrama de Actividad	Representa el flujo de control de una actividad a otra.	Utilizado para representar el sistema desde una secuencia de actividades.	
Diagrama de Colaboración	Muestra la interacción entre clases, interfaces y otros elementos.	Ayuda a comprender de mejor forma cómo interoperan los elementos de diseño.	

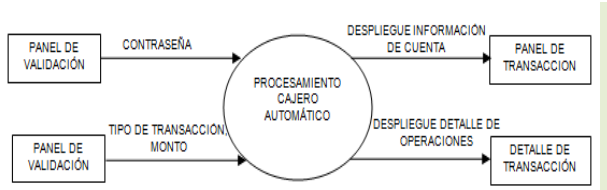
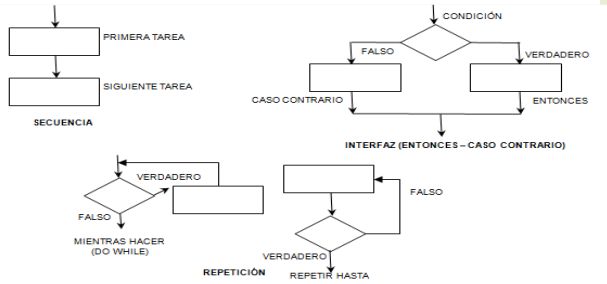
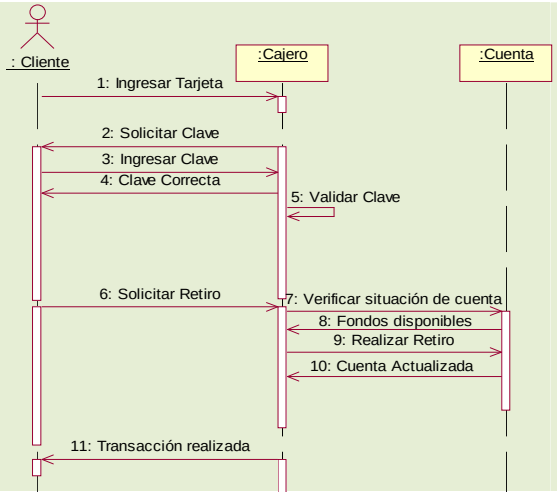
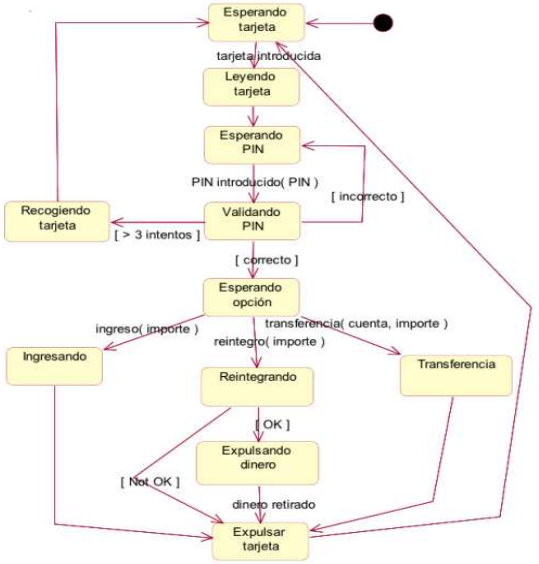
Diagrama de Flujo de Datos	Representa el movimiento y transformación de los datos.	Aporta una vista de entrada –proceso –salida del sistema	
Diagrama de Flujo/ Flujo Estructurado	Representa un flujo de control y las acciones asociadas.	Predecesor del Diagrama de Flujo de Datos.	
Diagrama de Secuencia	Muestra la interacción entre un grupo de objetos.	Utilizado para representar la secuencia y ordenamiento en el tiempo de mensajes entre objetos.	

Diagrama de Transición de Estado	Representa una máquina de estados.	Visualiza la secuencia de estados que un objeto atraviesa en su procesamiento.	
Lenguajes de Especificación Formal	Define interfaces y comportamiento de un componente en base a nociones matemáticas.	No es muy utilizado debido a su complejidad y necesidad de experiencia.	Lenguaje Z

Pseudocódigo	Lenguaje rudimentario que describe algoritmos de la operación de un sistema.	Utilizado para describir la operación de un programa en un lenguaje natural combinado con un lenguaje de programación	`caracter opción;`
			`entero monto;`
			
			`hacer mientras (opción distinta 3)`
			
			`inicio`
			
			`mostrar("1. Retiro 1")`
			`mostrar("2. Consulta 2")`
			`mostrar("3. Salir 3")`
			
			`leer(opción);`
			
			`si(opción = 1)`
			
			`MostrarSaldo(cuenta);`
			
			`sino(opción =2 )`
			
			`Retiro(cuenta,monto);`
			
			`sino(opción = 3)`
			`Terminar`
			
			`Fin`

Tabla 1.5. Descripciones de Comportamiento del Diseño de Software.

1.6 ESTRATEGIAS Y MÉTODOS DEL DISEÑO DE SOFTWARE

Para guiar el proceso de diseño se cuenta con varias estrategias generales y métodos, estos últimos son más específicos en cuanto a sugerir y ofrecer un conjunto de notaciones para ser utilizadas con el método, una descripción del proceso que se utilizará cuando se sigue el método y un conjunto de directrices en el uso del método.

1.6.1 ESTRATEGIAS GENERALES

A continuación se enumeran algunas estrategias generales utilizadas dentro del diseño de software (Swebok, 2004).

- *Divide y Vencerás y Refinamiento por pasos:* Divide y vencerás es una técnica utilizada para resolver un problema dividiéndolo en subproblemas cuyas soluciones permitirán hallar una solución al problema principal. El refinamiento por pasos se basa en una estructura jerárquica para desarrollar un programa complejo, desde un programa sencillo, mediante la incorporación incremental de características.
- *Estrategias Top – Down y Bottom – Up:* Top – Down permite descomponer un problema en módulos o segmentos de menor complejidad que ayudan en el proceso de elaborar una solución. Bottom – Up es una estrategia compositiva para construir un modelo del problema, se definen módulos que al ser combinados forman subsistemas.
- *Patrones y Lenguajes de Patrones:* Los patrones describen las estructuras estáticas y dinámicas de colaboración que resuelven un problema en particular que surge al crear aplicaciones. Un lenguaje de patrones es una combinación de patrones que conforman su gramática.
- *Abstracción de Datos y Ocultación de información:* La abstracción de datos es un conjunto de datos característicos que describen a un objeto. La ocultación de información permite que la información (procedimientos y datos) de un

módulo sólo sea accesible para ciertos módulos, es decir, solo intercambian la información necesaria para lograr la función del software.

- *Enfoques Iterativo e Incremental:* El enfoque iterativo permite tener una variedad de versiones del sistema, cada una desarrollada en momentos diferentes. En el enfoque incremental cada iteración mejora o añade funciones a la versión anterior.

1.6.2 DISEÑO ORIENTADO A FUNCIONES (ESTRUCTURADO)

El diseño estructurado se utiliza generalmente después de un análisis estructurado, cuyo producto es el diagrama de flujo de datos y las descripciones de procesos. Los investigadores han propuesto diversas estrategias y heurísticas, como por ejemplo fan-in/fan-out, para transformar un DFD en una arquitectura de software en general representada como una estructura chart.

Como un método de diseño, ésta es realmente una combinación de dos técnicas diferentes, interrelacionadas. La primera es el Análisis de Sistemas Estructurados (SSA), que se ocupa de la construcción de un modelo del problema mediante el uso de Diagramas de Flujo de Datos (DFD). La segunda es el diseño estructurado (SD), que a su vez se orienta hacia los aspectos relacionados con la búsqueda de soluciones (diseño detallado) mediante el uso de la estructura chart. Este método utiliza la estrategia de descomposición top-down.

Como se mencionó anteriormente, para transformar un DFD en una estructura chart, algunos investigadores han propuestas las siguientes heurísticas:

- *Fan-In:* es el número de módulos superiores inmediatos a un módulo, es decir, la cantidad de módulos que lo invocan.
- *Fan-Out:* es el número de módulos subordinados inmediatos a un módulo, es decir, la cantidad de módulos invocados.

1.6.3 DISEÑO ORIENTADO A OBJETOS

Existen numerosos métodos de diseño de software basado en objetos que han sido propuestos desde la década de 1980. De manera informal, de acuerdo a su elaboración, se los ubica en tres generaciones (Budgen, 2003):

- *Métodos de primera generación:* Desarrollados en la década de 1970 y principios de 1980, estos métodos tienen una tendencia evolutiva. Se caracterizan por tener formas limitadas de notación diagramática y procesos débiles. Un ejemplo de esta generación es el método HOOD (Hierarchical Object-Oriented Design).

  Este método no ofrece una forma eficaz para el uso de la propiedad de herencia y el concepto de jerarquía de clases. Sin embargo, se limita exclusivamente a las características de objetos como: la modularidad, el encapsulamiento y la abstracción.
- *Métodos de segunda generación*: Desarrollados a mediados de la década de 1990, estos métodos son considerados revolucionarios. El método Fusión desarrollado en 1994 integra y amplia los elementos de mayor éxito de las prácticas existentes hasta ese entonces. Al igual que HOOD, Fusión emplea una combinación de texto y diagramas, aunque la proporción de diagramas es mucho mayor. Fusión cuenta con un diagrama de clase (Denominado el modelo de objetos) que trata de proporcionar un punto de vista de la construcción de un sistema. Mientras que el modelado del comportamiento de clase hace uso de descripciones textuales.
- *El Proceso Unificado*: Surgió en la década de 1990 como resultado de unir las ideas y enfoques empleados por Booch, Jacobson y Rumbaugh. Se relaciona estrechamente con el desarrollo en paralelo de UML.

  El Proceso Unificado (PU) consta de cuatro fases basadas en proyectos de desarrollo (inicio, elaboración, construcción, transición), donde cada una completa un hito importante en un proyecto. Además, cada fase consta de un conjunto de una o más iteraciones, y cada ciclo de iteración contiene

elementos a partir de cinco flujos de trabajo (Requisitos, Análisis, Diseño, Implementación, Prueba), donde el grado de esfuerzo asignado a un flujo de trabajo depende de la fase.

1.6.4 DISEÑO BASADO EN COMPONENTES

Un componente de software es una unidad independiente, que posee interfaces bien definidas y dependencias que pueden ser construidas e implementadas de forma separada. El diseño basado en componentes busca proveer, desarrollar e integrar dichos componentes (Budgen, 2003).

Una de las principales razones para optar por este diseño es promover la reutilización, la misma que permite reducir los costos de elaboración del producto final (Budgen, 2003). Existen dos características que se deben tomar en cuenta para la reutilización:

- Funcionalidad bien definida, para facilitar la identificación de los componentes útiles, que satisfagan los requerimientos.
- Los componentes deben tener interfaces bien definidas que permitan su interrelación.

Sin embargo, existen varios problemas relacionados al diseño utilizando componentes:

- *Búsqueda del Componente:* es una tarea complicada encontrar un componente que satisfaga la funcionalidad indicada para un sistema. Es necesario identificar claramente las necesidades generales del problema, para luego buscar un conjunto de componentes que colectivamente resuelvan estas necesidades.
- *Integración de componentes*: pueden darse casos en los cuales el conjunto de componentes no logra satisfacer la funcionalidad requerida. De igual forma, existe la posibilidad de que dos o más componentes brinden la misma funcionalidad. Incluso, existe la posibilidad de una incompatibilidad arquitectónica entre componentes.

- *Predecir el comportamiento del sistema:* se hace difícil predecir el comportamiento al integrar varios módulos que aportan con características individuales.

1.6.5 OTROS MÉTODOS

Métodos Formales

Un método es formal si tiene sólidas bases matemáticas, usualmente proporcionadas por un lenguaje formal de especificación. Esta base ofrece los medios de definir con precisión nociones como consistencia y completitud y, con más relevancia, especificación, implementación, y corrección (Marciniak, 1994).

Para utilizar métodos formales se deben tomar en cuenta los siguientes aspectos (BOWAN, 1995):

- *Elegir la notación apropiada:* existen varios lenguajes formales de especificación (OCL, Z), se debe tomar en cuenta el tipo de aplicación que se especificará, la amplitud de uso del lenguaje y el vocabulario para su elección.
- *Formalizar lo necesario:* no es indispensable aplicar los métodos formales a todos los aspectos de un sistema. Es recomendable centrarse en los aspectos relacionados a la seguridad y aquellos cruciales para el funcionamiento del negocio.
- *Estimar los costos:* usar métodos formales incrementa los costos de desarrollo debido a que se necesitan herramientas y personal especializado.
- *Tener un experto en métodos formales:* es necesaria la consultoría permanente de un experto cuando se emplean métodos formales por primera ocasión.
- *Integración con otros métodos tradicionales:* en muchos casos es recomendable o hasta necesario integrar los métodos formales con métodos tradicionales.

- *No descuidar otros detalles:* aún utilizando métodos formales se debe tomar en cuenta el aseguramiento de la calidad, pruebas del sistema y la reutilización.

1.7 PROCESO DE DISEÑO

Un modelo general del proceso de diseño de software considera como entrada al documento de Especificación de Requerimientos; mientras que la salida a este proceso consiste en la elaboración de un conjunto de especificaciones detalladas, que sirvan para describir la forma de los eventuales componentes de un programa. En la Figura 1.3 se muestra un modelo general del proceso de diseño.

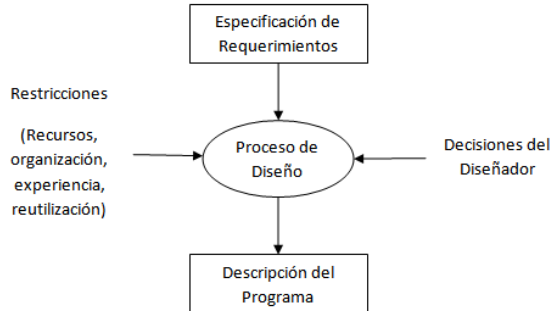


Figura 2.3. Modelo General del proceso de Diseño de Software. Tomado del libro "Software Design", Budgen, página 29.

Durante el proceso se realiza un refinamiento en el nivel de abstracción de la solución de software. A medida que se avanza, se reduce el nivel de abstracción, hasta alcanzar el nivel más bajo y detallado que permita generar código fuente.

En la práctica, al proceso de diseño de software se lo divide en dos etapas. La primera es el Diseño Arquitectónico, y la segunda el Diseño Detallado, como se puede observar en la Figura 1.4.

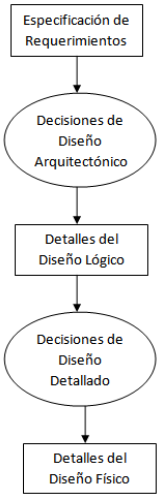


Figura 1.4. Fases del Proceso de Diseño de Software. Tomado del libro "Software Design", Budgen, página 30.

1.7.1 DISEÑO ARQUITECTÓNICO

Esta primera fase, conocida también como diseño lógico, representa un enlace crítico entre la Ingeniería de Requerimientos y la Ingeniería de Diseño. Consiste en elaborar un modelo de la solución con un alto nivel de abstracción. Esto implica que sólo las propiedades externas de los elementos del modelo son incluidas. Para esta primera aproximación se toma en cuenta la naturaleza y estructura en sí del problema a resolver, y no se enfoca profundamente en la forma que adoptará la solución.

La arquitectura abarca información acerca de cómo se relacionan entre sí los componentes de un sistema, se omite específicamente cierta información de aquellos componentes que no participan en la interacción. Además, en la arquitectura se definen patrones de diseño, que se pueden utilizar para satisfacer los requerimientos planteados, así como restricciones para aplicar estos patrones.

Según el estándar IEEE 12207.0-1996 el diseño arquitectónico consta de las siguientes tareas:

Diseño Arquitectónico del Sistema

- Establecer y documentar una arquitectura de alto nivel, identificando los elementos de hardware, software y operaciones manuales. Esta arquitectura debe ser planteada acorde a todos los requerimientos del sistema.
- Evaluar la arquitectura del sistema considerando trazabilidad hacia los requerimientos, consistencia, factibilidad.

Diseño Arquitectónico de Software

- Transformar y documentar los requerimientos de los elementos de software en una arquitectura que describa su estructura de alto nivel e identifique los componentes de software.
- Desarrollar y documentar un diseño de alto nivel para las interfaces externas a los elementos de software, y entre los componentes de cada elemento.
- Desarrollar y documentar un diseño de alto nivel de la base de datos.
- Desarrollar y documentar versiones preliminares de la documentación de usuario.
- Evaluar la arquitectura de software, diseño de interfaces, diseño de la base de datos de acuerdo a la trazabilidad a requerimientos, consistencia y factibilidad.

1.7.2 DISEÑO DETALLADO

En esta segunda etapa las piezas abstractas del problema, que fueron identificadas previamente en la primera fase, son direccionadas a unidades de base tecnológica, conocidas como diseño físico o detallado. Este el nivel más bajo de abstracción, donde se detalla el comportamiento de los componentes. Como resultado de este proceso se obtienen las especificaciones de diseño detallado para uso de los programadores en las siguientes fases del desarrollo.

Según el estándar IEEE 12207.0-1996 el diseño detallado consta de las siguientes tareas:

- Desarrollar y documentar el diseño detallado para cada componente de los elementos de software. Los componentes deben ser refinados en niveles bajos que contengan unidades de software, que puedan ser codificadas compiladas y probadas.
- Desarrollar y documentar el diseño detallado para las interfaces externas a los elementos de software, y entre unidades de software. Este diseño debe permitir la codificación sin necesidad de información posterior.
- Desarrollar y documentar el diseño detallado de la base de datos.

2 CAPITULO II: PROPUESTA PARA LA ELABORACIÓN DE LA ETAPA DE DISEÑO DE SOFTWARE

Para esta propuesta se ha planteado el proceso de diseño descrito en la Figura 2.1, el cual es detallado en los capítulos posteriores.

Figura 2.1. Resumen de la Propuesta de Diseño de Software (Elaborado por los Autores).

2.1 DOCUMENTACIÓN DE ENTRADA

Para la elaboración de la etapa de Diseño se considerarán los documentos producidos en la etapa de Requerimientos y en la etapa de Análisis, dentro del proceso de desarrollo de software. Estos documentos ayudarán al equipo de diseño a entender las necesidades del usuario y los procesos del negocio para poder diseñar el modelo de la solución.

2.1.1 DOCUMENTO DE ESPECIFICACIÓN DE REQUERIMIENTOS DE SOFTWARE

El documento de especificación de requerimientos de software (SRS) constituye el punto de partida para el desarrollo de un producto de software. Es de vital importancia para el equipo de desarrollo ya que facilita la comprensión de las funciones que el producto de software va a realizar, y los procesos que se van a automatizar.

El SRS debe reflejar un contrato entre clientes y desarrolladores acerca de la funcionalidad, las interfaces externas, el rendimiento, atributos y restricciones de diseño, que el software debe cumplir.

A continuación se describen las partes fundamentales que se deben tomar en cuenta para la especificación de requerimientos:

- *Alcance:* se deben especificar, a breves rasgos, las tareas que el software va a realizar, así como las tareas que no va a realizar. Además se especifican los beneficios, objetivos y metas que se pretende alcanzar con el software.
- *Perspectiva del Producto:* muestra la relación del producto con respecto a otros productos similares. Se debe indicar si el producto es independiente y totalmente autónomo, o es un componente de un sistema más grande.
  En el segundo caso, es necesario especificar las interfaces y los medios de comunicación que existen entre el producto y los otros componentes del sistema.

- *Restricciones:* se realiza una descripción general de cualquier elemento que limite las opciones del desarrollador. Estos elementos pueden ser políticas reguladoras, limitaciones de hardware, interfaces a otras aplicaciones, operaciones en paralelo, funciones de auditoría, funciones de control, requerimientos en el lenguaje de alto nivel, requerimientos de confiabilidad, criticidad de la aplicación, consideraciones de seguridad.
- *Funciones:* los requerimientos funcionales definen las acciones fundamentales que realizará el software cuando procesa la información.
- *Requerimientos de Rendimiento:* se especifican los requerimientos numéricos estáticos y dinámicos, por ejemplo, número de terminales soportadas, número de usuarios simultáneos, cantidad de información que se maneja.
- *Requerimientos de la base de datos lógica:* se especifican los requerimientos lógicos de la información que se va a manejar dentro de una base de datos.
- *Restricciones de Diseño:* especifica las restricciones que pueden ser impuestas por otros estándares, limitaciones de hardware, entre otras.
- *Atributos del sistema de software:* varios atributos que pueden ser usados como requerimientos, por ejemplo, confiabilidad, disponibilidad, seguridad, portabilidad y mantenimiento.

2.1.2 DOCUMENTOS DE LA ETAPA DE ANÁLISIS DE REQUERIMIENTOS DE SOFTWARE

La etapa de análisis permite comprender adecuadamente los procesos del negocio y el contexto del sistema que se va a desarrollar, y produce diagramas que modelen el dominio del problema.

Dependiendo de la orientación del sistema, el modelado de análisis difiere en las formas de representación. A continuación se presentan, en la Tabla 2.1 los diferentes enfoques que se pueden utilizar.

ORIENTACIÓN	MODELADO	USO	DIAGRAMA
Análisis Estructurado	Datos	Definir los objetos de datos que se procesan y su relación.	Diagrama Entidad-Relación
	Orientado al Flujo	Modelar los datos con una visión entrada-proceso-salida.	Diagrama de Flujo de Datos nivel 0
Análisis Orientado a Objetos	Basado en Escenarios	Determinar la interacción de los usuarios con el sistema.	Diagrama de Casos de uso, Diagrama de Interacción
	Basado en Clases	Modela el dominio del problema desde la perspectiva de relaciones entre clases	Diagrama de Clases de Análisis
	Comportamiento	Modela el comportamiento del sistema o cómo interactúa con eventos externos.	Diagrama de Secuencia del Sistema

Tabla 2.1. Documentación sugerida para la Etapa de Análisis de Requerimientos.

2.1.2.1 Análisis Estructurado

Utilizar este enfoque cuando se consideren como entidades separadas datos y procesos. Modelar los datos considerando sus atributos y relaciones. Modelar los procesos de tal forma que se visualice la transformación de los datos, mientras fluye el sistema. Se recomienda que el analista al menos realice el diagrama de flujo de datos, y si es oportuno, lo complemente con el diagrama entidad relación.

2.1.2.2 Análisis Orientado a Objetos

Utilizar este enfoque cuando se necesite analizar los requisitos desde la perspectiva de clases y objetos. Del analista dependerá el uso de una o varias formas de representación, sin embargo, se recomienda que al menos se modelen el diagrama de casos de uso y el diagrama de clases de análisis.

2.2 SELECCIÓN DE ESTRATEGIAS Y MÉTODOS

El uso de una o varias estrategias y métodos del diseño de software dependerá del enfoque al cual este dirigido el sistema a desarrollar.

A continuación se especifican en la Tabla 2.2 los métodos y estrategias utilizadas para cada uno de estos enfoques.

ENFOQUE	ESTRATEGIA	USO
Diseño Orientado a Funciones	Heurística Fan-in/ Fan-out	Utilizada para transformar un DFD en una Estructura Chart.
	Descomposición Top-Down	Utilizada para enfrentar problemas de gran complejidad o de gran tamaño.
Diseño Orientado a Objetos	Patrones de Diseño	• Utilizada para promover la reutilización • Solución a un problema de diseño a través de soluciones probadas a problemas específicos.
	Enfoque Iterativo e Incremental	Construir un sistema en base a iteraciones sucesivas que agregan funcionalidad y producen un incremento.
Diseño Basado en Componentes	Bottom-up	Utilizada para promover la reutilización o cuando se disponga de componentes previamente elaborados con funcionalidad e interfaces bien definidas.

Tabla 2.2. Estrategias de Diseño de Software

2.2.1 DISEÑO ORIENTADO A FUNCIONES (ESTRUCTURADO)

Este método se compone de dos técnicas: SSA (Análisis de Sistemas Estructurados) y SD (Diseño Estructurado), para la primera se recomienda utilizar los Diagramas de Flujo de Datos y para la segunda se deberán utilizar estructuras chart.

2.2.2 DISEÑO ORIENTADO A OBJETOS

A pesar de la existencia de varios métodos para el diseño orientado a objetos, se recomienda seguir el Proceso Unificado, que ha sido ampliamente utilizado, y que a través de un proceso dirigido por casos de uso, iterativo e incremental y centrado en la arquitectura; ofrece un marco adecuado para el desarrollo de proyectos. El lenguaje utilizado por este método para modelar la solución del problema es UML.

2.2.3 DISEÑO BASADO EN COMPONENTES

Se recomienda utilizar este tipo de diseño cuando un sistema se pueda construir a base de la integración y adaptación de componentes que tengan una funcionalidad particular, bien definida, y además puedan ser reutilizados posteriormente en otros desarrollos. Como estrategia fundamental en este tipo de diseño se tiene:

2.3 DISEÑO ORIENTADO A FUNCIONES

A continuación se detallan en la Figura 2.2 los pasos para el Proceso de Diseño cuando se escoge la Orientación Funcional.

Figura 2.2. Proceso de Diseño para la Orientación Funcional (Elaborado por los Autores).

2.3.1 SELECCIÓN DEL ESTILO ARQUITECTÓNICO

Existe una variedad de estilos arquitectónicos que permiten definir al sistema en términos de un patrón de organización estructural (Ver capítulo 1.3.2).

Para sistemas orientados a funciones es recomendable utilizar estilos arquitectónicos de “Llamado y Retorno”, donde se emplea un control sobre la secuencia del procesamiento del sistema, es decir, se tiene un programa principal desde el cual se realizan llamados a subprogramas, los cuales retornan datos.

Para usar el estilo arquitectónico de “Llamado y Retorno” con un programa principal y subrutinas, se deben tener en cuenta los siguientes aspectos:

- *Componentes:* determinar el programa principal y los subprogramas que se van a comunicar.
- *Conectores:* determinar las invocaciones a los subprogramas (llamados).
- *Control de ejecución:* determinar las jerarquías de invocación de los subprogramas (algoritmos).
- *Comunicación de datos:* determinar los parámetros que ayudan al intercambio de datos.
- *Coherencia de Diseño:* se debe utilizar el método de Análisis y Diseño estructurado (SSA/SD), incluyendo estrategias de descomposición Top-Down.

2.3.2 SELECCIÓN DE PUNTOS DE VISTA

Los puntos de vista son útiles para concentrarse en ciertos aspectos y características puntuales de la arquitectura del sistema, de tal manera que exista una mejor comprensión del problema desde diferentes representaciones (Ver capítulo 1.3.1).

En el caso del diseño orientado a funciones se van a seleccionar los puntos de vista funcionales y construccionales.

- *Puntos de vista funcionales:* representados por el diagrama de flujo de datos y pseudocódigo.

- *Puntos de vista construccionales:* representados mediante estructuras chart

2.3.3 DISEÑO ARQUITECTÓNICO

Para realizar el diseño arquitectónico de un sistema orientado a funciones se debe partir del diagrama de flujo de datos nivel 0, hasta obtener una arquitectura inicial con un alto nivel de abstracción, para luego ir refinándola sucesivamente hasta alcanzar un diseño detallado del sistema representado por una estructura chart y el uso de pseudocódigo.

A continuación se detalla el proceso de elaboración de la arquitectura:

- Como punto de partida, obtener de la etapa previa de análisis, un DFD de nivel 0, también denominado diagrama de contexto, que describa el problema en función de las operaciones que tiene que realizar el sistema. Este diagrama deberá mostrar el flujo de datos de entrada y salida, ver Figura 2.3.

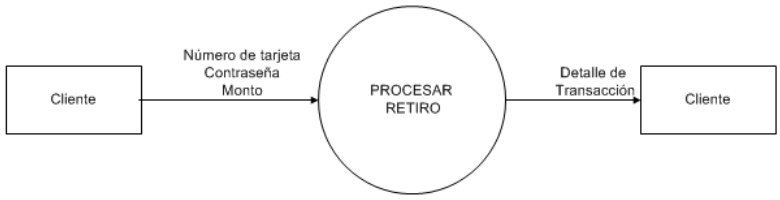


Figura 2.3 Ejemplo de DFD nivel 0 para un retiro en un cajero automático (Elaborado por los Autores).

- Generar niveles más detallados a partir del diagrama de contexto inicial, para esto se puede utilizar la estrategia Top-Down, en la cual al DFD de nivel cero se lo divide en subtareas funcionales, hasta obtener DFD's más simples que representen funciones específicas del sistema, ver figuras 2.4 y 2.5.
- Es necesario considerar que un refinamiento sucesivo puede llegar a generar una gran cantidad de módulos, lo cual generaría una mayor complejidad del problema, por esta razón es necesario detener la descomposición cuando se alcance un nivel adecuado de detalle del problema.

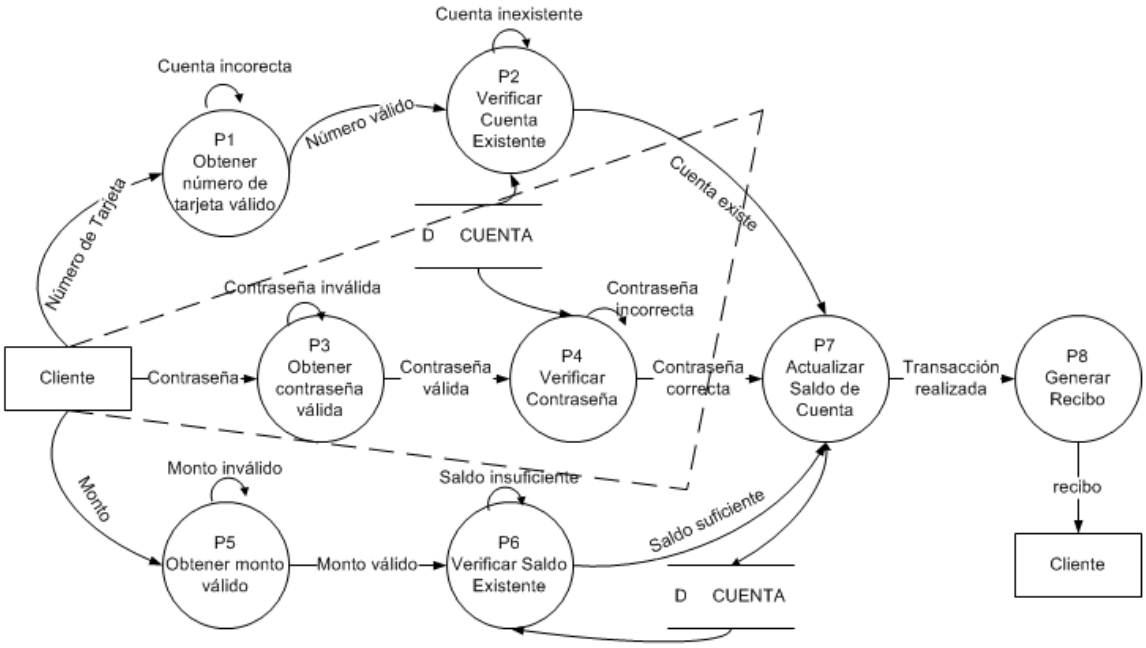


Figura 2.4. Ejemplo de un DFD detallado, dividido en subtareas funcionales (Elaborado por los Autores).

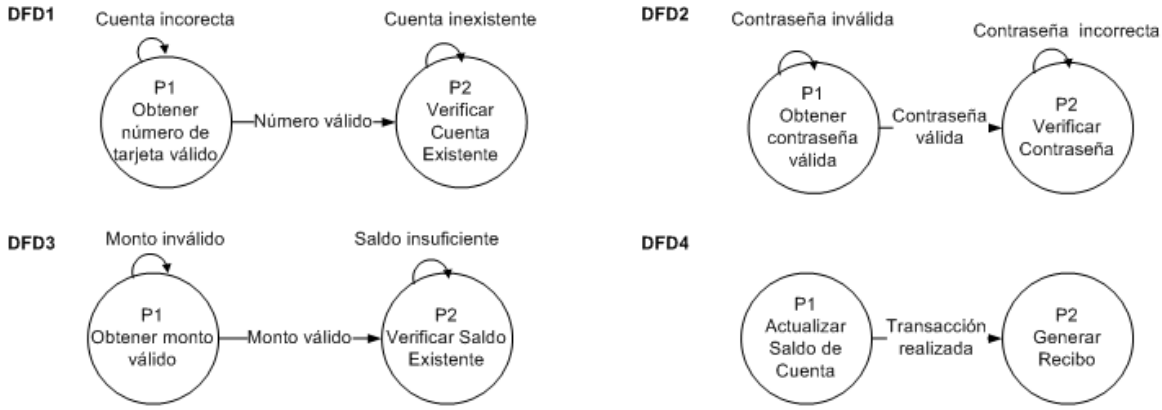


Figura 2.5. Ejemplo de la obtención de DFD simples mediante descomposición (Elaborado por los Autores).

- Generar un diccionario de datos pertinentes al sistema, con definiciones precisas. A continuación se presenta el formato base de un diccionario de datos:

ELEMENTOS DE DATOS	
Nombre	Nombre significativo del elemento
Alias	Nombre de referencia
Tipo/Tamaño	Tipo de datos y longitud máxima
Formato de Salida	Formato de cómo se presenta el dato en pantalla
Valor por defecto	Valor del dato si no se modifica su estado
Fuente	Origen del valor del dato
Seguridad	Actores que pueden modificar el elemento de datos
Responsable(s)	Usuarios responsables de modificar el elemento de datos
Validación de Datos	Se especifica el dominio, los valores permitidos o reglas de validación.
Descripción	Se provee información adicional

Tabla 2.3 Formato del Diccionario de Datos

PROCESOS	
Nombre del Proceso	Nombre del proceso, tal como parece en el DFD.
Propósito	Describir un resumen del propósito general del proceso
Número del proceso	Número de referencia que identifica el proceso
Flujo de Datos de Entrada	Nombres de los flujos de datos que entran al proceso
Flujo de Datos de Salida	Nombres de los flujos de datos que salen del proceso
Descripción del proceso	Se explican los detalles del proceso

Tabla 2.4. Formato del Diccionario de datos para especificar procesos.

FLUJOS DE DATOS	
Nombre	Nombre significativo del elemento.
Alias	Nombre de referencia.
Descripción	Descripción y propósito del flujo de datos.
Origen	Fuente del flujo de datos. Puede ser un proceso, una entidad o un almacén de datos.
Destino	El punto final del flujo de datos. Puede ser un proceso, una entidad o un almacén de datos.
Fuente	Origen del valor del dato
Record	Grupo de elementos relacionados al flujo de datos.
Frecuencia	Número esperado de ocurrencias para el flujo de datos por unidad de tiempo.

Tabla 2.5. Formato del Diccionario de Datos para especificar el flujo de datos

- Refinar los diagramas de flujo de datos obtenidos hasta alcanzar una especificación detallada. Proporcionar retroalimentación a los pasos anteriores.

2.3.4 DISEÑO DETALLADO

Una vez que el diagrama de flujo de datos está completo y tiene un nivel adecuado de detalle, se debe realizar un proceso de transformación hacia estructuras chart de la siguiente forma:

- Identificar las transacciones que estén involucradas en el problema e identificar los componentes del DFD que corresponden a cada transacción, para luego agruparlos y utilizarlos como entrada del siguiente paso, como se muestra en la Tabla 2.6.

TRANSACCIÓN	COMPONENTES DEL DFD
Gestión de Cuenta	• Obtener número de tarjeta válido • Verificar cuenta existente
Gestión de Contraseñas	• Obtener contraseña válida • Verificar contraseña
Validación de Monto	• Obtener monto válido • Verificar saldo existente
Generador de Recibos	• Actualizar saldo de cuenta • Generar Recibo

Tabla 2.6. Ejemplo de organización de procesos del DFD acorde a las transacciones.

- A los DFD's creados para cada transacción se los transforma en estructuras chart, tomando en cuenta el flujo de datos entre las operaciones y la jerarquía entre los subsistemas, como se muestra en la Figura 2.6.
- El primer paso para esta transformación es identificar la función central del sistema que actuará como el punto de partida para reorganizar los componentes del DFD. La función central del sistema es aquella que está en medio del flujo de entrada y salida de datos. Si no es posible identificar una función central, se debe crear una.

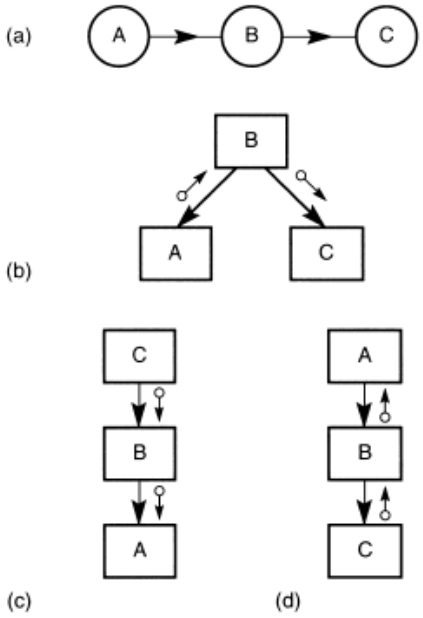


Figura 2.6. Ejemplo de selección del proceso de transformación central entre tres alternativas (b,c,d), partiendo de un DFD (a). Tomado del libro "Software Design",Budgen. Página 274.

- Posteriormente, se crea una jerarquía a partir de la función central, la cual se convierte en el cuerpo principal del programa y de la que se desprenden las demás funciones o subsistemas, ver Figura 2.7.

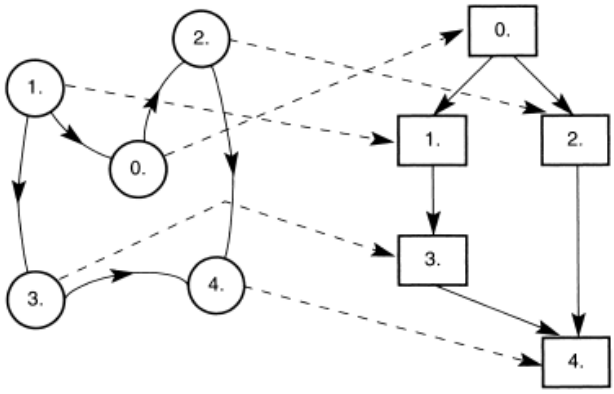


Figura 2.7. Ejemplo de la trasformación de un DFD a un diagrama de estructura jerárquico. Tomado del libro "Software Design",Budgen. Pagína 274.

- Organizar las llamadas a los subsistemas y revisar las jerarquías para asegurar que las estructuras chart estén bien creadas, como se muestra en la Figura 2.8.

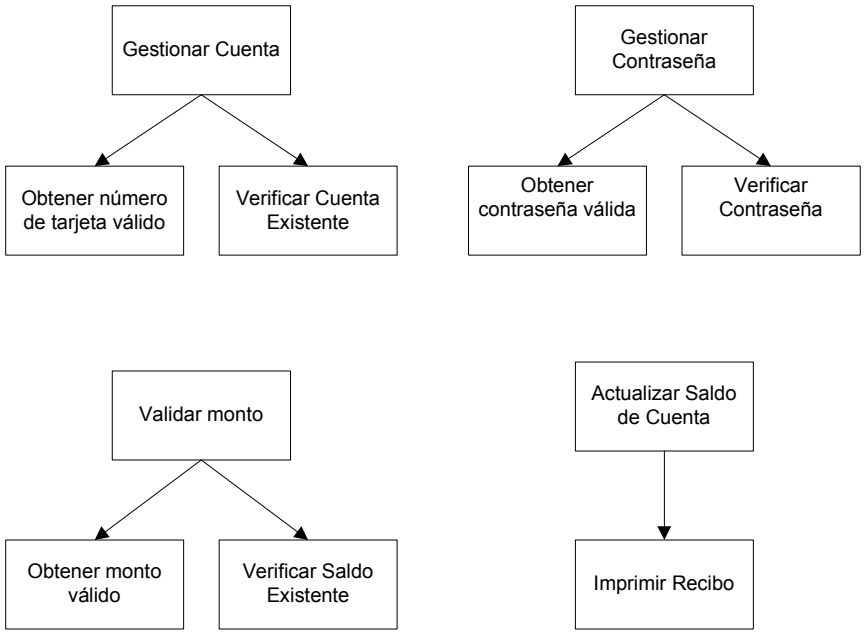


Figura 2.8. Ejemplo de estructuras chart para los subsistemas (Elaborado por los Autores).

- Finalmente, se deben combinar las estructuras chart creadas, y refinarlas, evitando tener duplicados e inconsistencias entre ellas, hasta obtener un diagrama como el de la Figura 2.9.

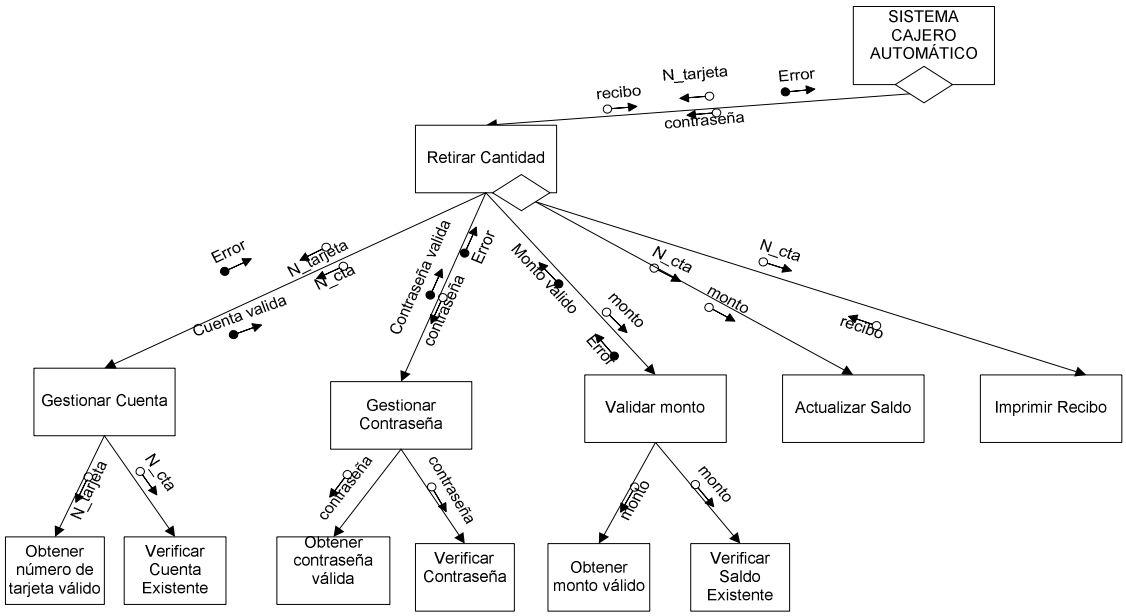


Figura 2.9. Ejemplo de un diagrama de estructura para un retiro en un cajero automático (Elaborado por los Autores).

2.3.5 PSEUDOCÓDIGO

Las estructuras chart permiten representar al sistema desde un punto de vista construccional, de bajo nivel, sin embargo antes de la implementación es recomendado desarrollar el pseudocódigo de los módulos del sistema.

Se debe escribir el pseudocódigo de tal forma que refleje la secuencia de las operaciones y la estructura jerárquica del sistema. La estructura del pseudocódigo deberá tener el contenido mostrado en las Figuras 2.10 y 2.11.

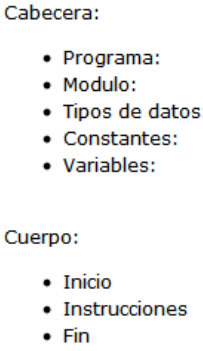


Figura 2.10. Estructura del pseudocódigo (Elaborado por los Autores).

```
PROGRAMA: Cajero_Automático
Modulo: Verificar_cuenta_Existe
Variables:
        n_cuenta: texto
        estado: booleano
Inicio
        Leer(n_cuenta)
        Buscar(n_cuenta:tabla_cuentas)
        Si(n_cuenta existe)
                retornar = "Cuenta existe"
                estado = verdadero

        Caso Contrario
                retornar ="Error cuenta no existe"
                estado = falso
Fin
```

Figura 2.11. Ejemplo de pseudocódigo para un módulo de verificación de cuenta (Elaborado por los Autores).

2.4 DISEÑO ORIENTADO A OBJETOS

A continuación se detallan en la Figura 2.12 los pasos para el Proceso de Diseño cuando se escoge la Orientación a Objetos.

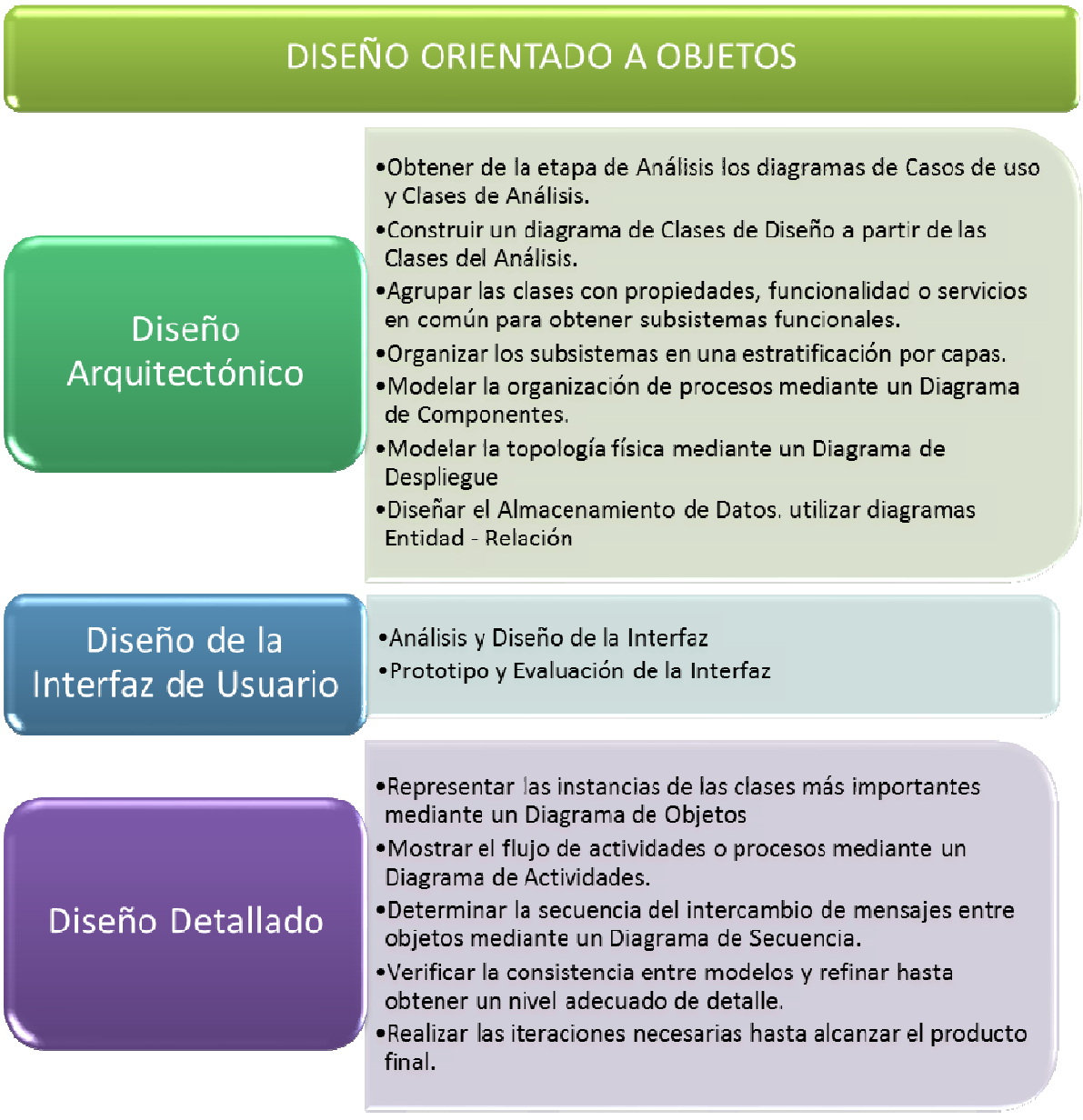


Figura 2.12. Proceso de Diseño Para la Orientación a Objetos (Elaborado por los Autores).

2.4.1 SELECCIÓN DEL ESTILO ARQUITECTÓNICO

Para sistemas orientados a objetos es recomendable utilizar estilos arquitectónicos que permitan fiabilidad, reusabilidad, escalabilidad y facilidad de modificación del sistema. Dentro de este grupo se puede elegir un estilo arquitectónico “Orientado a Objetos”, donde los componentes de un sistema encapsulan los datos y las operaciones que se deben realizar para manipular los datos.

Para usar el estilo arquitectónico Orientado a Objetos se deben tener en cuenta los siguientes aspectos:

- *Componentes:* Objetos o instancias de tipos de datos abstractos.
- *Conectores:* los objetos interactúan a través de invocaciones a métodos u operaciones.
- *Control de ejecución:* La coordinación entre objetos se consigue a través del paso de mensajes.
- *Comunicación de datos:* La comunicación entre objetos se consigue a través del paso de mensajes.
- *Coherencia de Diseño:* se recomienda utilizar el proceso unificado, basándose en estrategias iterativas e incrementales. Como lenguaje de modelado se recomienda usar UML.

2.4.2 SELECCIÓN DE PUNTOS DE VISTA

En el caso del diseño orientado a objetos se van a seleccionar los puntos de vista funcionales, construccionales, de comportamiento y de modelado de datos; organizados a través de las 5 vistas de diseño propuestas por UML como se muestran en la Figura 2.13.

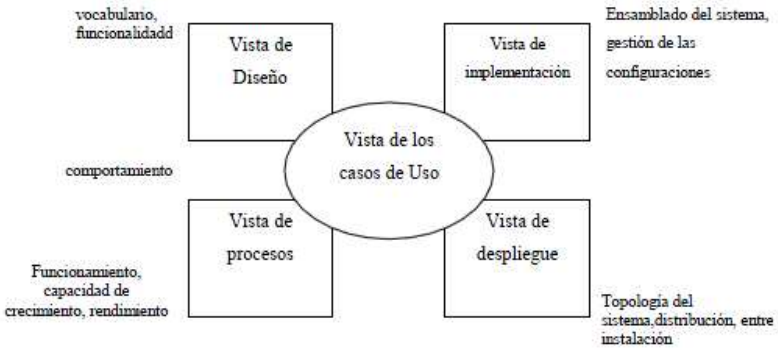


Figura 2.13. Vistas de diseño propuestas por UML. Tomado del libro "Diseño Orientado a Objetos con UML", Alarcón. Pagína 24.

- *Vista de Casos de uso:* describe el comportamiento del sistema desde la perspectiva del usuario final. Se representarán los aspectos estáticos mediante diagramas de casos de uso, mientras que los aspectos dinámicos con los diagramas de actividades o diagramas de estado.
- *Vista de Diseño:* modela los requerimientos funcionales del sistema en función del vocabulario del problema y de la solución. Se utilizará para su representación modelos estáticos, como diagrama de clases y objetos, y modelos dinámicos como diagramas de actividades o diagramas de secuencia.
- *Vista de procesos:* representa la distribución de los procesos e hilos de ejecución. Se utilizan los diagramas de la vista de diseño enfatizando las clases activas, las cuales muestran procesos e hilos.
- *Vista de implementación:* representa los componentes y archivos que se ensamblan para hacer disponible el sistema físico. Se utilizará el diagrama de componentes para modelar aspectos estáticos.
- *Vista de despliegue:* modela los nodos de la topología de hardware sobre la cual se ejecuta el sistema. Se utilizan los diagramas de despliegue para modelar el aspecto estático.
- *Vista de datos:* adicionalmente utilizaremos el diagrama entidad relación para modelar los datos.

2.4.3 DISEÑO ARQUITECTÓNICO

Dentro del Proceso Unificado, la actividad del diseño arquitectónico se realiza en la etapa de elaboración, y requiere mayor esfuerzo durante las primeras iteraciones del proceso, como se muestra en la Figura 2.14. En esta etapa se propone elaborar una arquitectura base con alto nivel de abstracción, la cual sirva como punto de partida para que a través de un refinamiento sucesivo se alcance el producto final.

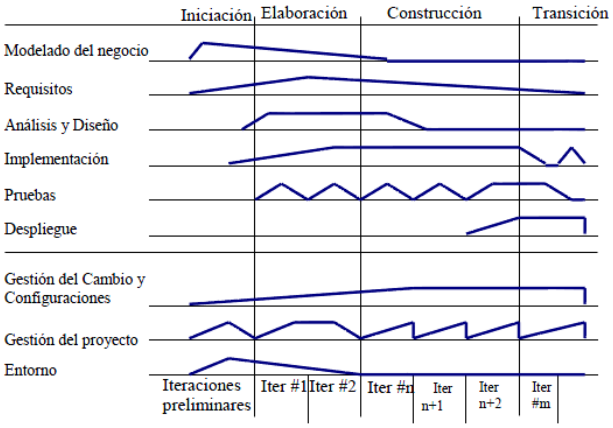


Figura 2.14. Ciclo de vida del Desarrollo de Software utilizando Proceso Unificado.

A continuación se detallan las tareas a realizarse durante el diseño arquitectónico:

- Obtener del modelo del análisis los Diagramas de casos de uso, que muestren un modelo del problema, como se puede observar en las Figuras 2.15 y 2.16.

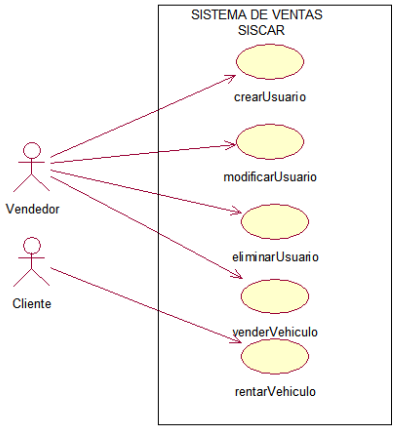


Figura 2.15. Diagrama de Caso de Uso del modelo de Análisis (Elaborado por los Autores).

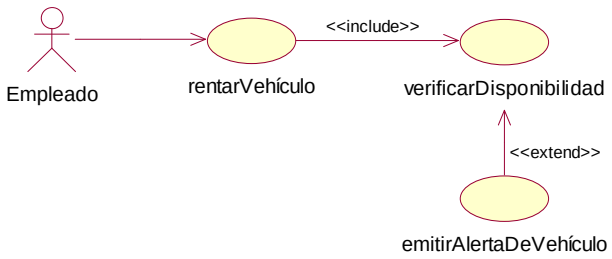


Figura 2.16. Diagrama de Caso de Uso Detallado (Elaborado por los Autores).

- De igual forma, obtener del modelo del análisis, un diagrama de clases de análisis, que modele el contexto del sistema, como se muestra en la Figura 2.17.

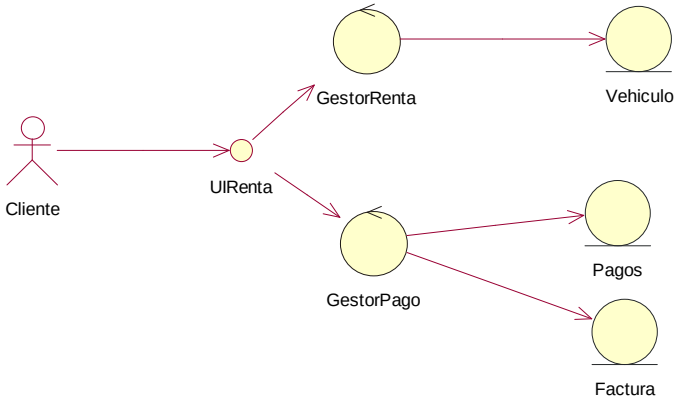


Figura 2.17. Diagrama de Clases de Análisis (Elaborado por los Autores).

- Basándose en el modelo del análisis inicial, refinar el diagrama de clases a un nivel de detalle adecuado para la etapa de diseño, para empezar a modelar la solución al problema.
- Tomando como modelo al diagrama de clases del análisis se construye el diagrama de clases de diseño. Para realizar esta tarea se deben identificar las clases (abstracciones clave) que van a formar parte de la solución del problema, no necesariamente todas las clases definidas en el análisis van a formar parte de la solución. A cada clase definida se le añaden atributos y operaciones, para lo cual se puede utilizar el método gramatical propuesto por Booch. Luego, se añaden las relaciones y asociaciones que existirán entre las clases indicando la multiplicidad y navegabilidad entre ellas. Un ejemplo de este diagrama se puede observar en la Figura 2.18.

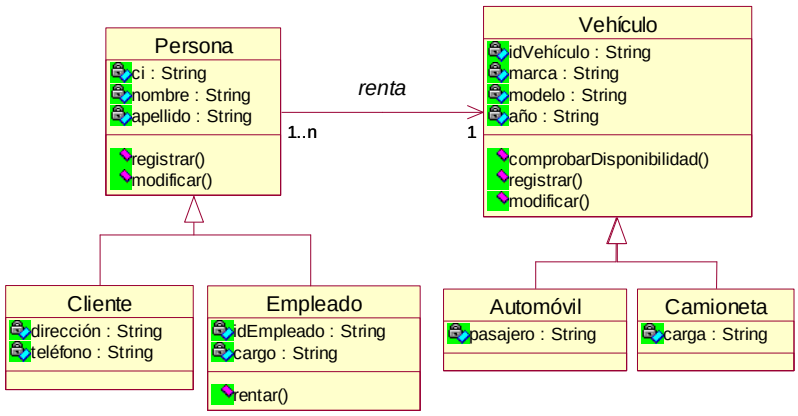


Figura 2.18. Diagrama de Clases Detallado (Elaborado por los Autores).

- Descomponer el sistema en subsistemas funcionales. Para realizar esta tarea se deben agrupar las clases que presenten propiedades en común, y que juntas presten un mismo tipo de servicio o funcionalidad.
    - Se debe procurar realizar un número pequeño de subsistemas.
    - Cada subsistema debe tener un mecanismo de comunicación con el resto, es decir una interfaz bien definida.
- Una vez que se han determinado los subsistemas, agruparlos mediante una estratificación de capas. Cada capa representa un nivel de abstracción diferente de la funcionalidad requerida, ver Figura 2.19. Para realizar esta tarea se deben seguir estos pasos:
    - Determinar el número de capas que se van a utilizar acorde al problema. Usualmente se utilizan tres capas principales, la capa de interfaz de usuario, capa de aplicación y capa de datos.
    - Nombrar cada capa y definir su funcionalidad.
    - Asignar los subsistemas a cada capa definida.
    - Definir las interfaces de comunicación entre capas.
    - Las capas y subsistemas se pueden representar utilizando paquetes UML.
    - Si es necesario, establecer futuras iteraciones para refinar el proceso.

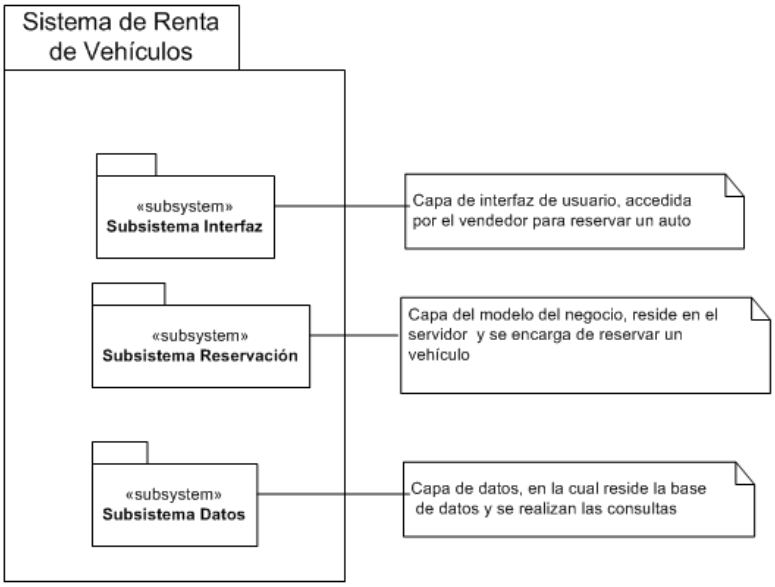


Figura 2.19. Ejemplo de estratificación por capas (Elaborado por los Autores).

- Adicionalmente, para complementar la arquitectura del sistema, se puede proporcionar una vista general de la implementación del sistema, a través de los diagramas de componentes y de despliegue modelando la topología física y organización de los procesos.

  Los diagramas de componentes modelan los elementos físicos que pueden encontrarse dentro de un nodo, por ejemplo ejecutables, bibliotecas, documentos; ver Figura 2.20. Los diagramas de despliegue representan la topología y organización del hardware, incluyendo los recursos computacionales; esta representación se hace a través de nodos como se indica en la Figura 2.21.

  Dependerá del equipo de diseño decidir si es necesario realizar esta tarea en la etapa de diseño, o postergarla para la etapa de implementación.

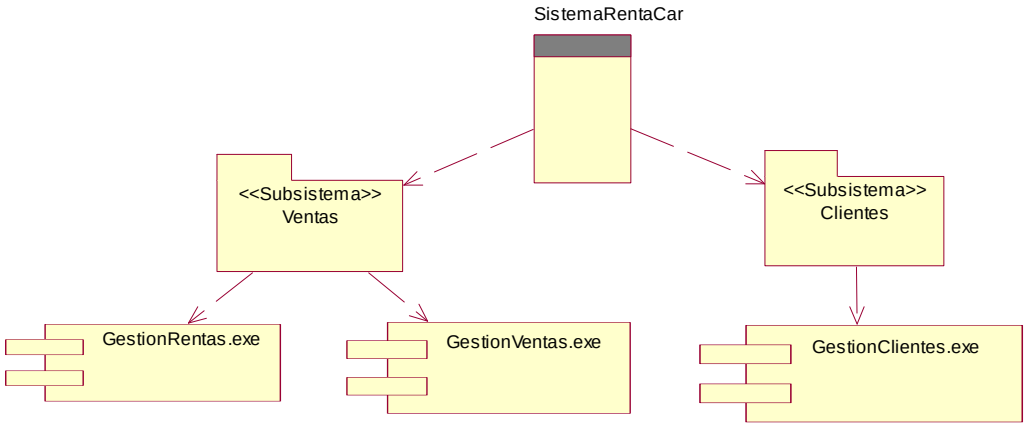


Figura 2.20 Ejemplo de Diagrama de Componentes (Elaborado por los Autores).

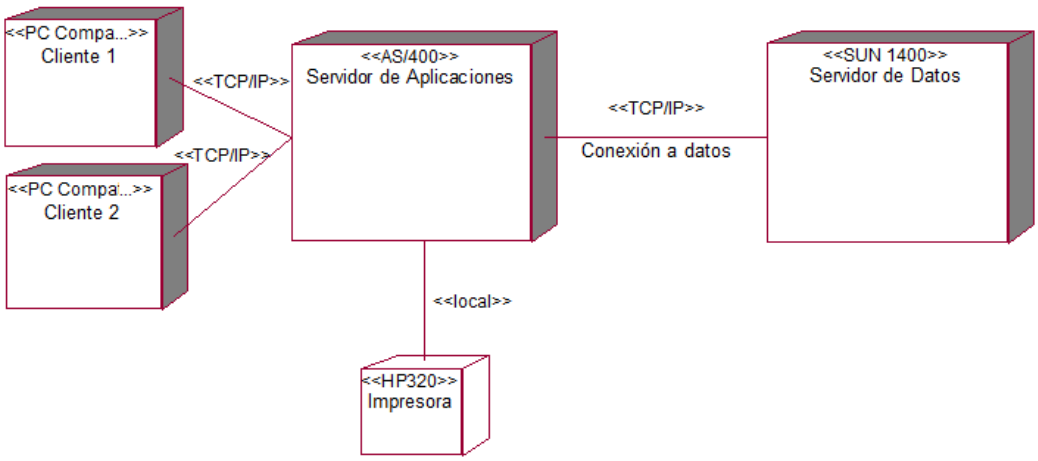


Figura 2.21. Ejemplo de Diagrama de Despliegue (Elaborado por los Autores).

- Diseñar el almacenamiento de los datos que se van a utilizar en el sistema. Ver Capítulo 2.4.5
- Diseñar la interfaz de usuario, basada en los casos de uso, teniendo en cuenta los roles de los actores y los distintos escenarios. Ver Capítulo 2.5.
- En el caso que el sistema se relacione, o forme parte de otros sistemas o dispositivos externos se deberán diseñar las interfaces necesarias para la comunicación.

2.4.4 DISEÑO DETALLADO

Dentro del Proceso Unificado la actividad del diseño detallado se realiza en los puntos iniciales de la etapa de construcción, ver Figura 2.14. En esta etapa se realiza un diseño detallado de objetos y sus interacciones, para lo cual se especifican más detalladamente los tipos de atributos, las operaciones y cómo se enlazan los objetos unos con otros. A continuación se describe el proceso de diseño detallado:

- Realizar un diagrama de objetos, en el cual se muestren instancias de los elementos encontrados en el diagrama de clases, que necesiten ser especificadas como una “instantánea” en un momento dado de la ejecución, como se muestra en la Figura 2.22.

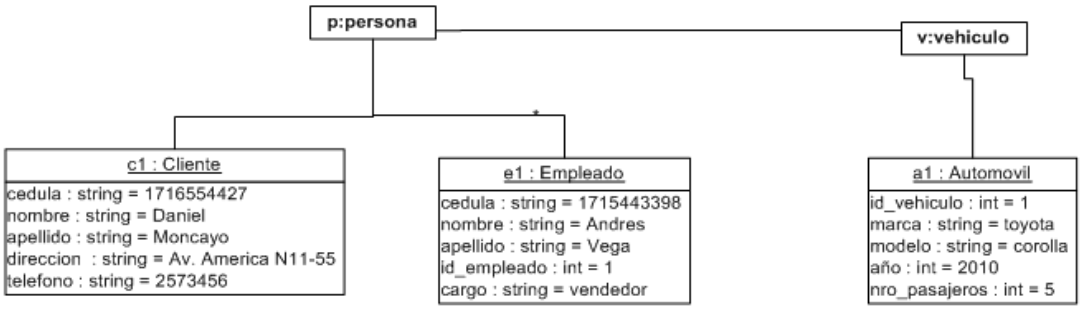


Figura 2.22. Ejemplo de Diagrama de Objetos (Elaborado por los Autores).

- Modelar la parte dinámica del sistema, mostrando el flujo de actividades o procesamientos que se llevan a cabo. Esto permitirá visualizar un flujo de control con objetos, ver Figura 2.23. Para este modelado se debe utilizar el diagrama de actividades.

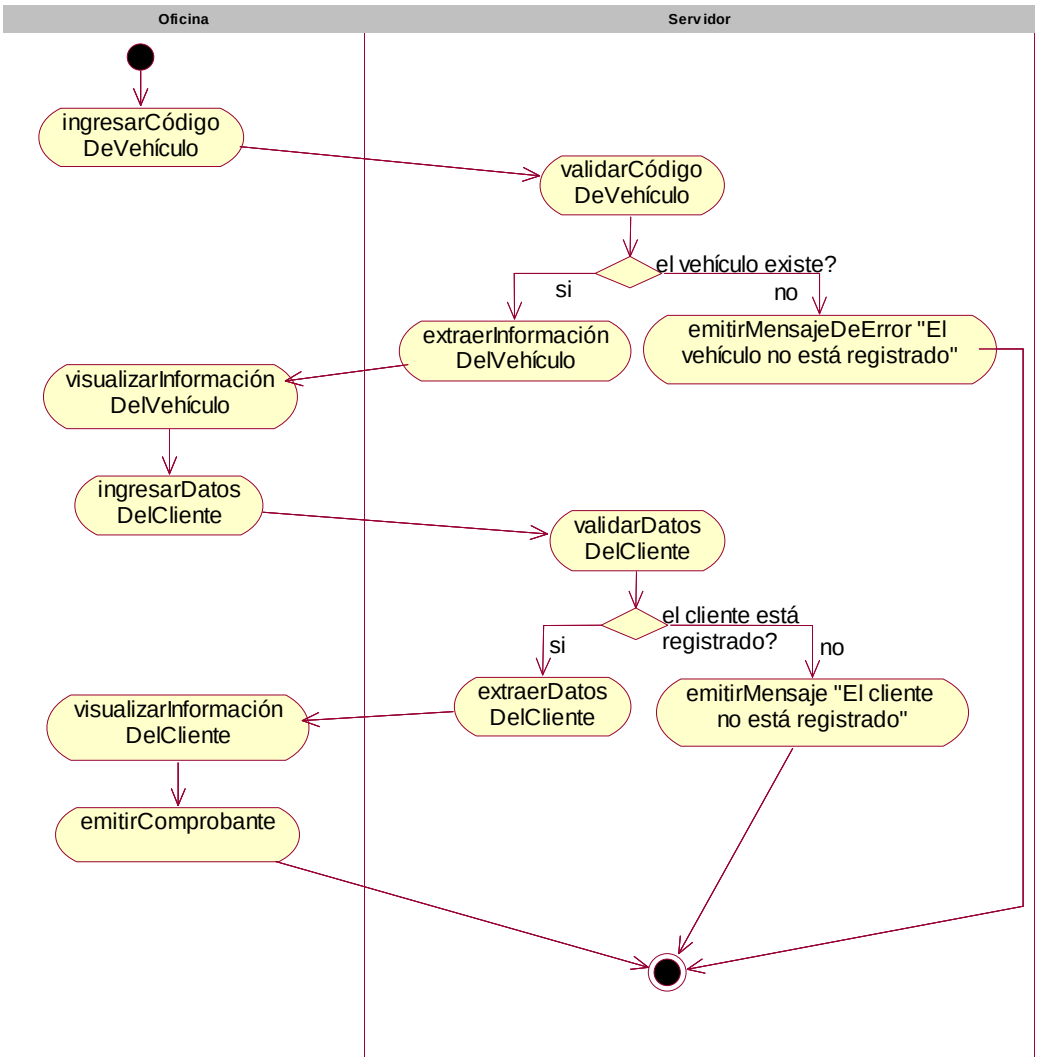


Figura 2.23. Diagrama de Actividades del Caso de Uso Rentar Vehículo (Elaborado por los Autores).

- Determinar la secuencia de mensajes, a lo largo del tiempo, que se intercambian entre los objetos del sistema. Para lograr esta descripción se debe utilizar el diagrama de secuencia para cada escenario del problema, Ver Figura 2.24.

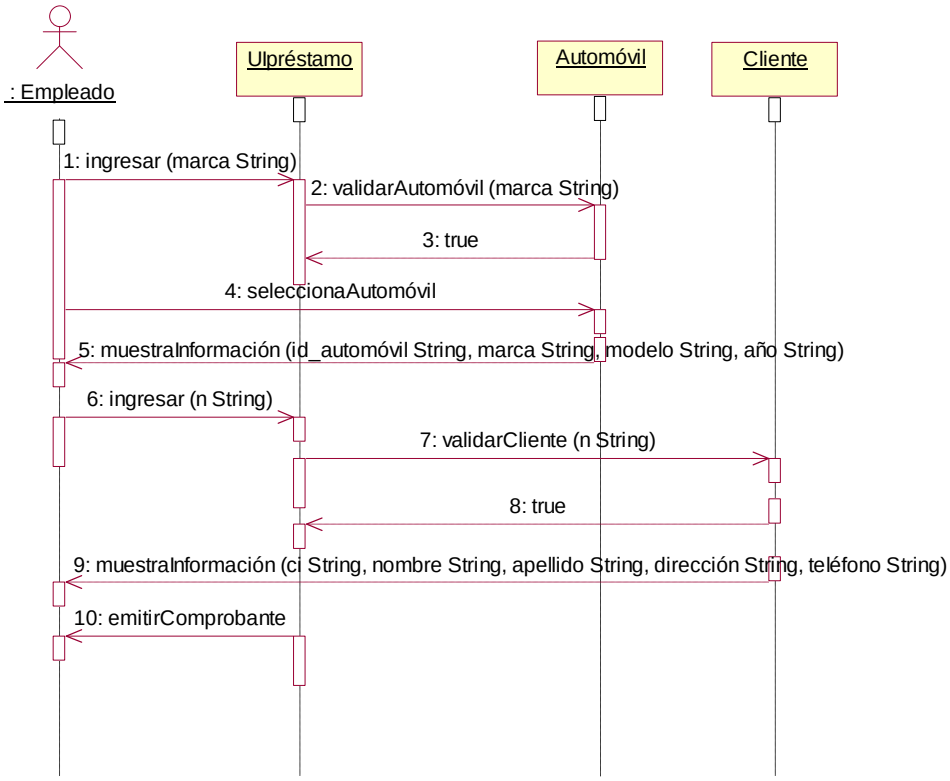


Figura 2.24. Ejemplo de Diagrama de Secuencia (Elaborado por los Autores).

- Una vez que las clases y los objetos se encuentran correctamente definidos, a un nivel más detallado, se revisan los diagramas obtenidos para verificar la consistencia entre modelos. Si es necesario se realizan las correcciones necesarias o se refinan los modelos.

2.4.5 MODELADO DE DATOS

La mayoría de los sistemas que se modelen tendrán objetos persistentes, lo que significa que estos objetos podrán ser almacenados en una base de datos con el fin de poderlos recuperar posteriormente. Generalmente se empleará una base de datos relacional, una base de datos orientada a objetos o una base de datos híbrida objeto-relacional para el almacenamiento persistente. UML es apropiado para modelar esquemas lógicos de bases de datos, así como bases de datos físicas.

Para modelar un esquema de base de datos, utilizando un diagrama Entidad Relación se siguen los pasos descritos a continuación:

- Basándose en el diagrama de clases de diseño, identificar aquellas clases persistentes que deben trascender en el tiempo de vida de las aplicaciones, como se muestra en la Figura 2.25.

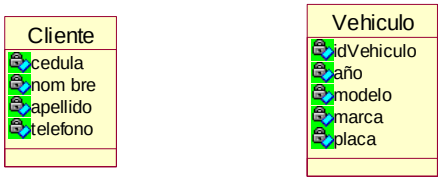


Figura 2.25. Ejemplo de Objetos de Datos (Elaborado por los Autores).

- Las clases seleccionadas se convierten en entidades dentro del diagrama ER. En el caso de las relaciones de herencia, se tienen tres alternativas; una de ellas es que tanto las superclases como las subclases se transformen en entidades. Otra alternativa es que sólo las superclases se conviertan en entidades en el diagrama ER. Finalmente la última alternativa es convertir en entidades sólo las subclases.
- A continuación se determinan los atributos que manejará cada entidad, para ellos se trasladan los atributos definidos en el diagrama de clases; adicionalmente se deberán especificar las claves primarias y foráneas propias del diagrama ER. Es importante que se detallen los atributos de cada entidad, indicando el tipo de dato, longitud, valores permitidos, entre otros. Ver Figura 2.26.

Figura 2.26. Ejemplo de Atributos de Objetos de Datos (Elaborado por los Autores).

- Las operaciones o métodos definidos dentro de las clases se convertirán en procedimientos almacenados en el diagrama ER. No todos los métodos podrán ser implementados en la base de datos, por lo que se debe escoger aquellos que sean útiles para manipular los datos.
- A continuación se establecen las relaciones y asociaciones entre las entidades, para lo cual se toman en cuenta las relaciones indicadas en el diagrama de clases. Ver Figura 2.27.

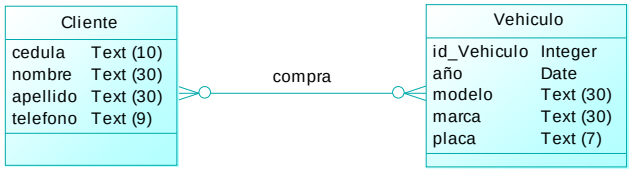


Figura 2.27. Ejemplo de Relación entre entidades (Elaborado por los Autores).

- Definir la cardinalidad entre las relaciones especificadas. Detectar patrones comunes que complican el diseño físico de las bases de datos, tales como asociaciones cíclicas, asociaciones uno a uno y asociaciones muchos a muchos. Donde sea necesario deben crearse abstracciones intermedias para simplificar la estructura lógica. Ver Figura 2.28.

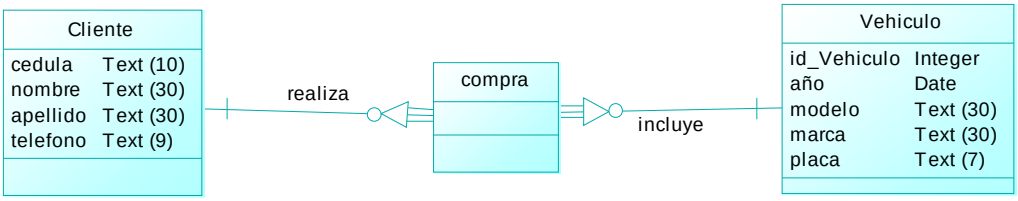


Figura 2.28. Ejemplo de Cardinalidad (Elaborado por los Autores).

- Añadir detalles propios del diagrama Entidad – Relación, que no se describen en el diagrama de clases.
- Donde sea posible, usar herramientas que ayuden a transformar un diseño lógico en un diseño físico.

2.5 DISEÑO BASADO EN COMPONENTES

Los métodos basados en componentes usan estrategias bottom-up que permiten integrar varios componentes de software ya existentes, que aporten con una funcionalidad específica, para formar un sistema completo que satisfaga los requerimientos planteados. Son recomendados cuando se busca la reutilización, además de la reducción del tiempo de desarrollo del sistema.

Existen varios métodos y estilos arquitectónicos utilizados, cada uno con sus propias etapas, sin embargo, se han determinado las siguientes etapas comunes que se deben realizar para trabajar con componentes, como se muestra en la Figura 2.29.

Figura 2.29. Pasos Diseño Basado en Componentes (Elaborado por los Autores).

2.5.1 ANÁLISIS DEL DOMINIO

Esta es una etapa previa a al proceso de desarrollo basado en componentes. En esta etapa se extrae y se analiza los requerimientos del usuario respecto al sistema que va a ser desarrollado.

- Se puede utilizar el diagrama de casos de uso como punto de partida para las etapas posteriores. Ver Figura 2.30.

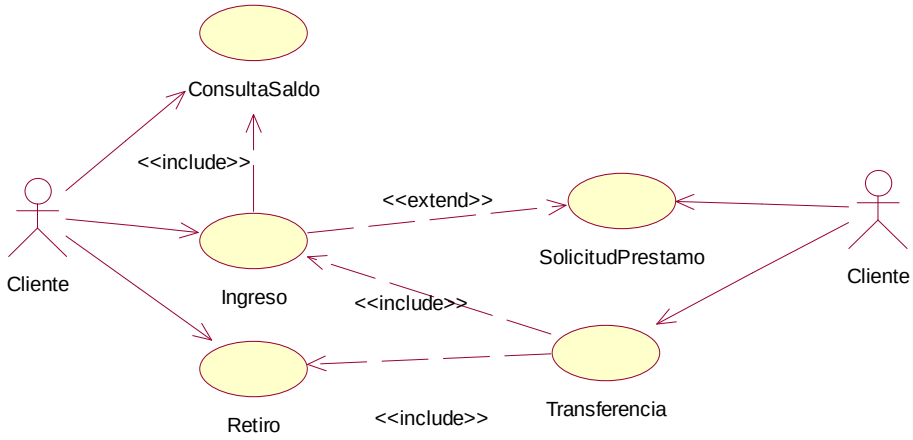


Figura 2.30. Ejemplo de diagrama de casos de uso para transacciones bancarias básicas (Elaborado por los Autores).

- Una vez que se determinan los requerimientos se debe distribuir la funcionalidad del sistema a través de varios componentes como se muestra en la Figura 2.31.

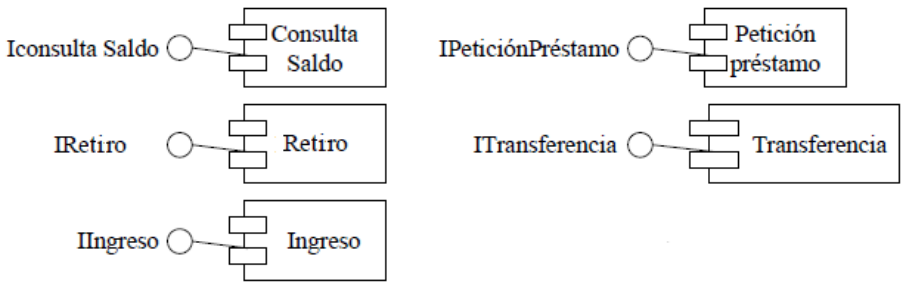


Figura 2.31. Ejemplo de distribución de la funcionalidad en componentes (Elaborado por los Autores).

- Determinar los componentes existentes, los que van a ser desarrollados y los que van a ser seleccionados para su adaptación.

2.5.2 SELECCIÓN DE COMPONENTES

Identificar las necesidades generales del sistema, para luego buscar un grupo de componentes que satisfagan esas necesidades. Para la búsqueda de un componente adecuado se deben identificar sus propiedades (funcionalidad, interfaz, servicios, grado de madurez), en base a los siguientes criterios:

- Herramientas de desarrollo e integración necesarias para el componente.
- Requisitos de ejecución como memoria, almacenamiento, tiempo de respuesta, protocolo de red.
- Requisitos de servicio como interfaces del sistema operativo y soporte de otros componentes.
- Requisitos de seguridad como control de acceso y protocolos de autenticación.
- Manejo de excepciones.

Cuando se quiere evitar las tareas de búsqueda se puede contratar un servicio de bróker de componentes que se dedican a buscar los componentes adecuados, existentes en el mercado, para solucionar un problema específico.

2.5.3 ADAPTACIÓN DE COMPONENTES

Una vez que los componentes son seleccionados, es necesario adaptarlos a las necesidades del sistema. Para ello primero se debe analizar si el componente es de caja blanca, caja gris o caja negra.

- Determinar si existe solapamiento entre componentes. El diseñador debe asegurar que no existan varios componentes realizando la misma función.
- Determinar si los componentes satisfacen todas las necesidades planteadas, caso contrario elegir otros componentes o desarrollar unos propios.

- Examinar cada componente para verificar si existe una mayor funcionalidad que la requerida. De ser así se debe determinar si se añade al sistema la funcionalidad adicional, o si se elimina dicha funcionalidad.
- Detectar incompatibilidades en la arquitectura. En un alto nivel de abstracción, determinar posibles problemas en la responsabilidad de cada componente para el manejo de eventos. En un bajo nivel de abstracción, identificar posibles problemas en la compatibilidad de los lenguajes de programación utilizados.

2.5.4 ENSAMBLAJE DE COMPONENTES

Para ensamblar los componentes de un sistema se utiliza la infraestructura proporcionada por un middleware. Existen dos tecnologías utilizadas: middleware orientado a mensajes (MOM), y la tecnología object request broker (ORB).

Para utilizar la tecnología ORB se deben definir las interfaces de los componentes (utilizando un IDL), luego se deben localizar y activar los servicios remotos, y finalmente se establece la comunicación entre clientes y servicios.

2.6 DISEÑO DE INTERFAZ DE USUARIO

El proceso de diseño de las interfaces de usuario es iterativo y se basa en cuatro actividades descritas a continuación.

2.6.1 MODELO DE ANÁLISIS DE INTERFAZ DE USUARIOS

Este modelo del análisis constituye la base para la posterior elaboración del diseño de las interfaces de usuario. Previamente, el diseñador deberá tener una idea de cómo funcionará el sistema para poder plantear las interfaces. A continuación se especifican los puntos clave a ser tomados en cuenta por parte del diseñador.

2.6.1.1 Análisis y modelado de usuarios

Se analiza el perfil de los usuarios tomando en cuenta el nivel de conocimiento del negocio y la expectativa que tienen del sistema. En base a los perfiles, se

categorizan los usuarios, para cada categoría se determinarán los requisitos generales a ser implementados en el sistema.

2.6.1.2 Análisis y modelado de tareas

En base a los requisitos generales definidos anteriormente, se procede a detallar las tareas que realizan los usuarios actualmente y que van a ser implementadas en el sistema por medio de las interfaces.

2.6.1.3 Análisis del Contenido de Pantalla

De acuerdo a las tareas a realizar por parte de los usuarios se debe identificar el tipo de contenido que se presentará en las interfaces. Así por ejemplo, gráficos, hojas de cálculo, audio o video.

De igual manera se debe especificar la fuente de la que provienen dichos objetos de datos, es decir, de componentes en otras partes de la misma aplicación, de alguna base de datos o transmitidos a la aplicación desde sistemas externos.

2.6.1.4 Modelado del entorno

Se debe analizar el entorno físico en el que se va a implementar el sistema tomando en cuenta aspectos de iluminación y comodidad para su manejo. Además, se debe tomar en cuenta la concurrencia, tiempos de respuesta y las salidas que generarán al realizar ciertos procesos.

2.6.2 DISEÑO DE LA INTERFAZ

En este punto se definen los objetos y acciones que realiza la interfaz y que tienen relación con los escenarios descritos por los usuarios. Además se definen eventos que suceden cuando el usuario realiza acciones que cambian el comportamiento de la interface. A continuación se muestra en la Figura 2.32 el proceso de Diseño de la Interfaz de Usuario.

Figura 2.32. Proceso de Diseño de la Interfaz de Usuario (Elaborado por los Autores).

2.6.3 PROTOTIPO DE LA INTERFAZ

Esta es una actividad iterativa que crea el primer modelo de pantalla de acuerdo a los escenarios descritos por parte de los usuarios y a datos obtenidos en los puntos anteriores. Se recomienda tomar en cuenta los siguientes puntos básicos al momento de realizar el prototipo de las interfaces.

- *Tiempo de Respuesta:* revisar la duración del tiempo desde que el usuario realiza una petición hasta que recibe una respuesta del sistema.
- *Opciones de Ayuda:* se recomienda integrarla en el sistema ya que al ejecutar una acción el usuario pueda consultar en una lista el tópico relacionado a dicho evento.

- *Manejo de Errores:* Desplegar mensajes de error que describan el problema y formulen una solución o indiquen de qué forma afecta la acción realizada por parte del usuario. Es importante que el mensaje este escrito en lenguaje que entienda el usuario.
- *Utilización de Menús y Comandos:* Depende de los perfiles de los usuarios, es decir si un usuario es avanzado se puede implementar ciertas tareas por medio de comandos, caso contrario se deberán emplear menús donde contengan los submenús necesarios para realizar las tareas requeridas.

2.6.4 EVALUACIÓN DE LA INTERFAZ

Una vez realizado el primer prototipo del sistema se procede a su validación por parte del usuario final. La evaluación puede ser realizada de manera cuantitativa o cualitativa, en el primer caso se puede utilizar cuestionarios, y para el segundo caso, se puede analizar acciones y tiempo de espera de los usuarios mientras interactúan con el prototipo del sistema. Sin embargo, la mejor forma de llevar a cabo la validación del prototipo es hablar con el usuario directamente.

En la evaluación esencialmente se deben analizar tres puntos importantes:

- El sistema cumplió con todas las tareas que requiere el usuario.
- El sistema tuvo facilidad en su utilización como en el aprendizaje.
- El sistema fue tomado por parte del usuario como una herramienta útil para las tareas que realiza actualmente.

Una vez que el usuario haya hecho sus observaciones respecto al prototipo, el diseñador las analiza y se procede a implementar un segundo prototipo del sistema con las correcciones respectivas. Se utilizarán los prototipos necesarios hasta alcanzar la funcionalidad requerida por el usuario.

2.7 EVALUACIÓN Y ANÁLISIS DE LA CALIDAD DEL DISEÑO DE SOFTWARE

Una manera de asegurar la calidad del producto de software es a través de métricas, las cuales permitan cuantificar el grado en el cual un sistema, componente o proceso posee un atributo dado. Con esto se logra determinar si el software se ajusta a los requerimientos del cliente y poder tomar decisiones para mejorar el producto final.

En esta propuesta se plantean las siguientes actividades para determinar la calidad en el diseño mediante métricas de software:

- *Formulación:* Determinar las métricas útiles y apropiadas para la etapa de diseño.
- *Colección:* Determinar el mecanismo para obtener los datos necesarios para derivar las métricas planteadas en el paso anterior.
- *Análisis:* En base a los datos obtenidos, calcular las métricas correspondientes y analizarlas dentro de un contexto adecuado.
- *Interpretación:* Interpretar los resultados para obtener una visión interna de la calidad del diseño.
- *Retroalimentación:* En base al análisis e interpretación, determinar recomendaciones para el producto y transmitirlas al equipo de desarrollo.

A continuación se detallan las principales métricas utilizadas en la etapa de diseño. Es recomendable escoger las métricas más significativas, dependiendo el tipo de software que se va a evaluar. Algunas de las métricas descritas se pueden postergar para etapas posteriores al diseño cuando se tengan los datos necesarios para realizar las mediciones.

2.7.1 MÉTRICAS DEL DISEÑO ORIENTADO A LA FUNCIÓN

Estas métricas son de “caja negra”, es decir, se enfocan en las propiedades externas de la arquitectura del sistema. Es recomendable utilizar el diagrama de estructura para el cálculo de las métricas descritas a continuación.

2.7.1.1 Métricas Arquitectónicas

Para determinar la complejidad en arquitecturas jerárquicas (llamada y retorno), se utilizan un conjunto de métricas descritas en la Tabla 2.7.

Métrica	Fórmula	Detalles
Complejidad Estructural de un módulo i	$S(i) = f_{out}^2$	$f_{out}(i)$ es el número de módulos inmediatamente subordinados o invocados directamente por el módulo i.
Complejidad de Datos de un módulo i	$D(i) = \frac{v(i)}{f_{out}(i) + 1}$	$v(i)$ es el número de variables de entrada y salida del módulo i
Complejidad del sistema	$C(i) = S(i) + D(i)$	Es la suma de la complejidad estructural y la de datos
Complejidad de diseño del módulo	$Dc = T * (FE * FS)^2$	Tamaño (T) se mide en líneas de código del módulo, el flujo de entrada (FE) es el total de elementos de datos que fluyen hacia el módulo, mientras que el flujo de salida (FS) es el total de elementos de datos que fluyen hacia otros módulos.

Tabla 2.7. Métricas de Complejidad Arquitectónica

Los valores obtenidos son directamente proporcionales al grado de complejidad arquitectónica del sistema, por consiguiente, si se obtienen valores altos, aumentarán los esfuerzos para integrar los módulos y aumentará la probabilidad de errores de integración.

Para comparar la forma de la arquitectura entre diferentes sistemas se utilizan las métricas descritas en la Tabla 2.8.

Métrica	Fórmula	Detalles
Tamaño	$tamaño = n + a$	n es el número de módulos y a es número de arcos.
Profundidad		El camino más largo desde el nodo raíz hacia un nodo hoja
Anchura		Número máximo de nodos en cualquier nivel de la arquitectura.
Impureza del árbol (I)	$I = n - a - 1$	Es la desviación del árbol de profundidad. Valores negativos indican un mayor acoplamiento, y por tanto mayor complejidad
Relación arco nodo	$r = \frac{a}{n}$	Mide de forma simple el acoplamiento de la arquitectura

Tabla 2.8. Métricas para medir la forma de la arquitectura

2.7.2 MÉTRICAS DEL DISEÑO ORIENTADO A OBJETOS

A diferencia de las métricas orientadas a funciones, donde las mediciones centrales se enfocan a los módulos funcionales del sistema, en las métricas orientadas a objetos las mediciones se enfocan a las clases y objetos.

El tamaño del sistema se puede determinar en base a las líneas de código, como en la orientación funcional, o se puede determinar en función del número de clases y operaciones del modelo de diseño.

2.7.2.1 Métricas Orientadas a Clases CK (Chidamber & Kemerer)

Se propone un conjunto de métricas orientadas a Clases descritas en la Tabla 2.9.

Métrica	Fórmula	Valor Umbral	Sobrepasa valor umbral
Métodos ponderados por clase	$MPC = \sum_{i=1}^{i=n} C_i$	MPC = 40	El método se vuelve difícil de entender, probar o modificar
Profundidad de Árbol de la Herencia		APH = 6	El acoplamiento aumenta, el sistema se vuelve complejo. Beneficioso para la reutilización.
Número de descendientes		NOC = 0	A medida que los valores de esta métrica sean más altos se tiene un mayor grado de reutilización
Acoplamiento entre clases		AEC = 5	Afecta la modularidad del sistema y se hace difícil modificarlo
Respuesta para una clase		RPC = 100	Aumenta la complejidad general del diseño, al igual que el esfuerzo para realizar pruebas
Falta de cohesión en métodos	$FCM = \frac{\left[\frac{1}{a}\sum_{j=1}^{a}\mu(Aj)\right]}{1-m}$ $para\ m > 1$	Valores cercanos a FMC =1	Falta de cohesión, incremento de la complejidad y la probabilidad de errores en el desarrollo

Tabla 2.9. Conjunto de Métricas CK.

- *Métodos ponderados por clase:* esta métrica relaciona el número de métodos existentes en una clase con el grado de complejidad de cada método de tal manera que determinen la complejidad total de la clase. Una clase C puede tener varios métodos definidos dentro de ella, denotados como

$M_1, M_2, M_3, \ldots\ldots, M_n$; cada método tiene su grado de complejidad definido por c_i, que puede ser calculado mediante el tamaño, complejidad de interface, complejidad del flujo de datos, o complejidad ciclomática.

- *Profundidad de Árbol de la Herencia:* esta métrica mide la distancia máxima desde el nodo que representa la clase hasta la raíz del árbol, dentro de una jerarquía de herencia de una clase C.
- *Número de descendientes:* es el número de subclases inmediatamente subordinadas a otra, dentro de una jerarquía de clases.
- *Acoplamiento entre clases:* dos clases están acopladas si los métodos de una de ellas usan los métodos o variables de instancia definidas en la otra clase, esto puede observarse generalmente analizando el código generado.
- *Respuesta para una clase:* mide la cardinalidad del conjunto de respuestas para una clase, es decir, el conjunto de todos los métodos que pueden ser invocados si un mensaje es enviado a un objeto dentro de la clase. De esta forma se puede medir la fuerza del enlace entre clases.
- *Falta de cohesión en métodos:* determina el grado de cohesión de una clase midiendo el número de atributos comunes usados por diferentes métodos, lo cual indica la calidad de la abstracción hecha en la clase. Se tiene un conjunto de métodos $\{M_i\}(i = 1, \ldots., m)$ que acceden a un conjunto de atributos $\{A_j\}(j = 1, \ldots., a)$, y $\mu(Aj)$ representa el número de métodos q acceden a cada atributo.

2.7.2.2 Otras Métricas Orientadas a Objetos

Se proponen métricas adicionales como:

- *Factor de Herencia de Método:* se define como proporción de la suma de todos los métodos heredados en todas las clases entre el número total de

métodos en todas las clases. Se calcula como $MFH = \frac{\sum M_i(C_i)}{\sum M_i(C_i)+\sum M_d(C_i)}$, donde, la sumatoria va desde $i = 1$ hasta $i = Tc$ (total de clases), $M_i(C_i)$ es el número de métodos heredados (y no refinados) en (C_i), $M_d(C_i)$ es el número de métodos declarados en la clase (C_i). El factor obtenido es un indicador del nivel de herencia y por consiguiente el grado de reutilización, a medida q los valores aumentan, aumenta la reutilización. Sin embargo demasiada reutilización del código genera problemas de comprensión y mantenimiento del sistema.

- *Factor de Herencia de Atributo:* proporción del número de atributos heredados entre el número total de atributos. Para el cálculo se utiliza la fórmula del factor de herencia de método, substituyendo los métodos por atributos.
- *Índice de especialización por clase:* determina la medida en la que las subclases redefinen el comportamiento de las superclases. Se calcula como $IEC = \frac{número\ de\ métodos\ redefinidos * anidamiento\ en\ la\ jerarquía}{número\ total\ de\ métodos}$, donde el anidamiento en la jerarquía es el nivel dentro del árbol de herencia de la clase. En niveles más profundos la clase se vuelve más especializada y es complicado modificar su comportamiento. Esta métrica puede ser usada para determinar la calidad en la herencia, ya que a medida que este factor aumenta significa que hay demasiados métodos redefinidos y por tanto las superclases pierden relación con las subclases.

2.7.3 MÉTRICAS DEL DISEÑO BASADO EN COMPONENTES

Estas métricas analizan el acoplamiento, cohesión y complejidad. Se enfocan en las características internas de los componentes por lo que requieren que se conozca su funcionamiento interno. Por esta razón es recomendable desarrollar las mediciones posteriormente a la etapa de codificación.

2.7.3.1 Métrica de Cohesión propuesta por Bieman y Ott

Para calcular esta métrica hay que determinar los siguientes parámetros:

- *Número de elementos:* Son todas las referencias a variables y constantes que se utilizan dentro de un módulo. En el caso que aparezcan varias veces las mismas variables, se deberá considerar a cada una de ellas como un elemento nuevo, y se lo identificará con un subíndice de acuerdo al orden de aparición.
- *Porción de datos:* es un recorrido hacia atrás de un módulo, en donde se determinan los valores de datos que afectan al estado del módulo cuando comienza el recorrido. Se empieza por una variable de resultado (retorno de una función o una variable de salida).
- *Señales de unión:* un elemento en común que aparece en dos o más porciones de datos de un módulo.
- *Señales de superunión:* un elemento en común que aparece en todas las porciones de datos de un módulo.

Una vez definidos los parámetros la cohesión se calcula como:

- *Cohesión funcional fuerte:* $CFF = \frac{número\ de\ señales\ de\ superunión}{número\ de\ elementos}$

- *Cohesión funcional débil:* $CFD = \frac{número\ de\ señales\ de\ unión}{número\ de\ elementos}$

Los resultados oscilarán entre un valor de 0 a 1. A medida que los valores de estas relaciones se acercan a 1 se puede determinar que el módulo posee una alta cohesión, caso contrario el módulo tendrá una baja cohesión, lo que significa que no hay elementos de datos comunes en las porciones de datos y los elementos del módulo no trabajan relacionados hacia una misma función. En la Figura 2.33 se propone un ejemplo para analizar estos conceptos.

```
1 procedure SumaProducto
  (N:integer; var SumN, ProdN:
  integer);
2 var I:integer
3 begin
4  SumN  :=0;
5  ProdN :=1;
6  for I :=1 to N do
7  begin
8   SumN  := SumN +I;
9   ProdN :=ProdN *I;
10  end;
11 end;
```

Figura 2.33. Ejemplo de un módulo de suma y producto (Elaborado por los Autores).

Para la figura anterior se tiene un total de 17 elementos: $\{N_1, SumN_1, ProdN_1, I_1, SumN_2, 0_1, ProdN_2, 1_1, I_2, 1_2, N_2, SumN_3, SumN_4, I_3, ProdN_3, ProdN_4, I_4\}$.

Las porciones de datos se determinan mediante un recorrido hacia atrás, empezando por las variables de resultado, en este caso $SumN, ProdN$, a partir de ellas se determinan todas las variables que influyen en su cálculo: $porcion\ de\ datos\ para\ SumN\ \{SumN_3, SumN_4, I_3, I_2, 1_2, N_2, SumN_2, 0_1, N_1, I_1, SumN_1\}$.

$porcion\ de\ datos\ para\ ProdN\ \{ProdN_3, ProdN_4, I_4, I_2, 1_2, N_2, ProdN_2, 1_1, N_1, I_1, ProdN_1\}$

Las señales de unión aparecen como elementos comunes entre estas dos porciones de datos: $señales\ de\ union\ \{I_2, 1_2, N_2, N_1, I_1\}$.

Las señales de superunión aparecen como elementos comunes entre todas las porciones de datos: $señales\ de\ superunion\ \{I_2, 1_2, N_2, N_1, I_1\}$.

La cohesión funcional fuerte se determina como $CFF = \frac{5}{17} = 0{,}294$

La cohesión funcional débil se determina como $CFD = \frac{5}{17} = 0{,}294$

En este ejemplo ambos valores se aproximan a 0 por lo tanto el módulo tiene una cohesión baja, es decir, el módulo no se especializa en una función específica.

2.7.3.2 Métrica de Acoplamiento propuesta por Dhama

El grado de acoplamiento mide la conectividad entre módulos del sistema, datos globales y el entorno exterior.

El acoplamiento de flujos de datos y control se mide mediante los parámetros:

- d_e= número de parámetros de datos de entrada.
- c_e= número de parámetros de control de entrada.
- d_s= número de parámetros de datos de salida.
- c_s= número de parámetros de control de salida.

El acoplamiento global considera los parámetros:

- g_d= número de variables globales usadas como datos.
- g_c= número de variables globales usadas como control.

El acoplamiento de entorno utiliza los parámetros:

- w= número de módulos que son llamados (dependencia hacia afuera)
- r= número de módulos externos que llaman al módulo (dependencia hacia adentro)

El factor de acoplamiento para un módulo se calcula como $= \frac{k}{M}$; donde k es una constante de proporcionalidad, para motivos de esta tesis tendrá el valor 1,

$M = d_s + (a * c_s) + d_s + (b * c_s) + g_d + (c * g_c) + w + r$; a, b y c son constantes que asumirán un valor empírico, para este estudio el valor de 2.

Valores altos de m_a significan un acoplamiento bajo del módulo, lo cual es una característica deseable.

2.7.4 MÉTRICAS DEL DISEÑO DE INTERFACES

Para la evaluación de interfaces de usuario, es recomendable comparar los prototipos de interfaz obtenidos, con los principios de diseño propuestos por varios autores. Si los prototipos se adaptan a estos principios, se pueden obtener interfaces adecuadas para el usuario. Además, es necesario realizar una retroalimentación con los usuarios finales del producto, ya que ellos serán los que evalúen finalmente la calidad de las interfaces.

Los principios básicos que una interfaz debe cumplir se pueden visualizar en la Tabla 2.10.

Principio	Descripción	Ejemplo

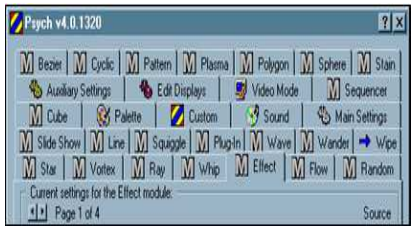
Anticipar	Anticipar acciones cotidianas de usuario para realizarlas de forma más rápida	
Autonomía	Permitir la autonomía en las acciones del usuario, sin sobrecargar la pantalla de elementos innecesarios	
Percepción de Color	Escoger una selección de colores que permitan visualizar claramente la información, tomando en cuenta a las personas con discapacidades visuales	
Consistencia	Los elementos de la interfaz deben guardar consistencia con la acción que representan, con el sistema operativo y con otros elementos estándar ya conocidos en otros programas similares	
Eficiencia de Usuario	Evitar el uso de mensajes o elementos innecesarios que distraigan el flujo del trabajo del usuario, e incrementen el tiempo de trabajo	

Tabla 2.10. Principios de buen Diseño de Interfaz.

Leyes de Fitt

Fitt propone un conjunto de leyes para alcanzar la ergonomía en las interfaces de usuario. Las leyes más importantes se detallan en la Tabla 2.11:

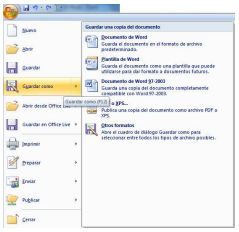
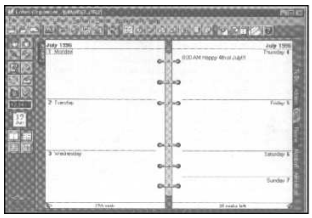
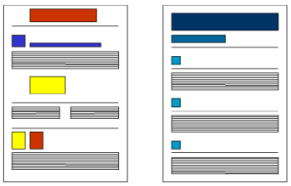
Leyes	Descripción	Ejemplo
Tiempo	Se recomienda utilizar elementos de mayor tamaño y facilidad de alcance para las acciones más comunes	
Interfaces Explorables	Facilitar la exploración y navegación por la interfaz a través de botones que permitan guardar el estado, deshacer acciones, volver a un punto o estado inicial, brindar ayudas.	
Objetos de Interfaz Humana	Simplificar la operación del sistema a través de objetos entendibles, consistentes y estables.	
Uso de Metáforas	Utilizar menús que permitan al realizar acciones en el sistema como las realizaría manualmente facilitando su aprendizaje	
Legibilidad	Uso de texto en alto contraste, fuentes de tamaño adecuado y colores que permitan visualizar fácilmente la información	
Interfaces Visibles	Utilizar el menor número de interfaces para realizar una acción.	

Tabla 2.11. Leyes de Fitt

2.8 DOCUMENTACIÓN DE SALIDA

2.8.1 DOCUMENTO DE DESCRIPCIONES DE DISEÑO DE SOFTWARE

Como resultado del proceso de diseño se obtendrá un documento que presente organizada y detalladamente la información y aspectos del diseño, y pueda ser comunicado a los distintos “stakeholders” del sistema.

Por motivos de estudio, y en concordancia con los estándares de la IEEE, se utilizará el estándar IEEE1016-2009 “Estándar para la Tecnología de la Información-Diseño de Sistemas-Descripciones del Diseño de Software (SDD)”.

Este estándar está organizado mediante puntos de vista y vistas de diseño (ver Capítulo 1.3.1), las cuales muestran diferentes características del sistema de acuerdo a los intereses del tipo de usuario a las que están dirigidas.

A continuación se describen las partes fundamentales que deberá tener el documento de descripciones de diseño de software en su estructura.

- *Portada:* En la portada del documento deberá constar un membrete indicando los siguientes campos:
  - Fecha de Emisión y Estado.
  - Organismo Emisor.
  - Autores.
  - Historial de Cambios.
- *Alcance:* proporcionar una descripción del sistema, en términos generales, indicando las metas, objetivos y beneficios del sistema.
- *Propósito:* identificar el propósito del SDD. Se puede comenzar con las líneas “Este documento de Diseño de Software describe la arquitectura y diseño detallado del software XXX…”.
- *Contexto:* determinar el contexto del sistema con el que se está trabajando, especificar cómo se relaciona el sistema con el entorno.

- *Stakeholders:* identificar los individuos, organizaciones o grupos que tengan algún tipo de interés en el diseño del sistema, y hacia los cuales va dirigido el documento. Una vez identificados los Stakeholders se debe especificar el tipo interés de cada uno de ellos en los aspectos del diseño.
- *Vistas:* el SDD debe estar organizado de acuerdo a vistas de diseño. El propósito de las vistas de diseño es permitir que los Stakeholders se enfoquen sólo en determinadas propiedades del sistema, de acuerdo a sus intereses.
- *Puntos de Vista:* determinar los puntos de vista para cada vista de diseño. Los puntos de vista deberán estar descritos como se indica a continuación:
  - Nombre del punto de vista.
  - Enfoques de Diseño: tópicos del punto de vista.
  - Entidades de diseño: son los elementos clave del diseño, por ejemplo, sistemas, subsistemas, librerías, módulos, componentes, clases, almacén de datos, programas, procesos.
  - Elementos de Diseño: conformados por cualquier ítem que aparezca dentro de un punto de vista, por ejemplo, una relación, un atributo o una restricción.
  - Atributos de Diseño: son las propiedades o características de los elementos de diseño. Entre los principales atributos constan el nombre, el tipo, el propósito y el autor.
  - Relaciones de Diseño: nombra una asociación entre dos entidades de diseño.
  - Restricciones de Diseño: reglas o restricciones impuestas desde un elemento de diseño hacia otro.
- *Justificación del Diseño:* describir brevemente las decisiones de diseño tomadas en cuanto a estilos arquitectónicos, vistas, puntos de vista, y otros criterios tomados.

Los puntos de vista recomendados se incluyen en la Tabla 2.12, que se presenta a continuación. No es necesario utilizar todos los puntos de vista descritos.

Punto de Vista de Diseño	Tópicos de Diseño	Lenguajes de Diseño
Contexto	Servicios del Sistema. Usuarios.	-UML Diagrama de casos de uso. -Diagrama de Contexto del Flujo de Datos en el Análisis Estructurado
Composición: - Funcional (lógico) -Tiempo de Ejecución (Físico)	Composición y ensamblaje de módulos en términos de subsistemas y componentes.	-Lógico: UML diagrama de Paquetes. UML Diagrama de Componentes. Lenguajes de Descripción Arquitectónica. Diagrama de Estructura. -Físico: UML Diagrama de Despliegue.
Lógico	Estructuras Estáticas(Clases, interfaces y sus relaciones)	-UML Diagrama de Clases. -UML Diagrama de Objetos.
Información con Distribución de Datos.	Información Persistente.	-Diagrama Entidad-Relación. -UML Diagrama de Clases.
Patrones	Reutilización de Patrones	- UML diagrama de Colaboración
Interface	Cooperación e interacción entre entidades para lograr interfaces.	-Interfaces Gráficas de Usuario
Estructura	Constitución interna de componentes y clases..	-UML Diagrama de Estructura. -UML Diagrama de Clases.
Interacción	Comunicación y paso de mensajes entre objetos.	-UML Diagrama de Secuencia. -UML Diagrama de Comunicación. -UML Diagrama de Actividades.
Dinámico de Estado	Transformación dinámica de estados.	-UML Diagrama de Estado de Máquina. -Matriz Tabla de Transición de Estado.
Algoritmo	Lógica Procedimental	-Tablas de decisión. -Diagrama de Warnier -Pseudocódigo
Recursos	Utilización de Recursos	-UML Diagrama de Clases. -UML Lenguaje de Restricción de objetos

Tabla 2.12. Puntos de Vista propuestos por la IEEE 1016-2009.

A continuación se describen los elementos de diseño para cada punto de vista mencionados anteriormente:

2.8.1.1 Punto de Vista de Contexto

Representa los servicios provistos por el sistema con respecto a actores y stakeholders que interactúan con el sistema dentro de su ambiente. Esta perspectiva es de caja negra, es decir sólo presenta la organización y características externas del sistema.

- *Entidades:* Actores (usuarios, stakeholders, sistemas externos), Servicios (Casos de uso, flujos de información entre actores y el sistema).
- *Relaciones:* Entradas y Salidas entre actores y el sistema.

- *Restricciones:* Calidad del Servicio, forma y medio de interacción con el entorno.

2.8.1.2 Punto de Vista de Composición

Describe la forma en la que el sistema se encuentra estructurado en partes, y determina los roles de cada parte. Este punto de vista es utilizado por desarrolladores y mantenedores para identificar las partes que conforman el sistema, y distribuir la funcionalidad, roles y responsabilidades, dentro de ellas.

- *Entidades:* subsistemas, componentes, módulos, librerías, marcos de trabajo, repositorios, funciones, catálogos y plantillas.
- *Relaciones:* realización, dependencia, agregación, composición y generalización.
- *Atributos:* Para cada entidad especificar una descripción detallada; indicando identificación, tipo, propósito, función y definición.

2.8.1.3 Punto de Vista Lógico

Representa las relaciones entre entidades de diseño dentro de una estructura estática. Direcciona el desarrollo y reutilización de abstracciones adecuadas y su implementación, por ejemplo mediante clases e interfaces.

- *Entidades:* Clases, interfaces, tipos de datos, objetos, atributos, métodos, clases de asociación, plantillas, namespace.
- *Relaciones:* Asociación, generalización, dependencia, realización, implementación, instancia de, composición y agregación
- *Atributos:* Nombre, rol, visibilidad, cardinalidad, tipo, estereotipo, valor agregado y parámetro.
- *Restricciones:* Restricciones de valor, relaciones exclusivas, valores iniciales, precondición, postcondición

2.8.1.4 Punto de Vista de Dependencia

Especifica las relaciones de interconexión y acceso entre entidades. Estas relaciones incluyen información compartida, orden de ejecución o parametrización de interfaces.

- *Entidades:* Subsistemas, componentes, módulos
- *Relaciones:* Usos, ofertas, requerimientos.
- *Atributos:* Para cada entidad describir: Nombre, tipo, propósito, dependencias, recursos.

2.8.1.5 Punto de Vista de Información

Se utiliza este punto de vista cuando existe un gran contenido de datos persistentes en el sistema. Se incluyen estructuras de datos persistentes, contenido de datos, estrategias para el manejo de datos, esquemas y definición de metadatos.

- *Entidades:* Elementos de datos, tipos y clases, almacenamiento de datos, mecanismos de acceso.
- *Relaciones:* Asociación, usos, implementación. Atributos de los datos, sus restricciones y relaciones estáticas.
- *Atributos:* Método de representación, valor inicial, formato, valores aceptables.

2.8.1.6 Punto de Vista Uso de Patrones

Representa la reutilización a nivel de patrones de diseño, estilos arquitectónicos y plantillas de marcos de trabajo.

- *Entidades:* Colaboraciones, clases, conectores, roles, plantillas de marcos de trabajo, patrones.
- *Relaciones:* Asociación, uso de colaboraciones, conector.
- *Atributos:* Nombre.
- *Restricciones:* Restricciones de colaboraciones.

2.8.1.7 Punto de Vista de Interface

Provee información a diseñadores, programadores y encargados de pruebas acerca del uso correcto de los servicios provistos por el sistema. Esta descripción incluye información de las interfaces internas y externas.

- *Entidades:* interfaces, funciones.
- *Relaciones:* Mecanismos de invocación e interrupción, comunicación a través de parámetros.
- *Atributos:* Descripción de rangos en entradas, significado de entradas y salidas, formato de cada entrada y salida, códigos de errores de salida.
- *Restricciones:* Protocolos de comunicación, formato de datos, valores aceptables y significado de cada valor.

2.8.1.8 Punto de Vista de Estructura

Documenta la constitución interna y organización del sistema en términos de elementos comunes.

- *Entidades:* Puerto, conector, interface, parte y clase.
- *Relaciones:* Conectado, parte de, englobado, provisto y requerido.
- *Atributos:* Nombre, tipo, propósito, definición.
- *Restricciones:* Restricciones de interfaces, reutilización y dependencia.

2.8.1.9 Punto de Vista de Interacción

Define estrategias de interacción entre entidades. Para diseñadores esto incluye evaluar la ubicación de las responsabilidades dentro de las colaboraciones.

- *Entidades:* Clases, métodos, estados, eventos, jerarquía, concurrenia y sincronización.

2.8.1.10 Punto de Vista Dinámico de Estado

Útil para sistemas reactivos, dónde hay respuestas a eventos; y para sistemas dinámicos, que incluyen modos, estados y transiciones.

- *Entidades:* Evento, condición, estado, transición, actividad, trigger.
- *Relaciones:* Parte de, interno, efecto, entrada, salida, anexo a.
- *Atributos:* Nombre, activo, inicial, final.
- *Restricciones:* Concurrencia, sincronización, estados invariantes, restricciones de transición, protocolos.

2.8.1.11 Punto de Vista Algorítmico

Descripción de diseño detallado de operaciones, detalles internos y lógica de las entidades de diseño.

- *Entidades:* Eventos, procesos, datos.
- *Relaciones:* Identificación, procesamiento y datos de todas las entidades de diseño.
- *Atributos:* Tiempos, secuenciamiento, pre-requisitos para iniciación de procesos, prioridad de eventos.

A continuación se detalla una plantilla que resume la distribución del SDD, como se puede observar en la Figura 2.34. Esta plantilla puede ser modificada de acuerdo a las necesidades de los desarrolladores, sin embargo debe cumplir con las pautas básicas marcadas anteriormente.

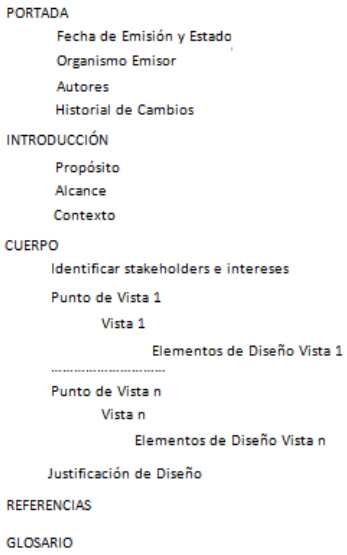


Figura 2.34. Plantilla de Contenido para un SDD aplicando el estándar IEEE-1016-2009

3 CAPITULO III: APLICACIÓN DEL PROCESO DE DISEÑO DE SOFTWARE A UN CASO DE ESTUDIO

3.1 DESCRIPCIÓN DEL CASO DE ESTUDIO

Para la aplicación del proceso de diseño se ha seleccionado la empresa PAPYRUS, la cual cuenta con tres sucursales en la ciudad de Quito.

PAPYRUS es una empresa dedicada a la comercialización de materiales e insumos de papelería. Además dicta cursos de capacitación, dirigidos a clientes, para la elaboración de manualidades en base a los productos que vende la empresa. En la actualidad el control y la administración de la empresa se llevan de forma manual, sin tener un correcto registro de productos, personal, clientes y ventas.

Con estos antecedentes, se plantea un sistema denominado e-papyrus cuya funcionalidad es automatizar el pedido de productos, inscripción para cursos didácticos, gestión de clientes y gestión de bodega para los productos que se expenden.

3.2 APLICACIÓN DE LA PROPUESTA

3.2.1 DOCUMENTACIÓN DE ENTRADA

3.2.1.1 Especificación de Requerimientos

En base al documento SRS (Especificación de Requerimientos de Software), que se encuentra en el Anexo A, se han extraído los requerimientos del sistema e- papyrus.

3.2.1.1.1 Requerimientos Funcionales

- *Gestionar información de clientes:* crear, modificar, eliminar registros de clientes.
- *Gestionar información del personal de la agencia:* crear, modificar, eliminar registros del personal.

- *Autenticar usuarios:* el sistema deberá solicitar la autenticación de usuario, y dependiendo del perfil, deberá habilitar las funciones del sistema. Se manejan los perfiles de cajero, bodeguero, y administrador.
- *Gestionar información de productos:* crear, modificar, eliminar registros de productos.
- *Gestionar información de cursos:* crear, modificar, eliminar registros de cursos de capacitación.
- *Mostrar catálogo de productos:* el sistema mostrará a los usuarios un catálogo de productos disponibles para la venta. Se podrá visualizar el catálogo general de productos, o buscar un producto específico.
- *Mostrar información de cursos de capacitación:* el sistema mostrará a los usuarios los cursos de capacitación disponibles. Se podrá visualizar el catálogo general de cursos, o buscar un curso específico.
- *Controlar stock de productos:* el sistema emitirá una alerta al bodeguero cuando la cantidad disponible de un producto específico sea inferior o igual a diez (10) unidades.
- *Matricular clientes en los cursos de capacitación:* el sistema deberá matricular a los clientes registrados en los cursos gratuitos de capacitación. EL cupo máximo de clientes por horario de un curso es de quince personas (15).
- *Facturar productos:* el sistema deberá procesar las compras que realicen los usuarios registrados, y emitir una factura de los productos adquiridos

3.2.1.1.2 Requerimientos no Funcionales

- *Número de Terminales Soportadas:* la empresa dispone de 10 terminales, por cada agencia, para el uso del sistema e-papyrus; por lo cual el sistema deberá ser instalado en dichas terminales.
- *Número de Usuarios simultáneos:* se estima que el número de usuarios conectados simultáneamente va a ser de 7 personas, por lo cual el sistema deberá manejar como mínimo este valor.

- *Cantidad y tipo de información para ser manejada:* el sistema maneja datos alfa numéricos e imágenes para los registros, por lo cual la base de datos también debe manejar estos parámetros. La cantidad de información procesada por día será variable dependiendo de las transacciones que se realicen. Como referencia se tomará el valor de 100 transacciones diarias.
- *Requerimientos lógicos de la base de datos*: la base de datos puede ser implementada en cualquier herramienta que el desarrollador decida, sin embargo, debido a la familiarización del personal de la empresa PAPYRUS, se sugiere que se utilice la herramienta Microsoft SQL Server.
- *Restricción de Hardware*: el sistema no deberá requerir equipos de hardware de muy altas prestaciones para su funcionamiento. Deberá funcionar sobre los equipos disponibles en la agencia PAPYRUS.
- *Restricción de Software*: el sistema funcionará sobre equipos con sistema operativo WINDOWS en sus versiones XP, VISTA o Siete.
- *Fiabilidad:* el sistema deberá realizar correctamente los cálculos, operaciones, transacciones y funciones especificadas, el 99% del tiempo que se lo utilice. Ante cualquier evento o falla, el sistema deberá emitir una alerta que informe al usuario el motivo del error.
- *Disponibilidad:* el sistema deberá estar activo y disponible cuando el usuario lo requiera. Deberá tener una disponibilidad de 24/7 es decir las veinticuatro horas del día, los 7 días de la semana. La disponibilidad aceptable por mes es del 99%.
- *Seguridad:* el sistema deberá administrar distintos perfiles de usuario, de tal manera que el usuario sólo pueda acceder a la información que le corresponde. Se deberá proteger el acceso a la base de datos mediante el manejo de contraseñas. Adicionalmente se debe activar el registro de auditoría en la base de datos para registrar los accesos y operaciones realizadas.
- *Capacidad de Mantenimiento:* el sistema debe brindar apertura para que, en caso de ser necesario, se realicen tareas de mantenimiento. Esta apertura

implica que el diseño del sistema permita la mejora y optimización del sistema, sin afectar su funcionalidad básica.

- *Portabilidad:* El sistema podrá ser instalado en equipos que funcionen bajo el sistema operativo Windows (XP, Vista, 7). Los datos deberán guardarse en formatos estándar que permitan su portabilidad.

3.2.1.2 Documentación del Modelo de Análisis

En base a los requerimientos planteados, el caso de estudio se ha enfocado dentro la perspectiva de clases y objetos, por lo cual se utilizarán los modelos basados en escenarios y los modelos basados en clases como documentación de entrada en la etapa del análisis.

3.2.1.2.1 Modelo Basado en Escenarios

Con la ayuda del usuario se han extraído los requerimientos del sistema en base a experiencias, y posibles situaciones que el sistema enfrentará. Por esta razón se va a utilizar un modelado donde se especifica la interacción de cada usuario con el sistema. A continuación se muestran los diagramas de casos de uso más significativos del sistema e-papyrus, los demás diagramas se encuentran en el Anexo B.

Diagramas de Casos de Uso

Matriculación

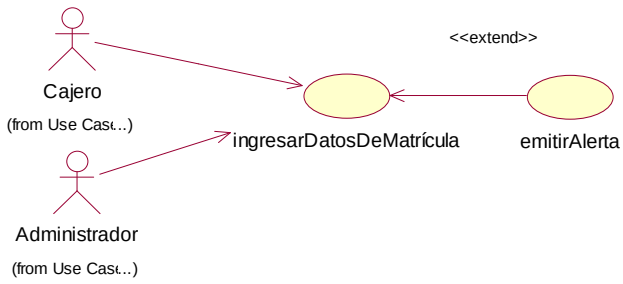


Figura 3.1. Caso de Uso – Matriculación (Elaborado por los Autores).

Facturación

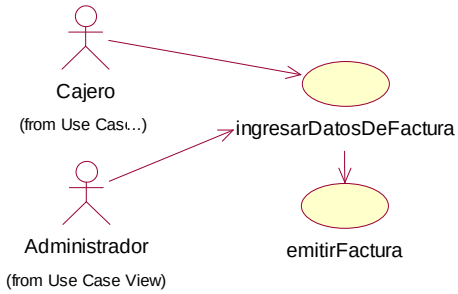


Figura 3.2. Caso de Uso – Facturación (Elaborado por los Autores).

Gestión de Productos

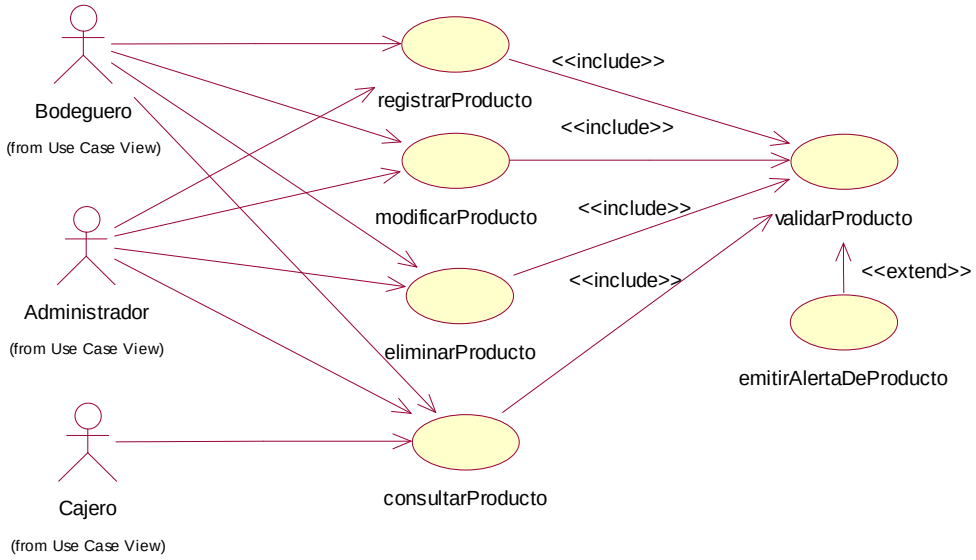


Figura 3.3. Caso de Uso – Gestión de Productos (Elaborado por los Autores).

3.2.1.2.2 Modelo Basado en Clases

A continuación se analiza el dominio del problema en función del diagrama de clases de análisis.

Autenticación de Usuarios

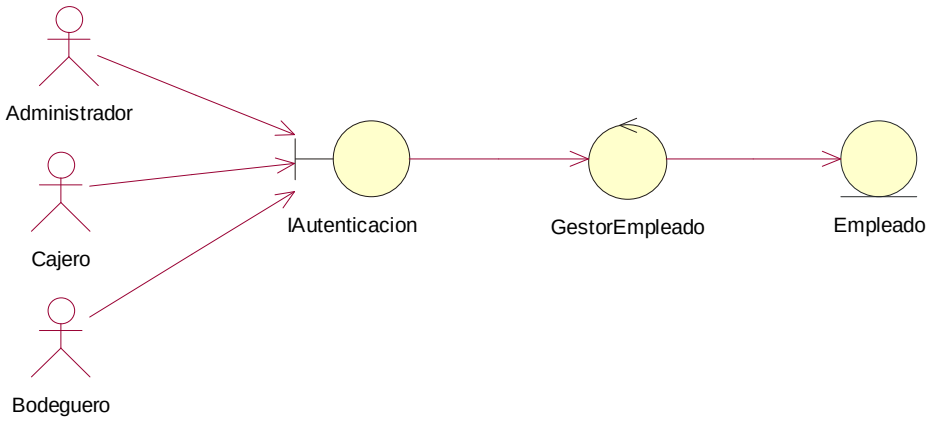


Figura 3.4. Modelo de Clases – Autenticación de Usuarios (Elaborado por los Autores).

Gestión de Productos

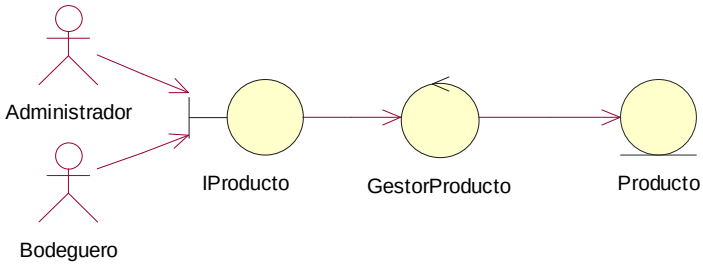


Figura 3.5. Modelo de Clases – Gestión de Productos (Elaborado por los Autores).

Gestión de Clientes

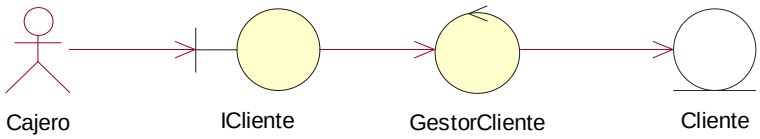


Figura 3.6. Modelo de Clases – Gestión de Clientes (Elaborado por los Autores).

Gestión de Usuarios

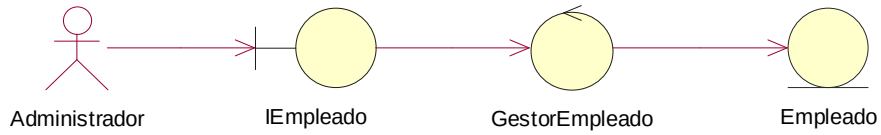


Figura 3.7. Modelo de Clases – Gestión de Usuarios (Elaborado por los Autores).

Gestión de Cursos

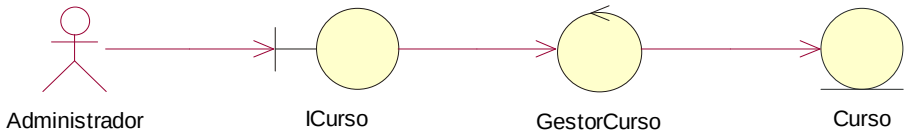


Figura 3.8. Modelo de Clases – Gestión de Cursos (Elaborado por los Autores).

Catálogo de Cursos

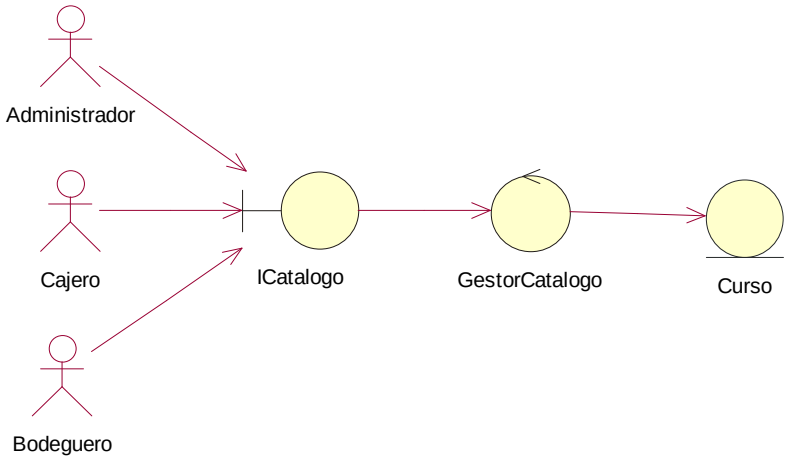


Figura 3.9. Modelo de Clases – Catálogo de Cursos (Elaborado por los Autores).

Controlar Stock de Productos

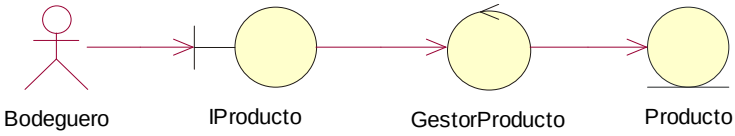


Figura 3.10. Modelo de Clases – Controlar Stock de Productos (Elaborado por los Autores).

Matriculación

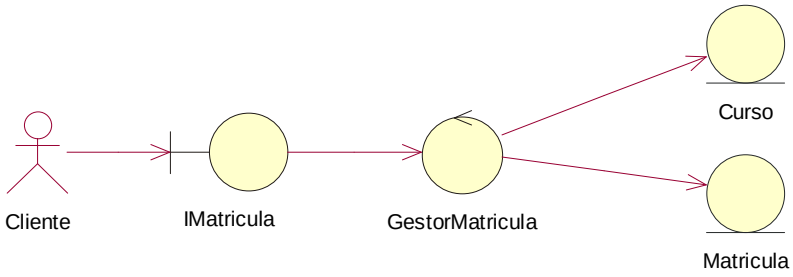


Figura 3.11. Modelo de Clases – Matriculación (Elaborado por los Autores).

Facturación

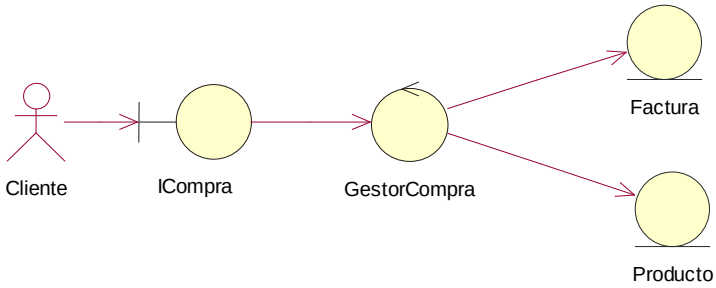


Figura 3.12. Modelo de Clases – Facturación (Elaborado por los Autores).

3.2.2 SELECCIONAR ESTRATEGÍAS Y MÉTODOS

El sistema e-papyrus es un sistema de información, donde lo más importante son los datos y las operaciones que se manejan; por tal razón, un enfoque orientado a objetos es adecuado para modelar la solución del problema.

Se propone al Proceso Unificado como método para el diseño orientado a objetos (Ver capítulo 2.2.2). Dentro de este proceso se utilizará la estrategia iterativa e incremental, a través de la cual se obtendrá el sistema final mediante sucesivas

iteraciones que refinen el modelo de la solución. El lenguaje utilizado para describir los modelos del sistema e-papyrus será el Lenguaje Unificado de Modelado (UML).

3.2.3 SELECCIÓN DEL ESTILO ARQUITECTÓNICO

Para el desarrollo de este sistema se utilizará el estilo arquitectónico Orientado a Objetos, utilizando elementos como clases, objetos y mensajes (ver capítulo 2.4.1). Sin embargo, se puede utilizar simultáneamente un estilo arquitectónico de tres niveles (ver capítulo 1.3.2), donde se tenga la capa de presentación, aplicación y datos.

3.2.4 SELECCIÓN DE PUNTOS DE VISTA

Para modelar la solución del sistema e-papyrus se usarán los puntos de vista propuestos por UML (ver capítulo 2.4.2), los cuales utilizarán los siguientes diagramas:

- *Vista de Casos de uso:* diagramas de casos de uso, diagramas de actividades.
- *Vista de Diseño:* diagrama de clases, diagrama de objetos, diagramas de actividades o diagramas de secuencia.
- *Vista de implementación:* diagrama de componentes, diagrama de secuencia.
- *Vista de despliegue:* diagramas de despliegue.
- *Vista de datos:* diagrama entidad relación.

3.2.5 DISEÑO ARQUITECTÓNICO

El caso de estudio del sistema e-papyrus se enfocará en la primera iteración de la etapa de elaboración dentro del Proceso Unificado, donde se debe obtener una arquitectura de alto nivel, que sirva como base para futuras iteraciones. Se seguirán los pasos descritos en el capítulo 2.4.3.

A partir de la documentación de análisis propuesta en el capítulo 3.2.1.2, se establecieron las clases, atributos y operaciones que puedan modelar una solución al problema de diseño, estas clases se interrelacionan formando el diagrama de clases de diseño, como se puede observar en la Figura 3.13.

3.2.5.1 Diagrama de Clases

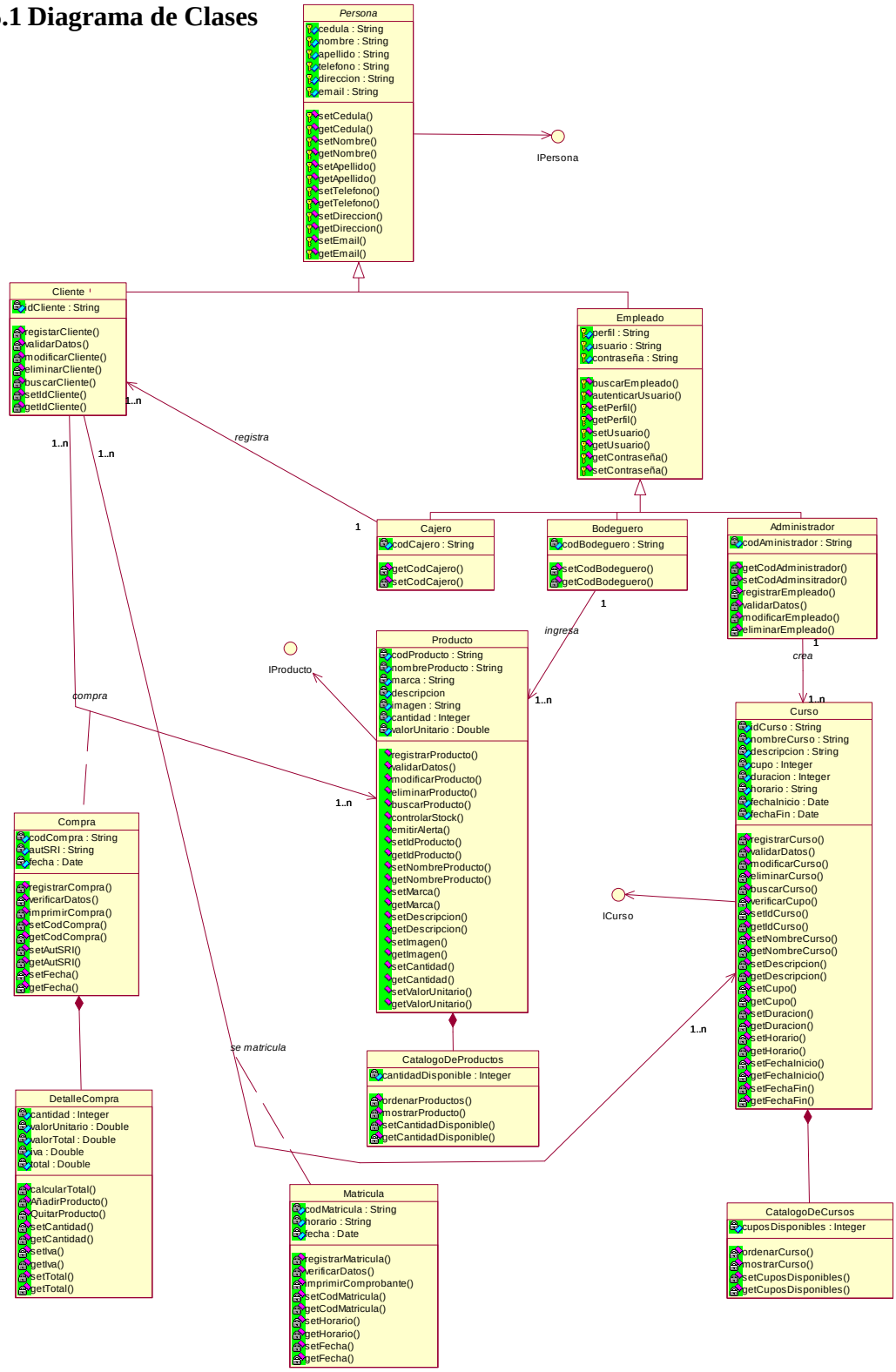


Figura 3.13. Diagrama de Clases de Diseño (Elaborado por los Autores).

3.2.5.2 Subsistemas e-papyrus

Mediante el diagrama de clases de diseño se pudieron determinar las clases que se interrelacionan entre sí y que tienen una funcionalidad común.

Las clases interrelacionadas entre sí formarán los subsistemas de e-papyrus de acuerdo a la Figura 3.14 descrita a continuación.

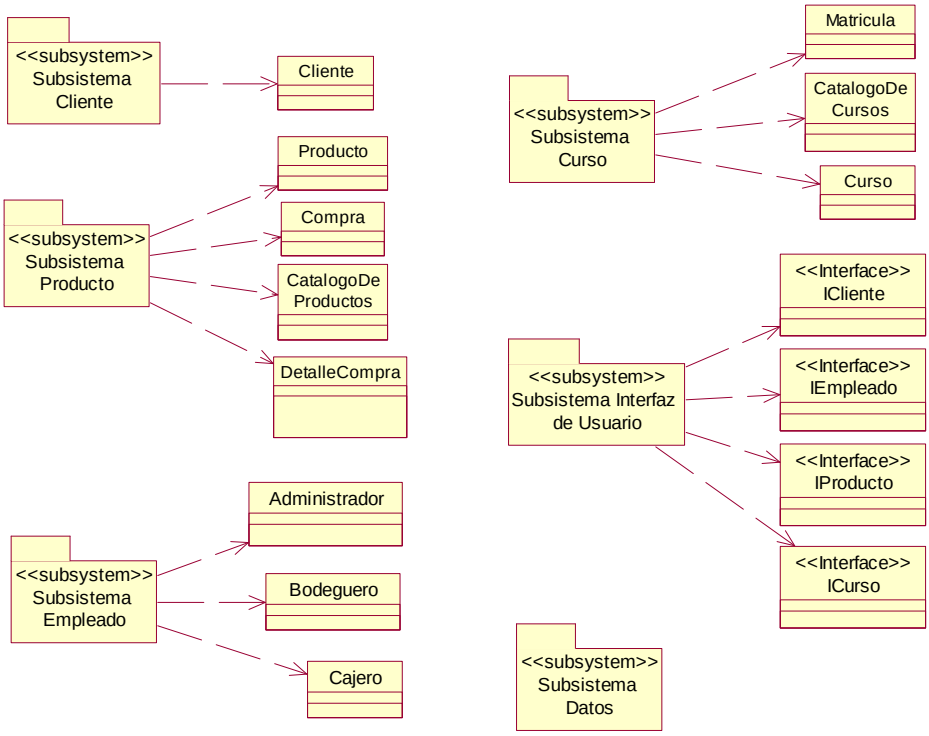


Figura 3.14. Clasificación de Clases de Diseño en Subsistemas Funcionales (Elaborado por los Autores).

3.2.5.3 Estratificación de Capas

Para el sistema e-papyrus se determinaron tres capas principales, capa de presentación, capa de aplicación y capa de datos, como se puede observar en la Figura 3.15.

- *Capa de Presentación:* esta capa estará relacionada directamente con los usuarios de e-papyrus, y será la encargada de recolectar los datos de entrada, enviarlos a la capa de aplicación, y presentar los datos de salida.

- *Capa de Aplicación:* en esta capa residen las clases relacionadas con el modelo del negocio, y es la encargada de recibir los datos de la capa de presentación, procesar los datos, enviar una respuesta y, si es necesario, enviarlos a la capa de datos para su almacenamiento.
- *Capa de Datos:* será la encargada de almacenar los datos persistentes, y presentarlos cuando otras capas lo requieran.

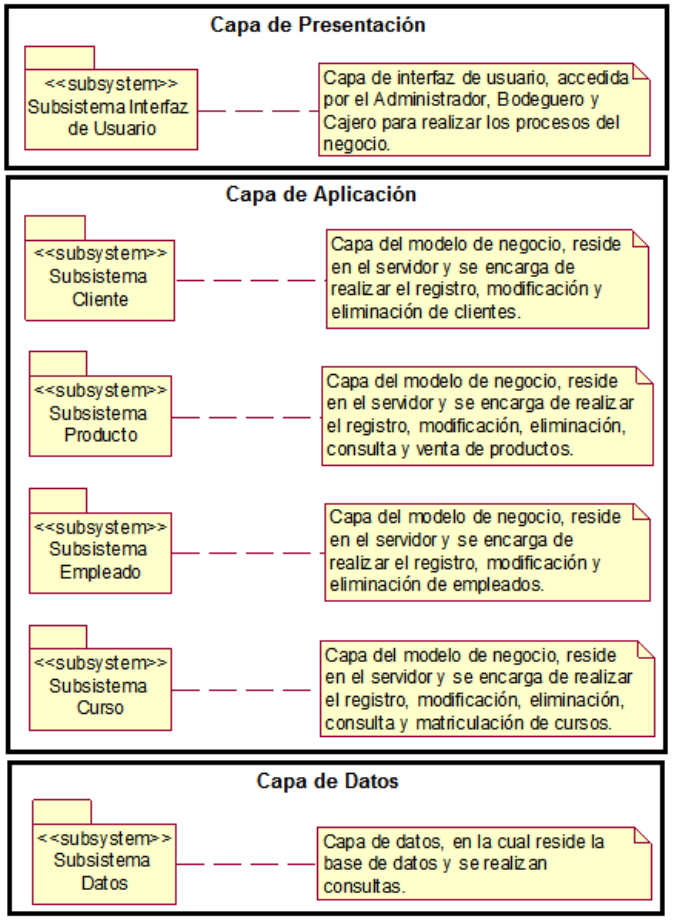


Figura 3.15. Clasificación de Subsistemas por Capas (Elaborado por los Autores).

3.2.5.4 Diagrama de Componentes

Una vez que se han determinado los subsistemas de e-papyrus, a continuación, se especifica la organización de los componentes dentro del sistema, como se muestra en la Figura 3.16.

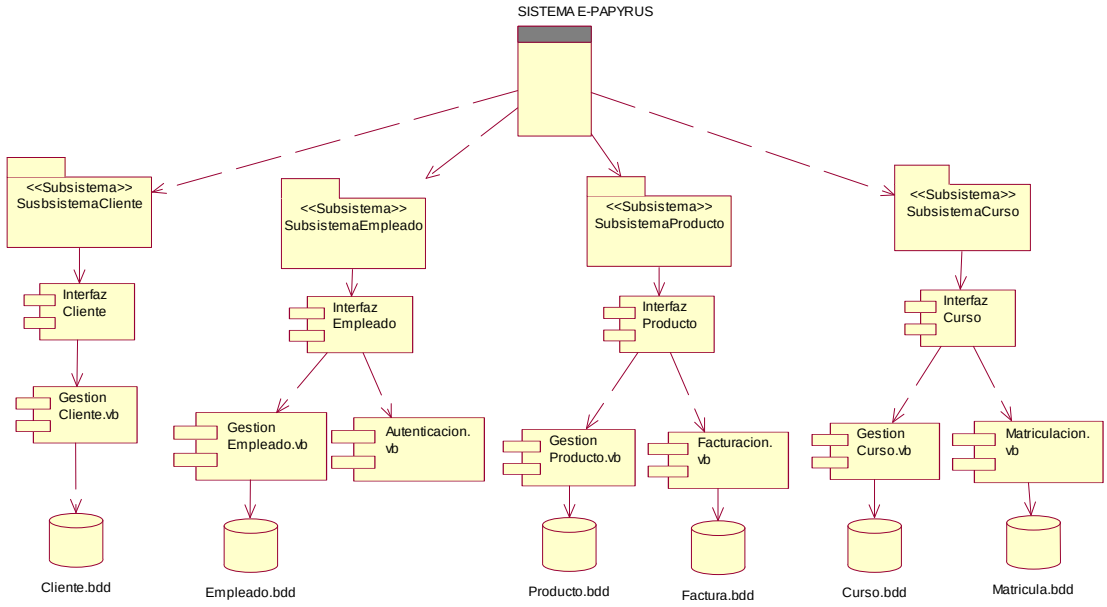


Figura 3.16. Diagrama de Componentes – Sistema e-papyrus (Elaborado por los Autores).

3.2.5.5 Diagrama de Despliegue

Para brindar una perspectiva más clara de la implementación física del sistema, y los elementos de hardware necesarios, se utilizará el diagrama de despliegue mostrado en la Figura 3.17.

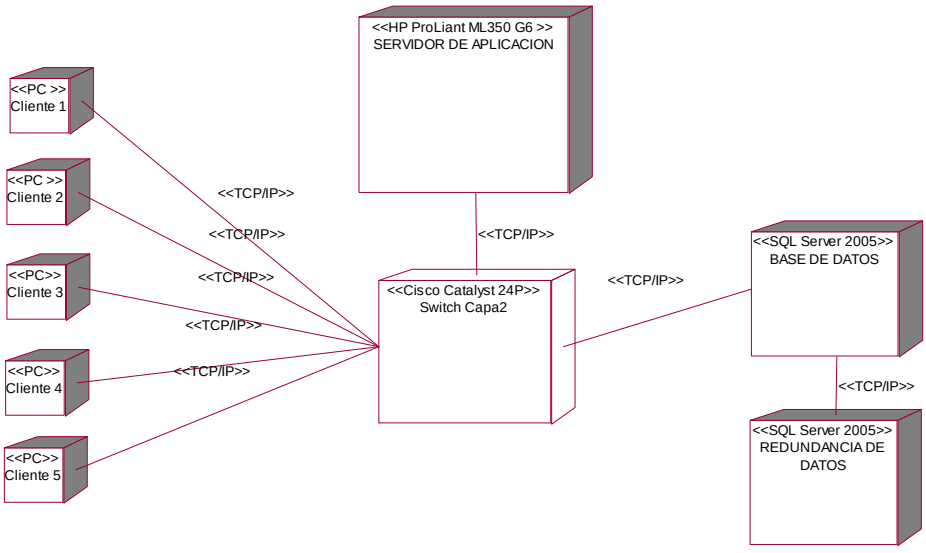


Figura 3.17. Diagrama de Despliegue – Sistema e-papyrus (Elaborado por los Autores).

3.2.6 DISEÑO DETALLADO

3.2.6.1 Diagrama de Objetos

Una vez definida la arquitectura del sistema, se realiza un diagrama de objetos para especificar de manera más detallada aquellas clases que necesiten ser instanciadas para su comprensión. A continuación se detallan las clases más significativas.

Facturación

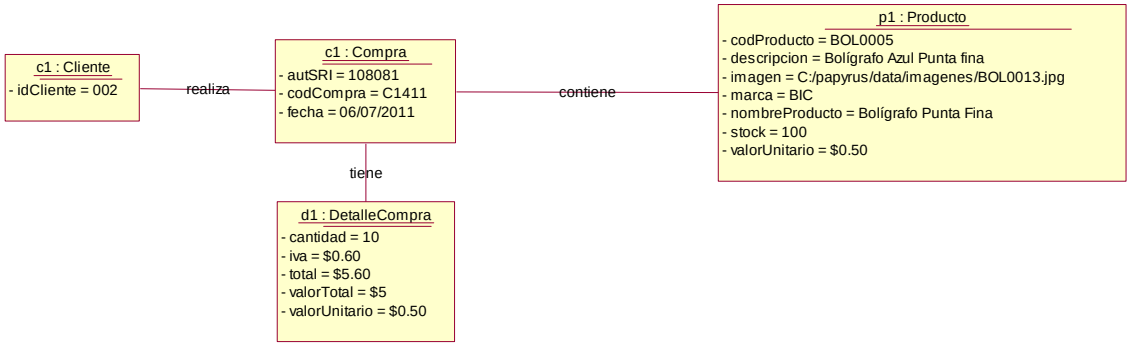


Figura 3.18. Diagrama de Objetos – Facturación (Elaborado por los Autores).

Matriculación

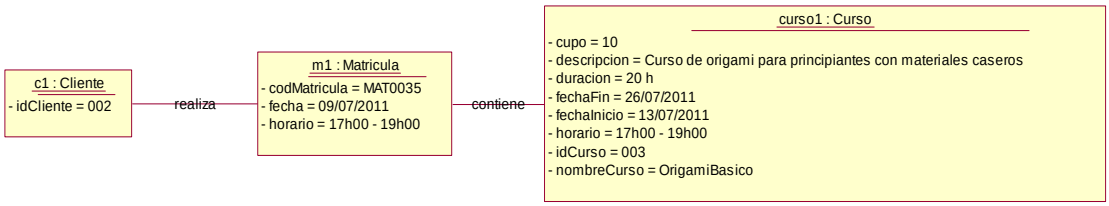


Figura 3.19. Diagrama de Objetos – Matriculación (Elaborado por los Autores).

Crear Producto

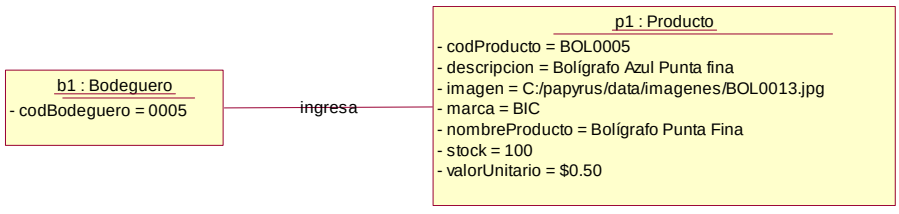


Figura 3.20. Diagrama de Objetos – Crear Producto (Elaborado por los Autores).

3.2.6.2 Diagrama de Actividades

A continuación se presentarán los diagramas de actividad de los casos de uso más relevantes expuestos anteriormente, los demás diagramas de actividad se encuentran colocados en el Anexo B.

Gestión de Productos – Registrar Producto

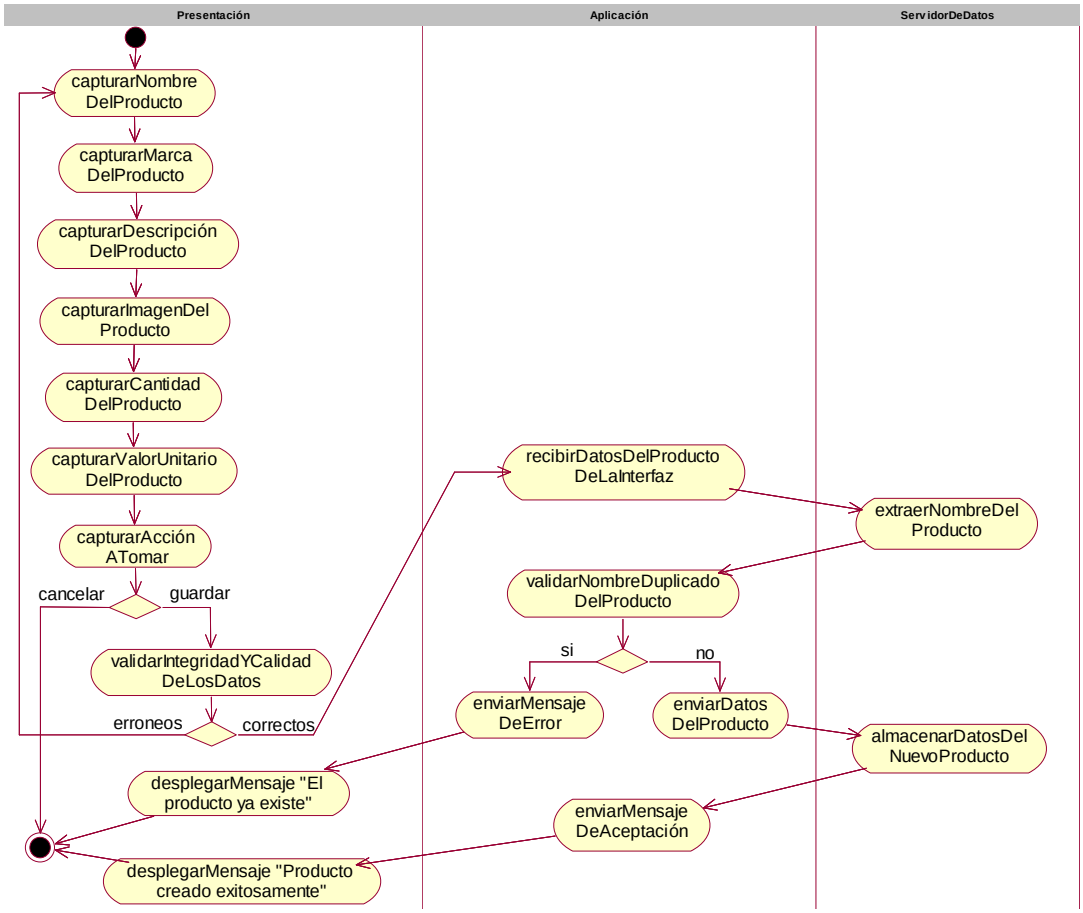


Figura 3.21. Diagrama de Actividades – Registrar Usuario (Elaborado por los Autores).

Facturación – Ingresar Datos de Factura

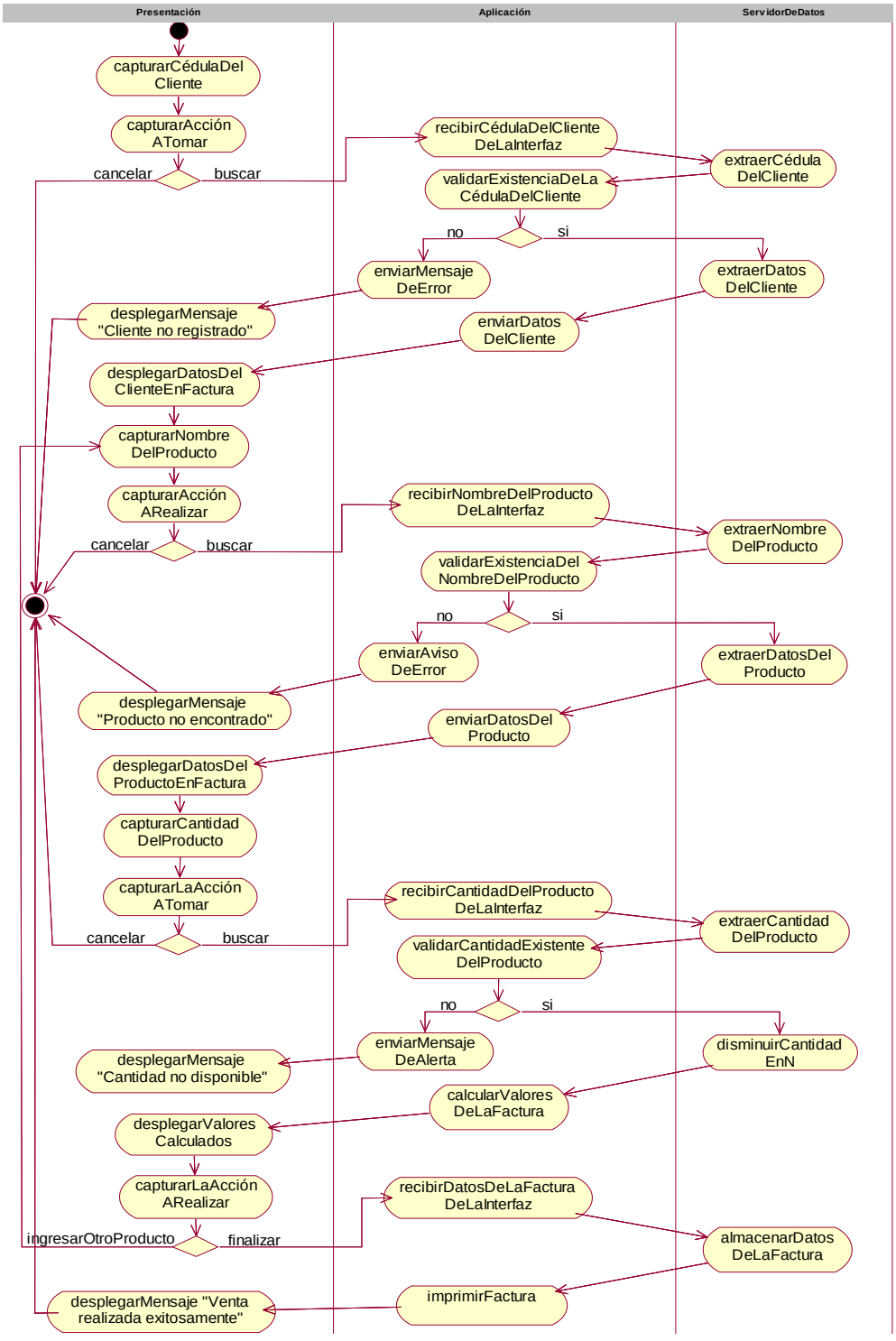


Figura 3.22. Diagrama de Actividades – Ingresar Datos de Factura (Elaborado por los Autores).

Matriculación – Ingresar Datos de Matrícula

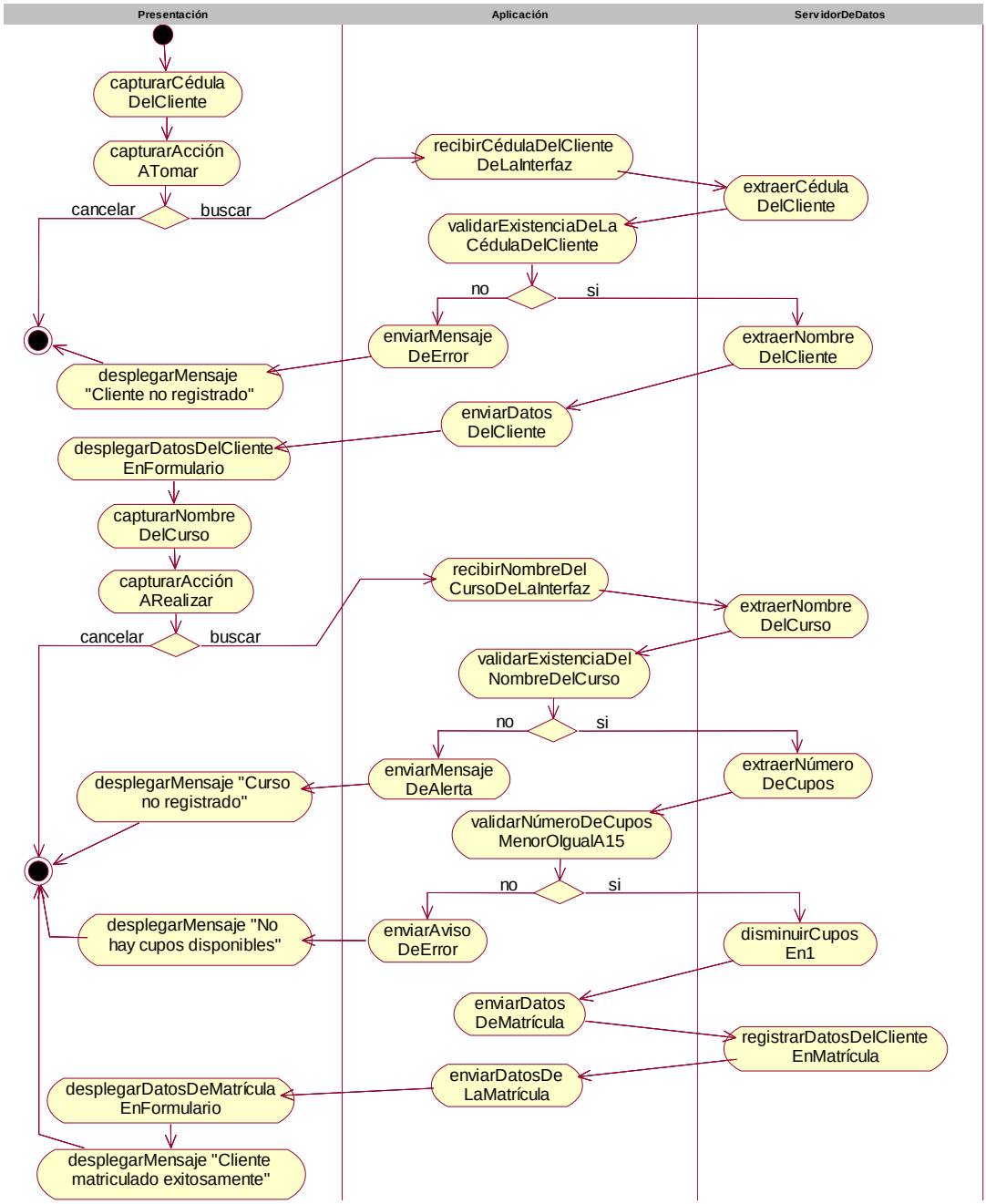


Figura 3.23. Diagrama de Actividades – Ingresar Datos de Matrícula (Elaborado por los Autores).

3.2.6.3 Diagramas de Secuencia

A continuación se detallan los diagramas de secuencia con los distintos escenarios para los principales casos de uso, los demás diagramas se encuentran en el Anexo B.

3.2.6.3.1 Registrar Producto

Notación:

np: Nombre del producto
m: marca
d: descripción
i: imagen
c: cantidad
vu: valor unitario

Nuevo Producto

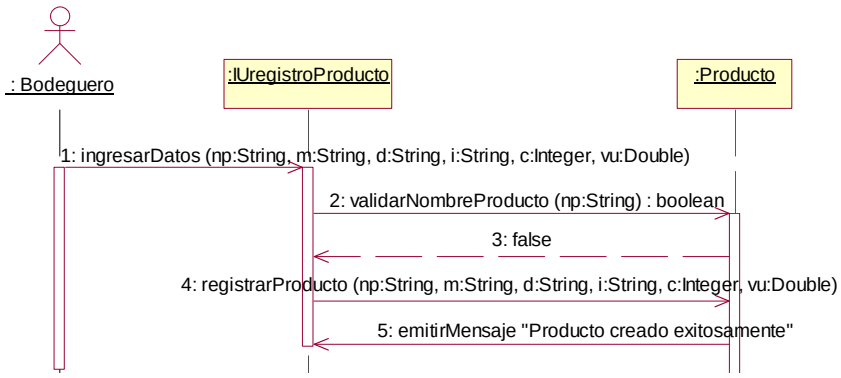


Figura 3.24. Diagrama de Secuencia – Nuevo Producto (Elaborado por los Autores).

Producto se Encuentra Registrado

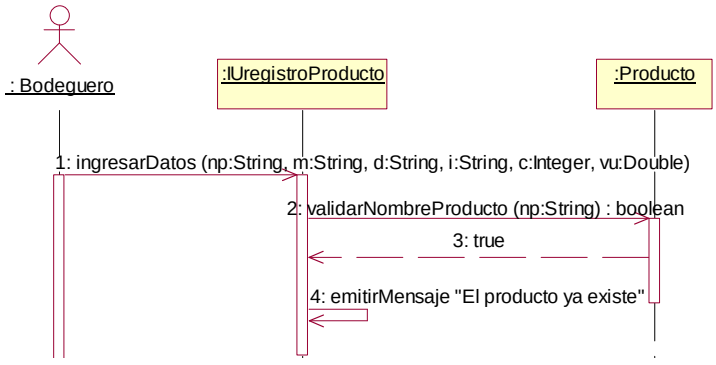


Figura 3.25. Diagrama de Secuencia – Producto se encuentra registrado (Elaborado por los Autores).

3.2.6.3.2 Facturación

Notación:

c: cédula
nc: nombre del cliente
np: nombre del producto
cd: cantidad disponible
ca: cantidad
t: valor total

Venta de productos

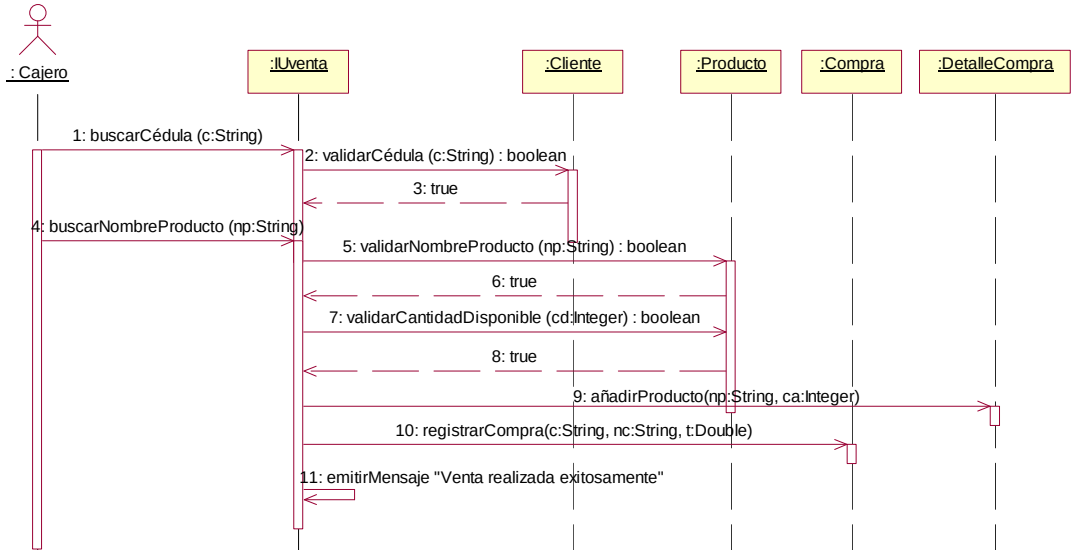


Figura 3.26. Diagrama de Secuencia – Venta de productos (Elaborado por los Autores).

Cliente no encontrado

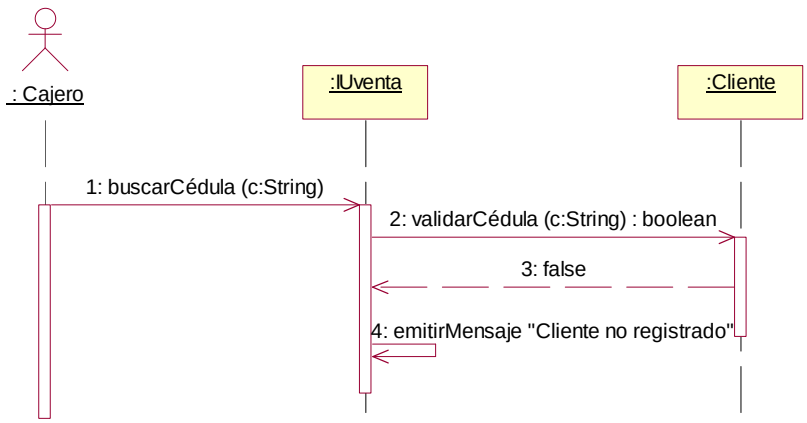


Figura 3.27. Diagrama de Secuencia – Cliente no encontrado (Elaborado por los Autores).

Producto no encontrado

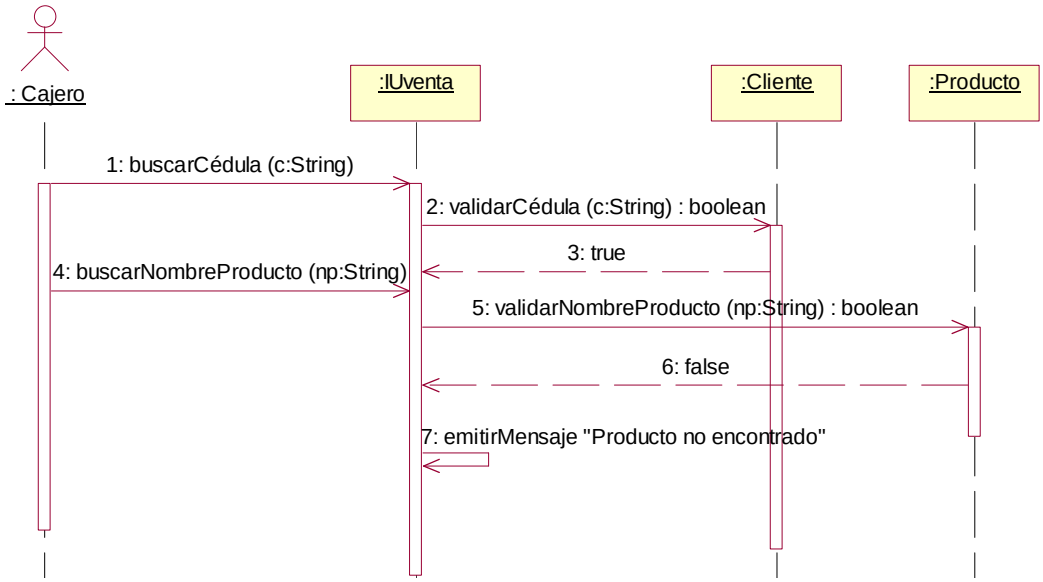


Figura 3.28. Diagrama de Secuencia – Producto no encontrado (Elaborado por los Autores).

Cantidad no disponible

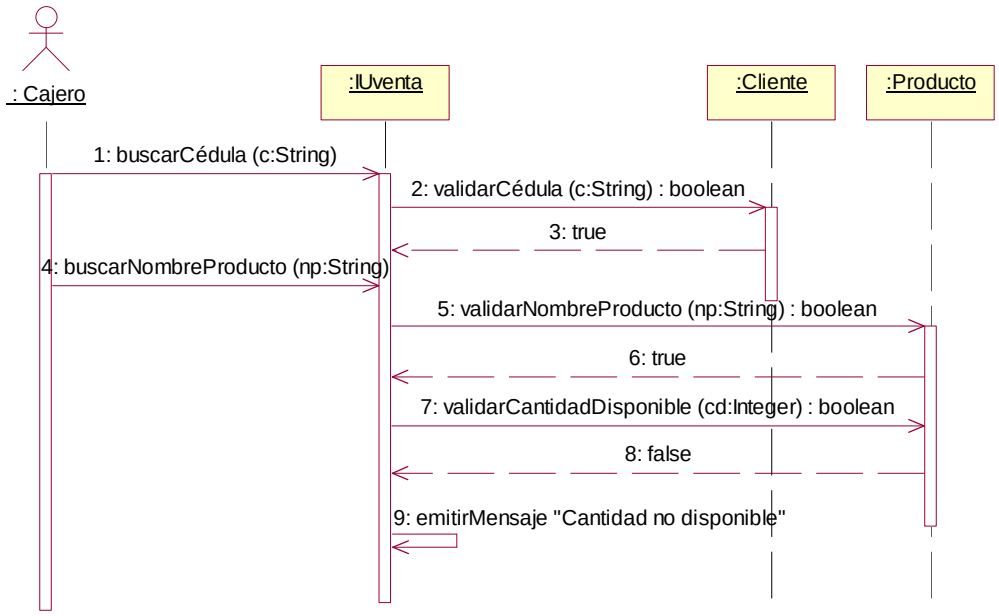


Figura 3.29. Diagrama de Secuencia – Cantidad no disponible (Elaborado por los Autores).

3.2.7 MODELADO DE DATOS

En base a los procedimientos descritos en el capítulo 2.4.5, se transformó el diagrama de clases en un modelo conceptual para poder almacenar los datos de las clases persistentes, como se muestra en la Figura 3.30.

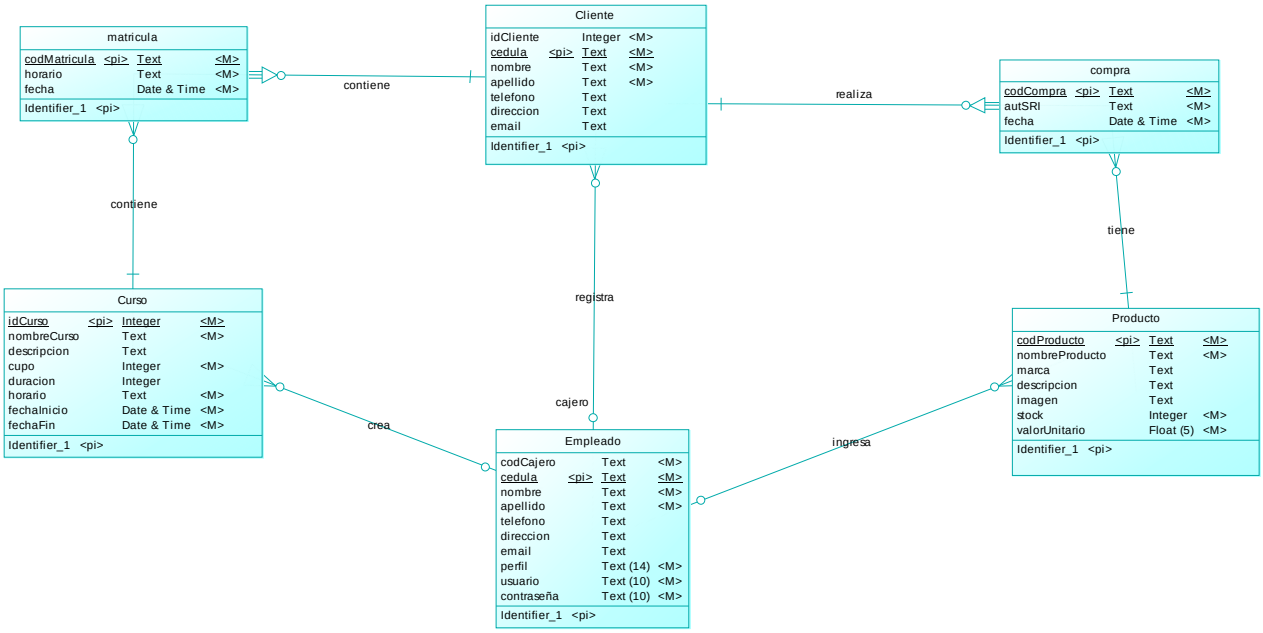


Figura 3.30. Modelo Conceptual – Sistema e-papyrus (Elaborado por los Autores).

3.2.8 DISEÑO DE LA INTERFAZ

El diseño de interfaces de usuario del sistema e-papyrus se ha realizado en base a las cuatro actividades descritas en el capítulo 2.6.

3.2.8.1 Modelo de Análisis de interfaz de usuarios

Inicia con el análisis de usuarios, en base a la clasificación de perfiles, nivel de conocimiento y tipo de usuario se establecen los requisitos a ser implementados en el sistema e-papyrus, como se muestra en la Tabla 3.1.

Perfil	Nivel de Conocimiento	Tipo de usuario	Requisitos
Administrador	Alto	Superusuario	Control sobre todos los procesos del negocio
Cajero	Medio	Usuario	Control sobre los procesos de venta, matriculación, gestión de clientes, consultas de cursos y productos
Bodeguero	Bajo	Usuario	Control sobre los procesos de reabastecimiento y gestión de productos

Tabla 3.1. Análisis de usuarios.

En base a la información de la Tabla 3.1 se procedió a detallar las tareas realizadas por los usuarios que serán implementadas en el sistema, como se muestra en la Tabla 3.2.

Perfil	Acciones	Tareas
Administrador	– Crear usuario	– Ingresar datos
	– Crear producto	– Guardar datos
	– Crear cliente	
	– Crear curso	
	– Modificar usuario	– Buscar datos
	– Modificar producto	– Modificar datos
	– Modificar cliente	– Guardar datos
	– Modificar curso	
	– Eliminar usuario	– Buscar datos
	– Eliminar producto	– Eliminar datos
	– Eliminar cliente	
	– Eliminar curso	
	– Consultar producto	– Buscar datos
	– Consultar curso	
	– Facturar	– Buscar cliente – Buscar producto – Ingresar cantidades – Guardar datos

	– Matricular	– Buscar cliente – Buscar curso – Seleccionar horario – Guardar datos
	– Controlar stock de producto	* Tarea configurada en la aplicación
Cajero	– Crear cliente	– Ingresar datos – Guardar datos
	– Modificar cliente	– Buscar datos – Modificar datos – Guardar datos
	– Eliminar cliente	– Buscar datos – Eliminar datos
	– Consultar producto	– Buscar datos
	– Consultar curso	
	– Facturar	– Buscar cliente – Buscar producto – Ingresar cantidades – Guardar datos
	– Matricular	– Buscar cliente – Buscar curso – Seleccionar horario – Guardar datos
Bodeguero	– Crear producto	– Ingresar datos – Guardar datos
	– Modificar producto	– Buscar datos – Modificar datos – Guardar datos
	– Eliminar producto	– Buscar datos – Eliminar datos
	– Consultar producto	– Buscar datos
	– Controlar stock de producto	* Tarea configurada en la aplicación

Tabla 3.2. Análisis de tareas por usuario.

Una vez que se han detallado las tareas de los usuarios, el siguiente paso es realizar el análisis del contenido de las pantallas para lo cual se han tenido las siguientes consideraciones:

- Las pantallas de autenticación y menú principal del sistema e-papyrus tendrán el logotipo de la empresa en su centro.
- Las pantallas de control de cantidad de producto y catálogo de producto contendrán imágenes.

- Los objetos de datos serán obtenidos de la base de datos y serán presentados al usuario por medio de la aplicación, manteniendo así un diseño de tres capas.
- Dado que el logo de la empresa se basa en tres colores fundamentales, verde, naranja y azul, se mantendrá esta relación de colores en las pantallas.
- Se utilizarán dos tipos de mensajes en el sistema e-papyrus, mensajes de error e informativos.
- Por motivos de facilidad de visualización del contenido del sistema se ha dispuesto colocar como fondo de las pantallas el color blanco.

3.2.8.2 Diseño de la interfaz

Se definieron las acciones que realizarán cada interfaz de acuerdo a cada escenario descrito por los usuarios, como se muestra en la Tabla3.3.

Escenario	**Interfaces**	**Acciones**
Autenticación	Autenticación	Permitir el ingreso al sistema mediante la autenticación de usuarios
Controlar stock de productos	Cantidad de producto	Conocer los productos que necesitan ser reabastecidos cuando su stock sea menos o igual a 10
Facturación	Vender producto	Registrar la venta de productos y emitir una factura por compra
Gestión de bodega	- Nuevo producto	- Registrar los datos de un nuevo producto
	- Editar producto	- Modificar los datos de un producto existente
	- Eliminar producto	- Borrar la información de un producto
	- Consultar producto	- Permitir la búsqueda de un producto específico o desplegar todos los productos existentes
Gestión de clientes	- Nuevo cliente	- Registrar los datos de un nuevo cliente
	- Editar cliente	- Modificar los datos de un cliente existente
	- Eliminar cliente	- Borrar la información de un cliente
Gestión de cursos	- Nuevo curso	- Registrar los datos de un nuevo curso
	- Editar curso	- Modificar los datos de un curso existente
	- Eliminar curso	- Borrar la información de un curso

	- Consultar curso	- Permitir la búsqueda de un curso específico o desplegar todos los cursos existentes
Gestión de usuarios	- Nuevo usuario	- Registrar los datos de un nuevo usuario
	- Editar usuario	- Modificar los datos de un usuario existente
	- Eliminar usuario	- Borrar la información de un usuario
Matriculación	Matricular cliente	Registrar la matriculación de un cliente y emitir un comprobante

Tabla 3.3. Diseño de la Interfaz

3.2.8.3 Prototipo de la interfaz

Para el primer prototipo de interfaces del sistema e-papyrus se han tomado en cuenta aspectos como opciones de ayuda, utilización de menús, lenguaje claro, entre otros. A continuación se presentan las pantallas más relevantes del sistema, para más información referirse al Anexo B.

Registrar Producto

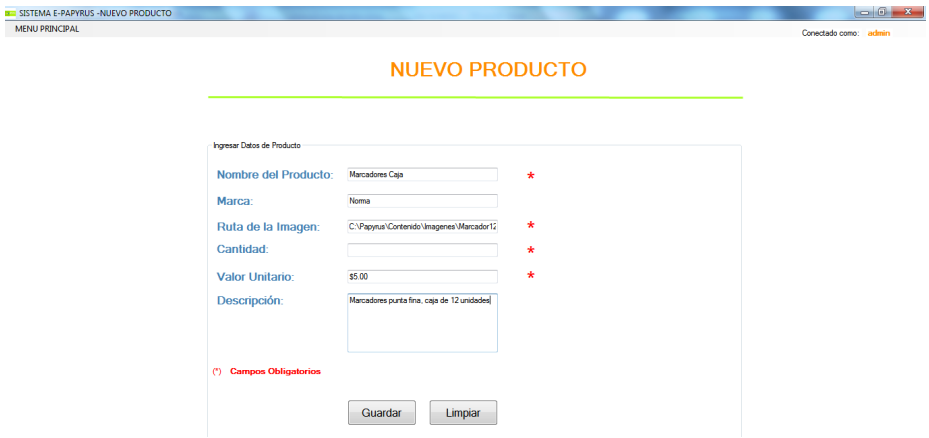


Figura 3.31. Interfaz Registrar producto (Elaborado por los Autores).

Matriculación

Figura 3.32. Interfaz Matriculación (Elaborado por los Autores).

Facturación

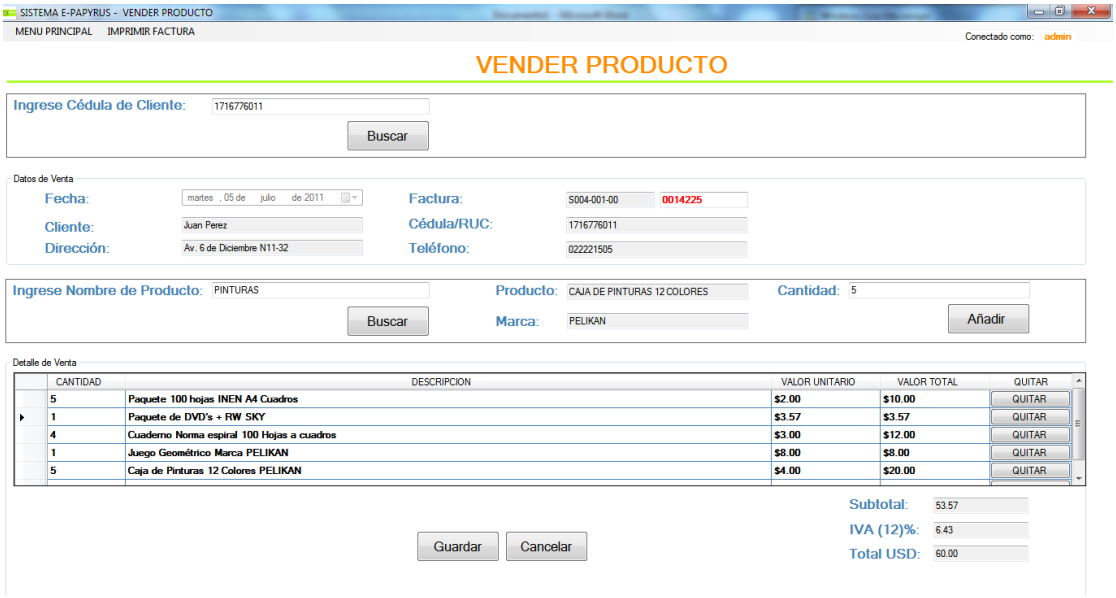


Figura 3.33. Interfaz Facturación (Elaborado por los Autores).

3.2.8.4 Evaluación de la interfaz

Se mantuvo una reunión con el personal clave de la empresa para evaluar el primer prototipo de las interfaces de usuario del sistema e-papyrus. Se corroboró que los requerimientos de los usuarios hayan sido implementados en el sistema, para lo cual se realizó una matriz casos de uso vs interfaces de usuario, como se muestra en la Figura 3.34.

	I01	I02	I03	I04	I05	I06	I07	I08	I09	I10	I11	I12	I13	I14	I15	I16	I17	I18
CU01	✓																	
CU02		✓																
CU03			✓															
CU04				✓	✓	✓	✓											
CU05								✓	✓	✓								
CU06											✓	✓	✓	✓				
CU07															✓	✓	✓	
CU08																		✓

Casos de Uso
CU01: Autenticación
CU02: Controlar stock de productos
CU03: Facturación
CU04: Gestión de bodega
CU05: Gestión de clientes
CU06: Gestión de cursos
CU07: Gestión de usuarios
CU08: Matriculación

Interfaces
I01: Autenticación
I02: Cantidad de producto
I03: Vender producto
I04: Nuevo producto
I05: Editar producto
I06: Eliminar producto
I07: Catálogo de productos
I08: Nuevo cliente
I09: Editar cliente
I10: Eliminar cliente
I11: Nuevo curso
I12: Editar curso
I13: Eliminar curso
I14: Catálogo de cursos
I15: Nuevo usuario
I16: Editar usuario
I17: Eliminar usuario
I18: Matricular Cliente

Figura 3.34. Matriz Casos de Uso vs Interfaces de Usuario (Elaborado por los Autores).

Al finalizar se comprobó que todos los requerimientos fueron implementados por las distintas interfaces del sistema e-papyrus.

Adicionalmente se revisó otros puntos durante la reunión cuyas observaciones se encuentran detalladas en el capítulo 3.2.9.

3.2.9 EVALUACIÓN Y ANÁLISIS DE LA CALIDAD

Para esta propuesta se evaluará la calidad del sistema “e-papyrus”, mediante métricas para el diseño de software, como se especifica en el capítulo 2.7.

3.2.9.1 Formulación y Colección de Métricas

Para el diseño del sistema “e-papyrus” se utilizó un enfoque orientado a objetos. Por esta razón se utilizarán métricas con este mismo enfoque, descritas en la Tabla 3.4.

Métrica	**Mecanismos de Colección**
Tamaño del sistema	Se obtiene a través del conteo del número de clases u operaciones, que pueden ser revisadas en el diagrama de clases.
Métodos ponderados por clase	Se obtiene a través de la estructura de los métodos que aparecen en el diagrama de clases de diseño.
Profundidad de árbol de herencia	Se analiza la estructura de las clases que poseen herencia. Esto se puede observar en el diagrama de clases.
Número de descendientes	Se analiza la estructura de las clases que poseen herencia. Esto se puede observar en el diagrama de clases.
Acoplamiento entre clases	Determinar los métodos y atributos compartidos entre clases, exceptuando aquellos que son heredados. Para esta tarea se puede utilizar el diagrama de clases de diseño, diagrama de actividades, diagrama de secuencia o el pseudocódigo.
Respuesta para una clase	Se calcula en base a los métodos invocados por las clases. Para ellos se puede utilizar el diagrama de clases de diseño, diagrama de actividades o el diagrama de secuencia.
Falta de cohesión entre métodos	Se calcula en base a los métodos de cada clase. Para ello se debe utilizar el diagrama de clases.
Factor de herencia de método	Se calcula en base a los métodos heredados de cada clase. Para ello se debe utilizar el diagrama de clases.
Factor de herencia de atributo	Se calcula en base a los atributos heredados de cada clase. Para ello se debe utilizar el diagrama de clases.

Tabla 3.4. Formulación y Colección de métricas para medir la calidad del sistema e-papyrus.

Adicionalmente se evaluará que el sistema “e-papyrus” cumpla con los parámetros descritos en el capítulo 2.7.4, para determinar la calidad del diseño de la interfaz de

ANÁLISIS E INTERPRETACIÓN

En base a los mecanismos de colección determinados en el paso anterior, se han obtenido los siguientes resultados para cada métrica de la Tabla 3.4.

Tamaño

Mediante el análisis del diagrama de clases del sistema e-papyrus se obtuvo la Tabla 3.5, la cual se utilizará para tener una noción del tamaño del sistema.

Clase	Total Atributos	Operaciones
Persona	6	12
Cliente	1	7
Empleado	3	8
Cajero	1	2
Bodeguero	1	2
Administrador	1	6
Producto	7	21
Curso	8	22
CatalogoDeProductos	1	4
CatalogoDeCursos	1	4
Matricula	3	9
Compra	3	9
DetalleCompra	5	9
Total: 13	**41**	**113**

Tabla 3.5. Resumen del Diagrama de Clases del sistema e-papyrus.

Análisis: El número d, e clases corresponde a un sistema de tamaño pequeño, que es aproximadamente entre 1 y 15 clases. El número de atributos por clase también corresponde a un sistema pequeño, mientras que el número de operaciones corresponde a un sistema de tamaño medio.

Interpretación: Según el número de clases se puede determinar que el sistema e-papyrus es un sistema de tamaño pequeño, sin embargo, cumple con todos los requerimientos y funcionalidad especificada por el usuario. También se pudo detectar que el número de operaciones es de tamaño medio debido a la implementación de métodos set() y get() por cada atributo; lo cual hace que el sistema aumente en tamaño, pero sin aumentar necesariamente la complejidad.

Métodos Ponderados por Clase

Para el cálculo de esta métrica se utilizó la fórmula $MPC = \sum_{i=1}^{i=n} C_i$, descrita en el capítulo 2.7.2. De la aplicación de esta métrica se obtuvieron los siguientes resultados:

CLASE: PERSONA		
Método	**Complejidad del Método**	**Valor de Complejidad C_i**
12 Métodos set/get	Simples	1 por cada Método
MPC =		**12**

Tabla 3.6. Valor de la métrica MPC para la clase Persona del sistema e-papyrus.

Análisis: En la Tabla 3.6 se puede observar que el valor de MPC= 12 no sobrepasa el valor umbral, que es 40.

CLASE: CLIENTE		
Método	**Complejidad del Método**	**Valor de Complejidad C_i**
registrarCliente	Simple	2
validarDatos	Simple	1
modificarCliente	Simple	1
eliminarCliente	Simple	1
buscarCliente	Simple	1
2 Métodos set/get	Simples	1 por cada Método
MPC =		**8**

Tabla 3.7. Valor de la métrica MPC para la clase Cliente del sistema e-papyrus.

Análisis: En la Tabla 3.7 se puede observar que el valor de MPC= 8 no sobrepasa el valor umbral, que es 40. Los métodos heredados en esta clase no se toman en cuenta para el cálculo de esta métrica.

CLASE: EMPLEADO		
Método	**Complejidad del Método**	**Valor de Complejidad C_i**
buscarEmpleado	Simple	1
autenticarUsuario	Simple	1
6 Métodos set/get	Simples	1 por cada Método
MPC =		**8**

Tabla 3.8. Valor de la métrica MPC para la clase Empleado del sistema e-papyrus.

Análisis: En la Tabla 3.8 se puede observar que el valor de MPC= 8 no sobrepasa el valor umbral, que es 40. Los métodos heredados en esta clase no se toman en cuenta para el cálculo de esta métrica.

CLASE: CAJERO		
Método	**Complejidad del Método**	**Valor de Complejidad C_i**

2 Métodos set/get	Simples	1 por cada Método
MPC =		**2**

Tabla 3.9. Valor de la métrica MPC para la clase Cajero del sistema e-papyrus.

Análisis: En la Tabla 3.9 se observa que la clase Cajero sólo posee métodos simples, por lo que su implementación será fácil de realizar. Los métodos heredados en esta clase no se toman en cuenta para el cálculo de esta métrica.

CLASE: BODEGUERO		
Método	**Complejidad del Método**	**Valor de Complejidad C_i**
2 Métodos set/get	Simples	1 por cada Método
MPC =		**2**

Tabla 3.10. Valor de la métrica MPC para la clase Bodeguero del sistema e-papyrus.

Análisis: En la Tabla 3.11 se puede visualizar que la métrica MPC= 2 no supera el valor umbral de 40. Los métodos heredados en esta clase no se toman en cuenta para el cálculo de esta métrica.

CLASE: ADMINISTRADOR		
Método	**Complejidad del Método**	**Valor de Complejidad C_i**
RegistrarEmpleado	Simple	2
ValidarDatos	Simple	1
ModificarEmpleado	Simple	1
EliminarEmpleado	Simple	1
2 Métodos set/get	Simples	1 por cada Método
MPC =		**7**

Tabla 3.11. Valor de la métrica MPC para la clase Administrador del sistema e-papyrus.

Análisis: En la Tabla 3.11 se puede visualizar que la métrica MPC= 7 no supera el valor umbral de 40. Los métodos heredados en esta clase no se toman en cuenta para el cálculo de esta métrica.

CLASE: CURSO		
Método	**Complejidad del Método**	**Valor de Complejidad C_i**

registrarCurso	Simple	2
validarDatos	Simple	1
modificarCurso	Simple	1
eliminarCurso	Simple	1
buscarCurso	Simple	1
verificarCupo	Simple	1
16 Métodos set/get	Simples	1 por cada Método
MPC =		**23**

Tabla 3.12. Valor de la métrica MPC para la clase Curso del sistema e-papyrus.

Análisis: En la Tabla 3.12 se puede observar que el valor de MPC=23 no sobrepasa el valor umbral, que es 40.

CLASE: PRODUCTO		
Método	**Complejidad del Método**	**Valor de Complejidad C_i**
registrarProducto	Simple	2
validarDatos	Simple	1
modificarProducto	Simple	1
eliminarProducto	Simple	1
buscarProducto	Simple	1
emitirAlerta	Simple	1
controlarStock	Simple	1
14 Métodos set/get	Simples	1 por cada Método
MPC =		**22**

Tabla 3.13. Valor de la métrica MPC para la clase Producto del sistema e-papyrus.

Análisis: En la Tabla 3.13 se puede observar que el valor de MPC= 22 no sobrepasa el valor umbral, que es 40.

CLASE: CATALOGODECURSOS		
Método	**Complejidad del Método**	**Valor de Complejidad C_i**
ordenarCurso	Simple	1
mostrarCurso	Simple	1
2 Métodos set/get	Simples	1 por cada Método
MPC =		**4**

Tabla 3.14. Valor de la métrica MPC para la clase CatalogoDeCursos del sistema e-papyrus.

Análisis: En la Tabla 3.14 se puede observar que el valor de MPC= 4 no sobrepasa el valor umbral, que es 40.

CLASE: CATALOGODEPRODUCTOS		
Método	**Complejidad del Método**	**Valor de Complejidad C_i**
ordenarProducto	Simple	1
mostrarProducto	Simple	1
2 Métodos set/get	Simples	1 por cada Método
MPC =		**4**

Tabla 3.15. Valor de la métrica MPC para la clase CatalogoDeProductos del sistema e-papyrus.

Análisis: En la Tabla 3.15 se puede observar que el valor de MPC= 4 no sobrepasa el valor umbral, que es 40.

CLASE: MATRICULA		
Método	**Complejidad del Método**	**Valor de Complejidad C_i**
registrarMatricula	Media	3
verificarDatos	Simple	2
imprimirComprobante	Simple	1
6 Métodos set/get	Simple	1 por cada Método
MPC =		**12**

Tabla 3.16. Valor de la métrica MPC para la clase Matricula del sistema e-papyrus.

Análisis: En la Tabla 3.16 se puede observar que el valor de MPC=12 no sobrepasa el valor umbral, que es 40.

CLASE: COMPRA		
Método	**Complejidad del Método**	**Valor de Complejidad C_i**
registrarCompra	Medio	3
verificarDatos	Simple	2
imprimirCompra	Simple	1
6 Métodos set/get	Simple	1 por cada Método
MPC =		**12**

Tabla 3.17. Valor de la métrica MPC para la clase Compra del sistema e-papyrus.

Análisis: En la Tabla 3.17 se puede observar que el valor de MPC= 12 no sobrepasa el valor umbral, que es 40.

CLASE: DETALLECOMPRA

Método	Complejidad del Método	Valor de Complejidad C_i
CalcularTotal	Simple	1
AñadirProducto	Simple	2
QuitarProducto	Simple	2
6 Métodos set/get	Simples	1 por cada Método
MPC =		**11**

Tabla 3.18. Valor de la métrica MPC para la clase DetalleCompra del sistema e-papyrus.

Análisis: En la Tabla 3.18 se puede observar que el valor de MPC= 11 no sobrepasa el valor umbral, que es 40.

Interpretación: Los resultados de MPC para todas las clases analizadas están por debajo del valor umbral para esta métrica, lo cual indica que todas las clases tienen baja complejidad, y que el esfuerzo necesario para su implementación y pruebas es bajo. También se detectó que ciertas clases poseen un número ligeramente elevado de métodos, lo cual aumenta el valor de MPC; sin embargo, a pesar de la cantidad, la complejidad de estos métodos es baja por lo que no afecta considerablemente a la complejidad de la clase.

Profundidad del árbol de herencia

Al analizar el diagrama de clases del sistema e-papyrus se obtuvieron los resultados mostrados en la Tabla 3.19.

CLASES DEL SISTEMA E-PAPYRUS		
Clase	**Profundidad de árbol de herencia**	**Valor umbral**
Persona	0	6
Cliente	2	6
Empleado	2	6
Cajero	3	6
Bodeguero	3	6
Administrador	3	6
Producto	0	6
Curso	0	6
CatalogoDeProductos	0	6
CatalogoDeCursos	0	6
Matricula	0	6
Compra	0	6
DetalleCompra	0	6

Tabla 3.19. Valor de la métrica APH para las clases sistema e-papyrus.

Análisis: En la tabla anterior se puede observar que existen 5 subclases que utilizan el mecanismo de herencia. Se obtuvo un valor máximo de profundidad de árbol de herencia de 3, lo cual es inferior al valor umbral de 6 para esta métrica.

Interpretación: Se puede observar que ninguna clase supera los valores umbrales de herencia, lo cual indica que no existen objetos complejos que sean difíciles de implementar o reusar. Por otro lado, se utiliza el mecanismo de herencia en 5 clases lo cual demuestra que se están aprovechando adecuadamente las ventajas de la orientación a objetos.

Número de Descendientes

Al analizar el diagrama de clases del sistema e-papyrus se obtuvieron los resultados mostrados en la Tabla 3.20.

CLASES DEL SISTEMA E-PAPYRUS	
Clase	**Número de Descendientes**
Persona	2
Cliente	0
Empleado	3
Cajero	0
Bodeguero	0
Administrador	0
Producto	0
Curso	0
CatalogoDeProductos	0
CatalogoDeCursos	0
Matricula	0
Compra	0
DetalleCompra	0

Tabla 3.20. Valor de la métrica NDH para las clases sistema e-papyrus.

Análisis: Analizando la tabla 3.20 se puede determinar que existen sólo dos superclases en el sistema, las cuales tienen un número pequeño de descendientes.

Interpretación: Cuando una clase posee una gran cantidad de subclases se aprovecha el mecanismo de la herencia, lo que representa una mayor reutilización de código. Sin embargo, surgen algunos inconvenientes como: mayor dependencia

entre las superclases y subclases lo que hace que el sistema sea difícil de modificar, otro problema es que a medida que se incrementa el nivel de herencia, se incrementa la complejidad del sistema y los recursos necesarios para realizar pruebas. En este caso se utiliza la herencia de forma discreta de tal forma que se fomente la reutilización sin aumentar considerablemente la complejidad.

Acoplamiento entre Clases

En base al diagrama de clases y el diagrama de actividades se puede determinar el acoplamiento entre clases, tal como se especifica en la Tabla 3.21.

ACOPLAMIENTO ENTRE CLASES DEL SISTEMA E-PAPYRUS			
Clase	**Clases Vinculadas**	**Valor Umbral**	**Acoplamiento**
Persona	2	5	Bajo
Cliente	3	5	Medio
Empleado	4	5	Alto
Cajero	1	5	Bajo
Bodeguero	1	5	Bajo
Administrador	1	5	Bajo
Producto	2	5	Bajo
Curso	2	5	Bajo
CatalogoDeProductos	1	5	Bajo
CatalogoDeCursos	1	5	Bajo
Matricula	2	5	Bajo
Compra	3	5	Medio
DetalleCompra	2	5	Bajo

Tabla 3.21. Tabla de acoplamiento entre clases del sistema e-papyrus.

Análisis: En la Tabla 3.21 se puede observar que ninguna de las clases del sistema sobrepasa el valor umbral de 5. La única clase con un valor alto de acoplamiento es Empleado, esto se debe a que se relaciona con otras clases a través de la herencia, por lo que es justificado ese resultado. Las demás clases tienen valores bajos de acoplamiento, lo cual es una característica deseable en un sistema.

Interpretación: El umbral de 5 nunca es sobrepasado, lo cual refleja que las clases del sistema poseen un nivel adecuado de independencia. El esfuerzo necesario para realizar el mantenimiento y pruebas es directamente proporcional al grado de acoplamiento, que en este caso al presentar valores bajos, se pueden realizar de forma simple; de igual forma, se facilita la reutilización del sistema.

Respuesta para una clase

En la Tabla 3.22 se especifican los valores de RFC para las clases del sistema e-papyrus. Hay que tomar en cuenta que el valor umbral de referencia establecido para RFC es de 100.

Clase	Métodos de la Clase	Métodos externos invocados	RFC (umbral 100)	Complejidad
Persona	12	0	12	Baja
Cliente	7	10	17	Baja
Empleado	8	5	13	Baja
Cajero	2	0	2	Baja
Bodeguero	2	0	2	Baja
Administrador	6	16	22	Baja
Producto	21	0	21	Baja
Curso	22	0	22	Baja
CatalogoDeProductos	4	7	11	Baja
CatalogoDeCursos	4	8	12	Baja
Matricula	9	7	16	Baja
Compra	9	7	16	Baja
DetalleCompra	9	5	14	Baja

Tabla 3.22. Valores de RFC para las clases del sistema e-papyrus.

Análisis: En la tabla anterior se puede observar que ninguna clase supera el valor umbral, y los valores de RFC son bajos. Las clases con mayores valores de RFC son aquellas que utilizan la herencia o aquellas que implementan muchos métodos set y get.

Interpretación: Los valores obtenidos no superan el valor umbral para la métrica RFC. Esto indica que la complejidad del sistema, basada en la comunicación entre clases, es relativamente baja. Por tanto, el esfuerzo para realizar pruebas y depuraciones va a ser bajo.

Falta de cohesión entre métodos

Para el cálculo de esta métrica se utilizó la fórmula $FCM = \frac{\left[\frac{t}{n}\sum_{j=1}^{a}\mu(Aj)\right]-m}{1-m}$, descrita en el capítulo 2.7.2. De esta Métrica se obtuvieron los valores especificados en la Tabla 3.23.

Clase	Número de Atributos a	Número de Métodos m	$\sum_{j=1}^{a} \mu(Aj)$	Valor de FCM	Valor Umbral
Persona	6	12	12	0.9	1
Cliente	1	7	4	0.50	1
Empleado	3	8	9	0.71	1
Cajero	1	2	2	0.00	1
Bodeguero	1	2	2	0.00	1
Administrador	1	6	4	0.40	1
Producto	7	21	34	0.81	1
Curso	8	22	44	0.78	1
CatalogoDeProductos	1	4	3	0.33	1
CatalogoDeCursos	1	4	3	0.33	1
Matricula	3	9	15	0.50	1
Compra	3	9	15	0.50	1
DetalleCompra	5	9	21	0.60	1

Tabla 3.23. Valores de FCM para las clases del sistema e-papyrus.

Análisis: En la tabla anterior se puede comprobar que en ninguna de las clases se supera el valor umbral de 1, es decir no existe una falta de cohesión en los métodos. Sin embargo, en cuatro clases los valores superan el valor de 0.7, y se aproximan al valor umbral, lo cual debe ser analizado por el equipo desarrollador.

Interpretación: A medida que los valores se aproximan al valor umbral de 1 la falta de cohesión entre métodos se hace evidente. En este caso, ningún valor llega al umbral permitido, pero cuatro clases se aproximan a este valor, lo que significa que cada método accede o utiliza pocos atributos de la clase. Lo ideal para que exista cohesión perfecta (FCM =0) sería que cada atributo de la clase sea utilizado por cada uno de los métodos. La definición de métodos set y get para cada atributo hace que el valor de FCM se incremente.

Factor de Herencia de Método

Para el cálculo de esta métrica se utilizó la fórmula $MFH = \frac{\sum M_i(C_i)}{\sum M_i(C_i)+\sum M_d(C_i)}$, descrita en el capítulo 2.7.2, de la cual se obtuvieron los valores especificados en la Tabla 3.24.

Clase	Métodos declarados por clase	Total de Métodos heredados y no redefinidos

	$M_d(C_i)$		$M_i(C_i)$
Persona	12		0
Cliente	7		10
Empleado	8		10
Cajero	2		18
Bodeguero	2		18
Administrador	6		18
Producto	21		0
Curso	22		0
CatalogoDeProductos	4		0
CatalogoDeCursos	4		0
Matricula	9		0
Compra	9		0
DetalleCompra	9		0
Total	**115**		**74**
MFH:		**0.39**	

Tabla 3.24. Valores de MFH para el sistema e-papyrus.

Análisis: Los resultados de la tabla anterior indican que los métodos del sistema e-papyrus son reutilizados en un 39%.

Interpretación: A medida que los valores de MFH aumentan significa que el grado de herencia, y el grado de reutilización también aumentan. En este caso un 39% de métodos reutilizados es un valor adecuado, ya que se consigue un equilibrio entre reuso y complejidad del sistema. Valores muy altos de MFH podrían generar un sistema complejo, difícil de mantener y con muchas dependencias.

Factor de Herencia de Atributo

Para el cálculo de esta métrica se utilizó la fórmula $AFH = \frac{\sum A_i(C_i)}{\sum A_i(C_i)+\sum A_d(C_i)}$, descrita en el capítulo 2.7.2, de la cual se obtuvieron los valores especificados en la Tabla 3.25.

Clase	**Atributos declarados por clase** $A_d(C_i)$	**Número de Atributos heredados y no redefinidos** $A_i(C_i)$
Persona	6	0
Cliente	1	6
Empleado	3	6

Cajero	1		9
Bodeguero	1		9
Administrador	1		9
Producto	7		0
Curso	8		0
CatalogoDeProductos	1		0
CatalogoDeCursos	1		0
Matricula	3		0
Compra	3		0
DetalleCompra	5		0
Total	**41**		**39**
AFH:		**0.48**	

Tabla 3.25. Valores de AFH para el sistema e-papyrus.

Análisis: Los resultados de la tabla anterior indican que los atributos del sistema e-papyrus son reutilizados en un 48%.

Interpretación: A medida que los valores de AFH aumentan significa que el grado de herencia y el grado de reutilización también aumentan. En este caso un 48% de atributos reutilizados es un valor adecuado, ya que se consigue un equilibrio entre reuso y complejidad del sistema. Valores muy altos de MFH podrían generar un sistema complejo, difícil de mantener y con muchas dependencias.

Interfaz de Usuario

Para el análisis de la calidad en la interfaz de usuario se utilizaron los parámetros establecidos en el capítulo 2.7.4, los cuales fueron evaluados por el equipo de diseñadores en conjunto con los usuarios finales del sistema. De esta evaluación se obtuvo la Tabla 3.26.

Parámetro	**Evaluación**
Anticipación	Cumple parcialmente
Autonomía	Cumple totalmente
Percepción del color	Cumple totalmente
Consistencia	Cumple totalmente
Eficiencia del Usuario	Cumple totalmente
Tiempo	Cumple parcialmente
Interfaces explorables	Cumple totalmente
Objetos UI	Cumple totalmente
Uso de metáforas	Cumple parcialmente
Legiblidad	Cumple totalmente
Interfaces visibles	Cumple totalmente

Tabla 3.26. Parámetros de calidad para las interfaces del sistema e-papyrus.

Análisis: Todos los parámetros de calidad en interfaces de usuario fueron considerados en el sistema e-papyrus, sin embargo no todas las interfaces implementan completamente los parámetros, como se detalla a continuación:

- *Anticipación:* se anticipan las acciones del usuario en las interfaces que poseen campos de búsqueda o de ingreso de información específica.
- *Autonomía:* en todas las interfaces se evita sobrecargar la pantalla con elementos excesivos que dificulten el aprendizaje del usuario. Las interfaces con mayor número de elementos, por la cantidad de entradas que manejan, son la de matriculación y facturación.
- *Percepción de color:* para todas las interfaces se utiliza una combinación de colores que resalta el texto y permite que se visualicen fácilmente los elementos, es perceptible aún para personas con deficiencias visuales.
- *Consistencia:* todas las interfaces utilizan símbolos y comandos estandarizados para las interfaces desarrolladas bajo plataforma Windows.
- *Eficiencia del Usuario:* todas las interfaces utilizan sólo los mensajes necesarios para la correcta utilización del sistema, y que no generen una interrupción en el flujo normal de trabajo. Esto ayuda a incrementar la eficiencia y reducir el tiempo para la realización de tareas.
- *Tiempo:* en la mayoría de interfaces se organizaron los elementos de uso común de tal forma que estén disponibles con mayor rapidez para las tareas del usuario. En algunas interfaces como Catálogo de Productos y Catálogo de Cursos fue posible realizar esta organización debido al gran número de elementos que se deben manejar en la interfaz.
- *Interfaces Explorables:* todas las interfaces poseen los botones necesarios para navegar fácilmente a través del sistema.
- *Objetos UI:* todos los elementos de la interfaz gráfica son fáciles de entender, consistentes y estables.

- *Uso de Metáforas:* en este sistema no es factible el uso de metáforas para realizar acciones del sistema, ya que su implementación tendría un costo adicional, innecesario, en recursos gráficos para poder visualizarlas.
- *Legibilidad:* la disposición de los elementos de cada interfaz es adecuada para el usuario, el texto tiene un contraste y tamaño adecuado.
- *Interfaces visibles:* todas las acciones descritas en los casos de uso pueden ser realizadas desde su respectiva interfaz, sin necesidad de ocupar otras interfaces que dificulten o retrasen la realización de la acción.

Interpretación: A pesar de que 3 parámetros son cumplidos parcialmente, debido a las razones especificadas en el análisis anterior, se puede decir que las interfaces del sistema e-papyrus cumplen con los parámetros de calidad. Esto conlleva a tener un sistema con un nivel adecuado de facilidad de uso, lo cual es una característica deseable dentro del diseño, que permite que el usuario se adapte al sistema, aprenda fácilmente a realizar tareas, y finalmente se sienta cómodo utilizando el entorno del sistema.

RETROALIMENTACIÓN

Tamaño

Al ser un sistema de tamaño pequeño se estima que el tiempo de desarrollo sea también corto. Por esta razón se sugiere al equipo desarrollador que para futuras versiones, si se implementan nuevos requerimientos y nueva funcionalidad, se trate de mantener un tamaño adecuado del sistema de tal forma que pueda ser manejable. El número de métodos que se implementan en el sistema es aparentemente grande, sin embargo, la mayoría de estos métodos son operaciones set() y get() las cuales no representan mayor complejidad en la implementación, y por tanto, no influyen en la complejidad del sistema.

Métodos Pesados por Clase

Los valores de MPC indican una complejidad aceptable de las clases del sistema e-papyrus, y se espera que no haya dificultad para la implementación y pruebas.

Se recomienda al equipo de desarrollo, que en iteraciones posteriores, se considere la posibilidad de simplificar y reducir el número de métodos dentro de las clases, para reducir los valores de MPC, con lo cual se logrará que el sistema tenga mayores posibilidades de reuso. Es posible lograr esta reducción observando métodos similares que puedan ser modelados mediante interfaces.

Profundidad del árbol de herencia

Altos valores de esta métrica indican objetos complejos que son difíciles de implementar y reutilizar. Por otro lado, valores bajos o nulos de esta métrica indican que se limita la capacidad de reutilización del sistema y no se aprovecha el mecanismo de herencia de la orientación a objetos.

El grado de herencia en este sistema es adecuado ya que no aumenta la complejidad del sistema. Se recomienda al equipo de desarrollo, que si se realizan iteraciones futuras, se mantenga el nivel de herencia, y si es necesario incrementarlo, que no sobrepase los valores umbrales establecidos.

Acoplamiento entre clases

Esta métrica ofrece una perspectiva del acoplamiento entre las clases del sistema, pero no ofrece un indicador preciso de la fuerza de acoplamiento entre cada relación. Es necesario mantener el acoplamiento lo más bajo posible, aunque siempre sea necesario un cierto grado de relación entre clases.

Se sugiere que el equipo de desarrollo mantenga este esquema fácil de manejar entre las clases del sistema, y que si bien el valor umbral de acoplamiento por clase es de 5 se evite sobrepasar el valor de 3 en futuras iteraciones, para garantizar un sistema de baja complejidad.

Número de Descendientes

En este sistema el número de descendientes es el adecuado para mantener un nivel adecuado de herencia, sin afectar futuras modificaciones del sistema. Se recomienda

al equipo desarrollador mantener esta métrica en los niveles adecuados para futuras iteraciones.

Respuesta para una Clase

Se recomienda al equipo de desarrollo que, si en futuras iteraciones, se aumentan nuevas clases en el sistema, se controle que los valores de RFC sean lo más bajos posibles para disminuir la complejidad de las clases y no afectar a todo el sistema.

Falta de Cohesión entre Métodos

No se detecta una falta de cohesión total en los métodos de las clases del sistema e-papyrus, sin embargo en la mayoría de clases existe una aproximación hacia los valores umbrales. Esta aproximación se debe a la utilización de métodos set y get para cada atributo de la clase, estos métodos aumentan los valores de FCM.

Se sugiere al equipo de desarrollo, que se implementen los métodos set y get, pero que no se los tome en cuenta para la el cálculo de esta métrica. Con esto se lograrán valores de FCM cercanos a 0.

Factor de Herencia de Método

Existe un porcentaje considerable de reutilización de métodos, sin embargo, se recomienda al equipo de desarrollo mantener un equilibrio entre la herencia y la complejidad del sistema para evitar demasiada dependencia entre sus elementos.

Factor de Herencia de Atributo

Existe un porcentaje considerable de reutilización de atributos, sin embargo, se recomienda al equipo de desarrollo mantener un equilibrio entre la herencia y la complejidad del sistema para evitar demasiada dependencia entre sus elementos.

Interfaz de Usuario

Una interfaz de usuario de calidad garantiza la facilidad de uso para las funciones del sistema, por lo que se recomienda al equipo de desarrollo seguir las pautas propuestas en el capítulo 2.7.4, sin olvidar la retroalimentación por parte del usuario final.

3.2.10 DOCUMENTACIÓN DE SALIDA

Como documentación de salida a la etapa de Diseño de Software se utilizará el estándar IEEE-1016-2009, “Descripciones de Diseño de Software”, en el cual se organizan los modelos y diagramas realizados para el sistema e-papyrus de acuerdo a los puntos establecidos en el capítulo 2.8.1. El documento completo puede ser revisado en el Anexo B de este documento.

3.3 ANÁLISIS DE RESULTADOS

Una vez que se ha aplicado la propuesta y se ha seguido detalladamente el proceso de diseño planteado se han obtenido los siguientes resultados:

- El proceso planteado pudo ser aplicado en su totalidad, lo cual certifica la aplicabilidad de la propuesta.
- Se obtuvo la primera iteración del diseño para el sistema e-papyrus, la cual es lo suficientemente completa para servir como base para futuras iteraciones.
- El diseño planteado tiene el nivel adecuado de detalle y calidad para servir como punto de partida para la implementación del sistema, garantizando que no existirán inconvenientes que afecten significativamente al desarrollo del sistema.
- El sistema e-papyrus tiene un tamaño pequeño y una complejidad baja, lo cual lo convierte en un sistema fácil de implementar y fácil de probar.
- El sistema e-papyrus posee facilidad de mantenimiento y facilidad de realizar correcciones al código al no tener grandes niveles de acoplamiento.

- La arquitectura del sistema le permite añadir fácilmente nuevas funcionalidades y requerimientos futuros, siempre y cuando se tengan en cuenta las sugerencias realizadas en la etapa de retroalimentación.
- El sistema posee un grado de cohesión adecuada, lo cual es muestra de buenas prácticas de diseño.
- El sistema utiliza la herencia adecuadamente para aprovechar los mecanismos de la orientación a objetos, sin aumentar la considerablemente la complejidad y dependencia entre elementos del sistema.

4 CAPITULO IV. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- El Diseño de Software es una etapa fundamental para el desarrollo de un producto de software, ya que permite plantear un modelo de la solución desde varios enfoques y puntos de vista dirigidos a distintos stakeholders.

- La correcta aplicación del Proceso de Diseño permite satisfacer los requerimientos del usuario y facilitar la elaboración de las etapas posteriores de desarrollo, como la implementación, pruebas y mantenimiento de software.
- Mediante la elaboración del caso de estudio se pudo constatar que el Diseño es un proceso iterativo, del cual se obtiene un producto final a través de un refinamiento sucesivo de soluciones y el planteamiento de varios prototipos.
- El proceso de diseño consta de dos actividades fundamentales; parte de la elaboración de una Arquitectura base de alto nivel, para terminar en un Diseño Detallado, con un nivel de detalle que permita su fácil implementación.
- Debido a que en el Ecuador el Diseño de Software se realiza de una manera poco formal, esta propuesta servirá como un modelo genérico a seguir, por su flexibilidad, escalabilidad y facilidad de aplicación.
- Mediante la realización del caso de estudio se pudo certificar la aplicabilidad de la propuesta, y se constataron los beneficios en tiempo de desarrollo, mitigación de errores en etapas futuras, y satisfacción de los requerimientos de usuario.
- Es necesario realizar un Análisis y Evaluación de la Calidad del Diseño de Software, para garantizar un producto con altos estándares de calidad, que satisfaga las necesidades del usuario y mitigue el riesgo de errores en las etapas posteriores de desarrollo.

4.2 RECOMENDACIONES

- Se recomienda aplicar esta propuesta para el Diseño de cualquier sistema de software, dada su versatilidad y facilidad de aplicación; y que los usuarios brinden una retroalimentación del proceso a fin de poder refinarlo y mejorarlo.
- Es recomendable que para sistemas de tamaño pequeño, se adapte el procedimiento, de tal manera que el diseño contribuya al desarrollo ágil y efectivo de una solución.

- Se recomienda que antes de realizar el Proceso de Diseño, se realicen adecuadamente las etapas de Requerimientos y de Análisis, ya que los requerimientos mal planteados y los continuos cambios en el análisis pueden generar grandes retrasos en el desarrollo del sistema.

- Se recomienda que en las etapas posteriores al diseño se realice una retroalimentación para asegurar que el producto de software sea desarrollado de acuerdo a los modelos de la solución requeridos, sin que se presenten diferencias o incongruencias entre las etapas que afecten el resultado final.

- Es recomendable que los diseñadores evalúen la calidad del producto de diseño, para garantizar que se tenga un producto confiable y bien elaborado.

- Se recomienda que se elabore correctamente la documentación de salida de la Etapa de Diseño, ya que este documento es de gran importancia para las etapas posteriores.

- Es recomendable escoger adecuadamente las herramientas de Diseño de software que se van a utilizar para realizar los diferentes diagramas, de tal manera que se agiliten y se realicen de manera correcta los modelos.

- Se recomienda una retroalimentación e interacción continua entre el equipo desarrollador y los clientes del sistema, para lograr que el diseño se adapte a las verdaderas necesidades del usuario.

BIBLIOGRAFÍA

LIBROS Y ESTÁNDARES

- ALARCÓN, Raúl. **Diseño Orientado a Objetos con UML**. Primera Edición. Grupo EIDOS. Madrid. 2000.
- BASS, Len; CLEMENTS Paul; KAZMAN, Rick. **Software Architecture in Practice.** Segunda Edición. Addison-Wesley. New York. 2003.

- BOOCH, Grady; RUMBAUCH, James; JACOBSON, Ivar. **The Unified Modeling Language User Guide**. Primera Edición. Addison-Wesley. Londres. 1998.
- BOOCH, Grady; RUMBAUCH, James; JACOBSON, Ivar. **The Unified Modeling Language User Guide**. Segunda Edición. Addison-Wesley. Londres. 2005.
- BOWMAN, Charles. **Algorithms and Data Structrures: An Approach in C.** Oxford University Press. Boston. 1997.
- BUDGEN, David**. Software Design.** Segunda Edición. Addison – Wesley. London. 2003.
- BUSCHMANN, Frank; MEUNIER, Regine; ROHNERT Hans. **Software Architecture a System of Patterns.** Volume 1. Wiley. Alemania. 1998.
- COAD, Peter; YOURDON, Edward. **Object-Oriented Design.** Segunda Edición. Prentice-Hall. London. 1998.
- FENTON, Norman; PFLEEGER, Shari Lawrence. **Software Metrics: A Rigorous & Practical Approach.** Segunda Edición. International Thomson Computer Press. Boston, USA. 1998
- FREEMAN, Peter; WASSERMAN, Anthony. **Tutorial on Software Design Techniques.** Cuarta Edición, IEEE Computer Society Press.
- IEEE Std 610.12. **IEEE Standard Glossary of Software Engineering Terminology.** IEEE Std 610.12, IEEE, 1990.
- IEEE Std 1016-2009. **IEEE Estándar para la Tecnología de la Información-Diseño de Sistemas-Descripciones del Diseño de Software.** IEEE Std 1016-2009, IEEE, 2009.
- IEEE Std 1016-1998. **IEEE Recommended Practice for Software Design Descriptions**. IEEE Std 1016-1998, IEEE, 1998.
- IEEE Std 1471. **IEEE Recommended Practice for Architectural Description of Software Intensive Systems**. IEEE Std 1471-2000, IEEE, 2000.
- IEEE 12207.0. **Standard for Information Technology – Software Life Cycle Processes.** IEEE/EIA 12207.0-1996, IEEE/EIA, 1996.
- JACOBSON, Ivar; CHRISTERSON, Magnus; JONSSON, Patrick; OVERGAARD, Gunnar. **Object-Oriented Software Engineering – A Use Case Driven Approach.** Addison – Wesley. London. 1992.
- JALOTE, Pankaj. **An Integrated Approach to Software Engineering**. Tercera Edición. Springer. New York. 2005.
- KAZMAN, Rick**. Software Architecture**. Software Engineering Institute. Carnegie Mellon University, 2000.
- MARCINIAK, John. **Encyclopedia of Software Engineering.** Vol. 1. J. Wiley & Sons. New York. 1994.
- MEYER, Bertrand. **Object-Oriented Software Construction.** Segunda Edición. Prentice-Hall. London. 2000.

- PRESSMAN, Roger. **Software Engineering. A Practitioner's Aproach.** Séptima Edición. McGraw-Hill. New York. 2010.
- SOMMERVILLE, Ian. **Ingeniería del Software**. Séptima Edición Addison – Wesley. Madrid. 2005.

DIRECCIONES ELECTRÓNICAS

- Guide to the Software Engineering Body of Knowledge (SWEBOK). http://www.computer.org/portal/web/swebok/htmlformat. Último ingreso 16 julio de 2011.
- Desarrollo Orientado a Objetos con UML. http://es.scribd.com/doc/50423698/Desarrollo-orientado-a-objetos-con-UML. Último ingreso 16 julio de 2011.
- Caso de estudio de Documento de Diseño de Sistemas. https://writer.zoho.com/public/lopera.rivera/Trabajo_Final_de_analisis1/noband. Último ingreso 16 julio de 2011.
- Diseño Basado en Componentes http://www.onekin.org/slide/AssociatedFiles/conf2000JISBD.xml.zip/JISBD00-DBCPartition.pdf. Último ingreso 16 julio de 2011.
- Enciclopedia de Ingeniería de Software. http://onlinelibrary.wiley.com/book/10.1002/0471028959. Último ingreso 16 julio de 2011.

GLOSARIO DE TÉRMINOS

Abstracción: captar los detalles más importantes con un cierto nivel de generalización, de tal forma que los detalles irrelevantes son ignorados.

Acoplamiento: fuerza de asociación en la conexión entre dos o más elementos.

Cohesión: grado de conectividad entre los elementos y funciones internas de un elemento simple.

Encapsulamiento: técnica de la orientación a objetos que agrupa, encapsula y oculta los elementos y detalles internos.

Estilo arquitectónico: define una familia de sistemas de software, refiriéndose a su organización estructural.

Línea de Productos de Software es un conjunto de sistemas de software que comparten características comunes y que son desarrolladas mediante la reutilización.

PU: Proceso Unificado. Método de diseño de software basado en objetos.

Punto de Vista: establece las convenciones mediante las cuales una vista es creada, representada y analizada.

SDD: descripciones del diseño de software.

SRS: documento de especificación de requerimientos de software.

SSA/SD: análisis de sistemas estructurados /diseño estructurado.

Stakeholder: personas o entidades interesadas en la realización de un proyecto que brinde una solución óptima a un problema específico.

UML: Lenguaje Unificado de Modelado.

Vista: es la representación o descripción del sistema desde una perspectiva simple, de tal forma que muestre sus propiedades específicas.

ANEXOS

Los anexos pueden ser encontrados en un CD adjunto al Proyecto de Titulación.

ANEXO A: Documento de especificación de requerimientos de software. Descripciones de los requerimientos de Software para el Sistema E-PAPYRUS

ANEXO B: Documento de salida de la etapa de diseño. Descripciones del Diseño de Software para el Sistema E-PAPYRUS