COLEGIO PARTENON, S.C.

INCORPORADO A LA
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
LICENCIATURA EN INFORMÁTICA

ANÁLISIS Y DISEÑO DEL SISTEMA DE CONTROL DE GESTIÓN PARA LA SUBSECRETARÍA DE ATENCIÓN CIUDADANA Y NORMATIVIDAD DE LA SECRETARÍA DE LA FUNCIÓN PÚBLICA

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
LICENCIADA EN INFORMÁTICA
P R E S E N T A :

ELIZETH ZEMPOALTECA ZEMPOALTECA

ASESOR:
ING. ELÍAS SÁNCHEZ SÁNCHEZ

MÉXICO. D.F. 2006

ANÁLISIS Y DISEÑO DEL SISTEMA DE CONTROL DE GESTIÓN PARA LA SUBSECRETARÍA DE ATENCIÓN CIUDADANA Y NORMATIVIDAD DE LA SECRETARÍA DE LA FUNCIÓN PÚBLICA

Dedicatoria

A Dios….

Quizá ya alguien ya lo menciono pero quiero agradecer de todo corazón por darme la oportunidad de existir y darme a un par de ángeles que son mis padres quienes han estado en cada uno de los momentos de debilidad como de fortaleza que he tenido en mí caminar por la vida.

A mis Padres….

Por darme no una, si no las dos manos durante mi camino quitando las piedras para poder seguir y lograr hacer de mí un ser con sueños y triunfos, gracias por cada uno de los consejos gracias mamá por haberme exigido, gracias por ser como eres, gracias por cada regaño. Y gracias a ti papá por darme la oportunidad de tener un padre ejemplar, gracias por escucharme y decirme tú puedes. Gracias por ser mis padres y darme la oportunidad de estudiar.

A mis hermanos….

Por estar conmigo en todo momento sin importar la hora para darme su apoyo y consejos; gracias Marcela por tu ejemplo de lucha gracias por decirme que los pies siempre están en el piso y no arriba y a ti Arturo por ser mi hermano y darme el ejemplo de ser más y que puedo lograr más. Gracias por ser mis hermanos.

Agradecimientos….

Agradezco a todos mis profesores por sus conocimientos aportados y por darme un ejemplo a seguir para desempeñar con satisfacción mi profesión.

Agradezco al Ing. Elías Sánchez Sánchez por la confianza que me brindo y deposito en mí, para realizar y llevar acabo este proyecto, por su guía y paciencia.

A mis amigos quienes contribuyeron enormemente al desarrollo de esta tesis, a todas aquellas personas que me brindaron su apoyo en todo momento para concluir este proyecto.

Muchas Gracias por darme su ejemplo de constancia y de hacerme recordar que aún cuando quede la última cuerda en el violín el concierto debe terminar con éxito.

Índice

Introducción

Es verdad que cada vez que miramos el mundo, nos da sorpresas. El ser humano por naturaleza siempre busca su bienestar, hemos visto durante años como ha automatizado de forma ingeniosa su trabajo. Ha descubierto y creado tecnología de punta que cubre las necesidades.

Pues bien es tan bien cierto que el ser humano ha crecido de tal manera que busco su propia autonomía.

El ¿Qué?, ¿Cuándo? y ¿Cómo?; son algunas preguntas frecuentes en el mundo de la informática; que tienen como único objetivo la creación y desarrollo de software que tiene como finalidad de dar una información automatizada como respuesta al problema.
En la actualidad se busca agilizar la información a las dependencias de gobierno las cuales dependen de áreas para poder realizar su trabajo correspondiente por lo que el gobierno mexicano ha establecido el compromiso de luchar frontalmente; así mismo ha emprendido una revisión de todos sus sistemas de cómputo para regular las licencias de programas que se utilizan en las tareas gubernamentales y erradicar el uso de programas no autorizados.

Después de un profundo análisis de las necesidades de las dependencias gubernamentales, el Gobierno Federal a través del Instituto Nacional de Estadística, Geográfica e Informática (INEGI), formalizaron en Diciembre del 2002, un convenio general.

Una de las dependencias de nuestro país es la Secretaría de la Función Pública que cuenta con la Subsecretaría de Atención Ciudadana y Normatividad la cual tuvo la necesidad de crear un nuevo Sistema de Control de Gestión ya que actualmente se esta estandarizando en lenguajes JAVA y con la ayuda de un administrador de base de datos determinada como MySQL.

Por lo que esta tesis presenta como punto esencial la creación de un nuevo sistema; está basada por tres capítulos y un caso práctico.

En el primer capítulo es nombrado como Fases en el Desarrollo de un Software en donde se abordan temas como, ¿Qué es un Sistema?, el ciclo de vida de un sistema, sus fases, la importancia de personas, proyecto, producto y el proceso, el por qué una arquitectura y cuál es su importancia así mismo el futuro que se desea para el desarrollo de software.

En el segundo capítulo lleva como nombre Metodologías en donde se explica que es una metodología, sus características, su clasificación y cuales son las más renombradas.

El tercer capítulo es llamado como Lenguaje de Modelado Unificado (UML), teniendo su origen en la empresa Rational cuando Booch y Rumbaugh decidieron unir sus métodos para conseguir un lenguaje estándar y sólido. Más tarde se incorporó Jacobson, lo que dio lugar a la versión 0.9 de UML en 1996; posteriormente se creó un consorcio con varias organizaciones interesadas en UML. La versión 1.0 de UML surgió en 1997 con la contribución de IBM, HP, Oracle, Microsoft y otras organizaciones.

El desarrollo de UML continúa actualmente bajo el control de IBM (que adquirió Rational); la última versión de UML es la 2.0.

Mostrando sus principales características el cual permite visualizar mediante su simbología el contenido y la estructura de software, la notación de UML permite definir modelos que serán claramente comprensibles por otros desarrolladores facilitando así el mantenimiento del sistema que describe, permite especificar los procesos de análisis, diseño y codificación de un software.

También permite determinar modelos precisos, sin ambigüedades, detallando las partes esenciales de los mismos, pueda generar código en distintos lenguajes de programación y tablas en una base de datos a partir de modelos, especificando los procesos de análisis, diseño, codificación permitiendo documentar dejando clara la arquitectura del sistema.

El cuarto capítulo presenta al caso práctico descrito con el nombre de Análisis y Diseño del Sistema de Control de Gestión Versión 2.0 en el cuál se busca la actualización del sistema existente, este sistema originalmente se encontraba programado en PL/SQL de Oracle; se busca como se menciona anteriormente la creación de un nuevo sistema basado en la estandarización del lenguaje de Java para poder manejar la versión 2.0 se planteo y diseño la objetividad del nuevo diseño de interfases Web así como el lenguaje de programación seleccionado y con la ayuda de una base de datos manipulada por MySQL que representa como un sistema de administración de base de datos.

Objetivo

El objetivo es realizar un nuevo Sistema de Control de Gestión, proporcionado la información más rápida a la solicitud del servicio necesitado por cada subdirección de la Secretaría de la Función Pública.

La creación del Sistema Control de Gestión surge como respuesta para cubrir dicha necesidad permitiendo de manera sencilla y ágil el control de los turnos de la información de los documentos enviados a la Secretaría Técnica, la cual se encargara de resolver o turnar entre las diferentes Áreas de La Subsecretaría de Atención Ciudadana y Normatividad, permitiendo identificar el tiempo que ha transcurrido en cada una de ellas, así como la consulta del trámite y la resolución que le ha dado a la misma.

Capítulo 1

Fases en el Desarrollo de un Software

1.1 Introducción

El qué, cuándo y cómo son algunas preguntas que contienen un gran significado en el mundo de la tecnología de la informática que tienen como único objetivo la creación y desarrollo de software.

En todo software se cumple un ciclo de vida que determina el inicio y el final y que durante este periodo existe la transformación, la evolución ó el desecho del mismo software.

Por lo que en cada proceso es necesario tener los conocimientos para el entendimiento de la necesidad solicitada por el usuario, como se menciona dentro del mundo del software el objetivo es construir un software ó mejorar uno existente.

Por tal motivo se realiza un proceso que proporciona normas para el desarrollo eficiente de calidad al igual que la captura que representa mejores prácticas actuales de la tecnología dentro de las cuatro "P", mejorando la arquitectura que en su consecuencia reduce el riesgo y hace un software más predecible, el efecto que se crea es el fomentar una nueva visión a futuro y ser capaz de evolucionar muchos años.

Un software es aquel que sirve como guía para todos los participantes, clientes, usuarios, desarrolladores y por que no decirlo ejecutivos, necesitamos que este ampliamente disponible de forma que todos los interesados puedan comprender su papel en el desarrollo en el que se encuentran implicados.

1.2 ¿Qué es un Sistema?

La palabra sistema se emplea actualmente en muchos ámbitos distintos; el termino sistema se emplea para designar un concepto, una herramienta genérica que se puede emplear para explicar, analizar mejor como es, ó que ocurre en una determinada área, entonces podemos decir que el sistema es un conjunto de casos que se relacionan entre si ordenadamente contribuyen a un determinado objetivo. A partir de esta definición podemos identificar cuales son los principales elementos presentes en cualquier sistema tales como los componentes del sistema, la relación entre ellos, que determinan la estructura del sistema y el objetivo sistema.

Dentro de los sistemas se identifican elementos importantes para comprender como son y como funcionan entorno all sistema que significa en donde esta ubicado y cuales son los límites ó frontera que existe entre el sistema y la contribución de su entorno.

Por lo tanto un sistema es un procedimiento formal para producir algún resultado, es usado como conocimiento para ayudar a resolver problemas en el cual que usa la tecnología como herramienta para implementar nuestra solución, por medio de un instrumento automatizado ó para realizar algo de la mejor manera posible. Utilizando los artefactos que se necesitan para representar en forma comprensible por máquinas u hombres, para las máquinas, los trabajadores y los interesados. Recordemos que las máquinas son las herramientas, compiladores u ordenadores destinatarios. La construcción del sistema es por lo tanto un proceso de construcción de distintos modelos para describir todas las perspectivas diferentes del sistema.[1]

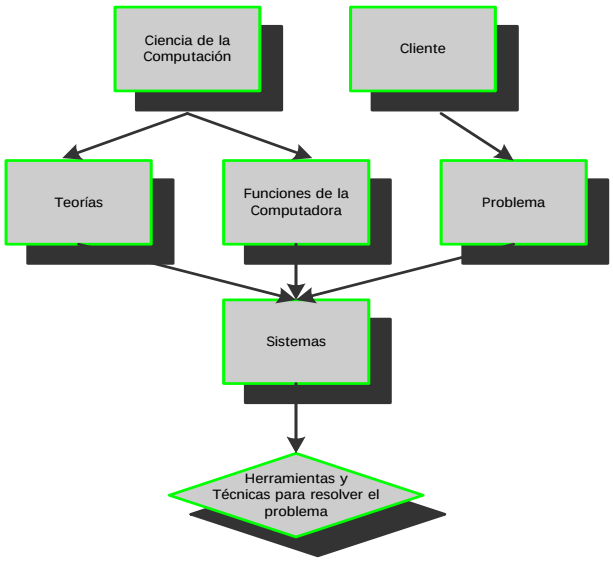


Fig. 1. Esquematización del Sistema

[1] Introducción a la Ciencia de la Computación de la manipulación de datos a la teoría de computadora
Pág. 22
Autor: Behrouz A. Forrouzan con colaboración de Shopia Cheng Fegan Thomson

1.3 Ciclo de Vida de un Sistema

El ciclo de vida de un software es la descripción de las distintas formas de desarrollo de un proyecto o aplicación informática, es decir, la orientación que debe seguirse para obtenerse, a partir de los requerimientos del cliente, puedan ser definido como el conjunto de fases, etapas, procesos y actividades requeridas para ofertar, desarrollar, probar, integrar, explotar y mantener un sistema.

Podemos decir que un ciclo de vida nos ayuda a comprender lo que esta sucediendo en una organización en donde se busca solucionar el problema bajo el enfoque por fases del análisis y diseño de lo cual se encuentra establecido bajo las siete etapas del ciclo de vida del desarrollo de sistemas.[2]

Un modelo de ciclo de vida del software representa todas las actividades y productos de trabajo necesarios para desarrollar un sistema de software.

Las siete etapas son descritas a continuación:

1. **La identificación, oportunidad y objetivos**: Primera etapa del ciclo de vida del desarrollo de un sistema en donde el analista tiene que ver con la identificación de problemas.

2. **La determinación de los requerimientos de información**: Segunda etapa del ciclo de vida es determinada por las herramientas utilizadas para definir los requerimientos como son el muestreo e investigación de datos relevantes, entrevistas, cuestionarios. Los principales involucrados son los analistas y los usuarios que realizan las preguntas ¿Qué?, ¿Dónde?, ¿Cuándo? y ¿Cómo?

3. **El análisis de las necesidades del Sistema**: Tercera fase del ciclo de vida se determina por las necesidades del sistema utilizando diagramas de flujo de datos para diagramar la entrada, proceso y salida de las funciones en forma gráfica estructurada. En este punto del ciclo el analista prepara una propuesta del sistema proporcionando alternativas.

4. **El diseño del sistema recomendado**: Cuarta fase del ciclo de vida determina la utilización de la información recolectada anteriormente para realizar el diseño lógico del sistema de información. El analista diseña procesamientos precisos para la captura de datos a fin de que los datos que van a entrar al sistema de información sean correctos; recordemos que la interfaz conecta al usuario con el sistema por lo tanto es extremadamente importante.

5. **El desarrollo y documentación del Sistema**: Quinta fase del ciclo de vida del desarrollo de sistemas, el analista trabaja con los programadores para desarrollar cualquier software original que se necesite las técnicas estructuradas para el diseño y documentación influyen diagramas estructurados teniendo en conclusión que los programadores tienen un papel principal en esta etapa conforme se diseñan, codifican y eliminan los errores de sintaxis.

6. **La prueba y mantenimiento del Sistema**: Sexta etapa del ciclo de vida manifiesta que el sistema de información debe ser aprobado, las pruebas son realizadas por los programadores solos y otra por los analistas junto a los programadores el mantenimiento del sistema y de su

[2] Ingeniería del Software Teoría y Practica
Pág. 196
Autor: Shari Lawrence Pfleeger

documentación comienza en esta fase y es efectuado rutinariamente a lo largo de la vida del sistema.

7. **La implementación y evolución del Sistema**: Séptima y última etapa del ciclo de vida de desarrollo de un sistema es donde se incluye el entrenamiento de los usuarios para el correcto manejo del sistema al igual la evolución se muestra en esta etapa para la realización de la modificación de programación o cambio de materiales para su diseño y uso.[3]

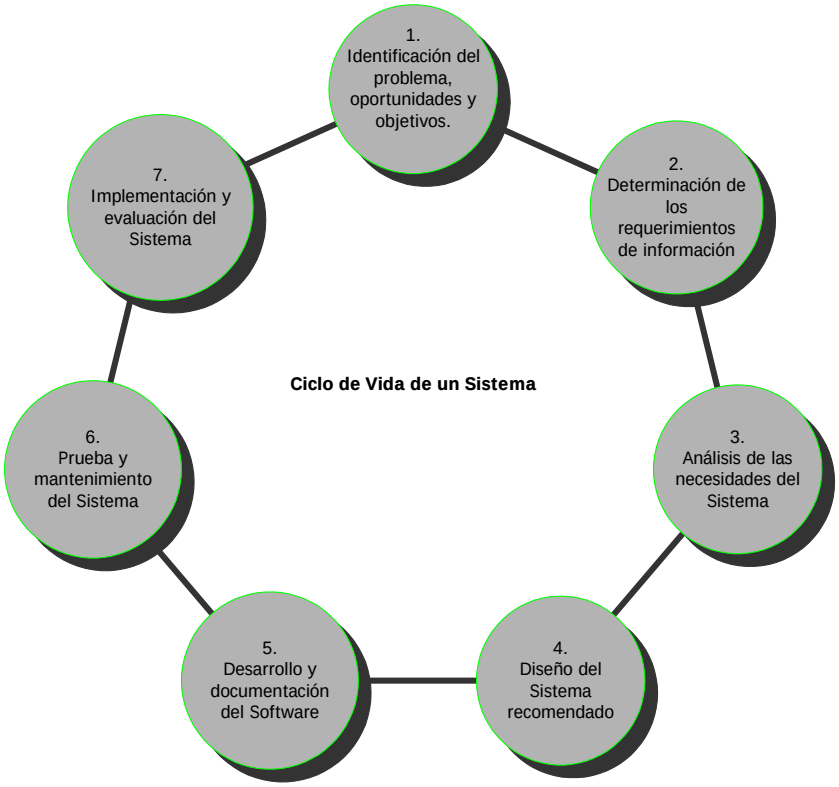


Fig. 2. Etapas del ciclo de Vida

[3] Análisis y Diseño de Sistemas de Información
Pág. 151-155 Autor: Cláudio F. Freijedo, Alicia B. Cortagerena.

Las principales funciones de un ciclo de vida de un sistema es el determinar el orden de las fases y procesos involucrados en el desarrollo del software y su evolución. Establecer los criterios de transición para pasar de una fase a la siguiente es decir criterios para seleccionar e iniciar las fases. El ciclo de vida de un sistema da respuesta a las siguientes preguntas; ¿Qué haré a continuación? y ¿Cuánto tiempo continuare haciéndolo?

Las ventajas que aporta el enfoque de ciclo de vida residen en lo siguiente:

- En las primeras fases, aunque no haya líneas de código, es necesario pensar en el diseño es avanzar en la construcción del sistema, pues posteriormente resulta más fácil la codificación.
- Asegurar un desarrollo progresivo, con controles sistemáticos, que permite detectar precozmente los defectos.
- Se controla el sobrepasar los plazos de entrega y los costes excesivos, mediante un adecuado seguimiento del proceso.
- La documentación se realiza de manera formal y estandarizada simultáneamente al desarrollo, lo que facilita la comunicación interna entre el equipo de desarrollo y la de este con los usuarios.

En conclusión podemos decir que un ciclo de vida de un software describe la vida de un sistema desde su concepción hasta su implementación, entrega, utilización y mantenimiento.

Un proceso de desarrollo de un sistema debe describirse de manera flexible que permita que las personas que lo diseñaron y contribuyeron a la creación del software utilizando las herramientas y técnicas preferidas.

Recordando que en el desarrollo en el ciclo de vida de un sistema implica un análisis, un diseño, una implementación y finalmente pruebas.

1.3.1 El Producto dentro del ciclo de vida

Cada ciclo produce una nueva versión del sistema, y cada versión es un producto preparado para su entrega. Consta de un cuerpo de código fuente incluido en componentes que puede compilarse y ejecutarse, además de manuales y otros productos asociados.

Sin embargo, el producto terminado no solo debe ajustarse a las necesidades de los usuarios, si no a las de todos los interesados, es decir, toda la gente que trabajara con el producto debería ser algo más que el código máquina que se ejecuta. El producto terminado incluye los requisitos, casos de uso, especificaciones no funcionales y casos de prueba. Incluye el modelo de la arquitectura y el modelo visual; esto permite a los usuarios utilizar y modificar el sistema de generación en generación.

De hecho el que los requisitos cambien es una incógnita para el desarrollo del software, al final los desarrolladores deben afrontar un nuevo ciclo.

1.3.2 Fases dentro del Ciclo de Vida de un Sistema

El desarrollo de un ciclo de vida implica un análisis, un diseño, una implementación y finalmente pruebas.

Fase de Análisis:

El proceso de desarrollo comienza con la **fase de análisis**, la cual muestra que debe hacer el paquete. En esta fase, el analista de sistemas define los requisitos que especifican lo que el sistema propuesto va a lograr. Los requisitos por lo general se establecen en los términos que el usuario comprende, existen cuatro pasos en la fase de análisis: definición de usuarios, definición de las necesidades, definición de los requisitos y definición de los métodos.[4]

Definición de Usuarios	Es un paquete de software en donde se puede diseñar para un usuario genérico o para un usuario especifico. Se debe definir con claridad.
Definición de las necesidades	Después de que se ha identificado el usuario, los analistas definen claramente las necesidades. En este paso, la mejor respuesta viene del usuario, es decir, define la claridad de sus expectativas del paquete.
Definición de los requisitos	Con base a las necesidades del usuario, el analista puede definir con precisión los requisitos para el sistema.
Definición de los métodos	Finalmente después de que se define los requisitos en términos claros, el analista puede elegir los métodos apropiados para cumplir estos requisitos.

Fase de Diseño:

La **fase de diseño** define el logro de la definición de la fase de análisis, determinando el sistema a utilizar, diseñando la base de datos / archivos teniendo en cuenta la modularidad y las herramientas.

Modularidad	La modularidad dentro del diseño utiliza un principio la Modularidad que se obtiene por medio de módulos los cuales son diseñados, probados y vinculados con programas iniciales, es decir que estén definidas sus tareas.
Herramientas	En esta fase de diseño utiliza varias herramientas, siendo la más común un diagrama estructurado en el cual muestra como dividir un paquete en pasos lógicos, cada paso es un módulo independiente el cual también muestra la interacción entre todos los módulos.

[4] Ingeniería del Software Teoría y Práctica
Pág. 196-198
Autor: Sharí Lawrence Pfleeger

Fase de Implementación:

En **la fase de implementación** se crean los programas reales existiendo dos pasos: herramientas y codificación.

Herramientas	Esta fase utiliza varias herramientas para mostrar el flujo lógico de los programas antes de la escritura real del código. Una herramienta aún generalizada, es el diagrama de flujo el cual utiliza símbolos gráficos estándar para representar el flujo lógico de datos a través de un módulo. La segunda herramienta utilizada por los programadores es el pseucódigo parte del idioma y parte lógica del programa que describe, con detalles algorítmicos precisos, que indican que hace el programa lo cual requiere definir con tanto que la conversión a un programa de computadora pueda lograrse fácilmente.
Codificación	Después de la producción en un diagrama de flujo, pseucódigo ó ambos, el programador en realidad escribe el código en un lenguaje específico para un proyecto. La opción de un lenguaje se basa en la eficiencia del lenguaje para esa aplicación en particular.

Fases de prueba:

Una vez que los programas se han escrito, deben probarse. La fase de prueba puede ser muy tediosa y consumir parte del tiempo del desarrollo del programa. Los programadores son completamente responsables de probar sus programas. En los proyectos de desarrollo grandes, con frecuencia hay especialistas llamados ingenieros de pruebas quienes son responsables de probar el sistema como un todo, es decir, de hacer pruebas para comprobar que todos los programas trabajen en conjunto. Hay dos tipos de pruebas: caja negra y caja blanca. La prueba de la caja negra es realizada por el ingeniero de pruebas del sistema y por el usuario. Las pruebas de la caja blanca es responsabilidad del programador.[5]

Pruebas de Caja Negra	Reciben el nombre por concepto de probar un programa sin saber que hay dentro y sin saber como funciona. En pocas palabras el programa es una caja negra en la cual no se puede ver, es decir, los planes de prueba se desarrollan considerando solo los requisitos por esta razón es muy importante tener un buen conjunto de requisitos definidos. El conocimiento en el desarrollo de sistemas y el entorno de trabajo del usuario para crear un plan de pruebas, este plan se utilizara después cuando el sistema se pruebe como un todo. Estas pruebas deben ser realizadas antes de que escriba el código.
Pruebas de Caja Blanca	Mientras en la caja negra suponen que no se debe saber nada acerca del programa, las pruebas de la caja blanca asumen todo acerca del programa. En este caso el programa es como un vidrio transparente en el cual todo es visible, las pruebas de la caja blanca son responsabilidad del programador, quien sabe exactamente que sucede dentro del programa. Todas las instrucciones posibles y todas las situaciones posibles se hayan probado. El programador diseñara un plan de prueba pero algo que puede hacer desde el principio es el escribir planes de prueba desde la etapa de diseño, a medida que construya su diagrama de estructura, diagramas de flujo, casos de uso.

[5] Ingeniería del Software Teoría y Practica
Pág. 192-196
Autor: Shari Lawrence Pfleeger

Representación gráfica de las fases del ciclo de Vida de un Sistema

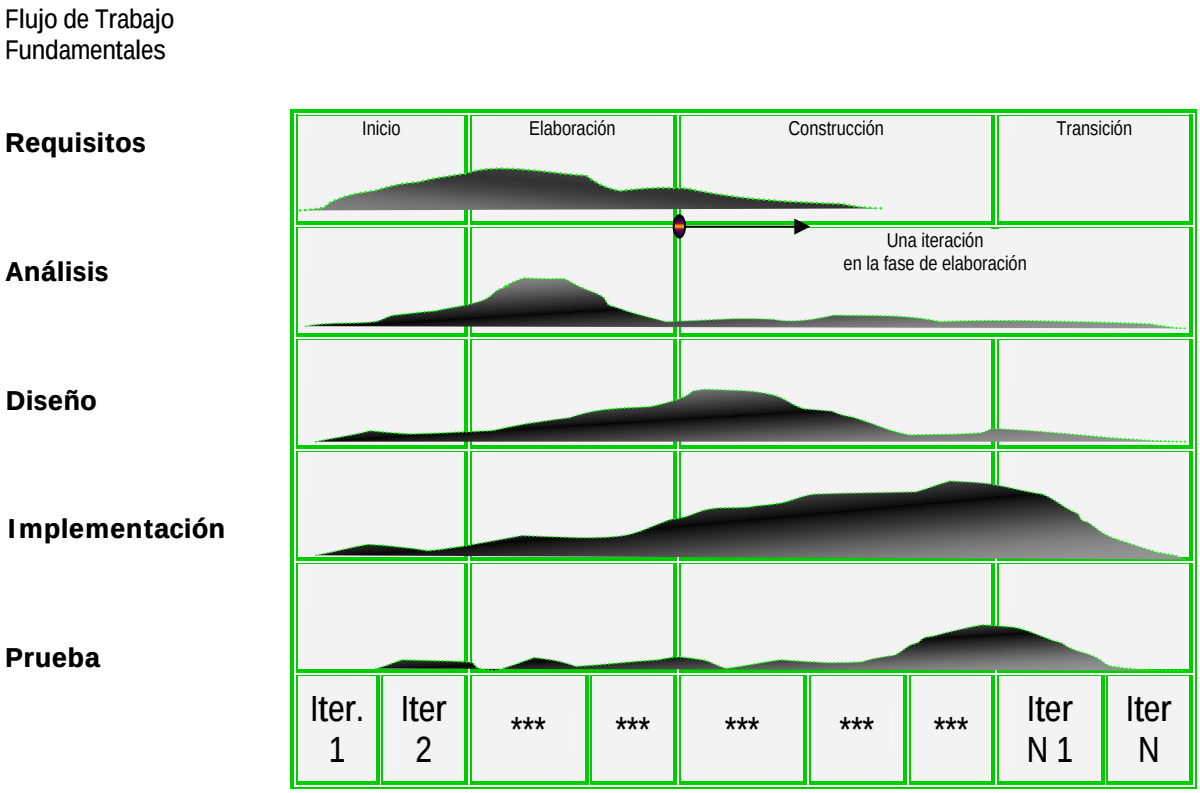


Fig. 3 Fases del ciclo de Vida de un Sistema[6]

[6] El proceso Unificado de Desarrollo de Software
Pág. 11
Ivar Jacobson, Grady Booch y James Rumbaugh

1.4 Las cuatro "P" para el desarrollo de un Sistema

El resultado final de un proyecto de software es un producto que toma forma durante su desarrollo gracias a la inversión de muchas personas. Un proceso de desarrollo es una guía realizada por el esfuerzo de las personas implicadas en el proyecto, a modo de plantilla que explica los pasos necesarios para determinar el proyecto.

Los términos personas, proyecto, producto y proceso son las cuatro "P" existentes dentro del desarrollo de un software.

Personas:

Los principales autores de un proyecto de software son los arquitectos, desarrolladores, ingenieros de prueba y el personal de gestión que les da soporte, además de los usuarios, clientes y otros interesados.

Las personas son realmente los que se encuentran implicados en el desarrollo de un software durante todo un ciclo de vida, financian el producto, lo planifican, lo desarrollan, lo gestionan, lo prueban, lo utilizan y se benefician con el. Por lo tanto el proceso que guía este desarrollo debe orientarse a las personas que son seres humanos, a diferencia del término abstracto trabajadores. Se manejan 6 puntos importantes para su funcionamiento.

Viabilidad del proyecto	La mayoría de la gente no disfruta trabajando en proyectos que parecen imposibles, una aproximación iterativa en el desarrollo permite juzgar pronto la viabilidad del proyecto.
Gestión del riesgo	De igual forma cuando la gente siente que los riesgos no han sido analizados y reducidos, la exploración de los riesgos significativos en las primeras fases atenúa este problema.
Estructura de los equipos	La gente trabaja de manera más eficaz en grupos pequeños de 6 a 8 miembros. Un proceso que produce trabajo significativo como el análisis de un determinado riesgo, el desarrollo de un subsistema o el llevar la iteración proporciona esta oportunidad. Una buena arquitectura con interfases bien definidas entre subsistemas y componentes hace posible una división del esfuerzo de este tipo.
Planificación del proyecto	La planificación de un proyecto no es realista, las técnicas que se utilizan en la fase de inicio y de elaboración permite a los desarrolladores tener una buena noción de cual debería ser el resultado del proyecto es decir, de lo que debería hacer la versión del producto.
Facilidad de compresión del proyecto	A los programadores les gusta saber lo que están haciendo; quieren tener una comprensión global. La descripción de la arquitectura proporciona una visión general para cualquier que se encuentre implicado en el proyecto.
Sensación de cumplimiento	En un ciclo de vida iterativo, la gente recibe retroalimentación frecuentemente, lo cual a su vez, hace llegar a conclusiones, la retroalimentación frecuente y las conclusiones obtenidas aceleran el ritmo de trabajo. Un ritmo de trabajo rápido combinado con las conclusiones frecuentes aumenta la sensación de progreso del proyecto.

Proyectos:

Un proyecto de desarrollo da como resultado una nueva versión de un producto. El primer proyecto dentro del ciclo de vida de desarrollo a veces llamado proyecto inicial desarrolla y obtiene el sistema o producto inicial.

Para una representación más completa de la gestión de proyecto. A través de un ciclo de vida, un equipo de proyecto debe preocuparse del cambio, las iteraciones y el patrón organizativo.

Una secuencia de cambio	Los proyectos de desarrollo de un sistema obtiene productos como resultados, pero un camino hasta obtenerlos es una serie de cambios. Este hecho de la vida de un proyecto debe tenerse en mente mientras los trabajadores progresan a través de las fases e iteraciones, cada ciclo, cada fase y cada iteración modifican el sistema de ser una cosa a otra distinta, el primer ciclo de desarrollo es un caso especial que convierte el sistema de nada en algo. Cada ciclo lleva a una versión y más allá de una secuencia de ciclos, el cambio continuo por generaciones.
Una serie de iteraciones	Dentro de cada fase de un ciclo, los trabajadores llevan a cabo las actividades de la fase a través de una serie de iteraciones. Cada iteración implementa un conjunto de casos de uso relacionados atenuando algunos riesgos. En una iteración los desarrolladores progresan a través de una serie de flujos de trabajo: requisitos, diseño, implementación y prueba. Debido que cada iteración pasa por cada uno de esos flujos de trabajo, podemos pensar en una iteración como si fuese un mini proyecto.
Un patrón organizativo	Un proyecto implica a un equipo de personas asignadas para lograr un resultado dentro de las restricciones del negocio, es decir, tiempo, coste y calidad. La idea de “proceso” es proporcionar un patrón dentro del cual las personas en su papel de trabajadores ejecutan un proyecto.

Producto:

El producto que se obtiene es un sistema software se refiere al sistema entero, y no solo al código que se entrega. Un sistema es todos los artefactos que se necesitan para representarlo en una forma comprensible por máquinas o cualquier interesado entre los trabajadores tenemos directores, arquitectos, desarrolladores, ingenieros de prueba.

Procesos:

Los procesos varían porque tienen lugar en diferentes contextos, desarrollan diferentes tipos de sistemas y se ajustan a diferentes tipos de restricciones como plazos, costes, calidad y fiabilidad, por consiguiente un proceso de desarrollo de software del mundo real debe ser adaptable y configurable para cumplir con las necesidades reales de un proyecto y/o organización concreta.[7]

[7] El proceso Unificado de Desarrollo de Software
Pág. 14
Autor: Ivar Jacobson, Grady Booch y James Rumbaugh

Los principales factores que influyen en un proceso son:

Factores organizativos	La estructura organizativa, la cultura de la empresa, la organización y la gestión del proyecto, las aptitudes y habilidades disponibles, experiencias previas y sistemas software existentes.
Factores del dominio	El dominio de la aplicación, procesos de negocio que deben soportar, la comunidad de usuarios y las ofertas disponibles de la competencia.
Factores de ciclo de vida	El tiempo de salida al mercado, el tiempo de vida esperado del software, la tecnología y la experiencia de las personas en el desarrollo de software, y las versiones planificadas y futuras.
Factores técnicos	Lenguaje de programación, herramientas de desarrollo, base de datos, marcos de trabajo y arquitecturas "estándar" subyacentes, comunicaciones y distribución.

1.5 Arquitectura de un Sistema

La arquitectura de un programa ó sistema informático es la estructura del sistema, que comprende componentes de software, las propiedades externamente visibles de estos componentes y las relaciones entre ellos involucrando la descripción de los elementos a partir de los cuales se construyen los sistemas, las interacciones entre dichos elementos, patrones que guían en su composición y restricciones entre dichos elementos.

La arquitectura global de un sistema es importante no solo porque es fácil implementarlo y probarlo si no también en cuanto a la velocidad y efectividad del mantenimiento y el cambio.
La calidad de la arquitectura puede hacerlo o deshacer un sistema, de hecho en Shaw y Garlan en 1996 presenta a la arquitectura como un disciplina en si, cuyos efectos se sienten y son relevantes a lo largo de todo el proceso de desarrollo.

La estructura arquitectónica debe reflejar los principios del buen diseño del sistema lo describe en términos de un conjunto de unidades arquitectónicas y un mapa que muestra como se va a relacionar unas con otras.

Cuanto más independientes son las unidades, más modular es la arquitectura y tanto más fácilmente se pueden diseñar y desarrollar las piezas por separado. [8]

1. Descomposición modular, basada en la asignación de funciones a los módulos.
2. Descomposición orientada por datos, basada en la estructura externa de los datos.
3. Descomposición orientada por eventos, basada en eventos que el sistema debe tratar.
4. Diseño de afuera hacia dentro basada en las entradas del usuario al sistema.
5. Diseño orientado a objetos, basado en la identificación de clases de objetos y sus interrelaciones.

Es lo que especifica el arquitecto en la descripción de la arquitectura. La descripción de la arquitectura permite al arquitecto controlar el desarrollo del sistema desde la perspectiva técnica.

[8] Ingeniería del Software Teoría y Practica
Pág. 36
Autor: Shari Lawrence Pfleeger

La arquitectura software se centra tanto en los elementos estructurales significativos del sistema, como subsistemas, clases, componentes y nodos, como en las colaboraciones que tienen lugar entre estos elementos a través de las interfaces. Los casos de uso dirigen la arquitectura para hacer que el sistema proporcione la funcionalidad y su uso deseado alcanzando a la vez por su objetivo rendimiento deseable. Una arquitectura debe ser completa, pero también debe ser suficiente flexible como para incorporar nuevas funciones y debe soportar la reutilización del software existentes.

1.5.1 Características de una Arquitectura

Existen una serie de características que ofrecen calidad a una arquitectura, éstas son las siguientes:

1. **Facilidad de modificación:** Una estructura de diseño de un sistema debe estar sujeta y disponible a realizar cualquier cambio necesario, donde puede ser preciso tener que separar funciones de la tecnología usada, aislar información.

2. **Reutilizable:** Las implementaciones obtenidas en un sistema ó unidades de diseño deben estar disponibles para poder ser reutilizado en otro sistema.

3. **Portable:** Un sistema debe ser portable hacia cualquier otra plataforma de hardware.

4. **Edificable:** El sistema debe ser fácil de implementar y construir, y se debe elegir bien las herramientas para realizar los procesos de construcción; aunque muchas veces está característica viene impuesta por el cliente, siempre es importante estudiarla y ver que es lo más conveniente.

5. **Fácil de Probar:** El sistema debe ser fácil de probar, la definición de las pruebas debe estar basada en la arquitectura del sistema.

6. **Manejable:** El sistema debe proveer una interfaz intuitiva y fácil de manejar para los usuarios.

7. **Buen rendimiento:** El rendimiento del sistema, una vez que está en la producción debe ser óptimo y adecuado sobre los tiempos de respuesta que proporcione.

8. **Segura:** El sistema debe prever cualquier acceso sin autorización y denegarle el acceso a funciones no permitidas para un usuario en concreto.

9. **Fiable:** El sistema debe obtener unos resultados fiables y concretos, además debe ser tolerante a fallos.

10. **Escalable:** El sistema debe permitir la escalabilidad para ello debe ser posible utilizar diferentes recursos duplicados, como distintos procesadores, memorias.

11. **Actualizable:** El sistema debe ser fácilmente actualizable con nuevas versiones de desarrollo.[9]

[9] Análisis y Diseño de Aplicaciones Informáticas de Gestión
Pág. 639-640
Análisis y Diseño de Aplicaciones Informáticas de Gestión
Una perspectiva de Ingeniería del Software

1.5.2 ¿Por qué es necesaria una Arquitectura?

Un sistema de software grande y complejo requiere de una arquitectura para que los desarrolladores puedan progresar hasta tener una visión común, un software es difícil de abarcar visualmente porque no existe un mundo de tres dimensiones; es único y sin precedentes en determinados aspectos.
Suele utilizar tecnologías poco probadas o una mezcla de tecnologías nuevas, tampoco es común que el sistema lleve a los últimos límites de la tecnología existentes. Además el sistema debe ser construido para acomodar grandes cantidades de clases que sufrirán cambios futuros a medida que los sistemas se hacen más complejos.[10]
Se necesita una arquitectura para:

- Comprender el sistema.
- Organizar el desarrollo.
- Fomentar la reutilización.
- Hacer evolucionar el sistema.

1.5.3 ¿Cómo se Obtiene?

La arquitectura se desarrolla de forma iterativa durante la fase de elaboración pasando por los requisitos, el análisis, el diseño, la implementación y las pruebas. Utilizando los casos de uso significativos para la arquitectura y un conjunto de otras entradas para implementar la línea base de la arquitectura o esqueleto del sistema. Los principales objetivos son:

- Recopilar la mayor parte de los requisitos que aún queden pendientes, formando los requisitos funcionales como casos de uso.
- Establecer una base de la arquitectura sólida, la línea base de la arquitectura para guiar el trabajo durante las fases de construcción y transición, así como en las posteriores generaciones del sistema.

Se tomara decisiones de la arquitectura basados en la comprensión del sistema en su totalidad, su ámbito y sus requisitos funcionales y no funcionales, como el rendimiento hasta llegar a un equilibrio entre los requisitos representados en los casos de uso y el modelo de casos de uso y la arquitectura. La arquitectura se obtiene bajo decisiones importantes sobre:

- La organización del sistema software.
- Elementos estructurales que compondrán el sistema y sus interfaces, junto con sus comportamientos tal y como se especifican en las colaboraciones entre estos elementos.
- La composición de los elementos estructurales y el comportamiento en subsistemas progresivamente más grandes.

[10] El proceso unificado de desarrollo de software
Pág. 58
Autor: Ivar Jacobson, Grandy Booch y James Rumbaugh

La arquitectura del software esta afectada no solo por la estructura y el comportamiento, si no también por el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, las restricciones y compromisos económicos y tecnológicos y la estética.

1.5.4 ¿Cómo se Describe?

La descripción de la arquitectura es sencillamente un extracto adecuado de los modelos del sistema.

La primera versión de la descripción de la arquitectura es un extracto de la versión de los modelos que tenemos al término de la fase de elaboración en el primer ciclo de vida.

La descripción de la arquitectura se parece mucho a los modelos normales del sistema, esta apariencia significa que la vista de la arquitectura solo contiene los casos de uso significativos para la arquitectura mientras que el modelo de casos uso final contiene todos los casos de uso.

La descripción de la arquitectura tiene cinco secciones, una para cada modelo. Tiene una vista del modelo de casos de uso, una vista del modelo de análisis, una vista del modelo de diseño, una vista del modelo de despliegue y una vista al modelo de implementación.

No incluye una vista del modelo de prueba por que no desempeña ningún papel en la descripción de la arquitectura y solo se utiliza para verificar la línea base de la arquitectura.

En conclusión decimos que la descripción de la arquitectura es una vista de los modelos del sistema, de los modelos de casos de uso, análisis, diseño, implementación y despliegue.

La descripción de la arquitectura describe las partes del sistema que es importante que comprendan todos los desarrolladores y otros interesados.[11]

[11] El proceso Unificado de Desarrollo de Software
Pág. 78
El Proceso Unificado de Desarrollo de Software
La guía completa del Proceso Unificado escrita por sus creadores

1.6 El futuro de los Sistemas también llamado Software

La investigación y el desarrollo de técnicas y métodos de ingeniería del software son constantes y suelen suponer interesantes avances en la resolución de problemas de desarrollo de software. Sin embargo, es habitual que en la práctica profesional no se incluya prácticamente ninguna de las recomendaciones más elementales de la ingeniería del software. En efecto, es habitual que el desarrollo del software, de hecho, las evoluciones de los procesos productivos de software realizadas a raíz del modelo de procesos de software confirman que el desarrollo de software suele estar básicamente en estado caótico. La dinámica del mercado y la tradición imperante en la gestión de las ofertas de software no fomenta el uso de planificaciones no se realiza en las mejores condiciones para los analistas y jefes de proyecto. Una de las sensaciones más habituales de los profesionistas de la informática en general, y el desarrollo de software en particular, consiste en la impresión de desbordamiento ante la avalancha constante de nuevas tecnologías, entornos, siglas, etc. Muchos sitios Web suelen ser atractivos pero han descuidado aspectos básicos de estructura y diseño que generan un infierno cuando hay que abordar el mantenimiento (otra vez el mantenimiento de software tan decisivo en la problemática de la crisis del software durante los años setenta y ochenta).
Precisamente ahora que la tremenda confusión en el desarrollo de software orientado a objetos va resolviéndose para bien o para mal, con la concepción global de **UML** [Ivar Jacobson, Grandy Booch y James Rumbaugh], como notación de trabajo. Parece que la ingeniería del software va avanzando y retrocediendo continuamente, movida continuamente por las variaciones en el mercado de tecnologías de la información. En cualquier caso la ingeniería del software tiene habitualmente que responder a posteriores cambios en la tecnología de software, es decir, deberá aportar métodos y técnicas para su desarrollo y mantenimiento una vez que se conozca adecuadamente las características de dichas novedades técnicas en el mundo del software. No obstante la verdadera tendencia que definitivamente parece que el software en general ha asumido no solo desde el punto de vista de ingeniería de software es la necesidad de constatar empíricamente las propuestas de programación, de técnicas, de métodos etc. Para los desarrolladores de software un posible futuro podría ser:

- La planificación de los proyectos se hace con rigor y este bien ajustada a la complejidad y tamaño de las aplicaciones y de los requisitos.

- Las especificaciones son el producto de un exhasivo y sólido trabajo de interacción con el cliente / usuario y cuenta con su aprobación, incluido el acuerdo sobre los criterios de aceptación de la aplicación.

- Se cuenta con las colecciones de componentes, funciones, realmente contralodos, especificados y gestionados que solventan buen parte de la generación del código, pasando los ingenieros de software a efectuar tareas de diseño más relacionadas con la arquitectura del software.

- En todo el desarrollo, el jefe de proyecto cuenta con una cuidadosa selección de medidas e indicadores que le permitan conocer y controlar en todo momento el estado real del trabajo.[12]

[12] Ingeniería del Software Teoría y Práctica
Pág. 710
Autor: Sharí Lawrence Pfleeger

Capítulo 2

Metodologías

2.1 Introducción

Dentro de la creación de un software siempre ha sido necesario basarse en una metodología para el desarrollo y la construcción de uno o varios ciclos de vida.

Pero al igual las metodologías de desarrollo de software son visiones particulares de estrategias de resolución de problemas y de modelos software aplicados a la construcción de software dirigido a resolver problemas del mundo real.

Las metodologías también consideran aspectos sociales y de organización del desarrollo de software que emplea métodos para el desarrollo de sistemas automatizados. La Metodología contempla una notación, un proceso y herramientas que crean para guiar la estimación de costos, manejo del proyecto en las tareas, medidas y métricas, las políticas y procedimientos para garantizar la calidad del software.

En conclusión es necesario establecer un enfoque disciplinado y sistemático para desarrollar un proyecto de software. Las metodologías de desarrollo, influyen directamente en este proceso de construcción, se elabora a partir del marco definido por uno o varios ciclos de vida explicados. Veremos qué características debe tener una metodología de desarrollo de software, y cómo varían en función del enfoque de desarrollo, y daremos una visión histórica de la evolución de las metodologías comentando las más representativas.

2.2 ¿Qué es una Metodología?

Las metodologías imponen un proceso disciplinario sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Pero también recordemos que una metodología es una colección de técnicas basadas sobre una filosofía ó una forma de trabajo, que se establece en conjunto sobre una plataforma llamado Ciclo de vida del desarrollo de un software, la metodología corresponde a un conjunto de modelos, lenguajes y otras herramientas que facilitan la representación de los datos de cada fase del proceso junto con las reglas que permiten el paso de una fase a la siguiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar inspirado por otras disciplinas de la ingeniería. Las metodologías han estado presentes durante mucho tiempo, no se han distinguido precisamente por ser muy exitosas, aún menos por su popularidad.

La crítica más frecuente a estas metodologías es que son burocráticas. Hay tanto que hacer para seguir una metodología que el entero desarrollo se retarda. La metodología es considerada como el conjunto de pasos y procedimientos que se deben seguir para el desarrollo de un software. Según Maddison define a la metodología como un "conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas de información". Cómo el dividir un proyecto en etapas, qué tareas se llevan a cabo en cada etapa, qué salidas se producen y cuando se deben producir, qué restricciones se aplican, qué herramientas se van utilizar y como gestionar y controlar un proyecto.

En conclusión podemos decir que una metodología son procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software teniendo en cuenta las mejores aplicaciones un mejor proceso de desarrollo y un proceso estándar en la organización siempre teniendo en cuenta que las metodologías cuentan con distintos objetivos; como el registro de requisitos, un método sistemático que controle la construcción correctamente documentada del sistema, la identificación de los cambios realizados del proyecto ó el sistema que cubre las necesidades requeridas por usuario.

Por lo que una metodología por lo tanto representa el camino para desarrollar un software de una manera sistemática, de forma general, podemos identificar tres necesidades principales que se intentan cubrir en una metodología:

- Mejores aplicaciones si consideramos mejores sistemas a los de mejor calidad, hay que tener en cuenta que el seguimiento de una metodología no basta para asegurar la calidad del producto final. En general. El valor de los sistemas de información resultantes dependen de la multitud de pequeños factores.

- Un mejor proceso de desarrollo que identifica las salidas de cada fase de forma que se pueda planificar y controlar el proyecto, así, los sistemas se desarrollan más rápidamente y con los recursos apropiados.

- Un proceso estándar en la organización lo que aporta claros beneficios. [13]

[13] Metodologías de Desarrollo de Software Pág. 81 Análisis y diseño de Aplicaciones Informáticas de Gestión Una perspectiva de Ingeniería del Software

2.2.1 Características de las Metodologías

Todo entorno de desarrollo incluye un conjunto de componentes que condicionan la construcción de software. En este esquema muestra la relación entre la metodología y el entorno de desarrollo, dentro del entorno, la organización mantiene a un equipo de desarrollo de software, los procedimientos de gestión determinan el tipo de soporte automatizado tanto hardware como software que debe proporcionar, especificando las herramientas necesarias a utilizar para el desarrollo.
La organización de desarrollo tiene dos opciones:

- Seleccionar entre un gran número de posibilidades y combinaciones de métodos de gestión, técnicas de desarrollo y soporte automatizado, para crear y desarrollar la metodología de desarrollo software más apropiado.
- Analizar y evaluar las metodologías existentes y adoptar en la organización a lo que más ajuste a las necesidades.

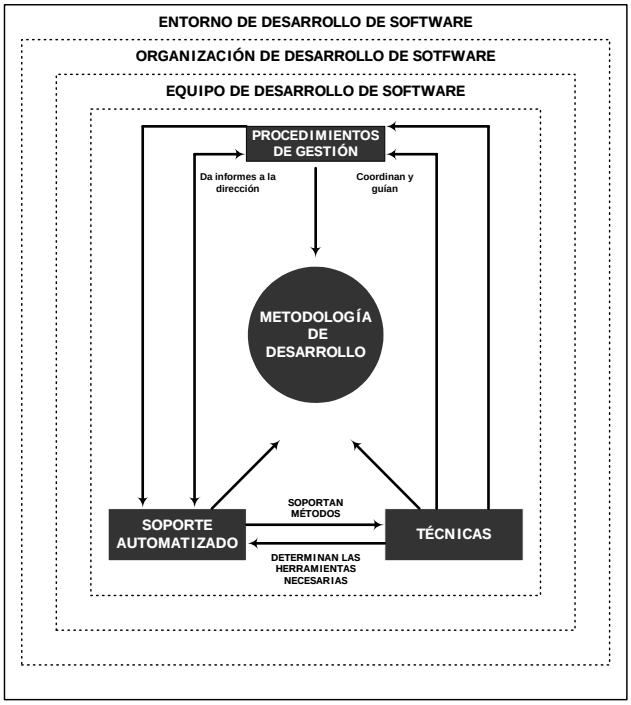


FIG. 4. Función de la Metodología

Las características deseables de una metodología:

1. La existencia de reglas predefinidas sobre sus fases, las tareas, los productos intermedios, las técnicas y herramientas los cuales tienen como fin y objetivo tener un estándar; el seguir pasos desde un planteamiento hasta el mantenimiento proporcionando resultados para su comprobación.

2. Cobertura total del ciclo desarrollo en donde hay que realizar desde el planteamiento de un sistema hasta su mantenimiento, proporcionando mecanismos para integrar los resultados de una fase a la siguiente, de forma que pueda referencial a fases previas y comprobar el trabajo realizado.

3. Verificaciones intermedias que se contemplaran en la realización de verificaciones sobre los productos generados en cada fase para comprobar su corrección, se realizan por medio de revisiones de software que detectan las inconsistencias, inexactitudes ó cualquier otro tipo de defecto que se genera durante el proceso de desarrollo, evitando que salga a relucir en la fase de pruebas.

4. Enlace con Procesos de Gestión en donde la metodología debe proporcionar una forma de desarrollar software de manera planificada, controlada y de calidad. Por ello, si bien no se incluyen procesos de gestión de proyectos en una metodología, esta debería dar pautas ó recomendaciones para enlazar las actividades de desarrollo del proyecto con actividades propias de su gestión, como son la planificación, control y seguimiento del proyecto, aseguramiento de la calidad del software ó gestión de configuración.

5. Comunicación efectiva proporciona un medio de comunicación efectiva entre los desarrolladores para facilitar el trabajo en grupo y con los usuarios.

6. Utilización sobre un abanico amplio de proyectos debe ser flexible, para que pueda emplearse sobre un amplio abanico de proyectos, tanto en variedad, tamaño y entorno. Una organización no debería utilizar metodologías diferentes para cada proyecto sino que la metodología se debe amoldar a un proyecto concreto.

7. Fácil formación para todo el personal de desarrollo de una organización, por lo que los desarrolladores deben comprender las técnicas y los procedimientos de gestión. La organización debe formar a su personal en todos los procedimientos definidos por la metodología.

8. Herramientas CASE debe estar soportada por herramientas automatizadas que mejoren la productividad del equipo de desarrollo y la calidad de los productos resultantes.

9. La metodología debe contener actividades que mejoren el proceso de desarrollo teniendo disponibles los datos que muestren la efectividad de la aplicación del proceso sobre un determinado producto.

10. Soporte al mantenimiento en donde se considera técnicas para la evaluación lógica del sistema, que se encuentran en un entorno cambiante, debido principalmente a los cambios tecnológicos y a las nuevas necesidades de los usuarios.

11. Soporte de la reutilización de software se debería incluir procedimientos para la creación, mantenimiento y recuperación de componentes reutilizables que no se limiten sólo al código.[2]

En todo desarrollo de aplicaciones pasa a través de una metodología algunas veces más rigurosas que otras pero es necesario seguir los pasos para detectar una metodología como son:

Especificar los requerimientos: Los requerimientos son las condiciones que debe cumplir con el sistema para satisfacer las necesidades de sus clientes y usuarios aún cuando lo que se busque sea la utilización de un sistema previamente preparado, probado y/o comercializado, la especificación de los requerimientos es fundamental para el éxito final de cualquier esfuerzo de información.

Muchas veces los clientes que incursionan en el desarrollo de los sistemas y hasta los propios analistas de sistemas y programadores, caen en la trampa de creer que basta con hablar acerca de lo que se quiere por sistema. Los requerimientos de los sistemas no solo deben identificarse si no que además deben especificarse apropiadamente utilizando lenguajes y técnicas especiales.

Otro de los errores frecuentes es la secuencia de la documentación formal generada en la mediad que se avanza en el proyecto, evaporándose así la oportunidad de capitalizar el esfuerzo y la inversión hecha hasta el momento. Hay que tener en cuenta que los requerimientos de un sistema pueden variar fundamentalmente considerando que estos provengan de futuros usuarios, operadores, administradores y/o gerentes teniendo a veces que satisfacer partes conflictivas.

Modular los procesos: La primera técnica a tomar en cuenta se conoce como modelaje de procesos. Para ello aplicamos una ayuda gráfica llamada el diagrama de flujo de datos ó **DFD**. Muchos analistas y programadores creen que la técnica consiste en pintar pelotitas y flechitas los componentes Diagrama de Flujo de Datos deben cumplir con reglas bastantes estrictas de construcción y significado, deben ser globalmente coherentes y reflejar de manera verédigna la lógica de los procesos que se quieren representar lo cual debe ponerse a prueba una y otra vez antes de dar por terminado su desarrollo.

Modelar los datos: El modelo de datos es la primera aproximación al diseño de la base de datos. Al igual que el modelo de procesos, también se aplica una técnica gráfica denominada el diagrama de entidades y relaciones ó **DER**. El Diagrama de Entidad Relación también tiene sus propios componentes gráficos y reglas de construcción así como significado. Igualmente no basta con pintar cajas y líneas; el Diagrama de Entidad Relación debe reflejar la estructura de datos y las reglas de negocio. Un DER debe someterse a prueba muchas veces antes de darlo por terminado.

Diseñar la base de datos: El diseño de la base de datos es el primer punto verdaderamente crucial en la construcción de un sistema. Metodológicamente se llega al modelo de datos, puede decir que en cierto modo es la implementación del Diagrama de Entidad Relación en un manejador de bases de datos. El diseño de la base de datos es el primer responsable de que se pueda ó no satisfacer los requerimientos de información de los usuarios del futuro sistema, otro no menos importante y a veces olvidado es la disponibilidad de datos. Las bases de datos deben cumplir con requisitos técnicos y muy

[2] Metodologías de Desarrollo de Software
Pág. 89-93
Análisis y diseño de aplicaciones informáticas de Gestión una perspectiva de Ingeniería del Software

importantes para la salud general del sistema como son las reglas de normalización. Una base de datos desnormaliza esta propensa a producir errores de inconsistencias, problemas de redundancias, recalculó y complejidades en su desarrollo.

Diseñar las interfaces Usuarias: En el lenguaje de las herramientas actuales de programación, las interfaces usuarias definen la manera como los usuarios se relacionan con el sistema. Básicamente podemos decir que se reducen a un juego de ventanas cuyos componentes permiten la interacción. Muchas veces el diseño de las ventanas este estrechamente relacionado con el de la base de datos en otras la relación no es tan evidente dependiendo de la naturaleza del tema subyacente así como de las necesidades de interacción requeridas por los usuarios.

Diseñar un repertorio de consultas e informes: Las consultas y reportes provienen de las exigencias de los usuarios en cuanto a obtención de información. Lo usual es identificar y definir tales exigencias y traducirlas a consultas y reportes. Particularmente nosotros contamos las consultas y reportes como parte de la interfase usuaria.

Implementar diseños: Si los diseños se emprenden y la implementación son simultáneos. Por otra parte la implementación de ciertas funcionalidades de la interfase usuaria solo se logra llenado acertadamente ciertas propiedades ó desarrollando código de programación.

Probar el Sistema: Siempre que se crea un nuevo producto y los sistemas no son una excepción al respecto lo correcto es someterlo a un periodo de pruebas en las cuales nos percatemos si cumplen con los requerimientos para los cuales fueron hechos y/o se corrigen las discrepancias detectadas.

Adiestrar a los Usuarios: Los usuarios deben recibir un adiestramiento que les permita conocer a fondo el uso de los diferentes componentes del sistema y sus interrelaciones.

Redactar Manuales para los Usuarios: Un elemento fundamental en un sistema y que frecuentemente falta ó es muy somero es una buena documentación para los usuarios. El manual para usuarios no solo debe decir lo que el sistema hace si no como lo hace.

Controlar el Acceso: Dependiendo del nivel de control de acceso los datos que se deben tener, ciertas acciones sobre los datos pueden estar reservados para personas autorizadas por lo cual puedes hacerse necesaria la incorporación de un sistema de control de acceso.

Implantar el Sistema: Implantar el sistema es todo un reto adicional. Un sistema puede ser todo un éxito como concepto, diseño e implementación y sin embargo fracasar en la etapa de implementación. Si un sistema no se implementa difícilmente podremos justificar el esfuerzo y costo invertido en el.

Recordemos que una metodología contempla las verificaciones de los productos generados en cada fase teniendo planificada, controlada y con calidad; teniendo en cuenta que se tiene que tener una comunicación entre los desarrolladores y el usuario para llegar a un objetivo con técnicas, procedimientos de gestión con un soporte por herramientas automatizadas teniendo la visión de mejorar el proceso de desarrollo pero no olvidando el mantenimiento que no es considerada técnica lógica para el sistema debido al cambio tecnológico y las necesidades del usuario llevando acabo como consecuencia a la reutilización del software. Hemos observado que las metodologías van evolucionando; primeramente tenemos un periodo de desarrollo convencional en donde solo eran

artesanales y no existían metodologías definidas, teniendo como consecuencia a la evolución el desarrollo estructurado en el cual esta basado en métodos de análisis y diseño estructurado, llegando así a la culminación del ciclo de vida, pero recordemos que existe la incógnita de la visión futura del software orientada a objetos.

2.3 Clasificación de las Metodologías

Todas las metodologías distinguen estas tres fases: Análisis, Diseño, Implementación.

Según Piattini se puede observar un cambio "filosófico entre las metodologías clásicas de análisis y diseño estructurado y las de orientación al objeto. En las primeras se examinan los sistemas desde el punto de vista de las funciones o tareas que deben realizar, tareas que van descomponiendo sucesivamente en otras tareas más pequeñas y que forman los bloques ó módulos de las aplicaciones. En la orientación al objeto, por su parte, cobra más importancia el aspecto de "modelado" del sistema, examinando el dominio del problema como un conjunto de objetos que interactúan entre sí.
En las metodologías tradicionales se produce una dicotomía entre los dos elementos constituyentes de un sistema: funciones que llevan a cabo los programas y datos que se almacenan en ficheros o base de datos.

Desarrollo Convencional	Se basa principalmente en funciones básicas en donde los programadores se encargan en codificar más no en recoger y comprender las necesidades de los usuarios, teniendo en este desarrollo 3 problemas serios para su enfoque: los resultados finales son impredecibles, es decir, no se sabe cuando se termina el proyecto ya que depende de los programadores más no de los usuarios. El darse cuenta de lo que esta pasando con el proyecto: Por lógica al no existir fases establecidas no se puede saber cual es el estado del proyecto teniendo en consecuencia la tardía detección de defectos que pueda presentar creando los altos costos estando ya en servicio por lo cual es necesario tener puntos de control. Los cambios organizativos: Son los que modifican el proceso de desarrollo, se maneja la estandarización de la documentación actualizada adecuadamente.
Desarrollo Estructurado	El desarrollo se basa a partir de la construcción de programas seguidos de los métodos sustentados las bases para el desarrollo automatizado. Teniendo una programación estructurada tanto estática como dinámica, es decir, que se entendible. Los métodos estructurados comenzaron a desarrollar-se a fines de los 70's con la programación Estructurada, luego a mediados de los 70's aparecieron técnicas para el Diseño primero y luego para el Análisis. Enfocados a implementaciones usando lenguajes de 3ra generación. Ejemplos de metodologías estructuradas gubernamentales: MERISE (Francia), MÉTRICA 3 (España), SSADM (Reino Unido).
Desarrollo Orientado a Objetos	La incógnita visual futura del software orientada a objetos a diferencia de desarrollo estructurado trata los procesos en forma modular tanto de información como de procesamiento. La orientación a objetos da comienzo con la programación de lenguajes. Teniendo como significado que el sistema se organiza como una colección de objetos que interactúan entre si y que contienen tanto estructuras de datos.

2.4 Descripción de las Metodologías

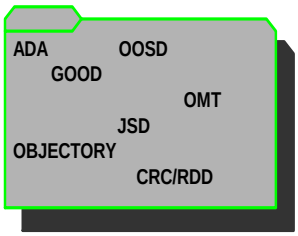
Existen diversas metodologías las cuales se han desarrollado de acuerdo a las necesidades de cada usuario determinando por uso de cada una de ellas la cual es la más conveniente dependiendo del tiempo en que se ha ido desarrollando la tecnología en el mundo real de la industria del software. Como son:

Metodologías	Traducción	Siglas	Autores
Advance Data Analysis	Lenguaje de Programación Estructurada	**ADA**	Jean Ichbiah
ADM3		**ADM3**	Donald Firesmith
Berard		**BOOM**	Berard
Business Object Notation	Notación de Objetos de Negocio	**BON**	Nerson
Colbert		**COLBERT**	E. Colbert
Class Responsibility and Collaboration	Técnica de clases, responsabilidad con la colaboración de clases.	**CRC**	Kent Beck & Ward Cunningham
Desfray		**DESFRAY**	Softeam
Software Development Methodology Research Group	Especificaciones Orientadas a Objetos Entidad Relación	**EROOS**	
Frame Object Analysis	Marco de Análisis de Objetos	**FOA**	Andleigh & Gretzingr
Fusion	Método de Fusion	**FUSION**	Coleman y otros
General Object Oriented Design	Método de Diseño General OO	**GOOD**	Seidewitz & Stark
Hierarchical Object Oriented Design	Diseño Jerárquico Orientado a Objetos	**HOOD**	ESA
Medhea	Metodología para el diseño de hiperdocumentos	**MEDHEA**	M. Piattini
Methodology for Object Oriented Software Engineering of Systems	Metodología de desarrollo de software de calidad	**MOSES**	Henderson – Sellers & Edwards
Metrica		**METRICA**	Allan Albrecht
Object Behaviour Analysis	Análisis del funcionamiento de Objetos	**OBA**	Rubin & Goldberg
Object Modeling Technique	Técnica de Modelado Orientada a Objetos	**OMT**	James Rumbaugh
Object Oriented Analysis	Análisis Orientado a Objetos	**OOA**	Coad & Yourdon
Object Oriented Design	Diseño Orientado a Objetos	**OOD**	Grady Booch
Object Oriented Jackson Structured Design	Diseño estructurado de Jackson Orientado a Objetos	**OOJSD**	Jackson
Object Oriented Process Environment and Notation	Proceso orientado al objeto, entorno, y notación	**OPEN**	Henderson - Sellers
Object Oriented Role Analysis and Modeling	Objeto orientado sobre el análisis en síntesis y estructuración	**OORAM**	Reenskaug
Object Oriented Role Analysis, Synthesis and Structuring		**OORASS**	Reenskaug et al
Object Oriented Software Engineering	Ingeniería del Software Orientada a Objetos	**OOSE**	Jacobson
Object Oriented Structured Design	Diseño Estructurado Orientado a Objetos	**OOSD**	Wasserman y otros
Object Oriented System Analysis	Sistema de Análisis Orientado a Objetos	**OSA**	Embley, Kurtz y Woodfield
Ptech		**OOA&D**	Martin & Odell
Responsibility Driven Design	Diseño conducido responsable	**RDD**	Wirfs-Brock y otros
Rational Unified Process	Proceso Racional Unificado	**RUP**	Booch, Rumbaugh y Jacobson
Scrum		**SCRUM**	Jeff Sutherland
Semantic Object Modelling Approach	Objeto semántico que modela acercamiento	**SOMA**	Ian Graham
Shlaer & Mellor Method	Método de Shlaer & Mellor	**SSM**	Shlaer & Mellor
Syntropy		**SYNTROPY**	Cook y otros
Systems Engineering OO	Ingeniería de Sistemas Orientado a Objetos	**SEOO**	LBMS
Texel		**TEXEL**	Texel
UML	Lenguaje de Modelado Unificado	**UML**	Grady Booch, Ivar Jacobson y James Rumbaugh
Visual Modeling Technique	Tecnología del Objeto usando la programación visual	**VMT**	IBM
Extreme Programming	Programación Extrema	**XP**	Kent Beck

2.4.1 De las cuales se encuentran subdivididas por generaciones

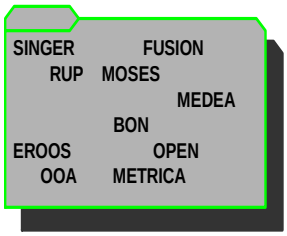
Primera Generación: La primera generación de lenguajes se remonta a los días en que se codificaba a nivel de máquina. Todavía continúan llevándose a cabo bastantes trabajos con lenguajes de primera generación. El código máquina y su equivalente más humanamente legible, el lenguaje ensamblador, representan la primera generación de lenguajes. Estos lenguajes dependientes de la máquina muestran el menor nivel de abstracción con el que se puede representar un programa. El lenguaje de máquina está formado por cadenas de ceros y unos por lo tanto para realizar un programa se necesitan de programadores altamente entrenados. Algunos ejemplos de lenguajes de esta generación presentaban las características de abstracción matemática, estructura física plana y consistían únicamente de datos globales y subrutinas o subprogramas. Como consecuencia de esto un error podía tener un gran efecto e influía en todo el programa, gracias a que las estructuras globales de datos eran accesibles por todas las subrutinas. Existen tantos lenguajes ensambladores como arquitecturas de procesadores con sus correspondientes conjuntos de instrucciones. Desde un punto de vista de la ingeniería del software, esos lenguajes sólo se deben usar cuando un lenguaje de alto nivel no satisfaga los requisitos o no esté disponible.

Primera Generación



Segunda Generación: La segunda generación de lenguajes fue desarrollada a finales de los años 50 y principios de los 60 y ha servido como base para todos los lenguajes de programación modernos (tercera generación). La segunda generación de lenguajes está caracterizada por su amplio uso, la enorme cantidad de bibliotecas de software y la gran familiaridad y aceptación. Prácticamente nadie pone en duda que son lenguajes de base, debido a su madurez y su aceptación. Han subsistido a 30 años de críticas y sigue siendo lenguajes de programación en el ambiente científico y de ingeniería. La versiones han resultado ser una potente herramienta para la resolución de problemas computacionales; aunque le faltaba el soporte directo de estructuras de control, tenía una tipificación de datos pobre, no facilitaba un soporte a la manipulación de cadenas y tenía algunas otras deficiencias. En muchos casos, ha sido forzado a ajustarse a áreas de aplicación para las que no fue nunca diseñado, por lo que muchas de las críticas que ha recibido los lenguajes han sido injustas.

Segunda Generación



Tercera Generación:

Los lenguajes de tercera generación (también denominados lenguajes de programación moderna o estructurada) están caracterizados por sus potentes posibilidades procedí mentales, abstracción de datos, compilación de módulos en forma separada, orientación a objetos y estructuración de datos. Los lenguajes de esta clase se pueden dividir en tres amplias categorías, lenguajes de alto nivel de propósito general, lenguajes de alto nivel orientados a los objetos y lenguajes especializados. Los lenguajes especializados, han sido diseñados para satisfacer requisitos especiales y, a menudo, tienen una formulación y una sintaxis únicas.

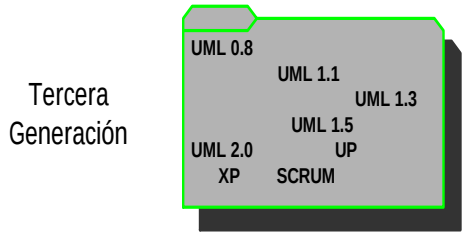


Fig. 5. Metodologías por Generación

2.4.2 Descripción de las metodologías

Año	Metodología	Autor	Descripción
1979	**METRICA** **Metrica**	**Allan Albrecht**	Nace para resolver la problemática que presentan los departamentos de Tecnologías de información que no utilizan ninguna metodología de desarrollo. El principal objetivo de la metodología es crear un entorno que permita al equipo de trabajo construir sistemas que den solución a los objetivos considerados prioritarios en la organización que se desarrollen cuando el usuario los necesite, tenga el soporte a cambios a futuro de la organización. La metodología de desarrollo de sistemas METRICA fue diseñada por la subdirección general de coordinación informática del ministerio para las administraciones públicas ante la necesidad de disponer con una metodología teniendo como característica principal es su flexibilidad ya que se adapta a gran variedad de sistemas y ciclos de vida.
1980	**CRC** **Class Responsibility and Collaboration** **Técnicas de clases, responsabilidad con la colaboración de clases**	**Kent Beck & Ward Cunningham**	A mediados de la década de 1980, Kent Beck y Ward Cunningham trabajaban en un grupo de investigación de Tektronix; allí idearon las tarjetas CRC y sentaron las bases de lo que después serían los patrones de diseño y XP. CRC denota "Clase-Responsabilidad-Colaboración", y es una técnica que reemplaza a los diagramas en la representación de modelos. En las tarjetas se escriben las Responsabilidades, una descripción de alto nivel del propósito de una clase y las dependencias primarias. La gran mayoría de los autores de metodologías de desarrollo orientadas a objetos, coinciden en que la identificación de un conjunto apropiado de clases y su correcta asignación de responsabilidades, son los pilares fundamentales de un diseño orientado a Objetos. Cada metodología tiene sus propios métodos o técnicas para lograr ambos propósitos. CRC ayuda a: **1. Identificar las clases que participan del diseño del sistema.** **2. Obtener las responsabilidades que deben cumplir cada clase.** **3. Establecer cómo colabora una clase con otras clases para cumplir con sus responsabilidades.** La técnica CRC pertenece a la segunda categoría y sirve como catalizadora de ideas. Si bien la técnica CRC tiene varios propósitos, en este artículo se hace énfasis en el uso de está técnica como una forma de enseñar el pensamiento orientado a objetos.
1983 **1986** **1991**	**ADA** **Advance Data Analysis** **Lenguaje de Programación Estructurada**	**Jean Ichbiah**	Ada es un lenguaje de programación estructurado y fuertemente tipado de forma estática que fue diseñado por Jean Ichbiah de CII Honeywell Bull por encargo del Departamento de Defensa de los Estados Unidos (DoD). Es un lenguaje multipropósito, orientado a objetos y concurrente, pudiendo llegar desde la facilidad de Pascal hasta la flexibilidad de C++. Fue diseñado con la seguridad en mente y con una filosofía orientada a la reducción de errores comunes y difíciles de descubrir. Para ello se basa en un tipado muy fuerte y en chequeos en tiempo de ejecución (desactívales en beneficio del rendimiento). La sincronización de tareas se realiza mediante la primitiva rendezvous. Ada se usa principalmente en entornos en los que se necesita una gran seguridad y confiabilidad como la defensa, la aeronáutica (Boeing o Airbus), la gestión del tráfico aéreo (El lenguaje fue diseñado bajo encargo del DoD. Durante los años 1970, este departamento tenía proyectos en una infinidad de lenguajes y estaba gastando mucho dinero en software. Para solucionarlo se buscó un lenguaje único que cumpliese unas ciertas normas recogidas en el documento Steelman. Después de un estudio de los lenguajes existentes en la época se decidió que ninguno las cumplía totalmente, por lo que se hizo un concurso público al que se presentaron cuatro equipos, cuyas propuestas se nombraron con un color: Rojo (Intermetrics), Verde (CII Honeywell Bull), Azul (SofTEch) y Amarillo (SRI International). Finalmente en mayo de 1979 se seleccionó la propuesta Verde diseñada por Jean Ichbiah de CII Honeywell Bull, y se le dio el nombre de Ada. Esta propuesta era un sucesor de un lenguaje anterior de este equipo llamado LIS y desarrollado durante los años 1970. El nombre se eligió en conmemoración de lady Ada Augusta Byron (1816-1852) Condesa de Lovelace, hija del poeta Lord George Byron, a quien se considera la primera programadora de la Historia, por su colaboración y relación con Charles Babbage, creador de la máquina analítica. El lenguaje se convirtió en un estándar de ANSI en 1983 (ANSI/MIL-STD 1815) y un estándar ISO en 1987 (ISO-8652, 1987).

1983	**GOOD** **General Object Oriented Design** **Método de Diseño General Orientado a Objetos**	**Seiderwitz & Stark**	Desarrollado en la NASA en 1983 el cuál muestra un método al igual que sucede con Booch partiendo de un conjunto preliminar de diagramas de flujo de datos por capas llegando a la identificación de los objetos implicados, los procesos principales no lleva a un método abstracto del funcionamiento del sistema y siguiendo las entradas y salidas de flujos asociados a estos procesos se puede construir un conjunto de diagramas por capas.
1991	**OMT** **Object Modeling Technique** **Metodología de Análisis y Diseño Orientada a Objetos**	**James Rumbaugh**	Es una metodología orientada a objeto muy difundida, de hecho es la que actualmente más se está utilizando por encima incluso de la de Booch. Se desarrollo en el "Research and Development Center de General Electric"; a finales de los ochenta se hace cargo de todo el ciclo de vida del software y durante ese tiempo mantienen la misma notación, tiene una parte de diseño no muy compleja. Se centra mucho en un buen análisis. Es de las denominadas "dirigida por datos". Se divide en cuatro fases consecutivas las cuales son: 1. **Análisis de Objetos:** Se describe el problema, se obtiene uno de los requisitos que no den lugar a dudas (rendimiento, funcionalidad, contexto, etc.). En toda la fase de análisis se describe el comportamiento del sistema como una caja negra. Se hacen los diagramas de objetos con su diccionario de datos así se obtiene el modelo de objetos en el se define la estructura de los objetos y clases así como las relaciones que les unen. Comprende tanto un diagrama de clases como un diccionario de datos que las explique, así como la creación de un modelo dinámico para la descripción de los aspectos de control y evolución del sistema incluyendo diagramas de eventos, diagramas de estado por cada clase o por la creación del modelo funcional que utiliza diagramas de flujo de datos. 2. **Diseño del Sistema:** Comprende la arquitectura básica. En esta fase se tomarán las decisiones estratégicas a alto nivel de diseño es decir, la estructura global del sistema. 3. **Diseño de Objetos:** Es el último paso antes de implementar, y sirve para definir completamente todas las características de los objetos. Se detallan los 3 modelos ya descritos en el análisis de objetos de cara a poder implementarlo y optimizar el programa (acceso a datos, iteraciones, control, recursos, etc.); todo esto ha de hacerse con dependencia del lenguaje o entorno que finalmente se codifique y ejecute la aplicación. 4. **Implementación:** Se codifica lo ya diseñado.
1992	**OOSE** **Object Oriented Software Engineering** **Desarrollo de Aplicaciones Orientadas a Objetos**	**Ivar Jacobson**	Jacobson plantea una metodología de desarrollo de aplicaciones orientada a objetos. La metodología OOSE propone que el desarrollo de sistemas basado en el uso de distintos modelos. El ciclo de vida que ofrece la metodología se basa en la sucesión de dichos modelos soportados son: Modelo de Requisitos, Modelo de Análisis, Modelo de Diseño, Modelo de Implementación, Modelo de Prueba.

1992	**OOD** **Object Oriented Design** **Diseño Orientados a Objetos**	**Coad & Yourdon**	Es un diseño orientado a objetos que cuenta con cuatro características que son la identificación de clases y objetos as relaciones existentes así mismo la implementación, en el cual maneja al conjunto de objetos que comparten una estructura común y un comportamiento común. 1. **Diagrama de Clases:** Consisten en un conjunto de clases y relaciones entre ellas. Puede contener clases, clases paramétricas, utilidades y metaclases. 2. **Especificación de Clases:** Es usado para capturar toda la información importante acerca de una clase en formato texto. 3. **Diagrama de Categorías**: Muestra clases agrupadas lógicamente bajo varias categorías. 4. **Diagramas de transición de estados.** 5. **Diagramas de Objetos:** Muestra objetos en el sistema y su relación lógica. Pueden ser diagramas de escenario, donde se muestra cómo colaboran los objetos en cierta operación; o diagramas de instancia, que muestra la existencia de los objetos y las relaciones estructurales entre ellos. 6. **Diagramas de Tiempo:** Aumenta un diagrama de objetos con información acerca de eventos externos y tiempo de llegada de los mensajes. 7. **Diagramas de módulos:** Muestra la localización de objetos y clases en módulos del diseño físico de un sistema. Un diagrama de módulos representa parte o la totalidad de la arquitectura de módulos del sistema. 8. **Subsistemas:** Un subsistema es una agrupación de módulos, útil en modelos de gran escala. 9. **Diagramas de procesos:** Muestra la localización de los procesos en los distintos procesadores de un ambiente distribuido.
	FOA **Frame Object Analysis**	**Andleigh & Gretzingr**	Metodologías como la "Frame Object Analysis Diagrams" de Andleigh y Gretzinger, pretende combinar la metodología Entidad-Relación con las metodologías de Diseño Orientado por Objetos para diseñar, de forma adecuada, los sistemas de información modernos: a través de marcos gráficos de distintos tipos, el diseñador modela simultáneamente las Entidades del sistema de información, sus relaciones y las funciones desde el punto de vista del usuario en forma de jerarquías de menús que determinarán la interface gráfica con el usuario. Después de este primer modelaje de Entidades y Funciones, esta metodología contempla su refinamiento a través de nuevos marcos para poder determinar las clases de Objetos que componen el sistema (incluyendo atributos y comportamiento de estos objetos, relaciones de herencia, composición, etc.).
	OOSA **Object Oriented System Analysis** **Sistema de Análisis Orientado a Objetos**	**Shlaer & Mellor**	Es una metodología bastante rígida. Sus assets resultan fáciles de verificar. Es válido para proyectos de muy diversos tamaños. Incluye la planificación dentro de sus fases mediante el uso de una técnica denominada “matriz de proyecto”. Su forma de trabajar consiste en la separación del problema a solventar en problemas independientes, denominados “dominios”, y la división de los mismos en subsistemas para poder analizarlos en profundidad. Una vez diferenciados los dominios, se procede al análisis de cada uno de ellos consiguiendo un modelo de cada dominio: 1. División del problema en 4 tipos de dominios: de Aplicación (usuario), de Servicio (interfaz), de Arquitectura (datos y su control.) y de Implementación (sistema operativo). 2. Análisis del Dominio de Aplicación: Se crean varios modelos, para cubrir las especificaciones del cliente un Modelo de Información de los objetos en el que se reflejará el análisis conceptual. Luego un Modelo de Estados que refleje el comportamiento del sistema y por último una descripción de acciones específicas, usando A-DFDs. Para toda esta fase se pueden usar herramientas CASE. La metodología también define una serie de modelos de subsistemas adicionales. 3. Confirmación del análisis: Validar que cumple los requisitos y tiene la forma adecuada. Se hace una simulación mediante secuencias del tipo “Estado _ inicial-Comportamiento-Acción”, para comprobar si realmente la aplicación va a funcionar como deseamos. 4. Extracción de requisitos para los Dominios de Servicio. Se representan los dominios como elipses. Ahora, a todos los requisitos implícitos no descritos por el cliente (aquellos que tratan sobre la interrelación entre los dominios) se representan por una flecha entre los dominios afectados, denominada ‘puente’. Deben quedar reflejados darse todos estos requisitos, tanto los cualitativos como los cuantitativos. 5. Análisis de los Dominios de Servicio: Una vez que ya tenemos los requisitos de los dominios del cliente, pasamos a analizar este segundo tipo de dominios en los que subdividimos nuestro problema.

Año	Metodología	Autor	Descripción
	MOSES **Methodology for Object Oriented software Engineering of System** **Metodología de desarrollo de software de calidad**	**Henderson – Sellers & Edward**	La metodología MOSES cubre todo el ciclo de vida del desarrollo de software orientado a objeto, no sólo análisis y diseño sino la gestión de proyectos, planificación de negocio, mantenimiento de la aplicación y futuras mejoras. Incluye una notación completa que además es soportada por algunas de las herramientas CASE que existen en el mercado, así como unas técnicas de gestión avanzadas que no están disponibles en una metodología. Muchas industrias están adoptando MOSES como metodología de desarrollo de software de calidad. MOSES cuenta con un ciclo de vida de desarrollo de software basado en el modelo de ciclo de vida fuente orientada a objeto, un ciclo de vida del producto con 3 etapas orientadas al negocio, 5 fases solapadas orientadas a la parte técnica y un conjunto de 20 actividades que componen una detallada guía de pasos a seguir. Pero sus ventajas son: **1. Incorpora diagramas de modelización de negocio.** **2. Consigue procesos mejores.** **3. Soporte a la reutilización.** **4. Las extensiones y modificaciones son más fáciles de gestionar.** **5. Enfoque a la calidad.** **6. Una arquitectura flexible.** **7. Gestión adecuada de proyectos complejos.** **8. Soporte a herramientas CASE.**
	FUSION **Método de Fusion**	**Coleman y otros**	Fusión proporciona un método de desarrollo de software orientado al objeto, que abarca desde la definición de requisitos a la implementación en un lenguaje de programación. Es considerada como una metodología de segunda generación por que proviene de: **OMT: Modelo de Objetos** **CRC: Interacción de Objetos** **BOOCH: Visibilidad** Los métodos formales de pre y post condiciones. Proporciona un desarrollo que se divide en : Análisis, Diseño e Implementación. Ofreciendo notaciones para los modelos que describen varios aspectos del software al igual proporciona herramientas de gestión.
	SMM **(Shlaer & Mellor Method)** **Método de Shlaer & Mellor**	**Shlaer & Mellor**	Es una metodología bastante rígida. Sus assets resultan fáciles de verificar. Es válido para proyectos de muy diversos tamaños. Incluye la planificación dentro de sus fases mediante el uso de una técnica denominada “matriz del proyecto”. Su forma de trabajar consiste en la separación del problema a solventar en problemas independientes, denominados “dominios” y la división de los mismos en subsistemas para poder analizarlos en profundidad. Una vez definidos los dominios, se produce el análisis de cada uno de ellos consiguiendo un modelo de cada dominio: **1. División del problema en dominios:** Hay cuatro tipos de dominios de aplicación que pertenece al usuario, de servicio que es la interfaz, de arquitectura que son los datos y el control y de implementación que pertenece al sistema operativo. **2. Análisis del dominio de aplicación:** Se crean varios modelos, para cubrir las especificaciones del cliente como el modelo de información de los objetos en el que se reflejará el análisis conceptual, el modelo de estados que refleja el comportamiento del sistema, por último una descripción de acciones especificas. **3. Confirmación del análisis:** Validar que cumple los requisitos y tienen la forma adecuada. Se hace una simulación mediante secuencias del tipo “Estado _ inicial – Comportamiento _ Acción” para comprobar si realmente la aplicación va a funcionar como deseamos. **4. Extracción de requisitos para los dominios de Servicios:** Se representan los dominios como elipses. **5. Análisis de los Dominios de Servicio:** Una vez que ya se tienen los requisitos de los dominios del cliente se pasa a analizar este segundo tipo de dominio en la subdivisión del problema. **6. Especificación del dominio de Arquitectura:** En este dominio se especifican facetas de diseño que se usaran diagramas. **7. Construcción del dominio de Arquitectura:** Tiene dos tipos de componentes el mecanismo que es el proceso que lleva el sistema y la estructura que es la colección de datos organizados. **8. Codificación de los Modelos:** La codificación.

	EROOS **Software Development Methodology Research Group** **Especificaciones Orientadas a Objetos Entidad Relación**		Tiene una notación como una serie de estrategias para el desarrollo de software desde el análisis hasta la implementación. Todas las especificaciones que se hacen son declarativas, permitiendo posponer así decisiones de diseño e implementación antes de la codificación habiendo realizado un excelente análisis. La potencia de EROOS queda resumida en 3 puntos: 1. **Combina el modelado del comportamiento con el modelo de las estructuras a representar:** Una integración total de estos dos modelos proporcionaría al método de especificación unas propiedades de composición y descomposición mucho más potentes. Esto llevaría a un software mejor estructurado, mantenible y reutilizable. 2. **Descripción del Software declarativa (no operacional).** La primera descripción de lo que el sistema ha de hacer en lugar de cómo; permitiendo la transformación sin perder de vista el problema a solventar. Esto da un nivel de abstracción adecuado a cada fase del desarrollo. 3. **Separación del dominio del problema de su solución.** Esto crea un sistema flexible y cambiable, pero con un núcleo sólido. La funcionalidad del sistema se reflejará en ese núcleo, permitiendo que el sistema se adapte a las peticiones del usuario, en lugar de comenzar desde el principio cada ver que surja una nueva demanda. Una de las virtudes de EROOS es que el dar una notación y reglas a seguir, solo puede darse una solución a un problema.
	BON **Business Object Notation** **Notación de Objetos de Negocio**	**Nerson**	Es una metodología de Ingeniería de Software que se inscribe en la categoría de metodologías puramente orientadas al objeto. Tienen su origen en la filosofía de diseño y programación por contrato. BON es un método y una notación para un análisis y diseño orientada a objetos de sistemas de alto nivel. Algunas de sus características más destacadas son: 1. **Énfasis en permitir un desarrollo sin transiciones.** 2. **La reutilización del software a gran escala.** 3. **Reflejo de la confianza necesaria para hacer que los componentes reutilizables se acepten y utilicen por la industria del software.** Sus principios fundamentales son: **1. Desarrollo sin costuras:** El desarrollo sin costuras es el principio de utilizar un conjunto consistente de conceptos y notaciones a través del ciclo de vida del software, evitando lo que su falta genera en métodos tradicionales. Una ventaja muy importante del **seamlessness** es que facilita los procesos de traducción automática. **2. Reversibilidad:** Este principio complementa el proceso sin costuras, garantizando que los cambios realizados en cualquier paso del proceso, incluso en la implementación o el mantenimiento puedan reflejarse hacia atrás en los primeros pasos, incluyendo el análisis. Si esto no fuera posible los primeros niveles de modelado pasan a ser obsoletos dejando solamente el código fuente de la implementación como especificación del sistema, con todos los inconvenientes que esto supone. **3. Software contractual:** Con este principio la especificación de un sistema está distribuida entre todas sus partes componentes. Las responsabilidades de cada abstracción (clase) están especificadas por contratos expresados en función de otras abstracciones. **4. Simplicidad:** El principio de simplicidad indica que hay que minimizar el número de conceptos utilizados. **5. Escabilidad:** El principio de la escabilidad se manifiesta en términos de la capacidad de soportar un formalismo para representar grupos de clases progresivos y soportar la división del problema, basada en capas de abstracción como manejo de la complejidad estructural.

	OOA **Object Oriented Analysis** **Análisis Orientado a Objetos**	**Coad & Yourdon**	Cuenta con cuatro estructuras básicas para que es el análisis orientado a objetos que maneja la identificación de objetos, identificación de la estructura, especialización y generalización de las estructuras así como las múltiples, la definición de temas, la definición de estructuras como la identificación de los atributos, posicionar los tributos, la identificación de las instancias de conexión, casos especiales, especificación de atributos y la definición de servicios que es la identificación del estado del objeto.
	MEDHEA **Medhea** **Metodología para el Diseño de Hiperdocumento**	**M. Piattini**	Es una metodología para el diseño de hiperdocumentos, incluye el diseño lógico de la estructura de la navegación, el diseño didáctico y la fragmentación de la información en nodos, pero no abarca el diseño de los procesos informáticos que harán posible que el hipertexto funcione. Los modelos lógicos creados con MEDHEA podrán ser implementados utilizando un sistema de gestión de hipertextos o mediante la tecnología Web con el formato HTML.
	VMT **Visual Modeling Technique** **Tecnología del Objeto usando la Programación Visual**	**IBM**	Una metodología comprensiva que integre la programación visual en el ciclo vital orientado al objeto del desarrollo del uso. VMT amplía el ciclo vital perceptiblemente agregando a él la reingeniería de los procesos del negocio corporativo; además, la metodología captura las reglas de negocio de la empresa y las utiliza para aumentar la construcción del utilizar-caso de una manera muy de gran alcance. El paradigma de programación visual, cuando es apoyado por las herramientas eficaces, llega a ser central a la productividad de los reveladores del objeto. Los autores explican de una manera clara y sistemática cómo utilizar la programación visual para realizar eficientemente la construcción del mundo real el modelar y del uso de piezas reutilizables del software. VMT es una metodología probada que es utilizada por muchas corporaciones en Norteamérica, Europa, América latina, y Asia. Los laboratorios de las soluciones del software de la IBM han seleccionado VMT como la metodología. **La técnica que modela visual**: **1.** Explica cómo construir modelos del objeto por conocimiento de dibujo de utilizar casos, de descripciones del dominio, y de declaraciones del problema. **2.** Describe cómo construir componentes del modelo y de la visión con la programación visual. **3.** Examina el diseño de los usos distribuidos del objeto con los corredores de la petición del objeto y de las idiomas distribuidas. **4.** Ilustra cómo planear procesos del negocio y demuestra cómo construir las arquitecturas de la empresa basadas en procesos y objetos. **5.** Discute el papel de las reglas de negocio en requisitos que capturan. **6.** Presenta las ediciones de gerencia para considerar para oponerse tecnología. **7.** Incluye un estudio de caso de la vida real que se desarrolle a través del libro y destaca todos los aspectos de la metodología de VMT.
	RUP **Rational Unified Process** **Proceso Racional Unificado**	**Booch, Rumbaugh y Jacobson**	Es un proceso de desarrollo de software y junto con el lenguaje de modelado unificado constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Sus principales características son: **1. Asignación de tareas y responsabilidades quién hace qué, cuándo y cómo.** **2. Implementación de mejoras** Es decir, un desarrollo iterativo, administración de requisitos, uso de una arquitectura basada en componentes, control de cambios, modelado visual del software y verificación de la calidad del software. Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos y roles. RUP divide el proceso de desarrollo en ciclos, teniendo un producto final al final de cada ciclo, cada ciclo se divide en fases que finalizan con un hito donde se debe tomar una decisión importante.

Año	Metodología	Autor	Descripción
1999	**XP** **Extreme Programming** **Programación Extrema**	**Kent Beck**	Extreme Programming (XP) es una metodología que desde hace un par de años ha tomado un lugar importante en el desarrollo de software.
	OPEN **Object Oriented Process Environment and Notation** **Proceso Orientado a Objetos, Ambiente y Notación**	**Herderson & Seller**	OPEN fue creada en un principio a partir de una mezcla de algunas metodologías de segunda generación (MOSES, SOMA y The Firesmith Method). Es esencialmente un armazón para la tercera generación de métodos de desarrollo de software en la orientación al objeto con un gran soporte para el proceso de modelado mediante el uso de modelos de ciclo de vida, mediante la captura de requisitos ofrece la habilidad de modelar o construir agentes inteligentes. OPEN extiende el concepto de metodología no solo incluyendo un modelo de procesos, sino también suministrando líneas para construir versiones del método que se ajusten a las necesidades del dominio industrial, teniendo como elementos principales de esta metodología como el ciclo de vida o metamodelo, técnicas, representación.
1994 **1995** **1997** **1998**	**UML 0.8** **UML 1.1** **UML 1.3** **UML 1.5** **UML 2.0** **Unified Modelingn Language** **Lenguaje de Modelado Unificado**	**Grandy Booch, Ivar Jacobson James Rumbaugh**	Tras la aparición de los lenguajes orientados a objetos se buscaron nuevas metodologías que permitiesen el análisis y diseño de aplicaciones bajo dichos lenguajes; estas metodologías fueron los primeros lenguajes de modelado orientados a objetos. Al no poder cubrir éstos todas las necesidades de los desarrolladores, surgió una nueva generación de lenguajes más potentes liderados por el método de Booch, el método OOSE de Jacobson y el método OMT de Rumbaugh; cada uno de estos métodos destacaba en algunos puntos pero fallaba en otros. UML se comenzó a gestar en la empresa Rational cuando Booch y Rumbaugh decidieron unir sus métodos para conseguir un lenguaje estándar y sólido. Más tarde se incorporó Jacobson, lo que dio lugar a la versión 0.9 de UML en 1996; posteriormente se creó un consorcio con varias organizaciones interesadas en UML. La versión 1.0 de UML surgió en 1997 con la contribución de IBM, HP, Oracle, Microsoft y otras organizaciones. El desarrollo de UML continúa actualmente bajo el control de IBM (que adquirió Rational); la última versión de UML es la 2.0.

2.5 Representación gráfica de la evolución de UML

El "UML" es un lenguaje de modelización genérico, independiente del proceso de construcción de software, no pretende definir el proceso de desarrollo. Pero asume la existencia de un proceso con estas características: Guiado por Casos de Uso, Centrado en una Arquitectura, Iterativo, Incremental. Teniendo así mismo como Objetivos el modelar sistemas, no solo software, con los conceptos de la Tecnología de Objetos, establecer una relación explícita entre los distintos conceptos, diagramas y elementos, resolver los problemas de escala inherentes a los sistemas complejos y de misión crítica y crear un lenguaje de modelización utilizable tanto para seres humanos como para computadoras.

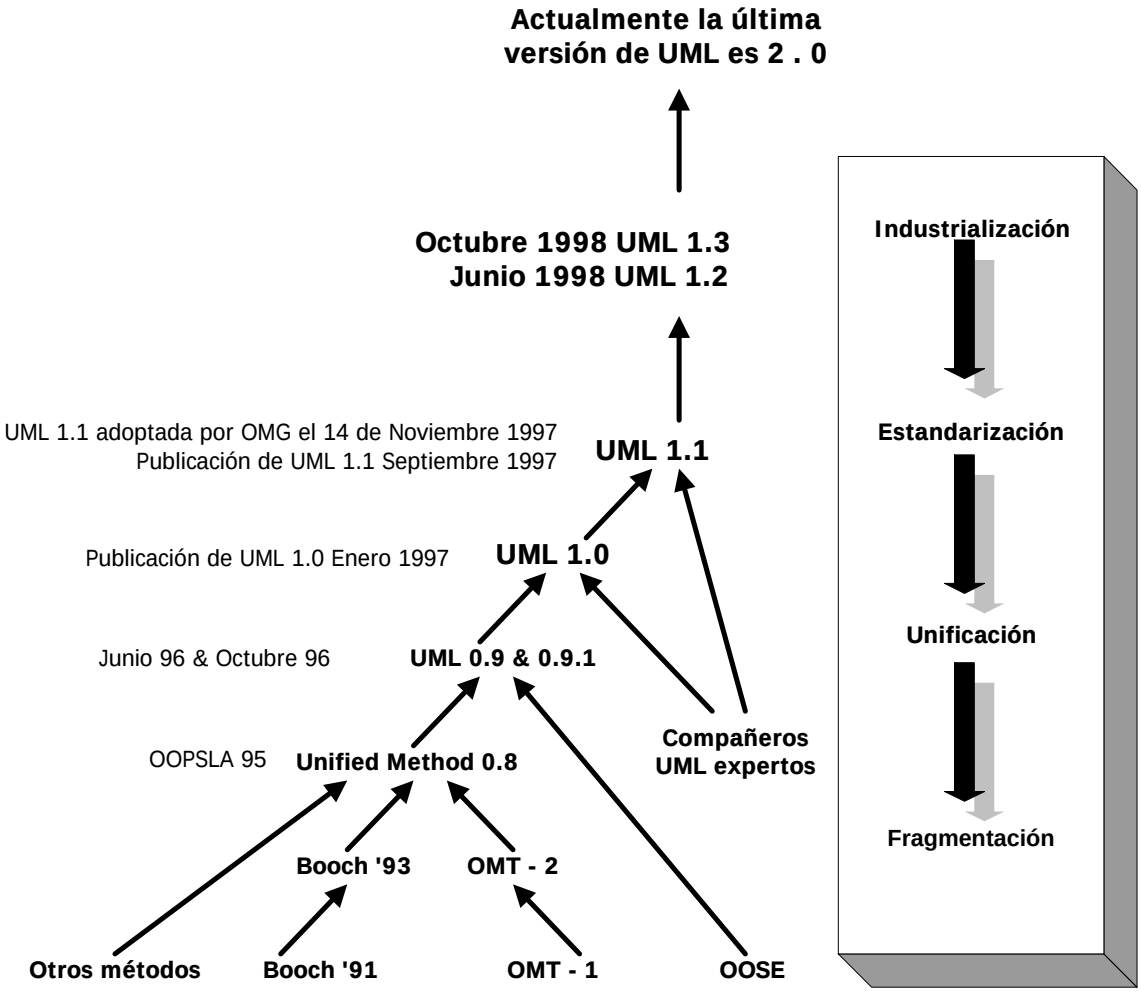


Fig. 6 Evolución de UML [3]

[3] www.rational.com.uml

Capítulo 3

UML Lenguaje de Modelado Unificado

3.1 Introducción

Este capítulo es esencial ya que determina que metodología se utilizara para llevar acabo el desarrollo de software. Es verdad el hecho de iniciar un software no es nada sencillo, ya que los desarrolladores por naturaleza hemos aprendido a trabajar encapsulado nuestros conceptos, modelos de lo que en diversas ocasiones tenemos problemas. Pues bien ahora nos damos a la tarea de buscar una metodología con el cual nos proporcione la comunicación y el marco para analizar el proyecto determinado. La elección que determinamos es UML; tenemos el conocimiento de que UML se deriva y unifica de tres metodologías de análisis y diseño orientado a objetos más extendidas; como la Metodología de Grady Booch para la descripción de conjuntos de objetos y sus relaciones; la Técnica de Modelado orientado a objetos de James Rumbaugh (OMT: Object_Modeling Technique); y aproximación de Ivar Jacobson (OOSE: Object_Oriented Software Engineering) mediante la metodología de casos de uso (use case). El desarrollo de UML comenzó a finales de 1994 cuando Grady Booch y Jim Rumbaugh de Racional Software Corporation empezaron a unificar sus métodos. A finales de 1995, Ivar Jacobson y su compañía Objectory se incorporaron a Racional es su unificación aportando el método OOSE. Dentro de las tres metodologías de partida, las de Booch y Rumbaugh pueden ser descritos como entradas en objetos ya que sus aproximaciones se enfocan hacia el modelado de los objetos que componen el sistema, su relación y colaboración. Por otro lado, la metodología de Jacobson es mas centrada a usuario ya que en todo su método de deriva de los escenarios de uso. UML nos sirve para proporcionarnos una notación y semántica suficientes para poder alcanzar una gran cantidad de aspectos del modelo contemporáneo de una forma directa y económica; aspectos relacionados en la tecnología de componentes a un costo bajo, nos permitirá realizar intercambios de modelos entre gran variedad de herramientas así como interfaces y bibliotecas. De lo mencionado anteriormente concluimos que UML es un proceso guiado por casos de uso, centrado en la arquitectura, iterativo e incremental. [16] El desarrollo de sistemas se enfoca en tres modelos diferentes del sistema:

- El modelo funcional: representado en UML con diagramas de caso de uso, describe la funcionalidad del sistema desde el punto de vista del usuario.
- El modelo de objetos: representado en UML con diagramas de clase, describe la estructura de un sistema desde el punto de vista de objetos, atributos, asociaciones y operaciones.
- El modelo dinámico: representado en UML con diagramas de secuencia, diagramas de gráfica de estado y diagramas de actividad, describe el comportamiento interno del sistema.

[16] Modelo de UML
Pág. 24
Ingeniería del Software Orientada a Objetos
Bruegge, Bernd y Dutoit, Allen H.

3.2 El Significado de UML en general

UML es un lenguaje para especificar, construir, visualizar y documentar artefactos de un sistema de software orientado a objetos para el desarrollo de un software en cual cuenta con un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software; se basa en una serie de pasos que se sigue para llegar a un resultado en la construcción de un sistema de información, UML pede verse como una metodología adaptable, es decir, que tiende a ser modificada por el sistema de información que va ser desarrollado. UML que cuenta con las siglas en inglés de Lenguaje Unificado de Modelado es la herramienta que usamos para representar un sistema de información objetivo, es decir, UML es un lenguaje estándar de modelado para software permitiendo a los desarrolladores visualizar los resultados de su trabajo en esquemas y diagramas estandarizados. En donde el proceso unificado es una metodología iterativa donde consta de una serie de pasos que se repiten hasta que los miembros del equipo de desarrollo estén seguros.[17]
UML cuenta con cinco notaciones fundamentales los cuales son: diagramas de casos de uso, diagramas de clase, diagramas de secuencia, diagramas de gráfica de estado y diagramas de actividad. Sabemos que UML es guiado por casos de uso que son artefactos denominados por la información que se utiliza ó es producida mediante un estándar con el cual es posible modelar todos los componentes del proceso de desarrollo de aplicaciones. Sin embargo hay que tener en cuenta un aspecto importante del modelado, en UML los procesos de desarrollo son diferentes según los distintos dominios de trabajo buscando como objetivos principales un método que sea capaz de modelar no solo sistemas de software sino otro tipo de sistemas de software reales de la empresa, crear un lenguaje para modelado utilizable a la vez por máquinas y por personas establecer un acoplamiento explicito de los conceptos y los artefactos ejecutables. Existen artefactos que son una información utilizada ó producida mediante un proceso de desarrollo de software, los artefactos de UML se especifican en forma de diagramas esto junto con la documentación sobre el sistema constituyen los artefactos principales utilizando UML diagramas de implementación, diagramas de comportamiento ó interacción, diagramas de casos de uso, diagramas de clase teniendo significados como:

Diagramas de implementación se derivan de los diagramas de procesos que muestran los aspectos físicos del sistema como los diagramas de componentes que muestran la dependencia entre los distintos componentes de software incluyendo componentes de código fuente, binario y ejecutable. Un componente es un fragmento de software que se utiliza para mostrar dependencias en tiempo de compilación. El diagrama de despliegue muestra la configuración de los componentes hardware, los procesos, los elementos de procesamiento en tiempo de ejecución y los objetos que existen en tiempo de ejecución. Dentro de los diagramas intervienen nodos, asociaciones de comunicación, componentes dentro de nodos y objetos que se encuentran a su vez dentro de los componentes.
Un nodo es un objeto físico en tiempo de ejecución, es decir una máquina, los diagramas de interacción ó comportamiento muestran las interacciones entre objetos ocurridos en un escenario de sistemas dentro de los cuales existen los diagramas de secuencia, de colaboración, de estado, de actividad. Un diagrama de secuencia representa una forma de iniciar el período durante el que un objeto esta desarrollando una acción directamente a través de un procedimiento. En este tipo de diagramas también intervienen los mensajes que son la forma en que se comunican los objetos, el objeto de origen solicita una operación del objeto destino. Existen distintos tipos de mensajes según

[17] El paradigma orientado a objetos
Página 57
Autor: James Martin
James J. Odell

como se producen en los tiempos simples, sincrónicos y asíncronos. Los diagramas de colaboración muestran la interacción entre varios objetos organizados alrededor de los objetos. A diferencia de un diagrama de secuencias, un diagrama de colaboraciones muestra las relaciones entre los objetos no la secuencia en los tiempos en que se producen los mensajes. Dentro de los diagramas de colaboración nos encontramos con objetos, enlaces y mensajes.

Un objeto es una instancia de una clase que participa como una interacción existiendo simples y complejos; un objeto es activo posee un hilo de control y es capaz de iniciar la actividad de control mientras que un objeto es pasivo si mantiene datos pero no coincida la actividad, un enlace es una instancia de una asociación que conecta dos objetos de un diagrama de colaboración, la existencia de un enlace entre dos objetos indica que puede existir un intercambio de mensajes entre los objetos conectados.
Los diagramas de iteración indican el flujo de mensajes entre elementos del modelo, el flujo de mensajes representa el envió de un mensaje desde un objeto a otro si entre ellos existe un enlace.

Los diagramas de actividad corresponden con los diagramas de estado donde los estados son estados de acción, un paso en la ejecución de lo que será un procedimiento, las transiciones vienen provocadas por la finalización de las acciones.

Los diagramas de actividad se utilizan para mostrar el flujo de operaciones que se desencadenan en un procedimiento interno del sistema, diagramas de estado representan la secuencia de estados por los que un objeto o una iteración entre objetos pasa durante su tiempo de vida en respuesta a eventos recibidos, un estado es cuando un objeto ó una interacción satisface una condición, desarrolla alguna acción o se encuentra esperando un evento. Mientras que los diagramas de uso es una secuencia de transiciones que son desarrolladas por un sistema es respuesta a un cuento que inicia un actor sobre su propio sistema. Los diagramas de casos de uso sirven para especificar la funcionalidad y el comportamiento de un sistema lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Una relación es una conexión entre los elementos del modelo, los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema al mostrar como reacciona una persona a eventos que produce en el mismo. Este diagrama interviene que un actor es una entidad extrema al sistema que se modela y que puede interactuar con el; las relaciones que pueden existir son un actos se comunica con un caso de uso, un caso de uso extiende otro caso de uso un caso a otro caso.

Mientras que los diagramas de clases representan un conjunto de elementos del modelado que son estáticos, como las clases y los tipos, sus contenidos y las relaciones que se establecen entre ellos. Algunos de los elementos que se pueden clasificar como estáticos que son los siguientes:

- Paquete: Es el mecanismo de que dispone UML para organizar sus elementos en grupos que representa un elemento de modelo. Un sistema es un único paquete que contiene el resto del sistema, por lo tanto, un paquete debe poder anidarse, permitiéndose que un paquete contenga otro paquete.

- Clases: Representa un conjunto de objeto que tiene una estructura, un comportamiento y unas relaciones con propiedades parecidas, describiendo un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y significado. En una clase es una implementación de un tipo. Los componentes de una clase.

- Atributos: Corresponde con las propiedades de una clase o un tipo se identifica mediante un nombre, existen atributos simples y complejos.
- Operación: También conocido como método es un servicio proporcionado por la clase que puede ser solicitado por otras clases y que produce un comportamiento en ellas cuando se realiza, las clases pueden tener varios parámetros formales, son las clases denominadas plantillas. Sus atributos y operaciones vendrán definidos según sus parámetros formales, entonces reciben el nombre de clase parametrizada instancia.
- Metaclase: Es una clase cuya instancias son clases. Sirven como depósitos para mantenerse las variables de clase y proporciona operaciones para inicializar estas variables. Se utiliza para construir metamodelos utilizan para definir otros modelos.
- Tipos: Es un descriptor de objetos que tienen un estado abstracto y especificadores de operaciones pero no su implementación. Un tipo establece una especificación de comportamiento para las clases.
- Interfaz: Representa el uso de tipo para describir el comportamiento visible externamente de cualquier elemento del modelo.
- Relación entre clases: Clases se relacionan entre si de distintas formas, que marcan los tipos de relaciones existes.
- Asociación: Relación que describe un conjunto de vínculos entre clases. Pueden ser binarias según se implica dos clases ó más. Las relaciones de asociación vienen identificadas por los roles, que son los nombres que indican el comportamiento que tienen los tipos ó las clases en el caso de rol de la asociación.

UML cuenta además con nuevas características que vienen a suplir carencias de las antiguas metodologías como:

- La definición de estereotipos que tiene como significado una nueva clase de elemento de modelado que debe basarse en ciertas clases ya existentes en el metamodelo y constituye un mecanismo de extensión del modelado.
- Responsabilidades.
- Mecanismos de extensibilidad como estereotipos, valores etiquetados y restricciones.
- Tareas y procesos.
- Distribución y concurrencia.
- Patrones / Colaboración.
- Diagramas de actividad.
- Clara separación de tipo, clase e instancia.
- Refinamiento.
- Interfases y componentes.

En conclusión mencionamos que el surgimiento de UML es un lenguaje que sintetiza y estandariza la notación utilizada en los más importantes métodos de desarrollo de sistemas orientados a objetos.

Por lo tanto UML surge como un intento de resolver problemas. UML es un lenguaje estándar de modelado de sistemas y es **"YA"** en este momento un estándar de tacto en la industria de desarrollo de software.

Sin embargo UML no es un método es simplemente un lenguaje de modelado que se puede aplicar a distintas metodologías que se apeguen a este estándar.

El proceso ó pasos a seguir son especificados por las metodologías por lo cual UML es un lenguaje y procedimientos creados por los más distinguidos investigadores y desarrolladores de sistemas orientados a objetos, ha sido probado y refinado en proyectos de gran complejidad y tamaño, en su conjunto podría ser una opción muy importante para dichas organizaciones en sus intento de adoptar una metodología de desarrollo de sistema.

3.3 Captura de Requisitos

3.3.1 Introducción

En este punto el objetivo principal de los que somos analistas de sistemas con los requisitos; es decir, el desarrollador junto con el cliente y con los futuros usuarios va a buscar una determinación de lo que quieren, lo cual tal vez no lo sea; lo que creen necesitar.

Es difícil como analista la realización de un sistema; es el paso más pequeño pero con gran significado y con la importancia más grande pieza fundamental para los desarrolladores de sistemas ya que obtendremos los requisitos los cuales deben ser meticulosamente cuidadosos ya que una palabra inapropiada puede deducir una mala interpretación por lo tanto veremos el significado de captura de requisitos, la visión que tiene, el papel que juega en la vida de un software, la utilización de modelos de dominio, el desarrollo del modelo de negocios, que son los artefactos y para que se utilizan, los trabajadores, un diagrama de estado y el flujo de trabajo de los requisitos; valla todo estos requisitos, perdón por la redundancia pero son necesarios he indispensables para llevar a cabo la tarea de captura de requisitos.

Sencillas palabras pero con gran significado en el desarrollo de un software.

Cada proyecto de software es diferente, esta singularidad proviene de las diferencias en el tipo de sistema, en la organización de desarrollo en la tecnología, en conclusión existen distintos puntos de captura de requisitos, en ocasiones comenzamos haciendo un modelo de negocios ó con un modelo que esta en desarrollo, en otras ocasiones el software es un sistema empotrado que no da soporte por lo que tendríamos el modelo de objetos así mismo un modelo de dominio ó incluso el usuario ya tendría casos de uso ya determinados y detallados que quizá solo podría tener un mínimo conocimiento de lo que quiere que haga su sistema detalladamente.[3]

Es ahí como desarrolladores somos capaces de intervenir con nuestras técnicas, experiencia y conocimiento para realizar la difícil tarea de capturar lo requisitos y reducir los riesgos posteriores que al fin y al termino serenarían costos.

[3] Captura de requisitos
Pág.107
Autor: Ivar Jacobson, Grady Booch y James Rumbaugh

3.3.2 Estructuración de Captura de Requisitos

¿Qué es un caso de Uso?

Un caso de Uso describe una funcionalidad particular que se supone que el sistema debe realizar ó exhibir, modelando el diálogo de un usuario, un sistema externo u otra entidad tendrán con el sistema a ser desarrollado. Cada caso de uso describe un posible escenario de la interacción de la entidad externo con el sistema. A menudo el caso de uso se representa con un dibujo de los objetos, involucrados y acompañados de una breve descripción textual como guión de escenario que indica como se realiza la función. Para cada escenario, el caso de uso identifica todos los eventos que pueden ocurrir así como las respuestas del sistema. Los casos de uso es su totalidad constituyen una descripción completa de todas las formas posibles de utilización del sistema por todas las posibles entidades. Así, puede decirse que la colección de los casos de uso pinta un cuadro de la funcionalidad completa del sistema. Los casos de uso son particularmente útiles para comunicarse con los clientes, diseñadores y probadores de sistemas. Los clientes pueden leer los casos de uso para comprobar si el sistema se diseñara para incluir todas las funciones deseadas, los diseñadores de sistemas pueden usar los casos de uso para diagramar los objetos o ser manipulados y los elementos de datos a ser almacenados. Los diagramas de casos de uso tienen cuatro elementos: actores, casos, extensiones y usos.[4]

- **Actor:** un actor es un rol de una entidad que juega ó actúa con respecto al sistema.
- **Caso:** el caso es una pintura de algún aspecto de la funcionalidad del sistema que resulta visible para el actor cuya perspectiva es lo que refleja en el caso de uso.
- **Extensión:** una extensión justamente extiende un caso de uso para explicar una perspectiva distinta o más profunda.
- **Uso:** un uso es realmente la reutilización de un uso que ya se encuentra definido, por lo tanto, el caso de uso se puede utilizar como una ayuda para comprender al cliente y sus problemas.

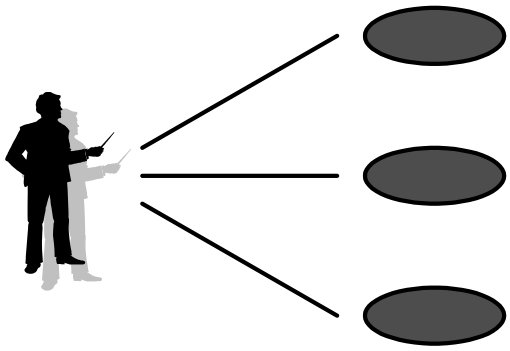


Fig. 7 La silueta representa al actor, los óvalos representan los casos de uso, las líneas representan la extensión

[4] Ingeniería de Software
Pág. 306
Ingeniería de Software Teoría y Práctica

3.3.3 Captura de Casos de Uso

Daremos una visión general de cómo se desarrolla el trabajo a través de todos los flujos de trabajo basados en la dirección de los casos de uso en donde se realiza la captura de requisitos funcionales en los casos de uso aunque también son necesarios otros requisitos, en donde veremos las necesidades de los usuarios así como la de los clientes para lo cual necesitamos el modelo de casos de uso. El modelo de casos de uso ayuda tanto al cliente, usuarios pero sobre todo a nosotros los desarrolladores a donde llegamos a un objetivo final no olvidando que cada usuario esta representado por un actor que son siluetas y los actores utilizan el sistema para interactuar con el caso de uso que esta representado por óvalos o rectángulos los cuales nos están mencionando las actividades que realizan los actores para llegar a fin común.

3.3.4 Diagrama de caso de uso

Los casos de uso se utilizan durante la obtención de requerimientos y el análisis para representar la funcionalidad del sistema. Los casos de uso se enfocan en el comportamiento del sistema desde el punto de vista externo. El diagrama de casos de uso describe una función proporcionada por el sistema que produce un resultado visible para un actor.[5]

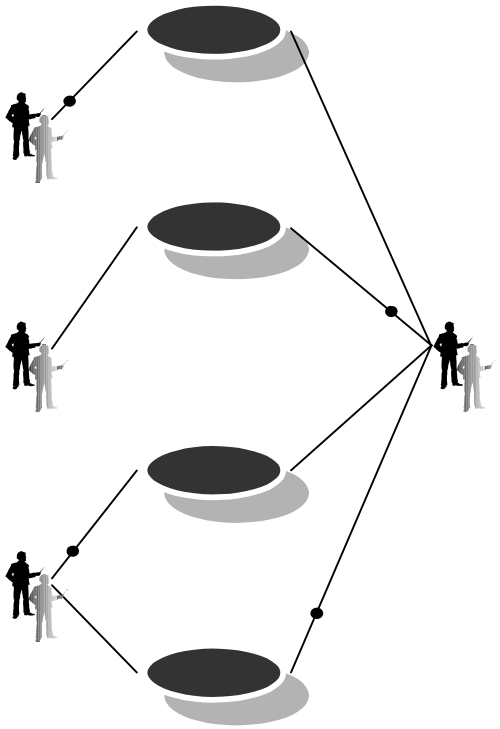


Fig. 8 Diagrama de Casos de Uso

[5] Una panorámica del UML
Pág. 25
Ingeniería del Software Orientada a Objetos
Bruegge, Bernd y Dutoit, Allen H.

3.3.5 ¿Qué es un Artefacto?

Los artefactos fundamentalmente se utilizan en la captura de requisitos, son el modelado de casos de uso y actores. Es pieza fundamental de información tangible que es creada, modificada y usada por los trabajadores al realizar actividades representando un área de responsabilidad y candidato a ser tendido en cuenta para el control.

Un artefacto puede ser un modelo, un elemento ó un documento; podrían ser importantes en la descripción formal de algunos casos de uso como cuando utilizamos diagramas de actividad ó diagramas de estado.

Facilita la identificación de casos de uso correctos y con una descripción detallada, los casos de uso permite a los desarrolladores de software y los clientes lleguen a un acuerdo sobre los requisitos es decir, las condiciones y posibilidades que debe cumplir el sistema.

Pero también un artefacto cuenta con la descripción de la arquitectura su misma palabra lo dice contiene la vista del modelo de casos de uso que al ser tan grandes forman paquetes representados por fólder los cuales describen la funcionalidad importante en donde se cumple una crítica ó contenga algún requisito especial solicitado por el usuario pero también influye en la parte del ciclo de vida del software el cual utiliza un glosario para la ayuda de las definiciones de los términos importantes para nosotros los analistas y desarrolladores en donde utilizamos los conceptos para determinar los riegos posibles y existentes dentro del desarrollo del software donde llegamos a la propuesta del prototipo de interfaces, hablado sencillamente del diseño de las pantallas posibles a diseñar.

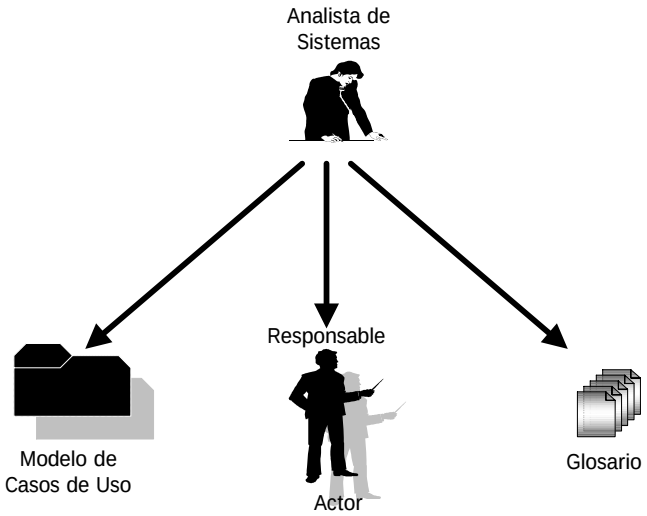


Fig. 9 Responsabilidades del analista de sistemas durante la captura de requisitos en casos de uso

3.3.6 Trabajadores

El significado de trabajador es el puesto puede ser asignado a una persona ó equipo, que requieren responsabilidades y habilidades para realizar determinadas actividades a desarrollar determinados artefactos.[6]
Pero hay que tener cuidado un trabajador no es un individúo es la representación abstracta de una persona que con ciertas capacidades que se requieren un caso de uso en el proceso unificado para el desarrollo del software.

El trabajador representa el conocimiento las habilidades que alguien necesita para interactuar con un sistema y se haga cargo desde principio a fin, sabiendo la esquematización detallada del mismo sistema estos pueden ser los analistas, el especificador de casos de uso y el diseñador de las interfaces. Los analistas de sistemas es el responsable de los requisitos que están modelados en los casos de uso, lo que incluye todos los requisitos funcionales y no funcionales que son casos de uso específicos, pero también tiene la tarea de delimitar al sistema encontrando casos de uso y actores para la realización del modelo de casos de uso.[7]

En el trabajo de captura de registro no puede estar dirigido por un solo individuo ya que esta respaldados por demás trabajadores asumen responsabilidades de las descripciones detalladas de uno o más casos de uso, estos se les denomina especificador de casos de uso.
Cada especificador de casos de uso necesita trabajar estrechamente con los usuarios.

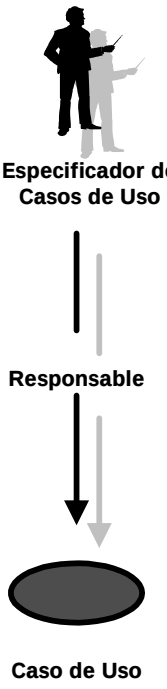


Fig. 10 Responsabilidades de un especificador de casos de uso durante la captura de requisitos

[6] Glosario General de UML
Página 433
Proceso Unificado de Desarrollo
[7] Captura de requisitos como casos de uso
Página 135
Autor: Ivar Jacobson, Grady Booch y James Rumbaugh

Los diseñadores de las interfaces dan la forma visual de las interfaces de los usuarios. Esto implica el desarrollo de los prototipos de interfaces de usuario para algunos casos de uso.

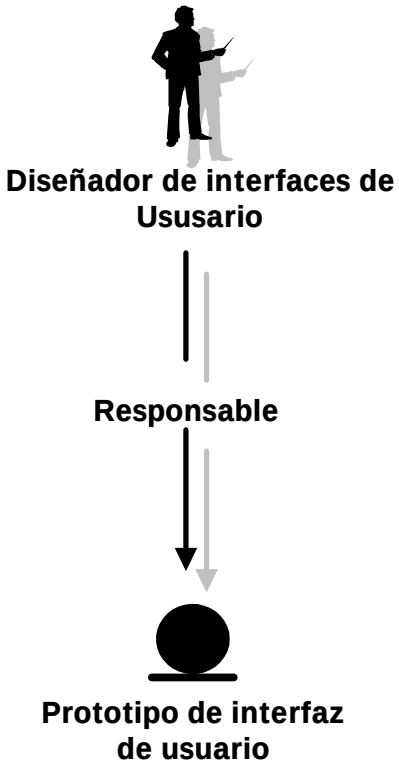


Fig. 11 Diseñador de Interfases

3.3.7 Modelo de Dominio

Un modelo de dominio captura los tipos más importantes de objetos en el contexto de un sistema.

Los objetos de dominio representan los casos de uso que existen a los eventos que suceden en el entorno en que trabaja el sistema.[8]

Muchos de los objetos de dominio ó clases pueden obtenerse de una especificación de requisitos mediante la entrevista con los expertos de dominio. Las clases del dominio aparecen en tres formas típicas.

- ♣ Objetos de negocio que representan casos que se manipulan en el negocio.
- ♣ Objeto del mundo real y conceptos de los sistemas debe hacer un seguimiento.
- ♣ Sucesos que ocurrirán ó han ocurrido.

El modelo del dominio se describe mediante diagramas de clases.

Estos diagramas muestran a los clientes, usuarios, revisores y a desarrolladores las clases de dominio y como se relacionan unas con otras mediante asociaciones.[9]

El modelo de Dominio se realiza habitualmente en reuniones organizadas por los analistas de lo que genera la documentación de resultados, el objetivo principal del modelo de dominio es comprender y describir las clases más importantes dentro de los sistemas.

En pocas palabras el modelo de dominio utiliza clases de dominio y un glosario de términos el cual ayuda a:

- ♣ Describir los casos de uso y al diseñar la interfaz de usuario.
- ♣ Para sugerir clases internas al sistema en desarrollo durante el análisis.

3.3.8 Desarrollo del Modelo de Negocios

El Modelo de Negocios es una técnica para comprender los procesos de negocio de la organización a la que se le va a implantar el sistema.

El objetivo principal del Modelo de Negocios es el identificar los casos de uso del software y las entidades relevantes que el software debe soportar de forma que podríamos modelar solo lo necesario para comprender el modelo de casos de uso y el modelo de objetos.
Pero la pregunta es ¿Qué es un Modelo de Negocios?; una pregunta sencilla quizá en donde solo la respuesta es simple la descripción de negocios de una empresa en términos de la descripción de las

[8] Unified Modeling Languaje User.
Pág. 112
Autor: Grady, Booch, James Rumbaugh and Ivar Jacobson
[9] The Object Advantage- Business process Reengineering with Object Technology
Página 100
Autor: Ivar Jacobson

actividades que realizan para determinar sus actividades, es decir, casos de uso interactúan con el sistema de otra forma los actores conocen como esta esquematizado y como funciona el sistema.

A grandes rasgos podemos mencionar que el Modelo de Negocios esta formado por dos puntos principales los cuales describen su funcionalidad:

- ♣ La confección de los casos de uso identifica actores y los casos de uso que utilizan los mismos actores.

- ♣ Desarrolla un modelo de objetos integrado por trabajadores, entidades y casos de uso.

Concluimos así de sencillo el Modelo de Dominio es muy parecido al Modelo de Negocios pero como en todo nadie ni nada es perfecto ni mucho menos iguales; aun así como los gemelos son iguales físicamente pero su forma de ser es diferente pues bien aquí aplica lo mismo existen diferencias, el Modelo de Dominio tiene atributos pero casi nunca utiliza operaciones y sin en cambio el Modelo de Negocios utiliza a actores pero esencialmente los casos de uso que utilizan los mismos actores.

3.3.9 Diagramas de Estado

Un Diagrama de Estados muestra la secuencia de estados por los que pasa un caso de uso ó un objeto a lo largo de su vida, indicando qué eventos hacen que se pase de un estado a otro y cuáles son las respuestas y acciones que genera.
En cuanto a la representación, un diagrama de estados es un grafo cuyos nodos son estados y cuyos arcos dirigidos son transiciones etiquetadas con los nombres de los eventos. Un estado se representa como una caja redondeada con el nombre del estado en su interior. Una transición se representa como una flecha desde el estado origen al estado destino.

La caja de un estado puede tener 1 ó 2 compartimentos. En el primer compartimiento aparece el nombre del estado. El segundo compartimiento es opcional, y en él pueden aparecer acciones de entrada, de salida y acciones internas.

Una acción de entrada aparece en la forma entrada/acción _ asociada donde acción _ asociada es el nombre de la acción que se realiza al entrar en ese estado. Cada vez que se entra al estado por medio de una transición la acción de entrada se ejecuta.

Una acción de salida aparece en la forma salida/acción _ asociada. Cada vez que se sale del estado por una transición de salida la acción de salida se ejecuta. Una acción interna es una acción que se ejecuta cuando se recibe un determinado evento en ese estado, pero que no causa una transición a otro estado. Se indica en la forma nombre_de_evento/acción _ asociada.

3.3.10 Captura de Requisitos en el ciclo de vida del Software

El modelo de casos de uso se desarrolla a lo largo de varios incrementos del desarrollo donde las iteraciones añadirán nuevos casos de uso y detallaran las descripciones de los casos de uso de uso existente:

- ♣ Durante la fase de inicio, los analistas identifican la mayoría de de los casos de uso para delimitar el sistema y el alcance del proyecto y desarrollar los más importante.

- Durante la fase de elaboración, los analistas capturan la mayoría de los requisitos restantes para que los desarrolladores puedan estimar el tamaño del esfuerzo de desarrollo de lo que se requiera. El objetivo es capturar el 80% de los requisitos necesarios para el usuario así como también descrito las actividades que realizara el sistema dentro de ese 80% por medio de casos de uso detallados teniendo siempre presente la comunicación con el usuario y ser congruente a lo que se esta solicitando que haga el sistema.

- Los requisitos restantes se capturan durante la construcción y la elaboración del proyecto.

- Casi no existen captura de requisitos en la fase de transición a menos que los requisitos capturados cambien a solicitud del usuario, verificando si son necesarios en el sistema y si se pueden cambiar sin afectar a la demás construcción del sistema.

3.4 Análisis

3.4.1 Introducción

Durante el desarrollo del esta fase el análisis es importante ya que analizaremos a más detalle la captura de requisitos, realizaremos la creación del diamante sabemos que para la creación del diamante es un trozo de carbón guardado pero bajo un proceso de calor se realiza la creación de la piedra bruta que se da a la tarea de ser pulida hasta llegar a ser una preciosa piedra pues es lo mismo que sucede con la captura de requisitos se necesita de un pulido para llegar a un objetivo como la comunicación del usuario y la especificación de lo que desea el usuario.

El análisis nos ayuda a estructurar los requisitos de manera que facilite su comprensión, preparación, modificación y mantenimiento basada en clases, análisis y paquetes para dar la forma del sistema de su totalidad.

Analizaremos lo que es un análisis, el papel que juega con el ciclo de vida de software, artefactos de análisis, la clase de interfaz, clases de entidad, clases de control; así como sus diagramas de colaboración, de interacción, la utilización de paquetes.

3.4.2 Significado de Análisis

El lenguaje que utilizamos en el análisis se basa en un modelo de objetos que llamamos un modelo de análisis.

El modelo de análisis nos ayuda a refinar los requisitos nos permite razonar sobre los sistemas incluyendo los recursos. El modelo de análisis también nos ayuda a estructurar los requisitos, nos proporcionan una estructura centrada en el mantenimiento en aspectos tales como la flexibilidad ante los cambios y la reutilización.

Esta estructura no solo es útil para el mantenimiento de los requisitos si no también es utilizada para la entrada de las actividades de diseño e implementación.

Pues bien los requisitos del modelo de análisis son importantes por varios motivos:

- Un modelo de análisis ofrece una especificación más precisa de los requisitos que la que tenemos como resultado de la captura de requisitos.

- Un modelo de análisis se describe utilizando el lenguaje de los desarrolladores y puede por tanto introducir un mayor formalismo y ser utilizado para razonar sobre los funcionamientos internos del sistema.

- Un modelo de análisis estructura los requisitos de un modo que facilita su comprensión, su preparación, su modificación y en general su mantenimiento.

- Un modelo de análisis puede considerarse como una primera aproximación al modelo de diseño.

Las iteraciones iniciales de la elaboración se encuentran en el análisis, eso contribuye a obtener una arquitectura estable y sólida y facilita una comprensión en profundidad de los requisitos.

El propósito y objetivo del análisis debe alcanzarse de algún modo en este proyecto. Pero de manera exacta de ver y de emplear el análisis puede diferir de un proyecto a otro y nosotros analizaremos tres puntos.

1. El proyecto utiliza el modelo de análisis para descubrir los resultados del análisis manteniendo la consistencia de este modelo a lo largo de todo un ciclo de vida del software.

2. El proyecto utiliza el modelo de análisis para describir los resultados del analista pero considera a este modelo como una herramienta transitoria e intermedia quizás de más interés durante la fase de elaboración. Más adelante cuando el diseño y la implementación están en marcha durante la fase de construcción se deja de actualizar el análisis.

3. El proyecto no utiliza en absoluto el modelo de análisis para describir los resultados del análisis. En cambio el proyecto analiza los requisitos como parte integrada en la captura de requisitos ó en el diseño. En el primero de los casos, hará falta un mayor formalismo en el modelo de casos de uso.

   El resultado del flujo de trabajo del análisis es un modelo de objetos conceptuales que analiza los requisitos mediante el refinamiento y estructuración. El modelo de análisis incluye los siguientes elementos:

   - Paquetes de análisis y paquetes de servicio, dependencias y contenidos. Los paquetes de análisis pueden aislar los cambios en un proceso de negocio, el comportamiento de un actor ó en un conjunto de casos de uso estrechamente relacionados.

   - Clases de análisis, responsabilidades, atributos, relaciones y requisitos especiales. Cada una de las clases de control, entidad e interfaz aíslan los cambios al comportamiento y la información que representan.

   - Realizaciones de casos de uso- analista que describen como se refinan los casos de uso en términos de colaboraciones dentro del modelo de análisis y de sus requisitos especiales.

   - La vista de la arquitectura del modelo de análisis, incluyendo sus elementos significativos para la arquitectura.

3.4.3 Estructuración de Análisis

Artefactos de Análisis: La estructura impuesta por el modelo de análisis se define mediante una jerarquía como se muestra en la figura.

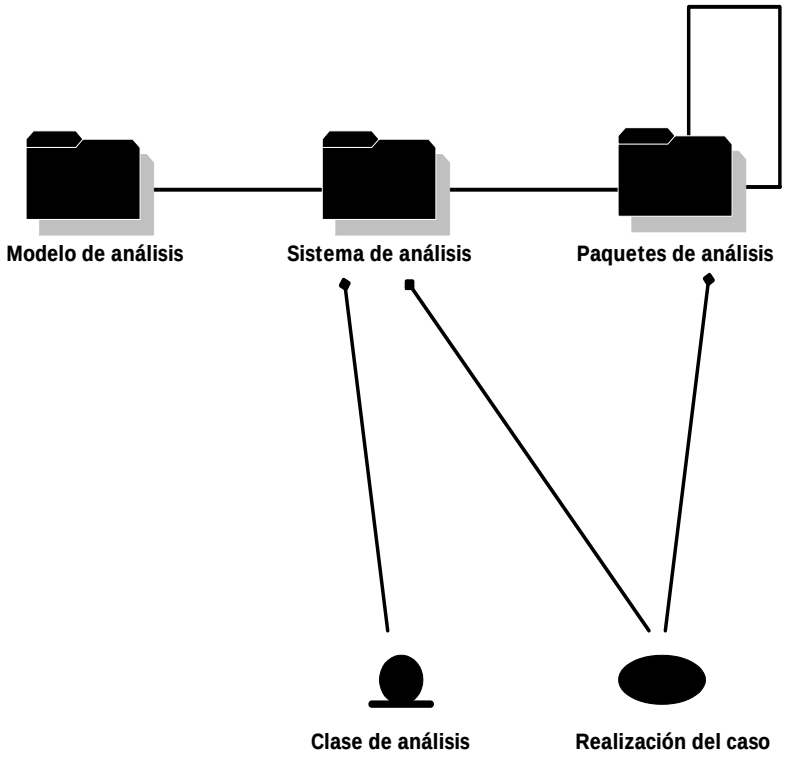


Fig. 12 Jerarquía del modelo de análisis[10]

El modelo de análisis se representa mediante un sistema de análisis es por lo tanto una forma de organizar el modelo de análisis en partes más manejables que representan abstracciones de subsistemas y posibles capas completas del diseño del sistema.

3.4.4 Artefactos de análisis

Una clase de análisis representa una abstracción de una ó varias clases y/ó subsistemas del diseño del sistema. Las cuales presentan las siguientes características:

- Una clase de análisis se centra en el tratamiento de los requisitos funcionales y pospone los no funcionales contando con requisitos especiales hasta llegar a las actividades de diseño e implementación subsiguiente.

- Esto hace que una clase de análisis sea más evidente en el contexto del dominio del problema, más conceptual.

[10] El proceso Unificado de Desarrollo de Software
Pág. 172
Autor: Ivar Jacobson, Grady Booch y James Rumbaugh

- ♣ Una clase de análisis define atributos aunque los atributos también son de un nivel bastante alto.

- ♣ Una clase de análisis participa en relaciones, aunque esas relaciones son más conceptuales que sus contrapartidas de diseño e implementación.

- ♣ Las clases de análisis siempre encajan en uno de tres estereotipos básicos: de interfaz, de control ó de entidad, cada estereotipo implica una semántica específica.

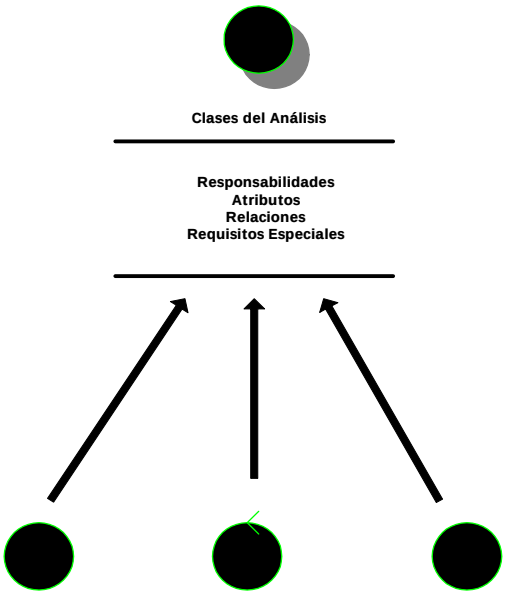


Fig. 13 Esquematización de Artefactos[11]

3.4.5 Clases de Entidad

El análisis se describe los casos de usos en términos de las clases del sistema de información. El proceso unificado hay tres tipos de clases: entidad, borde y control.

Una clase de entidad es un modelo de la información perdurable. La extracción de las clases de entidad consiste en tres pasos que conlleva a cabo de manera iterativo y por incrementos:

- ♣ Modelado Funcional: Representa los escenarios de todos los casos de uso.

[11] El proceso Unificado de Desarrollo de Software
Pág. 174
Autor: Ivar Jacobson, Grady Booch y James Rumbaugh

- ♣ Modelado de Clases: Determina las clases entidad y sus atributos, define las interrelaciones entre las clases de entidad presenta la información en forma de un diagrama de clases.

- ♣ Modelado Dinámico: Determina las operaciones realizadas por ó con cada clase ó subclase en donde presenta esta información en la forma de un diagrama de estado.

Como estamos leyendo todo lleva a cada paso por ser sencillamente un proceso todo va ligado finamente para cumplir con el objetivo.

El segundo paso es la extracción de las clases de entidad en el modelado de clase.

El objetivo es extraer las clases de entidad, determinar sus interrelaciones y hallar sus atributos. Por lo que este paso es usar el método de extracción de sustantivos en dos etapas.

1. Los informes se van a generar con el fin de mejorar la efectividad del proceso de tomo de decisiones donde se ve las clasificaciones representativas.

2. Los informes contienen información, clasificada y representativa.

La efectividad, proceso e información son sustantivos abstractos y por lo tanto es poco probable que sean clases de entidad ya que recordemos que deben ser tangibles.[12]

Las clases de entidad se utilizan para modelar información que posee una vida larga y que es a menudo persistente.

Las clases de entidad modelan la información y el comportamiento asociado de algún fenómeno ó concepto, como una persona, un objeto del mundo real ó un suceso del mundo real.

En la mayoría de los casos, las clases de entidad se derivan directamente de una clase de entidad del negocio correspondiente, tomada del modelo de objetos del negocio.

Pero la diferencia fundamental entre las clases de entidad y las clases de entidad de negocio es que las primeras representan objetos manejados por el sistema en consideración.

De lo que en consecuencia las clases de entidad reflejan la información de un modo que benefician a los desarrolladores al diseñar e implementar el sistema, incluyendo un soporte.

Las clases de entidad suelen mostrar una estructura de datos lógica y contribuyen a comprender que la información depende del sistema.[13]

[12] UML and the Unifed Process
Pág. 117
[13] Análisis
Pág. 175

3.4.6 Clases de Interfaz

Las clases de interfaz se utilizan para modelar la interacción entre el sistema y sus actores. Esta interacción a menudo implica recibir información y peticiones de usuarios y los sistemas externos.

Las clases de interfaz modelan las partes del sistema que dependen de sus actores, lo cual implica que clarifican y reúnen los requisitos en los límites del sistema.
Por tanto, un cambio en un interfaz de usuario en comunicaciones queda normalmente aislado en una clase de interfaz.

Las clases de interfaz representan a menudo abstracciones de ventanas, formularios, paneles, interfaces de comunicaciones, interfaces de impresoras, sensores, terminales se mantiene en un nivel bastante alto y conceptual.[14]

3.4.7 Clases de Control

Las clases de control comúnmente son fáciles de extraer como clases de borde. Las clases de control representan coordinación, secuencia, transacción, control de objetos y se utilizan con frecuencia para encapsular el control de un caso en concreto. Las clases de control también se utilizan para representar derivaciones y cálculos complejos, como la lógica del negocio, que no puedan asociarse con ninguna información concreta, de larga duración, almacenada por el sistema. Los aspectos dinámicos del sistema se modelan con clase de control, debido a que ellas manejan y coordinan las acciones y los flujos de control principales, y delegan trabajo a otros objetos.[15]

3.4.8 Diagramas de Colaboración y de Interacción

Las secuencias de acciones en el caso de uso comienzan cuando un actor invoca el caso de uso mediante de algún tipo de mensaje al sistema. En el análisis preferimos mostrar esto con diagramas de colaboración ya que nuestro objetivo fundamental es identificar requisitos y responsabilidades sobre los objetos y no identificar secuencias de interacción detalladas y ordenadas cronológicamente.
En los diagramas de colaboración mostramos las interacciones entre objetos cuando enlacen entre ellos y añadiendo mensajes a estos enlaces. En relación con la creación y finalización de los objetos del análisis dentro de una realización de casos de uso, objetos diferentes tienen diferentes ciclos de vida.

- ♣ Un objeto de interfaz no tiene por qué ser particular de una realización de caso de uso, sin embargo, los objetos de interfaz a menudo se crea y se finaliza dentro de una sola realización de uso.

- ♣ Las clases de control suelen encapsular el control asociado en donde el caso de uso inicia y termina

[14] Análisis
Pág. 177
Autor: Liliana Favre
[15] El proceso Unificado del desarrollo de Software
Página 175
Autor: Ivar Jacobson, Grandy Booch y James Rumbaugh

3.5 Diseño

3.5.1 Introducción

El enfoque particular del análisis comprende y procesan la realidad a través de los conceptos que adquieran implantar por diversos medios, entre máquinas, computadoras y personas. Sin embargo el objetivo del diseño se restringe al software de aplicación. En el diseño modelamos el sistema y encontramos su forma para que soporte todos los requisitos incluyendo los requisitos no funcionales y otras restricciones.
La entrada esencial en el diseño es el resultado del análisis, esto es el modelo de análisis proporciona una comprensión detallada de requisitos. Pero lo más importante impone una estructura del sistema detallada de los requisitos. Los propósitos del diseño son:

- Adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia tecnológicas de interfaz de usuario, tecnologías de gestión de transacción.

- Crear una entrada apropiada y un punto de partida para actividades de implementación subsiguiente capturada los requisitos ó subsistemas individuales, interfaces y clases.

- Ser capaces de descomponer los trabajos de implementación en partes más manejables que pueden ser llevadas a cabo por diferentes equipos de desarrollo.

3.5.2 Significado de Diseño

Dentro del diseño se utilizan los artefactos que interactúan y se incrementan hasta que tengan un formato que los programadores pueden utilizar dentro de este aspecto existe el incremento de la identificación de operaciones y sus asignaciones de clases apropiadas.

En el paradigma tradicional la fase de diseño consta de dos pasos: primero se lleva a cabo el diseño arquitectónico. En segundo paso de diseño es el paradigma tradicional del diseño detallado.

Durante el se debe especificar el formato exacto de cada atributo de los diagramas buscando el refinamiento del análisis hasta que el material este en una forma que los programadores pueden implementar. Además de la utilización de requisitos incluyendo la selección del lenguaje de programación.[16]
El diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción. Esto contribuye a una arquitectura estable, sólida crea un plano del modelo de implementación. Más tarde, durante la fase de construcción, cuando la arquitectura es estable y los requisitos están bien entendidos, el centro de atención se desplaza a la implementación.

[16] Fases de proceso unificado
Pág. 217

El modelo de diseño esta muy cercano al de implementación, lo que es natural para guardar y mantener el modelo de diseño a través del ciclo de vida completo del software.
Esto es especialmente cierto en la ingeniería de ida y vuelta, donde el modelo de diseño se puede utilizar para visualizar la implementación y para soportar las técnicas de programación gráfica.

El principal resultado del diseño es el modelo de diseño que se esfuerza la estructura del sistema impuesta por el modelo de análisis y que sirve como esquema para la implementación. El modelo de diseño incluye subsistemas del diseño, subsistemas de servicio y dependencias, interfaces y contenidos. Los subsistemas del diseño de las dos capas superiores obtenidas de los paquetes del análisis, clases del diseño, incluyendo las clases activas y sus operaciones atributos y requisitos de implementación. Algunas clases activas se obtienen a partir de clases de análisis.

La vista arquitectónica del modelo de diseño que describen como se diseñan los casos de uso en términos de colaboraciones dentro del modelo de diseño. El diseñado también obtiene como resultado un modelo de despliegue que describe todas las configuraciones de red sobre las cuales deberán implementarse el sistema. Con nodos, sus características y sus conexiones, una correspondencia inicial de clases activas sobre los nodos, la vista arquitectónica del modelo de despliegue que influye elementos relevantes para la arquitectura.

3.5.3 Estructuración del Diseño

Artefactos

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en como los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tiene impacto en el sistema a considerar.
Además el modelo de diseño sirve la abstracción de la implementación del sistema y es de ese modo utilizada como una entrada fundamental de las actividades de implementación.

El modelo de diseño se representa por un sistema de diseño que denota el subsistema de nivel más alto del modelo.

La utilización de los subsistemas de diseño se representa abstracciones del subsistema y componentes de la implementación del sistema.

En el modelo de diseño, los casos de uso son realizados por las clases de diseño y sus objetos representados por colaboraciones en el modelo de diseño y denota la realización caso de uso_ diseño.

Los artefactos del modelo de diseño se presentan en detalle en las siguientes secciones. La clase de diseño es una abstracción como ya se menciono sin costuras de una clase o construcción similar en la implementación del sistema.

El lenguaje utilizado para especificar una clase del diseño es lo mismo que el lenguaje de programación, por consecuente las operaciones, parámetros, atributos, tipos y demás son especificados utilizando la sintaxis del lenguaje de programación elegido.

3.5.4 Diagramas de clase

Usamos diagramas de clase para describir la estructura del sistema. Las abstracciones que especifican la estructura y el comportamiento común de un conjunto de objetos. Los objetos son instancias de las clases que se crean, modifican, destruyen los valores de sus atributos y sus relaciones con otros objetos. Los diagramas describen el sistema desde el punto de vista de objetos, clases, atributos, operaciones y sus asociaciones.
En UML las clases y objetos se muestran mediante cuadros que incluyen tres comportamientos. El comportamiento central muestra sus atributos y el comportamiento inferior muestra sus operaciones. Los diagramas de clases se usan para describir la estructura del sistema. Durante el análisis los ingenieros de software construyen diagramas de clase para formalizar el conocimiento del dominio de aplicación.[17]

3.5.5 Diagramas de interacción

La secuencia de acciones en un caso de uso comienza cuando el actor invoca el caso de uso mediante el envió de algún tipo de mensaje al sistema. Después el objeto de diseño llama algún otro objeto para su interacción entre ellos representando los diagramas en forma consecutiva detalladamente. En los diagramas de secuencia mostramos las interacciones entre objetos mediante transferencia de mensajes entre objetos ó subsistemas.

3.9.6 Trabajadores

En el diseño el arquitecto es el responsable de la integridad de los modelos de diseño y de despliegue garantizando que los modelos son correctos, consistentes y legibles en su totalidad, al igual que el modelo de análisis puede incluirse para sistemas grandes y complejos un trabajador asume la responsabilidad del subsistema del más alto nivel del diseño del sistema.

El arquitecto también es el responsable del diseño del sistema y el despliegue, es decir de la existencia de las partes significativas para la arquitectura. El objetivo de la arquitectura es esbozar los modelos de diseño y despliegue y su arquitectura mediante la identificación de nodos y sus configuraciones de red, subsistemas, interfaces, clases de diseño significativas para la arquitectura como clases activas.

3.5.7 Flujo de Trabajo

La utilización inicia la creación de los modelos de diseño y de despliegue endosan los nodos del modelo de despliegue, los subsistemas principales y sus interfaces, las clases del diseño importantes como las activas y los mecanismos genéricos de diseño del modelo de diseño. A lo largo del flujo de trabajo del diseño, nuevos candidatos para ser subsistemas, interfaces, clases y mecanismos de diseño genéricos.

[17] Modelado con UML
Pág. 50
Ingeniería del Software Orientada a Objetos
Bruegge, Bernd y Dutoit, Allen H.

3.6 Implementación

3.6.1 Introducción

En la implementación empezamos con el resultado del diseño e implementación de los sistemas en términos de componentes, es decir, ficheros de código de fuente, scripts, ficheros de código binario, ejecutables y similares.

Los propósitos de la implementación es planificar las integraciones de sistema en cada iteración comprendiendo con un enfoque incremental lo que da lugar a un sistema que se implementa en la sucesión de pasos pequeños y manejables.

La distribución del sistema asignado componentes ejecutables a nodos en el diagrama de despliegue. Implementar las clases y subsistemas encontrados durante el diseño, probar los componentes individualmente integrados para poder compilarlo y enlazarlos en uno a más ejecutables.

Los Diagramas de Implementación se usan para modelar la configuración de los elementos de procesado en tiempo de ejecución (Rut-time processing elements) y de los componentes, procesos y objetos de software que viven en ellos.

En el diagrama 'deployment' (despliegue), empieza el modelando nodos físicos y las asociaciones de comunicación que existen entre ellos. Para cada nodo, puedes indicar qué instancias de componentes que se ejecutan en el nodo. También puedes modelar los objetos que contiene el componente.

Los Diagramas de Implementación se usan para modelar solo componentes que existen como entidades en tiempo de ejecución, en compilación ó en tiempo de enlazado.

Puede también modelar componentes que migran de nodo a nodo u objetos que migran de componente a componente usando una relación.

3.6.2 Significado de Implementación

La implementación es el centro durante las iteraciones de construcción, aunque también se lleva acabo trabajo de implementación durante la fase de elaboración para la creación de ejecutables de la arquitectura y durante la fase de transición para tratar defectos tardíos como los encontrados con distribuciones beta del sistema, la implementación de un sistema actualmente se basa en términos de componentes y subsistemas en forma natural mantener el modelo de implementación a lo largo del ciclo de vida del software.

El resultado principal de la implementación es el modelo de implementación de lo cual tenemos a los subsistemas de implementación y sus dependencias, interfaces y contenidos.

Componentes incluyendo ficheros ejecutables y las dependencias entre ellos, la vista de la arquitectura del modelo de implementación de lo cual produce como resultado un refinamiento de la vista de la arquitectura del modelo de despliegue donde los componentes ejecutables son asignados por nodos.

3.6.3 Estructuración de la Implementación

Artefactos

El modelo de implementación describe como los elementos del modelo de diseño como las clases se implementan en términos de componentes como ficheros de código fuente ejecutables.

El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de construcción y modularización disponibles en el entorno de implementación así como el lenguaje de programación utilizados y como dependen los componentes unos de otros.

El modelo de implementación define una jerarquía. En los subsistemas de implementación proporcionan una forma de organizar los artefactos del modelo de implementación en trozos más manejables.

Un subsistema puede estar formado por componentes, interfases y otros subsistemas.
Además el subsistema puede implementar y así proporcionar las interfaces que representan la funcionalidad que exportan en forma de operaciones.

Es importante entender que un subsistema de implementación se manifiesta a través de un mecanismo de empaquetamiento concreto en un entorno de implementación determinado tales como:

- Un paquete en Java.
- Un directorio de ficheros.
- Un subsistema en un entorno de desarrollo.
- Paquete de herramientas

3.6.4 Interfases

Las interfases pueden ser utilizadas en el modelo de implementación para especificar las operaciones implementadas por componentes y subsistemas de implementación ya que pueden tener dependencias de uso.
Un componente que implementa una interfaz ha de implementar correctamente todas las operaciones definidas por la interfaz, un subsistema de implementación que proporciona una interfaz tiene también que contener los componentes que proporcionen la interfaz u otros subsistemas.

3.6.5 Vista arquitectónica de la Implementación

La descripción de la arquitectura contiene una visión de la arquitectura del modelo de implementación el cual representa sus artefactos significativos arquitectónicamente.

Los artefactos son sumamente considerados en el modelo de implementación significativos, la descomposición del modelo de implementación en subsistema, sus interfaces y las dependencias de ellos en general es muy significativa para la arquitectura ya que siguen la traza de los subsistemas de diseño uno a uno.

3.6.6 Trabajadores

Durante la fase de implementación el arquitecto es responsable de la integridad del modelo de implementación y asegura que el modelo como un todo es correcto, completo y legible como en el análisis y el diseño para sistemas grandes y complejos en el podemos traducir un trabajador adicional puede asumir la responsabilidad de subsistemas de nivel alto verificando que el modelo sea correcto cuando implementa la funcionalidad descrita en el modelo de diseño y en los requisitos adicionales pero también es responsable de la arquitectura del modelo de implementación es decir, de la existencia de sus partes significativas arquitectónicamente como se presento en la vista del modelo de despliegue.

3.6.7 Flujo de trabajo

El objetivo principal de la implementación es implementar el sistema este proceso es iniciado por el arquitecto embozado por los componentes clave en el modelo de implementación.

El integrador de sistemas observa la construcción describiendo la funcionalidad que daría la implementación y que partes del modelo de implementación estarían afectadas, los componentes resultantes son probados y pasados al desarrollador.

El resultado principal de la implementación es el modelo de implementación de lo cual tenemos a los subsistemas de implementación y sus dependencias, interfaces y contenidos.

Componentes incluyendo ficheros ejecutables y las dependencias entre ellos, la vista de la arquitectura del modelo de implementación de lo cual produce como resultado un refinamiento de la vista de la arquitectura del modelo de despliegue donde los componentes ejecutables son asignados por nodos.

3.7 Prueba

3.7.1 Introducción

En el flujo de trabajo de la prueba verificamos como el resultado de la implementación es probando en cada construcción incluyendo tanto construcciones internas como intermedias así como las versiones finales del sistema a ser entregadas a terceros.

Los objetivos de la prueba son: planificar las pruebas necesarias en cada interacción incluyendo las pruebas de integración y las pruebas de sistema las pruebas de integración son necesarias para cada construcción dentro de la iteración mientras que las pruebas de sistema son necesarias solo al final de la iteración.

3.7.2 Significado de Prueba

Diseñar e implementar las pruebas creando los casos de prueba que especifican que prueba, creando los procedimientos especifican como realizar las pruebas y creando si es posible que los componentes de prueba sean ejecutables para automatizar las pruebas.

Realizar las diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Las construcciones en las que se detectan defectos son probadas de nuevo y posiblemente devueltas a otro flujo de trabajo, como diseño ó implementación de forma que los defectos importantes puedan ser arreglados.

Durante la fase de inicio puede hacerse la planificación inicial de las pruebas cuando se define el ámbito del sistema, las pruebas se llevan acabo sobre todo cuando una construcción es el resultado d la implementación es sometido a pruebas de integración y de sistema.

Esto quiere decir que la realización de pruebas se centra en las fases de elaboración cuando se prueba la línea base ejecutable de la arquitectura y de la construcción cuando el grueso del sistema esta implementado. Teniendo encuenta la eliminación de casos de uso de pruebas obsoletas así como los correspondientes procedimientos de prueba y componentes de prueba, el refinamiento de algunos casos de prueba de regresión o la creación de nuevos casos de uso para una nueva construcción.

3.7.3 Estructuración de Pruebas

Artefacto

El modelo de pruebas describe principalmente como se prueban los componentes ejecutables en el modelo de implementación con pruebas de integración y de sistema. El modelo de pruebas puede describir también como han de ser aprobados aspectos específicos del sistema. Un caso de prueba especifica una forma de probar el sistema, incluyendo la entrada, resultado con la que han de aprobar las condiciones bajo las que se han de probar. En un caso de prueba de este tipo incluye la verificación del resultado de las interacciones entre los actores y el sistema en donde se satisfacen las precondiciones especificadas por los casos de uso.

También se pueden especificar otros casos de prueba para probar el sistema como un todo.

- Las pruebas de instalación verifican que el sistema puede ser instalado la plataforma del cliente y en que sistema funcionara correctamente.

- Las pruebas de configuración verifican que el sistema funciona correctamente en diferentes configuraciones.

- Las pruebas negativas intentan provocar que el sistema falle para poder así revelar sus debilidades.

3.7.4 Trabajadores

Un diseñador de pruebas es responsable de la integridad del modelo de pruebas asegurando que el modelo cumple con su propósito.

Los diseñadores de pruebas también planean las pruebas lo que significa que deciden los objetivos de prueba apropiados y la planificación de la prueba a demás los diseñadores de pruebas selecciona y describen los casos de prueba y de procedimiento que se necesitan.

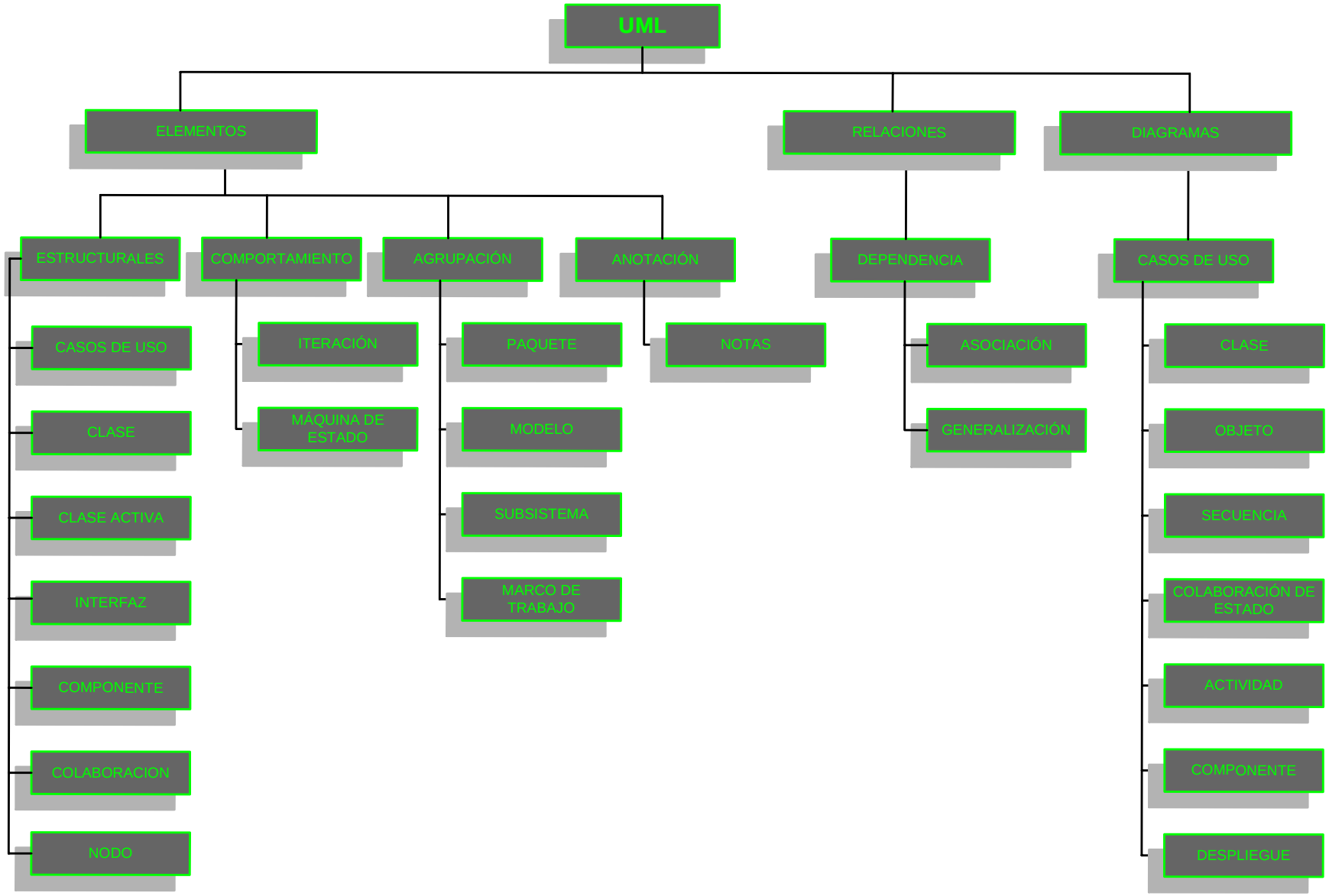
Existen ingenieros de componentes, ingenieros de pruebas de integración, ingeniero de pruebas de sistemas.

3.7.5 Flujo de trabajo

El principal objetivo de la prueba es realizar y evaluar las pruebas como se describen el modelo de pruebas.

El ingeniero de pruebas inicia esta tarea planificando el esfuerzo de prueba en cada iteración y describe entonces los casos de prueba necesarios y los procedimientos de prueba correspondientes para llevar acabo las pruebas.

Todo esto se hace para cada construcción entregada como resultado de trabajo de la implementación. A todo esto podemos concluir el conocimiento estratégico de la Metodología de UML, la cual nos da una visión amplia del por que es caracterizada por ser guiada por casos de usos.


UML
ELEMENTOS
RELACIONES
DIAGRAMAS
ESTRUCTURALES
COMPORTAMIENTO
AGRUPACIÓN
ANOTACIÓN
DEPENDENCIA
CASOS DE USO
CASOS DE USO
CLASE
CLASE ACTIVA
INTERFAZ
COMPONENTE
COLABORACION
NODO
ITERACIÓN
MÁQUINA DE ESTADO
PAQUETE
MODELO
SUBSISTEMA
MARCO DE TRABAJO
NOTAS
ASOCIACIÓN
GENERALIZACIÓN
CLASE
OBJETO
SECUENCIA
COLABORACIÓN DE ESTADO
ACTIVIDAD
COMPONENTE
DESPLIEGUE

Capítulo 4
Caso Práctico

Análisis y Diseño del Sistema de Control de Gestión Versión 2.0

Introducción

La Subsecretaria de Atención Ciudadana y Normatividad tiene como necesidad en el año 2004; la creación del Sistema Control de Gestión el cual surge como respuesta para cubrir dicha necesidad permitiendo de manera sencilla y ágil el control de los turnos de la información de los documentos enviados a la Secretaría Técnica, la cual se encargara de resolver ó turnar entre las diferentes Áreas de La Subsecretaría de Atención Ciudadana y Normatividad, permitiendo identificar el tiempo que ha transcurrido en cada una de ellas, así como la consulta del trámite y la resolución que le ha dado a la misma.

Su objetivo del sistema es el proporcionar la información más rápida a la solicitud del servicio necesitado por cada área.

Donde se tendrá el conocimiento para dar la solución expedita y eficiente a la solicitud.

El Sistema Control de Gestión a tenido modificaciones y se busca la actualización del sistema existente, este sistema originalmente estaba bajo la programación de PL/SQL de Oracle; se busca como se menciona un nuevo sistema para poder manejar un nueva versión del sistema de Control de Gestión se planteo y diseño la objetividad del nuevo diseño de interfases Web así como el lenguaje de programación de Java y se selecciono MySQL que es un sistema de administración de base de datos teniendo como características:

- Velocidad: Los desarrolladores sostienen que Mysql es posiblemente la base de datos más rápida.

- Facilidad de uso: MySQL es un sistema de base de datos de alto rendimiento pero relativamente simple y es mucho menos complejo de configurar y administrar sistemas grandes.

- Coste: MySQL es gratuito para la mayoría de usos internos.

- Capacidad de gestión de lenguajes de consulta: MySQL, lenguaje elegido para todos los sistemas de base de datos modernos, al igual que puede ser empleado con aplicaciones que admitan la conectividad de base de datos (ODBC) y un protocolo de comunicación de bases de datos desarrollado por Microsoft.

- Capacidad: Se pueden conectar muchos clientes simultáneamente al servidor. Los clientes pueden utilizar varias bases de datos simultáneamente se puede acceder en forma interactiva a MySQL empleando diferentes interfaces que le permiten introducir consultas y visualizar los resultados: cliente de línea de comando, navegadores Web, clientes de sistema de X Windows. Además de que cuenta con una amplia variedad de interfaces de programación para lenguajes como C, Perl,

Java, PHP y Python; por lo cual tiene la posibilidad de elegir entre usar un software cliente pre-empaquetado ó escribir sus propias aplicaciones a medida.

- Conectividad y seguridad: MySQL esta completamente preparado para el trabajo en red y las bases de datos pueden ser accedidas desde cualquier lugar en Internet por lo que se puede compartir sus datos con cualquier y en cualquier parte; pero MySQL disponen de control de acceso de forma que aquellos que no deberían ver sus datos no los vean.

- Portabilidad: MySQL se ejecuta en muchas variantes de Unix así como otros sistemas no-Unix como Windows y OS/2. MySQL se ejecuta en hardware que va desde PC hasta servidores de alta capacidad.

- Distribución abierta: MySQL es fácil de obtener; simplemente emplee su navegador Web. Si no entiende como funciona alguna cosa ó tiene curiosidad por algún algoritmo, puede conseguir el código fuente e investigarlo. Si no le gusta como funciona las cosas, puede cambiarlas.

Con la combinación de Java que se encuentra presente y actualmente en mercado de la tecnología. Ya que Java es el resultado de un ciclo de desarrollo teniendo como lenguaje de programación orientado a objetos que se emplea muchos elementos comunes de otros lenguajes, Java se puede adquirir completamente gratuita por Sun e Internet, su finalidad es la producción de aplicaciones completas de alto nivel con el que se puede escribir programas convencionales además de ventajas sobre otros lenguajes independientes.

4.2 Captura de Requisitos

Para capturar los requisitos de manera eficaz, los analistas necesitan un conjunto de técnicas y artefactos que les ayude a obtener una visión suficientemente buena del sistema para avanzar en los flujos de trabajo subsiguientes. Llamamos a este conjunto de artefactos colectivamente Captura de Requisitos los cuales son:

- Modelo de Dominio
- Modelo de Negocio
- Modelo de Casos de Uso
- Requisitos Adicionales

Estos artefactos son necesarios para establecer el contexto del sistema cuyo propósito fundamental es el tener un flujo de trabajo.

La Captura de Requisitos es la guía de desarrollo hacia el sistema correcto. Esto se consigue mediante una descripción de los requisitos del sistema, es decir, las condiciones o capacidades que el sistema debe de cumplir suficientemente buena como para que pueda llegarse a un acuerdo entre el cliente incluyendo a los usuarios y los desarrolladores sobre qué debe y qué no debe hacer el sistema.

4.3 Modelo de Dominio

Un modelo de dominio captura los tipos más importantes de objetos en el contexto de un sistema.

Los objetos de dominio representan los casos de uso que existen a los eventos que suceden en el entorno en que trabaja el sistema.[33]

Muchos de los objetos de dominio ó clases pueden obtenerse de una especificación de requisitos ó mediante la entrevista con los expertos de dominio. Las clases del dominio aparecen en tres formas típicos.

- Objetos de negocio que representan casos que se manipulan en el negocio.
- Objeto del mundo real y conceptos de los sistemas debe hacer un seguimiento.
- Sucesos que ocurrirán ó han ocurrido.

[33] Unified Modeling Languaje User.
Pág. 112
Autor: Grady, Booch, James Rumbaugh and Ivar Jacobson

4.4 Modelo de Negocio

El Modelo de Negocios es una técnica para comprender los procesos de negocio de la organización a la que se le va a implantar el sistema.

El objetivo principal del Modelo de Negocios es el identificar los casos de uso del software y las entidades relevantes que el software debe soportar de forma que podríamos modelar solo lo necesario para comprender el modelo de casos de uso y el modelo de objetos.

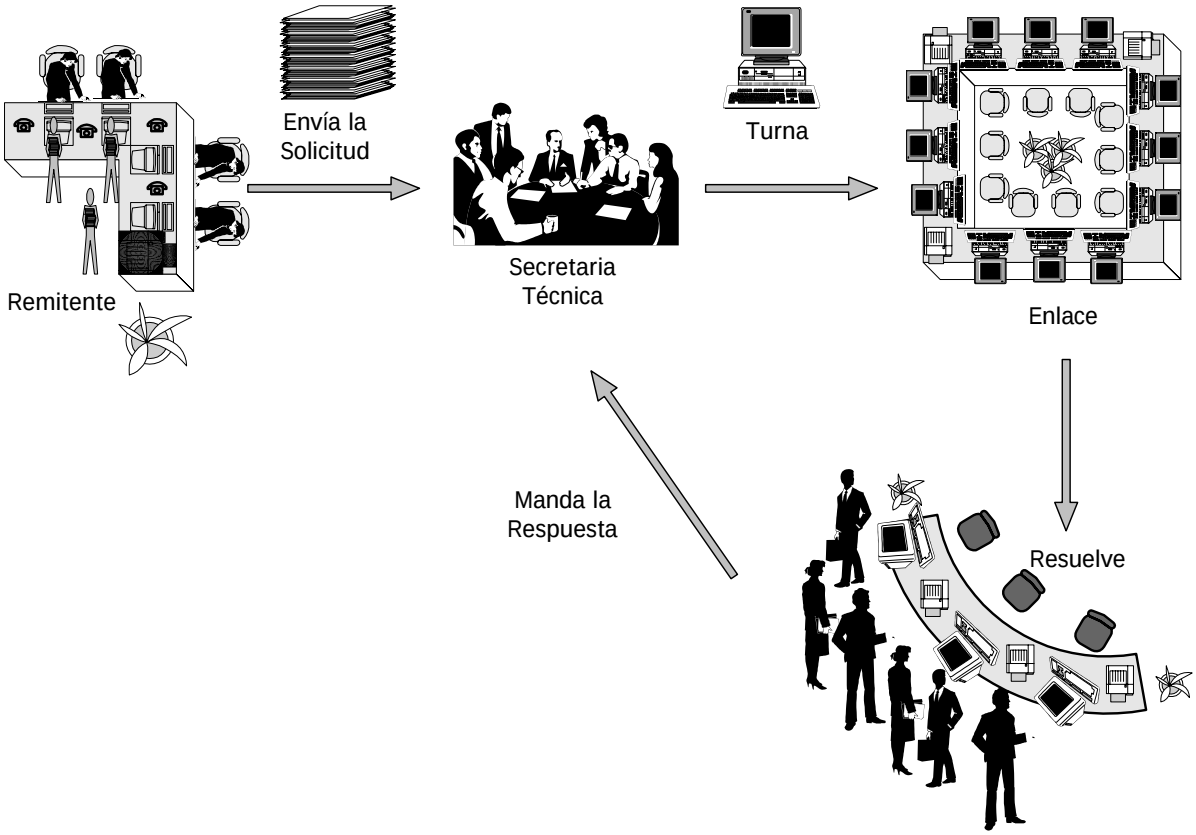


Fig.14 Representación del Modelo de Negocio del Sistema de Control de Gestión V 2.0

4.5 Modelo de Casos de Uso

Los casos de uso proporcionan un medio intuitivo y sistemático para capturar los requisitos funcionales con un énfasis especial en el valor añadido para cada usuario individual ó para cada sistema externo mediante la utilización de los casos de uso, los analistas se ven obligados a pensar en términos de quienes son los usuarios y cuales son sus objetivos para que pueden cumplir su papel clave en la dirección del resto del trabajo de desarrollo ha sido motivo importante para su aceptación en la mayoría de los métodos de la ingeniería moderna del software.

Recordando que un caso de uso es un artefacto de actores representados por siluetas y las acciones realizadas por los actores que son representadas por óvalos de los cuales conforman el sistema.

Los casos de uso son “fragmentos” de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores.

De manera más precisa, un caso de uso específica una secuencia de acciones que el sistema debe llevar acabo con sus actores, incluyendo alternativas dentro de la secuencia.

Los casos de Uso que utilizaremos en el Sistema de Control de Gestión V 2.0; surge como respuesta para cubrir las necesidades del remitente hacia la Secretaría Técnica la cuál se cuenta con un Administrador de Sistemas que tiene como tarea el dar de alta, baja ó modificar las claves del personal encargado de analizar, resolver ó en su caso determinado turnar las solicitudes para la resolución dentro de las diferentes áreas de la Subsecretaría de Atención Ciudadana y Normatividad de la Secretaría de la Función Pública.

Administrador de Sistemas: Tiene como tarea el realizar las altas, bajas ó modificaciones de las claves del personal que se encargara de la resolución de las solicitudes.

Remitente: Tiene como tarea el enviar las solicitudes a la Secretaría Técnica.

Secretaría Técnica: Tiene como tarea el recibir todas las solicitudes que lleguen por cualquier medio para poder analizar, resolver ó en determinado caso turnar al área competente para la resolución de la misma.

Enlaces: Cuenta como principal tarea el recibir todas las solicitudes asignadas por la Secretaría Técnica ya sea que se encuentren pendientes de resolución inmediata ó en el determinado criterio ser devueltas a la Secretaría Técnica para que sean nuevamente turnadas al área determinada para su resolución de la solicitud y poder mandar la respuesta a la Secretaría Técnica.

4.5.1 Actores del Modelo de Casos de Uso

- Administrador de Sistemas
- Remitente
- Secretaría Técnica
- Enlaces

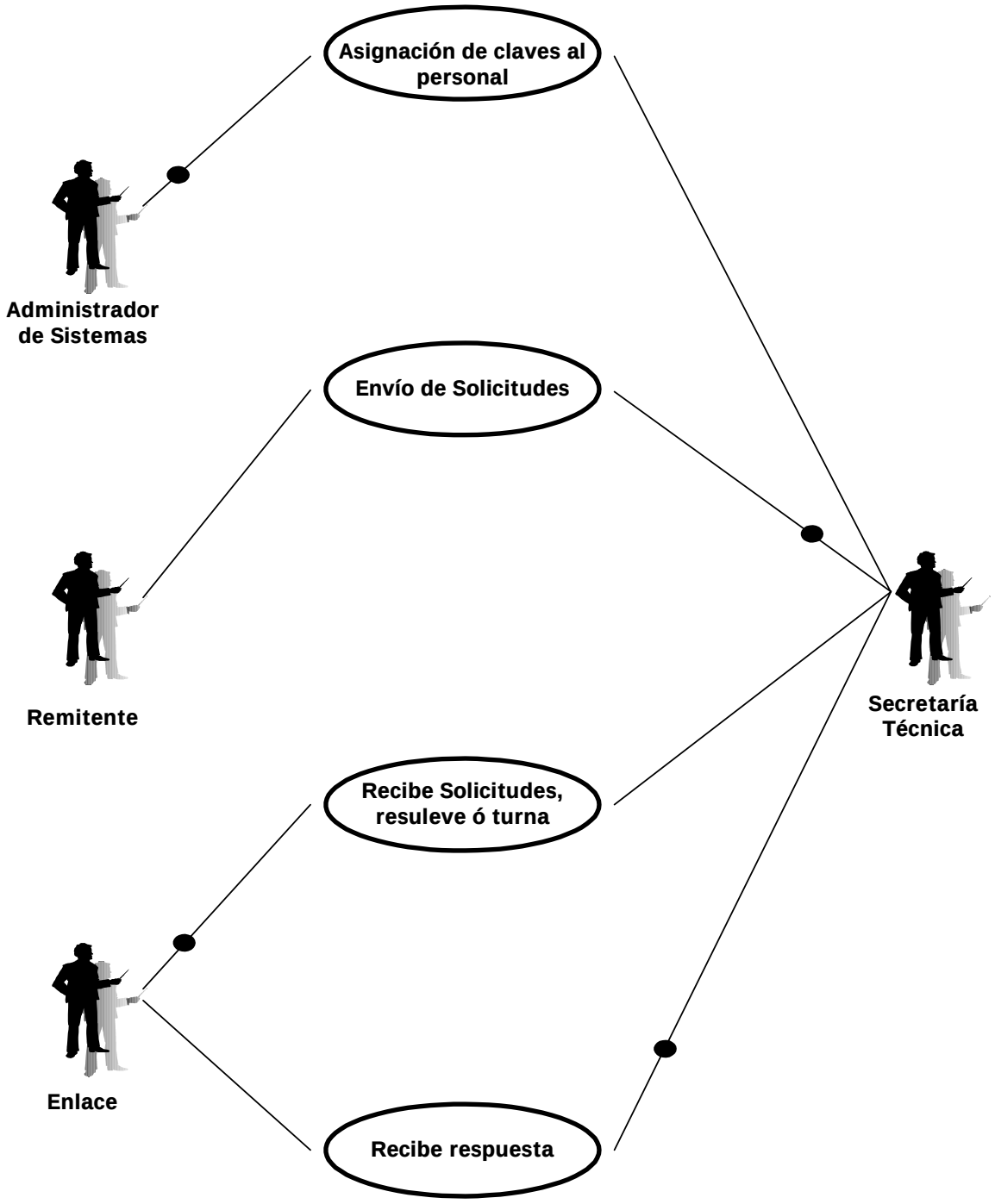


Fig. 15 Diagrama del Modelo de Casos de Uso

4.5.2 Descripción de los Casos de Uso

El remitente por medio electrónico ó por cualquier otro medio de la Secretaría realiza el envío de las solicitudes a la Secretaría Técnica.

Caso de Uso: **Asignación de las claves al personal**

El administrador de sistemas puede determinar el alta, baja ó modificaciones de las claves de acceso para el personal que va interactuar con el sistema para la resolución de las solicitudes enviadas a la Secretaría Técnica ó por cualquier otro enlace.

1. Si es un nuevo usuario que va interactuar con el sistema se realiza la captura de datos.
2. Al colocar el Usuario y su correo electrónico existente el sistema automáticamente le asignara la contraseña, la cuál será enviada a su cuenta de correo electrónico del usuario.
3. Podrá realizar la modificación de su contraseña si el usuario lo requiere al igual que recuperar la contraseña en caso de no recordarla.

Caminos Alternativos: Si al momento de realizar la captura de datos y al dar aceptar manda el mensaje de datos inválidos, nuevamente el usuario capturara sus datos. En caso de querer modificar la contraseña ó de no recordar la contraseña el sistema le asignara una nueva la cual será enviada a la cuenta de correo del Usuario.

Caso de Uso: **Envío de la Solicitud**

El remitente envía a la Secretaría Técnica las solicitudes destinadas a la Subsecretaría de Atención Ciudadana y Normatividad la cuál tiene como tarea el archivar, resolver, turnar ó regresar la solicitud a la Secretaría Técnica siempre que no pertenezca al área envía al inicio.

1. Secretaría Técnica recibe las solicitudes del remitente.
2. Secretaría Técnica puede resolver la solicitud si pertenece a su área.
3. En caso de no pertenecer al enlace enviado se regresara la solicitud a la Secretaría Técnica para ser turnada nuevamente al enlace correspondiente.

Caso de Uso: **Recibe la Solicitud, Resuelve o Turna**

Enlace recibe las solicitudes de las cuales resuelve en el instante ó en su defecto de no tener la solución es turnado al enlace competente para resolver la solicitud.

1. Se recibe las solicitudes que envía el remitente a la Secretaría Técnica ó cualquier otro enlace.
2. Se revisa y analiza.
3. Se resuelve la solicitud en caso de no ser competente al área se turna al enlace correspondiente.

Caminos Alternativos: Enlace resuelve las diferentes solicitudes enviadas, Enlace manda la respuesta a la Secretaría Técnica.

Caso de Uso: **Recibe Respuesta**

La Secretaría Técnica recibe la respuesta enviada por el Enlace.

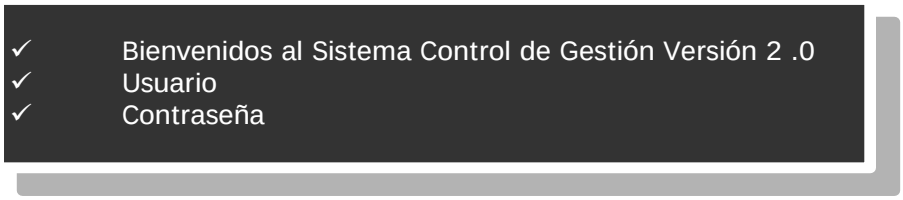
1. Se recibe la respuesta del Enlace.
2. Se envía la respuesta a Secretaría Técnica.
3. Se incorpora la respuesta al Sistema de Control de Gestión (SCG V2.0).

4.6 Interfaz Lógica

Cuando los actores interactúan con el sistema utilizaran y manipularan elementos de interfaces de usuario que representan atributos (de casos de Uso).
Los actores pueden experimentar estos elementos de las interfases de usuario como iconos, listas de elementos u objetos en un mapa de 2D y pueden manipularlos por sección arrastrando ó hablando con ellos. El diseñador de interfases de usuario identifica y especifica estos elementos actor por actor, recorriendo todos los casos de uso a los que el actor puede acceder e indicándolo los elementos apropiados de la interfaz de usuario para cada caso de uso, desempeñando un papel diferente en cada uno. Así los elementos de la interfaz de usuario puede diseñarse para jugar varios roles.

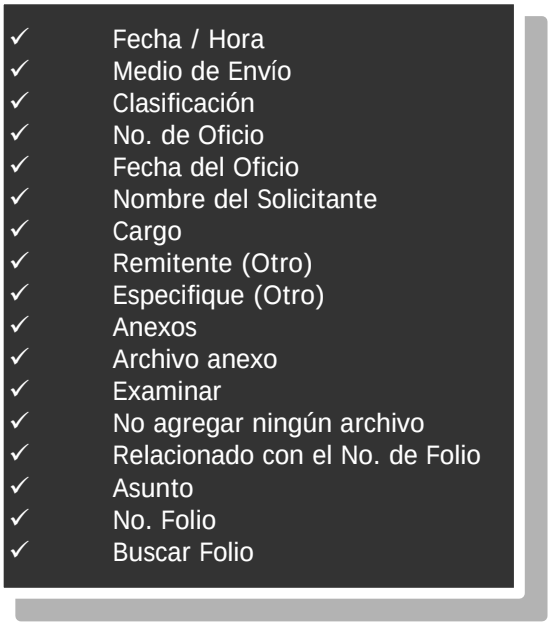
4.6.1 Interfaz Lógica de Usuarios

4.6.2 Interfaz Lógica del Sistema de Control de Gestión Versión 2.0



4.6.3 Interfaz Lógica: Opción del Sistema

4.6.4 Interfaz Lógica: Capturar la Solicitud



4.6.5 Interfaz Lógica: Guardar la Solicitud

- ✓ Grabar la Solicitud
- ✓ Fecha / Hora
- ✓ La Solicitud se ha guardado satistactoriamente.Con el No. de Folio.

4.6.6 Interfaz Lógica: Recepción de Solicitudes

- ✓ Fecha / Hora
- ✓ No. de Folio
- ✓ Fecha de Captura
- ✓ Origen
- ✓ Solicitud de Información
- ✓ Días Transcurridos
- ✓ Plazo Máximo
- ✓ Estatus
- ✓ Aviso
- ✓ Aceptar Recepción de Documentos
- ✓ Nombre de Usuario
- ✓ Contraseña
- ✓ Aceptar Firma

4.6.7 Interfaz Lógica: Recepción de Solicitudes _ detallé

- ✓ Fecha y Hora
- ✓ No. de Folio
- ✓ Fecha de Registro
- ✓ Días de Plazo
- ✓ Nombre del Solicitante
- ✓ Cargo
- ✓ Remitente
- ✓ Anexos
- ✓ Archivo anexo
- ✓ No. de Oficio
- ✓ Fecha de Oficio
- ✓ Relacionado con
- ✓ Solicitud
- ✓ Medio de Envío
- ✓ Clasificación
- ✓ Instrucción
- ✓ Resolución
- ✓ Carácter de trámite
- ✓ Normal
- ✓ Urgente
- ✓ Extra urgente
- ✓ Para su
- ✓ Atención
- ✓ Conocimiento
- ✓ Turnar
- ✓ Instrucciones
- ✓ Historial de Instrucciones

4.6.8 Interfaz Lógica: Resolución

- ✓ Fecha y Hora
- ✓ No. de Folio
- ✓ Fecha de Registro
- ✓ Días de Plazo
- ✓ Menú de Inicio
- ✓ Fecha de Oficio
- ✓ No. de Oficio
- ✓ Resolución
- ✓ Archivo anexo de resolución
- ✓ Examinar
- ✓ No agregar ningún archivo

4.6.9 Interfaz Lógica: Búsqueda de Solicitud

- ✓ Fecha / Hora
- ✓ No. de Oficio
- ✓ Buscar
- ✓ Fecha de Oficio
- ✓ Buscar
- ✓ Remitente
- ✓ Buscar
- ✓ Nombre del Solicitante
- ✓ Buscar
- ✓ Asunto
- ✓ Buscar
- ✓ Relacionado con el Folio
- ✓ Buscar
- ✓ Folio del _ al
- ✓ Período del dd/mm/aaaa al dd/mm/aaaa
- ✓ Aceptar
- ✓ Limpiar

4.6.10 Resultado de la búsqueda (por asunto)

- ✓ Fecha y Hora
- ✓ No. de Folio
- ✓ Oficio
- ✓ Remitente
- ✓ Asunto
- ✓ Nombre del Cargo
- ✓ Anexos
- ✓ No. de registros

4.6.11 Seguimiento de la Solicitud

- ✓ Fecha y Hora
- ✓ No. de Folio
- ✓ Fecha de Captura
- ✓ Fecha de Recepción en el Área en Turno
- ✓ Días Transcurridos
- ✓ Área origen / Área en turno
- ✓ Situación

4.6.12 Seguimiento de la Solicitud _ detallé

- ✓ Fecha y Hora
- ✓ Fecha en turno
- ✓ Área origen
- ✓ Área en turno
- ✓ Días en el área
- ✓ Fecha en que contestó el área
- ✓ Situación

4.6.13 Reportes

- ✓ Fecha y Hora
- ✓ Reporte Estadístico
- ✓ Reporte Estadístico por Área y Origen
- ✓ Pendientes por Área
- ✓ Concluídos por Área
- ✓ Período del dd/mm/aaaa al dd/mm/aaaa

4.6.14 Reportes_ detalle

- ✓ Fecha y Hora
- ✓ Reporte Estadístico
- ✓ Coordinación Administrativa
- ✓ Recibidos
- ✓ Conocimiento
- ✓ Atención
- ✓ Resueltos
- ✓ Pendiente
- ✓ Vencidos
- ✓ Subdirección Administrativa de Recursos Financieros
- ✓ Recibidos
- ✓ Conocimiento
- ✓ Atención
- ✓ Resueltos
- ✓ Pendiente
- ✓ Vencidos
- ✓ Subdirección Administrativa de Recursos Humanos
- ✓ Recibidos
- ✓ Conocimientos
- ✓ Atención
- ✓ Resueltos
- ✓ Pendiente
- ✓ Vencidos
- ✓ Total General
- ✓ Recibidos
- ✓ Conocimientos
- ✓ Atención
- ✓ Resueltos
- ✓ Pendiente
- ✓ Vencidos

4.7 Interfaz Física

Los diseñadores de interfaz de usuario preparan unos esquemas aproximados de la configuración de elementos de las interfases de usuario que constituyen interfases de usuario útiles para los actores.

Después bosquejan elementos adicionales necesarios para combinar varios elementos adicionales para incluir contenedores de los elementos de interfases de usuario completos.

Estos elementos adicionales pueden incluir contenedores de los elementos de la interfaz de usuario. Estos esquemas pueden incluir contenedores de los elementos de la interfaz de usuario.

Estos esquemas pueden preparase después o a veces concurrentemente del desarrollo de las notas adhesivas identificadas durante el diseño de la interfaz de usuario lógica.

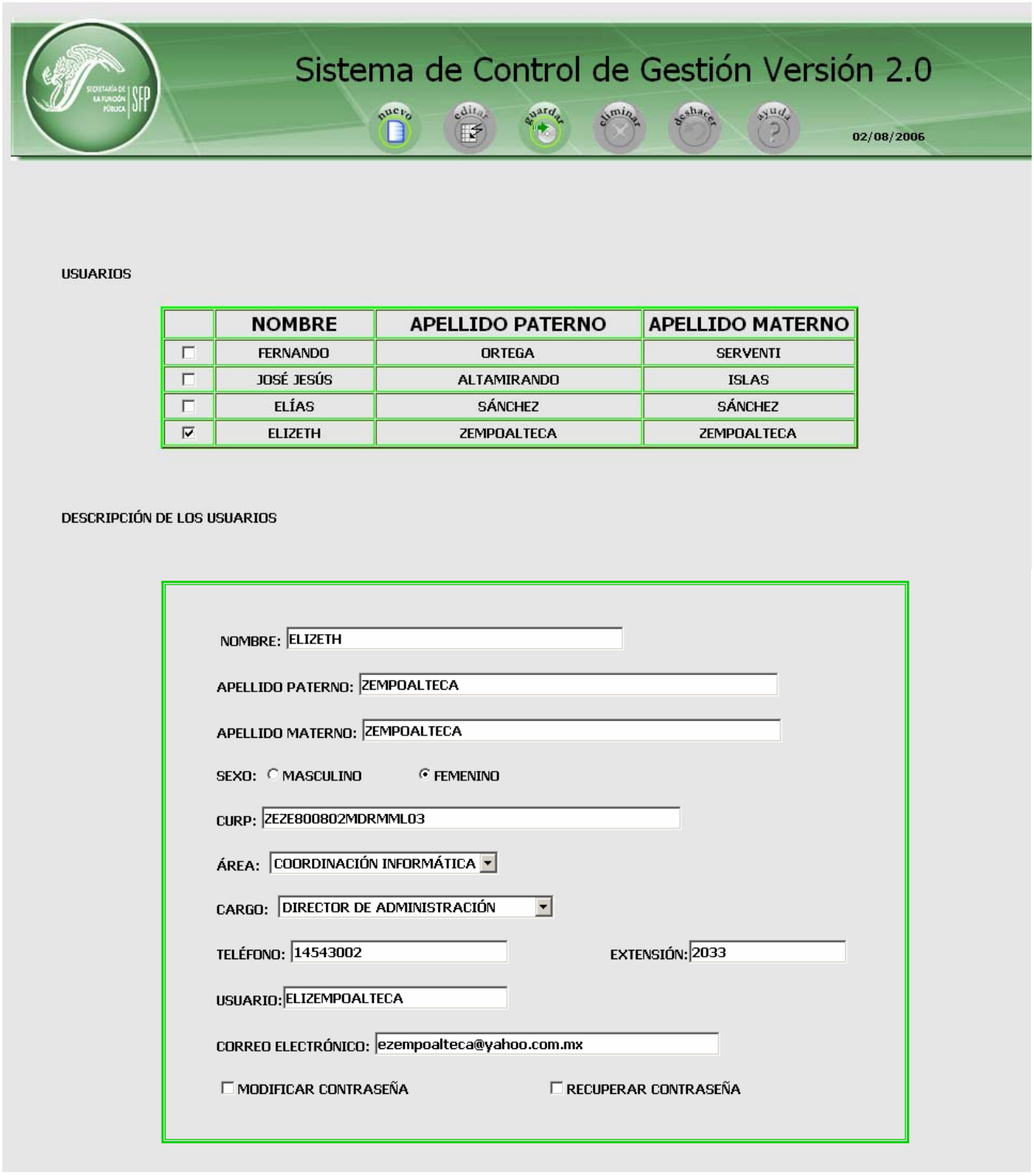
Puede haber varios propósitos, quizá para cada actor para verificar que cada actor pueda ejecutar el caso de uso que necesita.

El esfuerzo del prototipo y esquemas en los primeros momentos pueden prevenir muchos errores que serán mas caros de corregir después. Los prototipos pueden revisar después de que los casos de uso y permitir que se corrijan después de que los casos pases a un diseñado, los revisores deben verificar que cada interfaz de usuario:

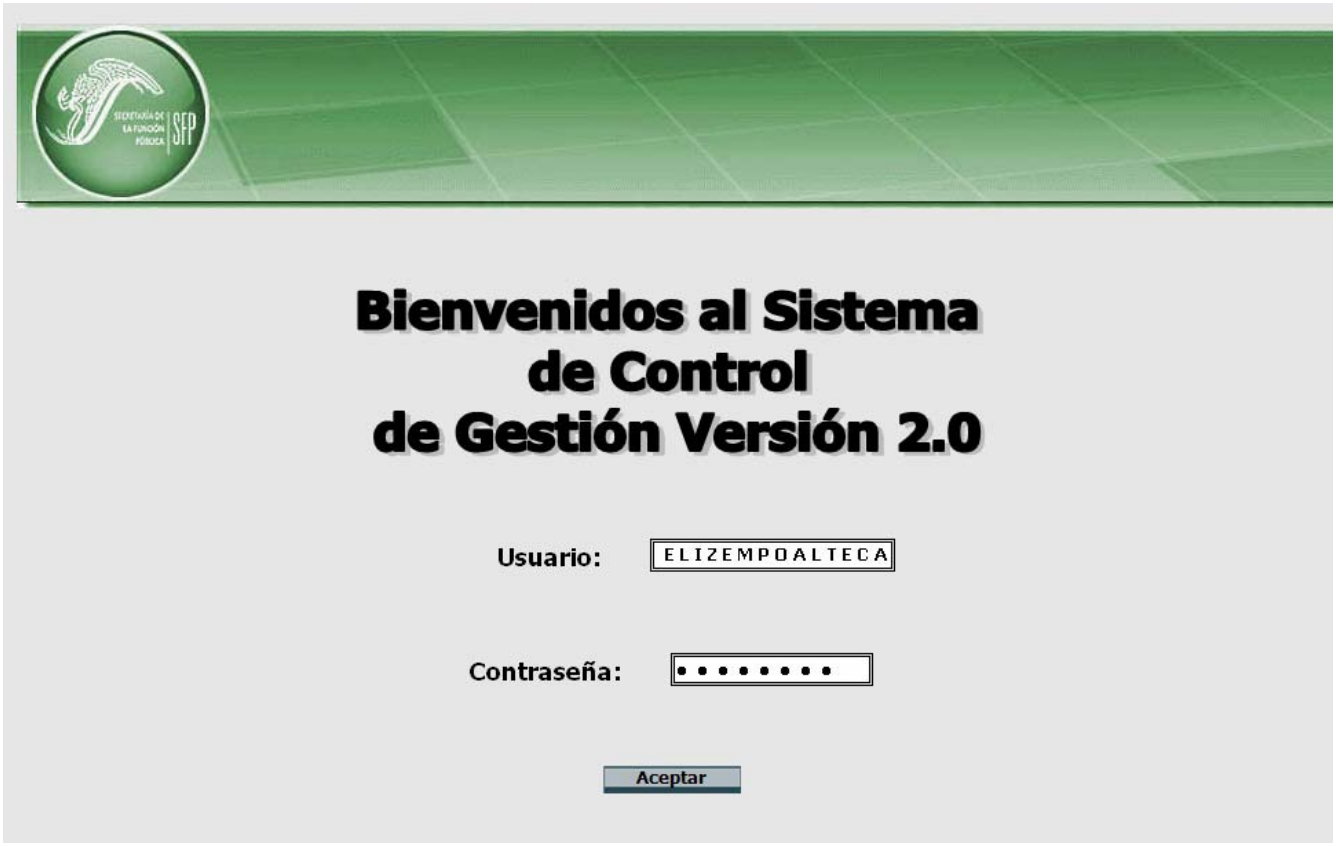
Permite que el actor navegue de forma adecuada.

Proporciona una apariencia agradable y en una forma consistente de trabajo con la interfaz de usuario como a orden de tabulación y teclas rápidas.

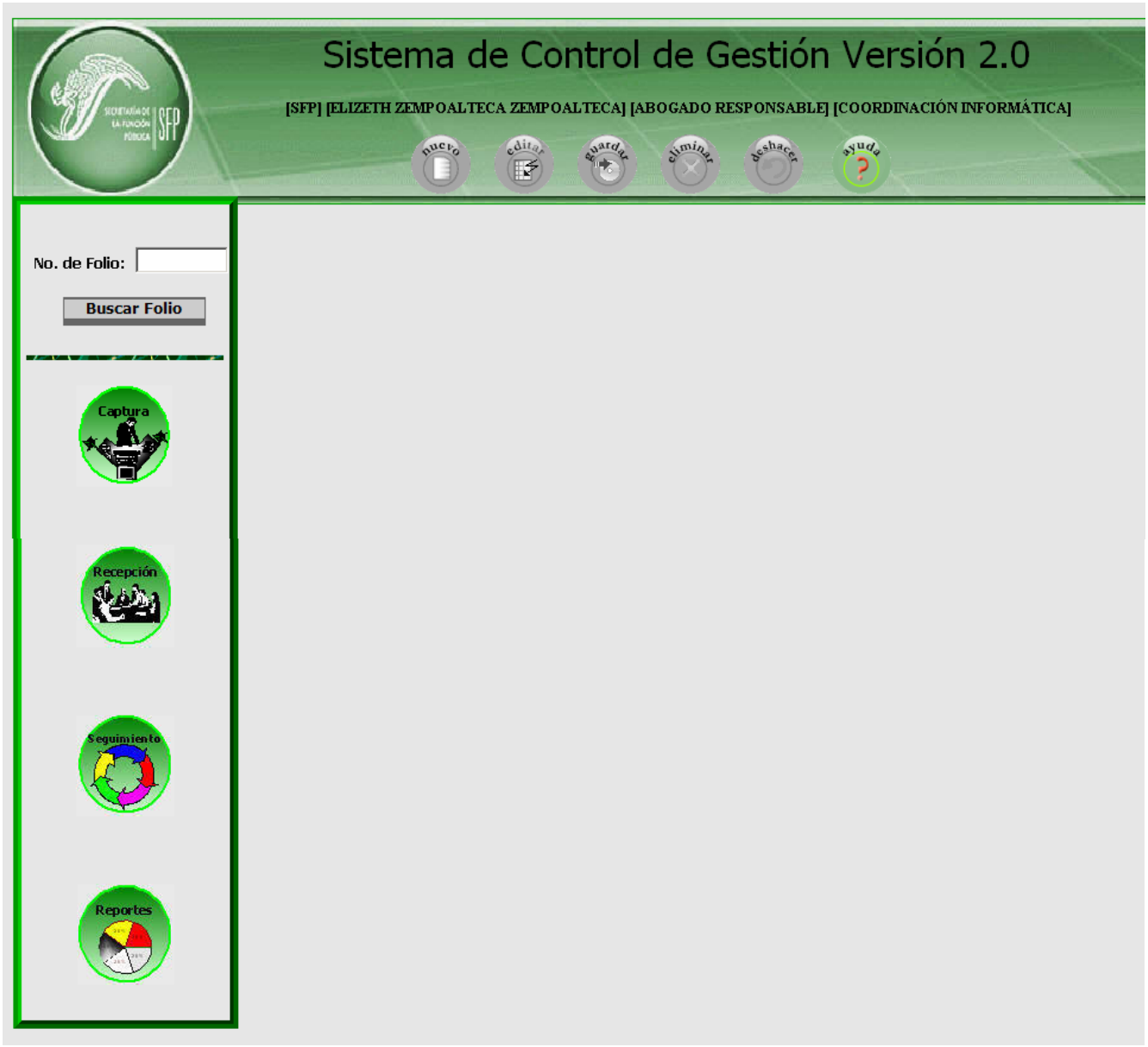
4.7.1 Interfaz Física: Usuario



4.6.2 Interfaz Física: Sistema de Control de Gestión Versión 2.0

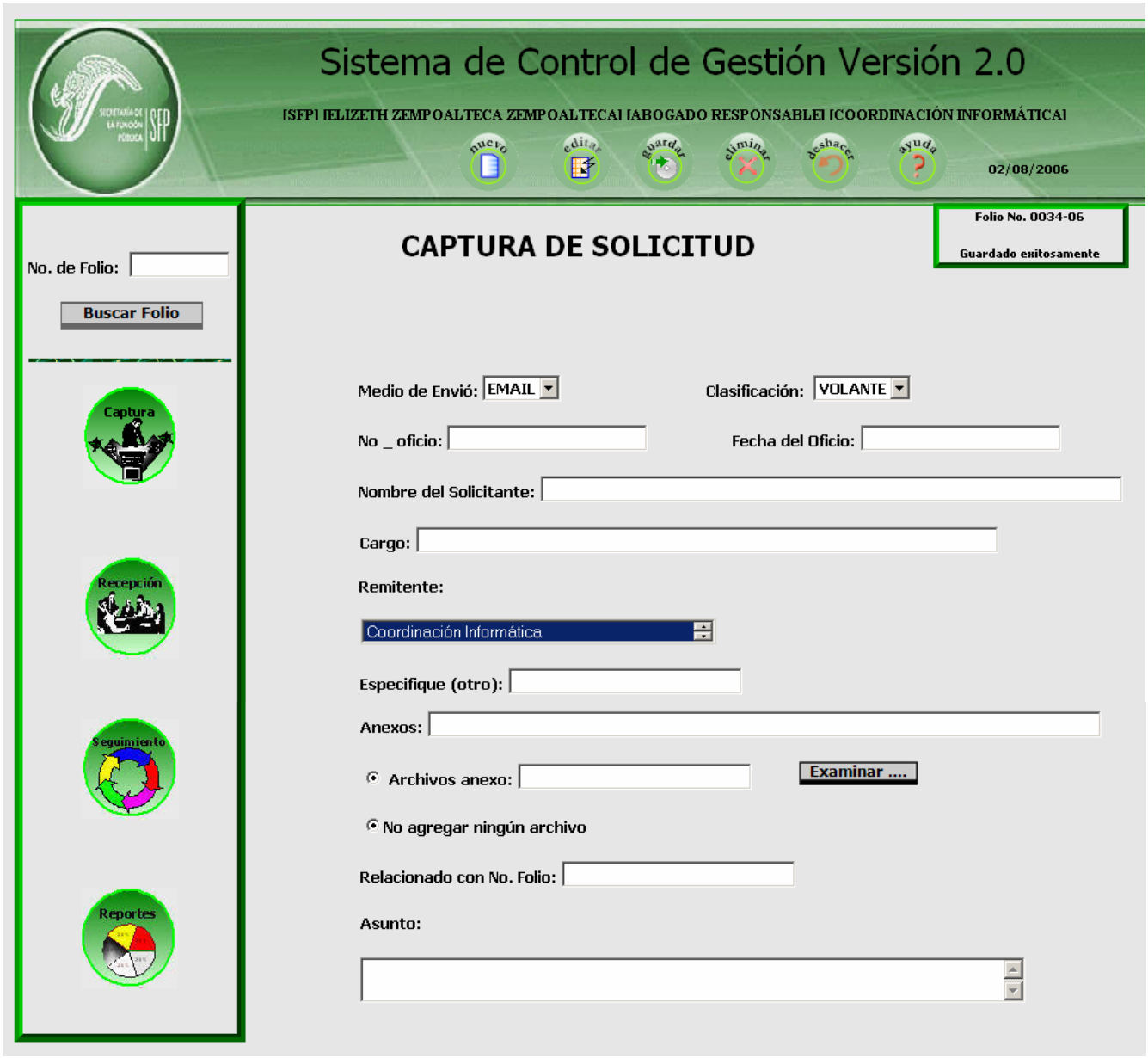


4.7.3 Interfaz Física: Opciones del Sistema



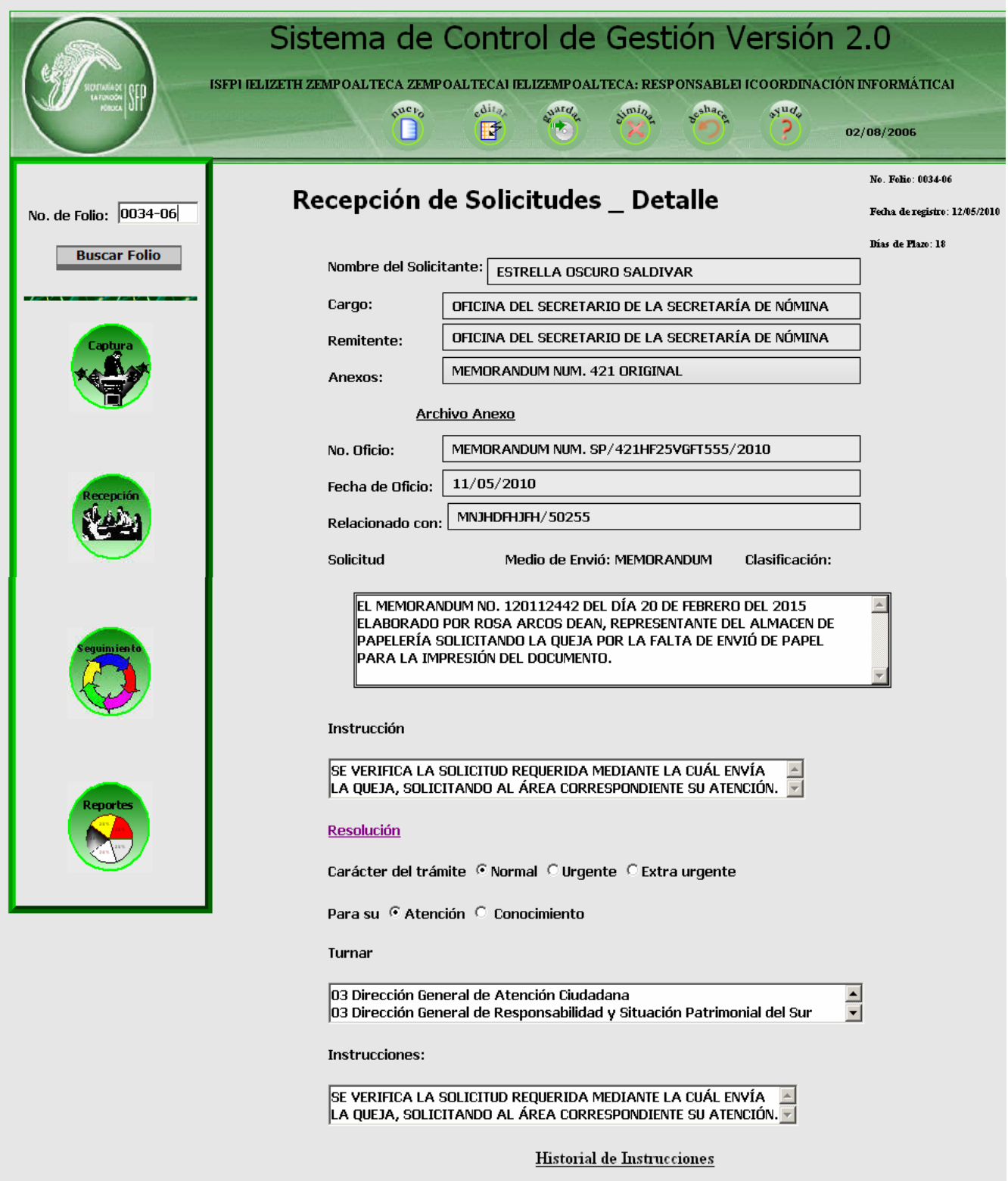
4.7.4 Interfaz Física: Captura de Solicitud

4.7.5 Interfaz Física: Guardar Solicitud

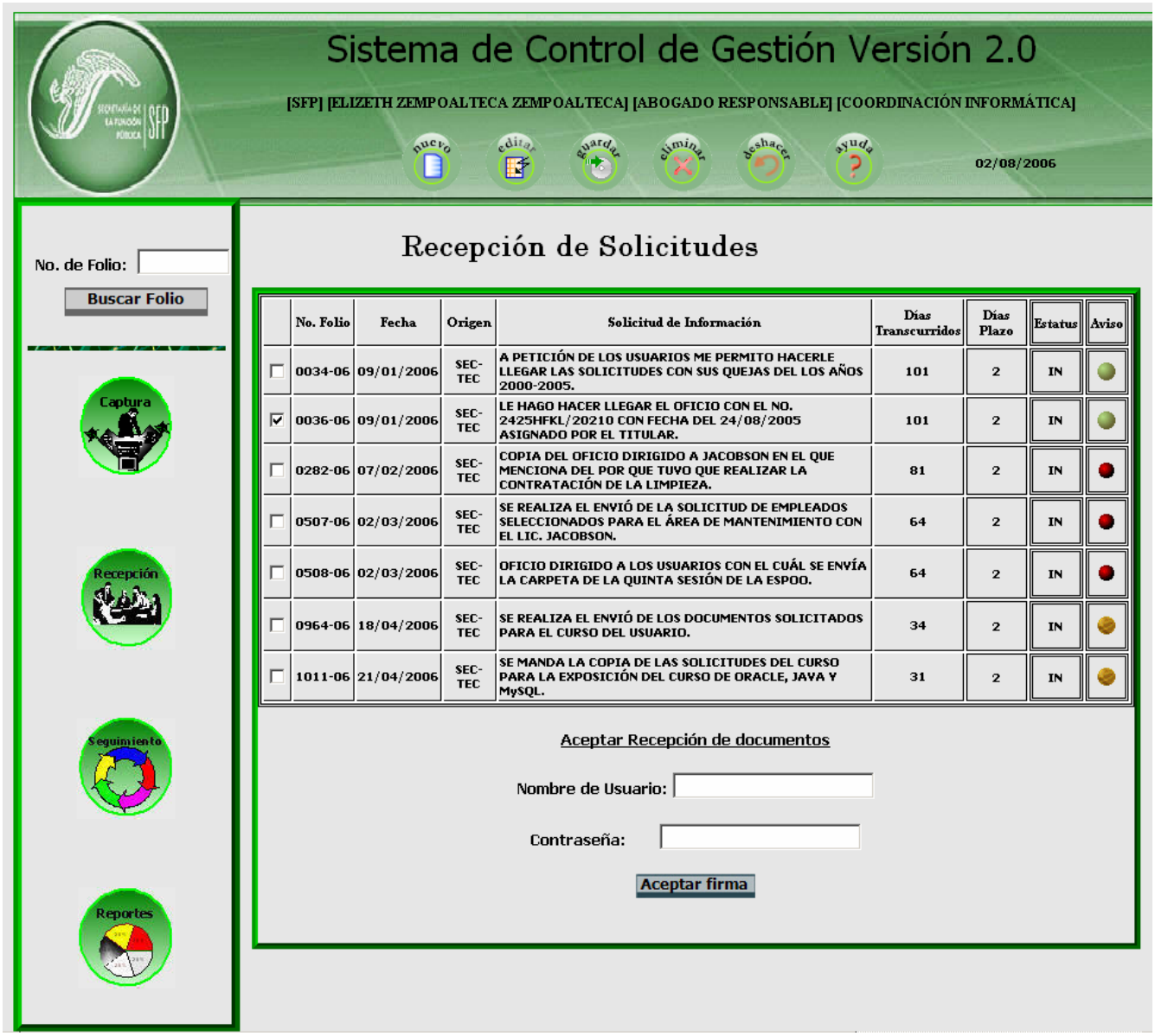


4.7.6 Interfaz Física: Búsqueda por No. Folio en Captura de Solicitud

4.7.7 Interfaz Física: Respuesta de la Búsqueda por No. de Folio en la Captura de Solicitud

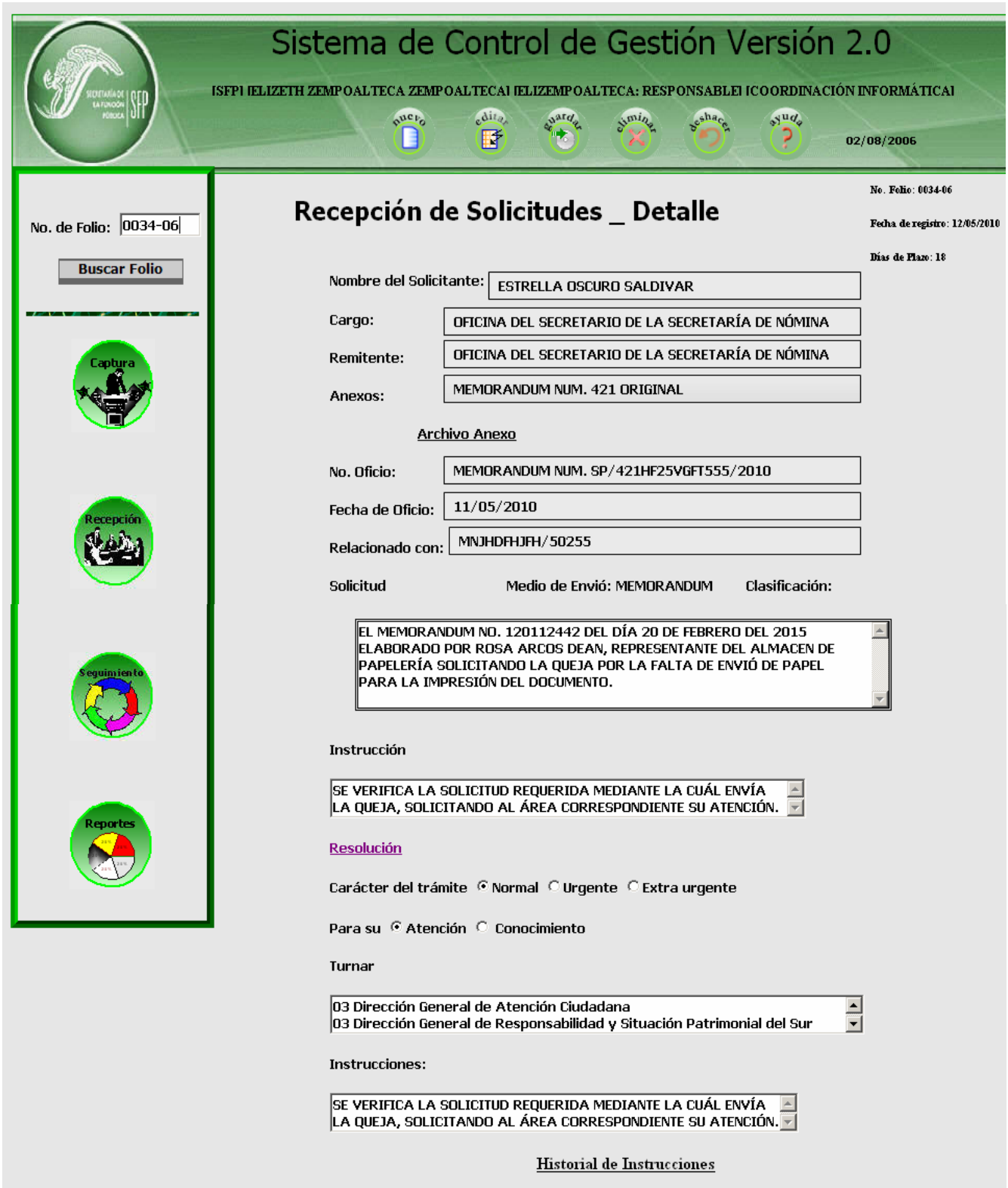


4.7.8 Interfaz Física: Recepción de Solicitudes

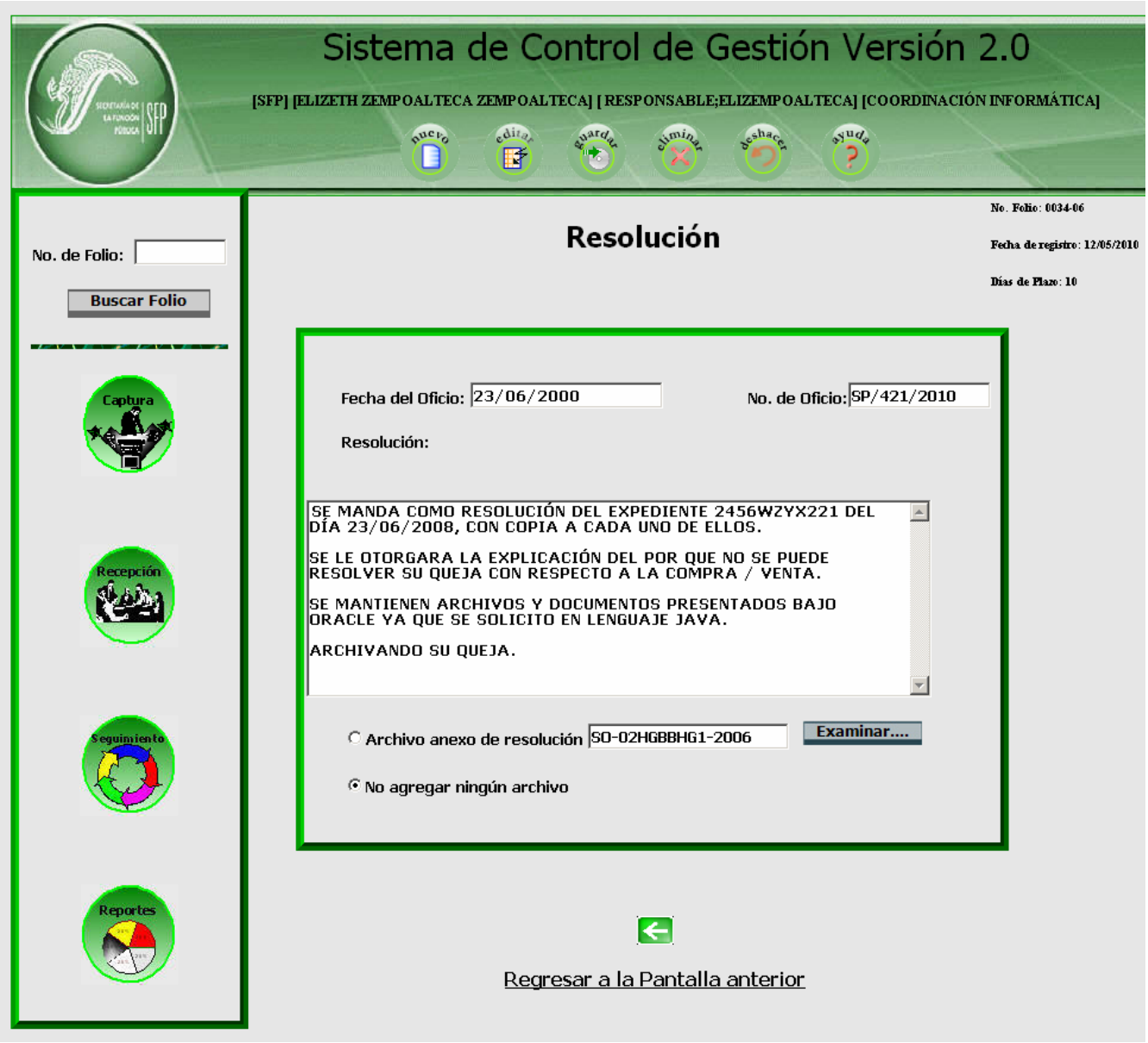


4.7.9 Interfaz Física: Acepta la firma en la Recepción de Solicitudes

4.7.10 Interfaz Física: Recepción de Solicitudes _ detalle al seleccionar por No. de Folio

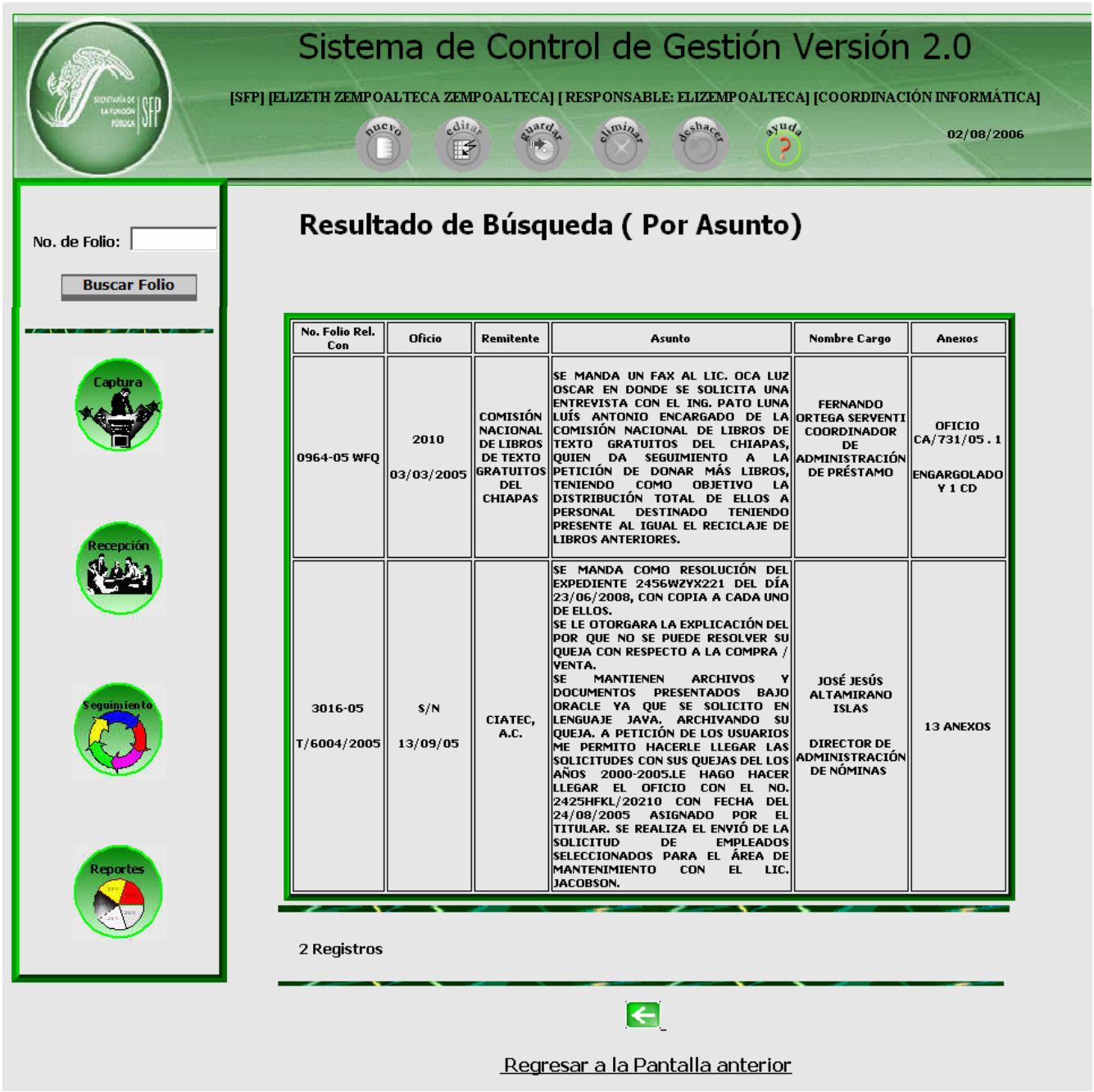


4.7.11 Interfaz Física: Resolución obtenida por la Recepción de Solicitudes _ detalle



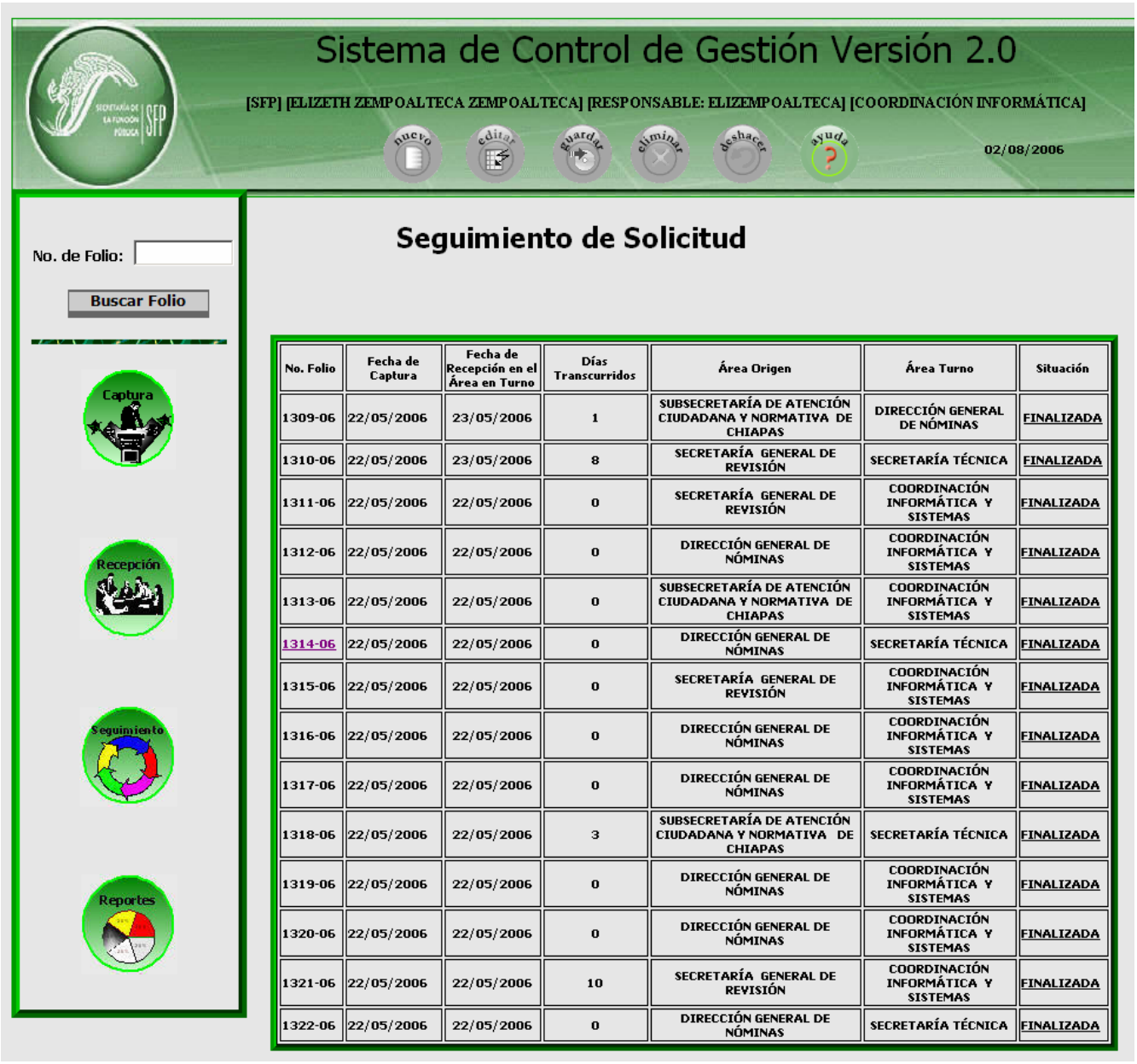
4.7.12 Interfaz Física: Búsqueda de Solicitud

4.6.13 Interfaz Física: Resultado de Búsqueda por Asunto



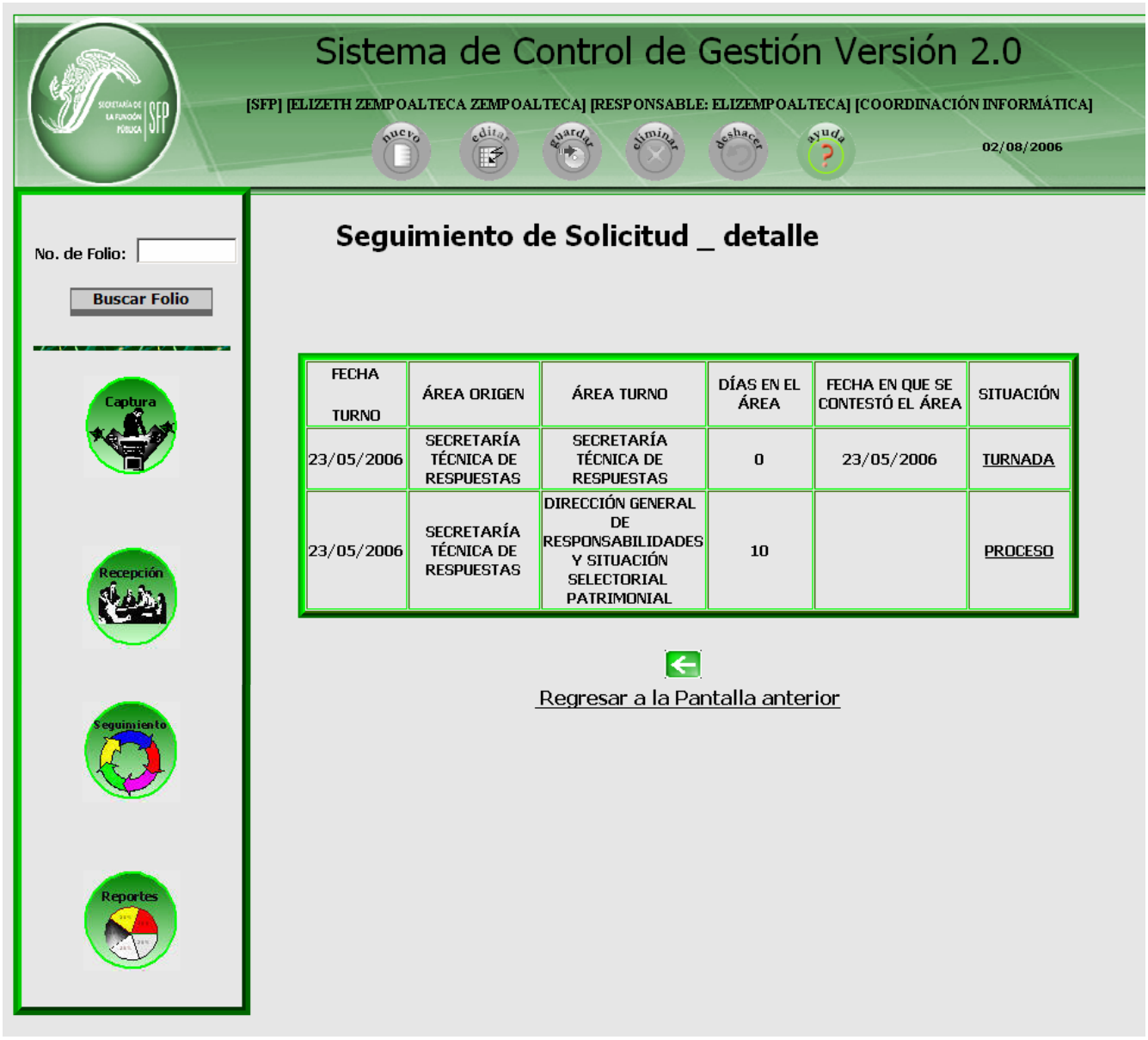
No. Folio Rel. Con	Oficio	Remitente	Asunto	Nombre Cargo	Anexos
0964-05 WFQ	2010 03/03/2005	COMISIÓN NACIONAL DE LIBROS DE TEXTO GRATUITOS DEL CHIAPAS	SE MANDA UN FAX AL LIC. OCA LUZ OSCAR EN DONDE SE SOLICITA UNA ENTREVISTA CON EL ING. PATO LUNA LUÍS ANTONIO ENCARGADO DE LA COMISIÓN NACIONAL DE LIBROS DE TEXTO GRATUITOS DEL CHIAPAS, QUIEN DA SEGUIMIENTO A LA PETICIÓN DE DONAR MÁS LIBROS, TENIENDO COMO OBJETIVO LA DISTRIBUCIÓN TOTAL DE ELLOS A PERSONAL DESTINADO TENIENDO PRESENTE AL IGUAL EL RECICLAJE DE LIBROS ANTERIORES.	FERNANDO ORTEGA SERVENTI COORDINADOR DE ADMINISTRACIÓN DE PRÉSTAMO	OFICIO CA/731/05 . 1 ENGARGOLADO Y 1 CD
3016-05 T/6004/2005	S/N 13/09/05	CIATEC, A.C.	SE MANDA COMO RESOLUCIÓN DEL EXPEDIENTE 2456WZYX221 DEL DÍA 23/06/2008, CON COPIA A CADA UNO DE ELLOS. SE LE OTORGARA LA EXPLICACIÓN DEL POR QUE NO SE PUEDE RESOLVER SU QUEJA CON RESPECTO A LA COMPRA / VENTA. SE MANTIENEN ARCHIVOS Y DOCUMENTOS PRESENTADOS BAJO ORACLE YA QUE SE SOLICITO EN LENGUAJE JAVA. ARCHIVANDO SU QUEJA. A PETICIÓN DE LOS USUARIOS ME PERMITO HACERLE LLEGAR LAS SOLICITUDES CON SUS QUEJAS DEL LOS AÑOS 2000-2005.LE HAGO HACER LLEGAR EL OFICIO CON EL NO. 2425HFKL/20210 CON FECHA DEL 24/08/2005 ASIGNADO POR EL TITULAR. SE REALIZA EL ENVIÓ DE LA SOLICITUD DE EMPLEADOS SELECCIONADOS PARA EL ÁREA DE MANTENIMIENTO CON EL LIC. JACOBSON.	JOSÉ JESÚS ALTAMIRANO ISLAS DIRECTOR DE ADMINISTRACIÓN DE NÓMINAS	13 ANEXOS

4.7.14 Interfaz Física: Seguimiento de la Solicitud



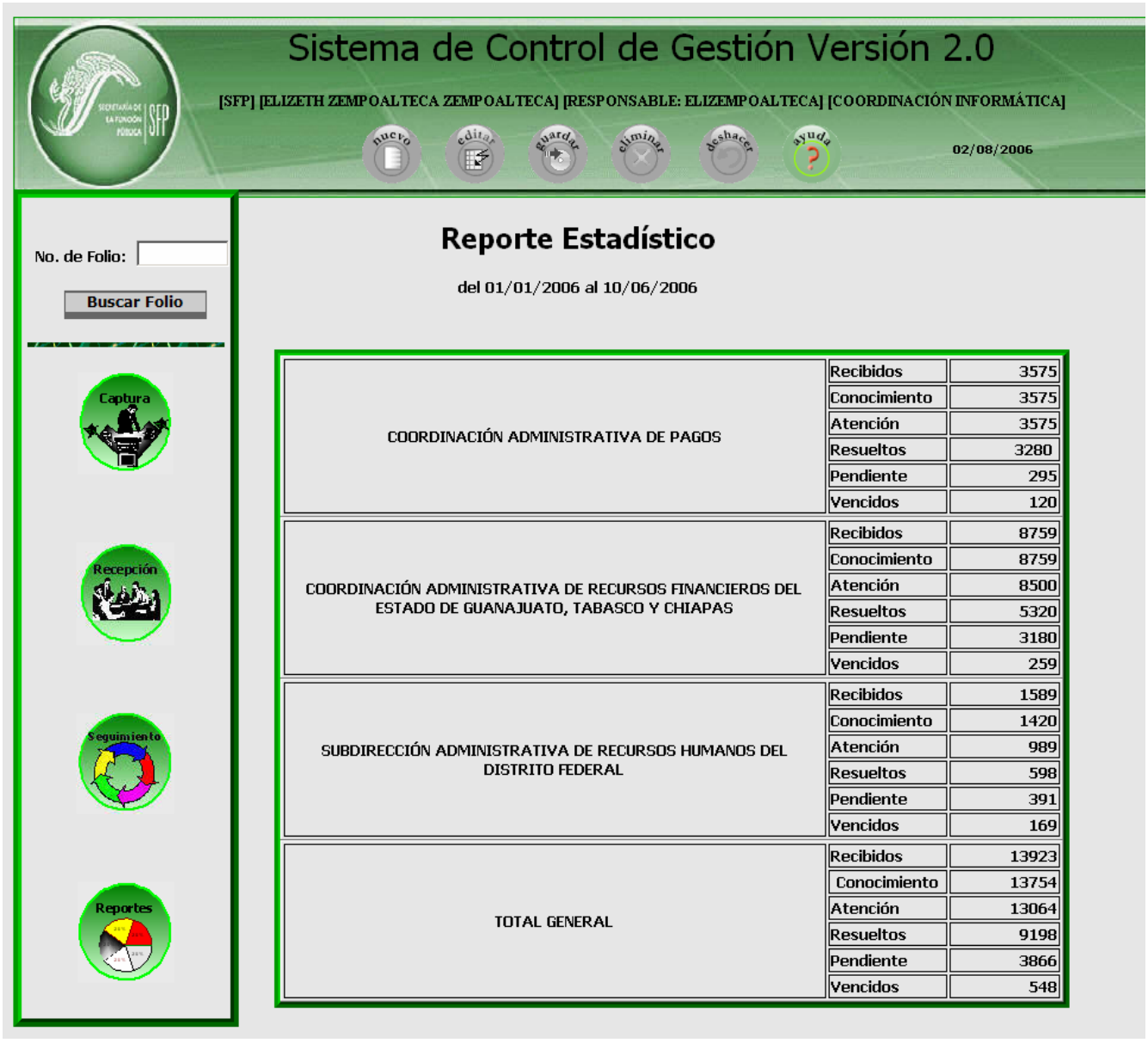
No. Folio	Fecha de Captura	Fecha de Recepción en el Área en Turno	Días Transcurridos	Área Origen	Área Turno	Situación
1309-06	22/05/2006	23/05/2006	1	SUBSECRETARÍA DE ATENCIÓN CIUDADANA Y NORMATIVA DE CHIAPAS	DIRECCIÓN GENERAL DE NÓMINAS	FINALIZADA
1310-06	22/05/2006	23/05/2006	8	SECRETARÍA GENERAL DE REVISIÓN	SECRETARÍA TÉCNICA	FINALIZADA
1311-06	22/05/2006	22/05/2006	0	SECRETARÍA GENERAL DE REVISIÓN	COORDINACIÓN INFORMÁTICA Y SISTEMAS	FINALIZADA
1312-06	22/05/2006	22/05/2006	0	DIRECCIÓN GENERAL DE NÓMINAS	COORDINACIÓN INFORMÁTICA Y SISTEMAS	FINALIZADA
1313-06	22/05/2006	22/05/2006	0	SUBSECRETARÍA DE ATENCIÓN CIUDADANA Y NORMATIVA DE CHIAPAS	COORDINACIÓN INFORMÁTICA Y SISTEMAS	FINALIZADA
1314-06	22/05/2006	22/05/2006	0	DIRECCIÓN GENERAL DE NÓMINAS	SECRETARÍA TÉCNICA	FINALIZADA
1315-06	22/05/2006	22/05/2006	0	SECRETARÍA GENERAL DE REVISIÓN	COORDINACIÓN INFORMÁTICA Y SISTEMAS	FINALIZADA
1316-06	22/05/2006	22/05/2006	0	DIRECCIÓN GENERAL DE NÓMINAS	COORDINACIÓN INFORMÁTICA Y SISTEMAS	FINALIZADA
1317-06	22/05/2006	22/05/2006	0	DIRECCIÓN GENERAL DE NÓMINAS	COORDINACIÓN INFORMÁTICA Y SISTEMAS	FINALIZADA
1318-06	22/05/2006	22/05/2006	3	SUBSECRETARÍA DE ATENCIÓN CIUDADANA Y NORMATIVA DE CHIAPAS	SECRETARÍA TÉCNICA	FINALIZADA
1319-06	22/05/2006	22/05/2006	0	DIRECCIÓN GENERAL DE NÓMINAS	COORDINACIÓN INFORMÁTICA Y SISTEMAS	FINALIZADA
1320-06	22/05/2006	22/05/2006	0	DIRECCIÓN GENERAL DE NÓMINAS	COORDINACIÓN INFORMÁTICA Y SISTEMAS	FINALIZADA
1321-06	22/05/2006	22/05/2006	10	SECRETARÍA GENERAL DE REVISIÓN	COORDINACIÓN INFORMÁTICA Y SISTEMAS	FINALIZADA
1322-06	22/05/2006	22/05/2006	0	DIRECCIÓN GENERAL DE NÓMINAS	SECRETARÍA TÉCNICA	FINALIZADA

4.7.15 Interfaz Física: Seguimiento de la Solicitud _ detalle



4.7.16 Interfaz Física: Reporte Estadístico

4.7.17 Interfaz Física: Detalle del Reporte Estadístico

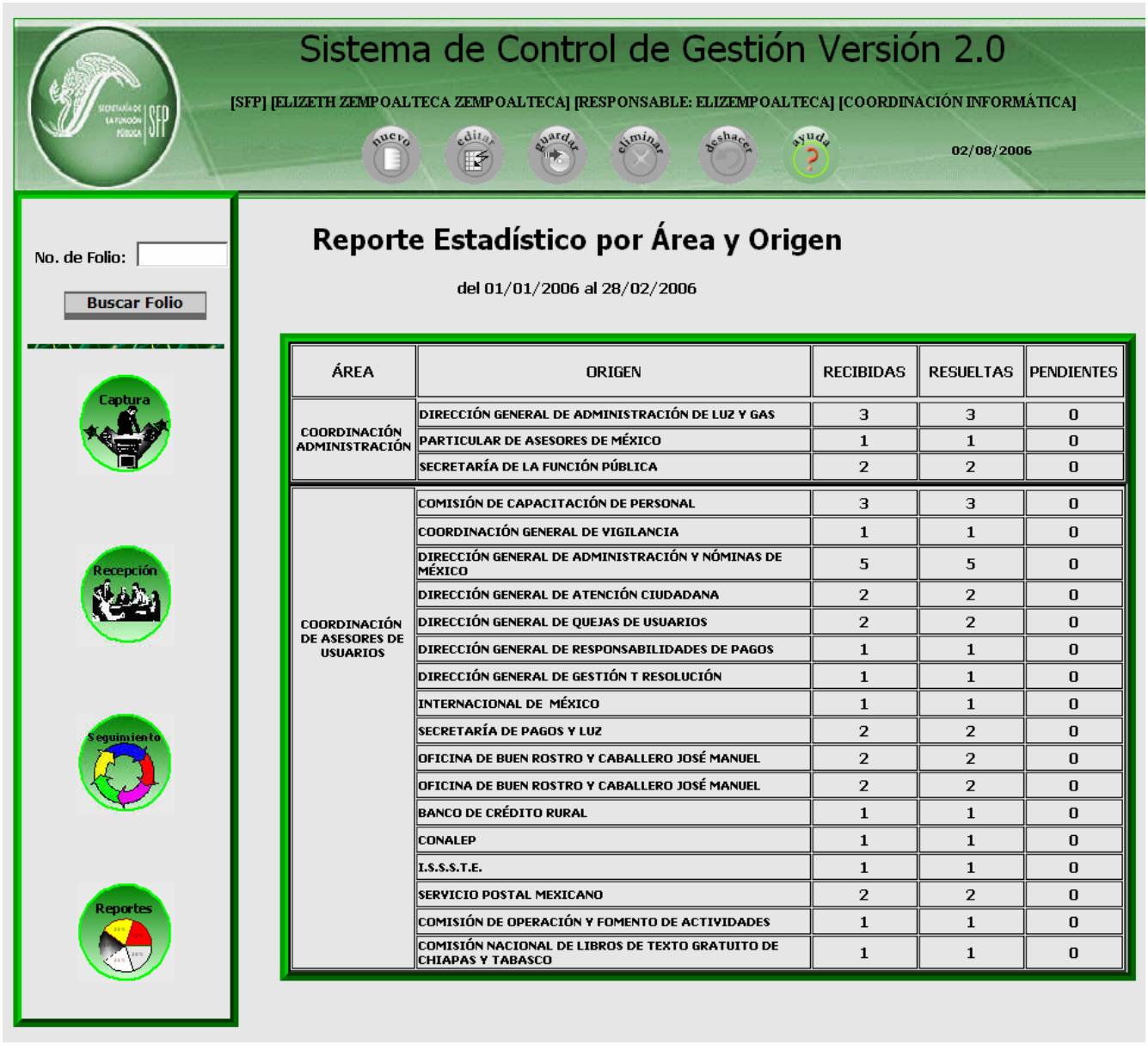


Reporte Estadístico

del 01/01/2006 al 10/06/2006

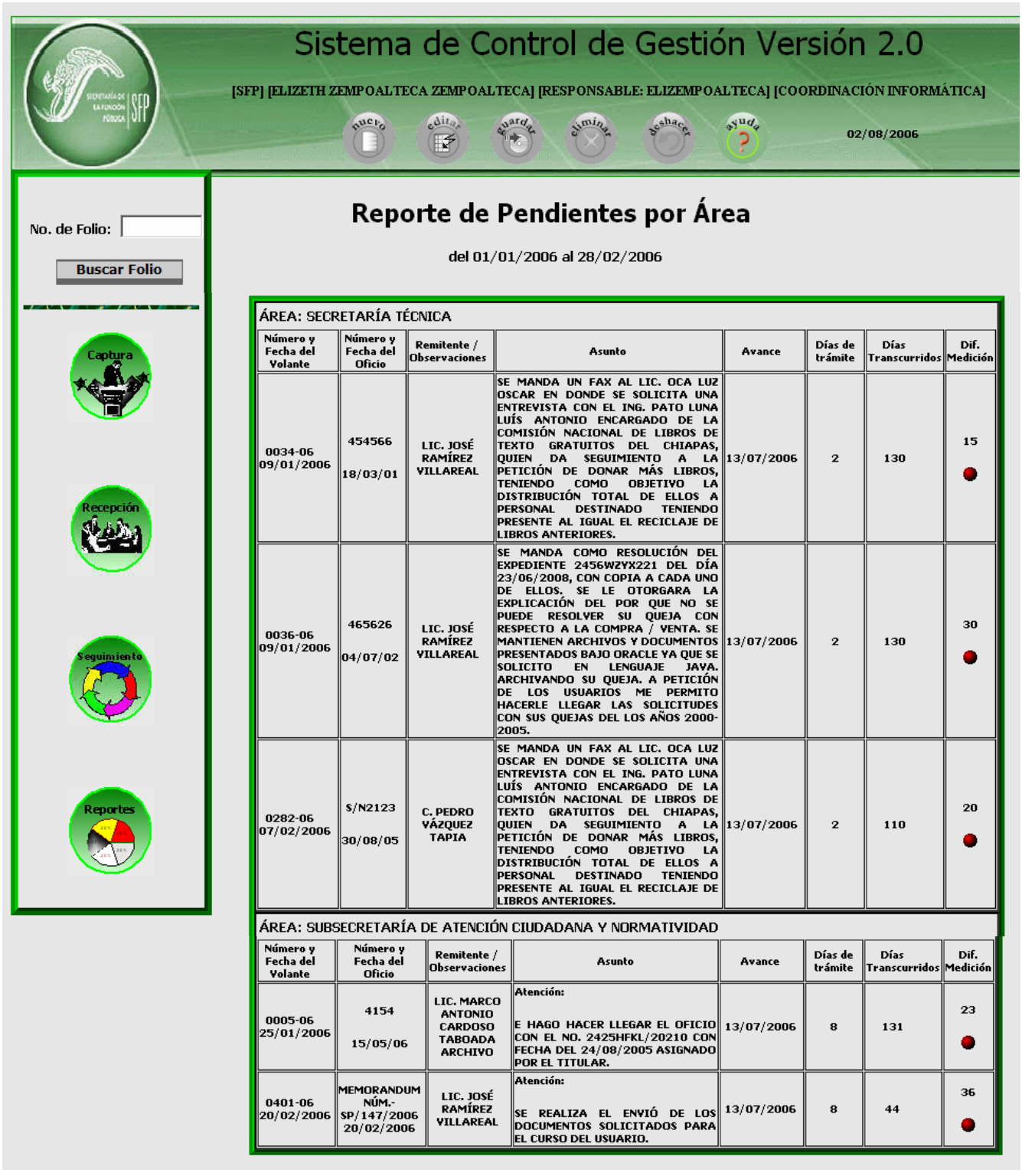
COORDINACIÓN ADMINISTRATIVA DE PAGOS	Recibidos	3575
	Conocimiento	3575
	Atención	3575
	Resueltos	3280
	Pendiente	295
	Vencidos	120
COORDINACIÓN ADMINISTRATIVA DE RECURSOS FINANCIEROS DEL ESTADO DE GUANAJUATO, TABASCO Y CHIAPAS	Recibidos	8759
	Conocimiento	8759
	Atención	8500
	Resueltos	5320
	Pendiente	3180
	Vencidos	259
SUBDIRECCIÓN ADMINISTRATIVA DE RECURSOS HUMANOS DEL DISTRITO FEDERAL	Recibidos	1589
	Conocimiento	1420
	Atención	989
	Resueltos	598
	Pendiente	391
	Vencidos	169
TOTAL GENERAL	Recibidos	13923
	Conocimiento	13754
	Atención	13064
	Resueltos	9198
	Pendiente	3866
	Vencidos	548

4.7.18 Interfaz Física: Reporte Estadístico por Área y Origen

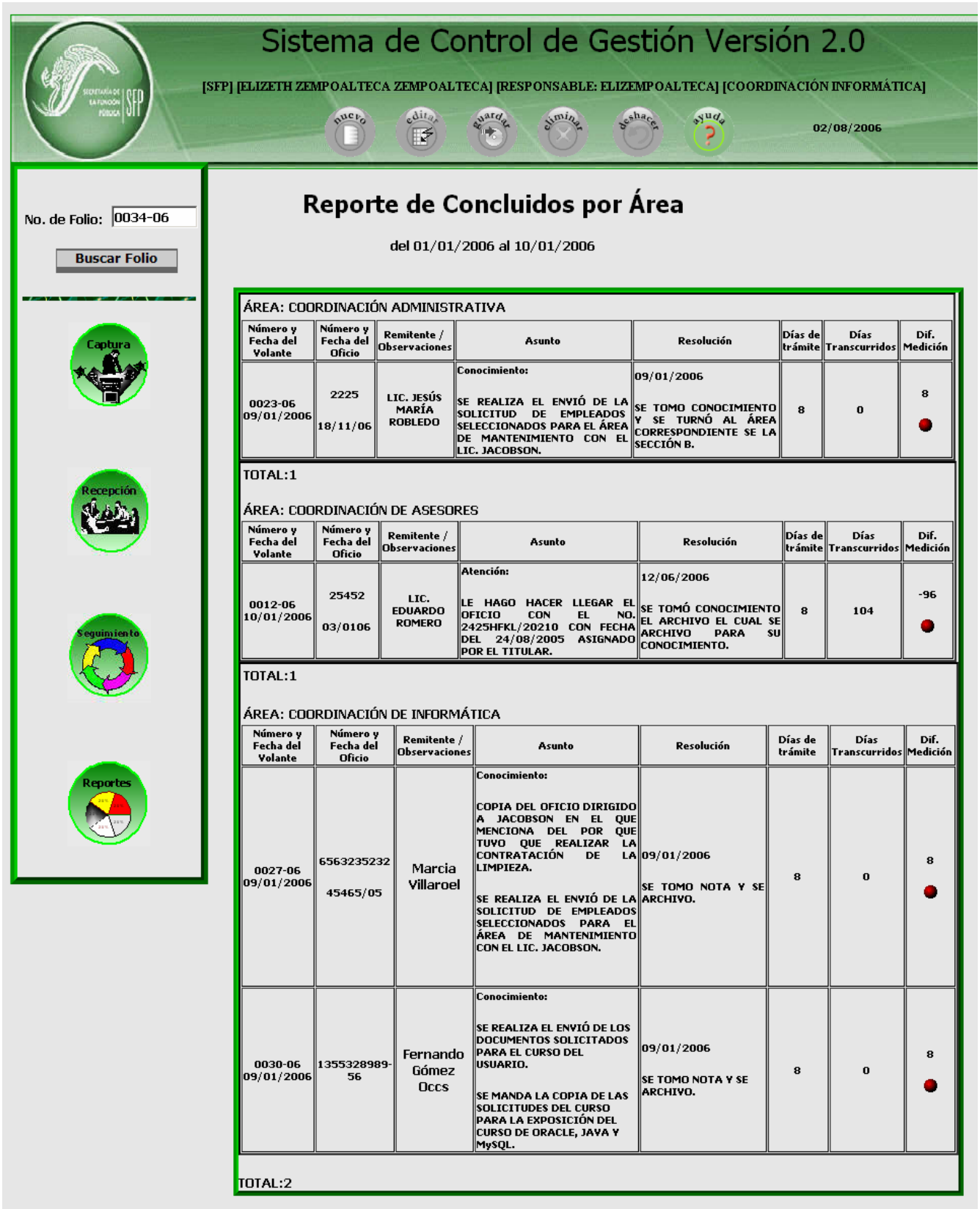


ÁREA	ORIGEN	RECIBIDAS	RESUELTAS	PENDIENTES
COORDINACIÓN ADMINISTRACIÓN	DIRECCIÓN GENERAL DE ADMINISTRACIÓN DE LUZ Y GAS	3	3	0
	PARTICULAR DE ASESORES DE MÉXICO	1	1	0
	SECRETARÍA DE LA FUNCIÓN PÚBLICA	2	2	0
COORDINACIÓN DE ASESORES DE USUARIOS	COMISIÓN DE CAPACITACIÓN DE PERSONAL	3	3	0
	COORDINACIÓN GENERAL DE VIGILANCIA	1	1	0
	DIRECCIÓN GENERAL DE ADMINISTRACIÓN Y NÓMINAS DE MÉXICO	5	5	0
	DIRECCIÓN GENERAL DE ATENCIÓN CIUDADANA	2	2	0
	DIRECCIÓN GENERAL DE QUEJAS DE USUARIOS	2	2	0
	DIRECCIÓN GENERAL DE RESPONSABILIDADES DE PAGOS	1	1	0
	DIRECCIÓN GENERAL DE GESTIÓN T RESOLUCIÓN	1	1	0
	INTERNACIONAL DE MÉXICO	1	1	0
	SECRETARÍA DE PAGOS Y LUZ	2	2	0
	OFICINA DE BUEN ROSTRO Y CABALLERO JOSÉ MANUEL	2	2	0
	OFICINA DE BUEN ROSTRO Y CABALLERO JOSÉ MANUEL	2	2	0
	BANCO DE CRÉDITO RURAL	1	1	0
	CONALEP	1	1	0
	I.S.S.S.T.E.	1	1	0
	SERVICIO POSTAL MEXICANO	2	2	0
	COMISIÓN DE OPERACIÓN Y FOMENTO DE ACTIVIDADES	1	1	0
	COMISIÓN NACIONAL DE LIBROS DE TEXTO GRATUITO DE CHIAPAS Y TABASCO	1	1	0

4.7.19 Interfaz Física: Reporte de Pendientes por Área



4.7.20 Interfaz Física: Reporte de Concluídos por Área



Reporte de Concluidos por Área

del 01/01/2006 al 10/01/2006

ÁREA: COORDINACIÓN ADMINISTRATIVA

Número y Fecha del Volante	Número y Fecha del Oficio	Remitente / Observaciones	Asunto	Resolución	Días de trámite	Días Transcurridos	Dif. Medición
0023-06 09/01/2006	2225 18/11/06	LIC. JESÚS MARÍA ROBLEDO	Conocimiento: SE REALIZA EL ENVIÓ DE LA SOLICITUD DE EMPLEADOS SELECCIONADOS PARA EL ÁREA DE MANTENIMIENTO CON EL LIC. JACOBSON.	09/01/2006 SE TOMO CONOCIMIENTO Y SE TURNÓ AL ÁREA CORRESPONDIENTE SE LA SECCIÓN B.	8	0	8

TOTAL:1

ÁREA: COORDINACIÓN DE ASESORES

Número y Fecha del Volante	Número y Fecha del Oficio	Remitente / Observaciones	Asunto	Resolución	Días de trámite	Días Transcurridos	Dif. Medición
0012-06 10/01/2006	25452 03/0106	LIC. EDUARDO ROMERO	Atención: LE HAGO HACER LLEGAR EL OFICIO CON EL NO. 2425HFKL/20210 CON FECHA DEL 24/08/2005 ASIGNADO POR EL TITULAR.	12/06/2006 SE TOMÓ CONOCIMIENTO EL ARCHIVO EL CUAL SE ARCHIVO PARA SU CONOCIMIENTO.	8	104	-96

TOTAL:1

ÁREA: COORDINACIÓN DE INFORMÁTICA

Número y Fecha del Volante	Número y Fecha del Oficio	Remitente / Observaciones	Asunto	Resolución	Días de trámite	Días Transcurridos	Dif. Medición
0027-06 09/01/2006	6563235232 45465/05	Marcia Villaroel	Conocimiento: COPIA DEL OFICIO DIRIGIDO A JACOBSON EN EL QUE MENCIONA DEL POR QUE TUVO QUE REALIZAR LA CONTRATACIÓN DE LA LIMPIEZA. SE REALIZA EL ENVIÓ DE LA SOLICITUD DE EMPLEADOS SELECCIONADOS PARA EL ÁREA DE MANTENIMIENTO CON EL LIC. JACOBSON.	09/01/2006 SE TOMO NOTA Y SE ARCHIVO.	8	0	8
0030-06 09/01/2006	1355328989-56	Fernando Gómez Occs	Conocimiento: SE REALIZA EL ENVIÓ DE LOS DOCUMENTOS SOLICITADOS PARA EL CURSO DEL USUARIO. SE MANDA LA COPIA DE LAS SOLICITUDES DEL CURSO PARA LA EXPOSICIÓN DEL CURSO DE ORACLE, JAVA Y MySQL.	09/01/2006 SE TOMO NOTA Y SE ARCHIVO.	8	0	8

TOTAL:2

4.8 Análisis

4.8.1 Introducción

En esta etapa, analizamos los requisitos que se describieron en la captura de requisitos, refinándolos y estructurándolos.

El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que nos ayude a estructurar el sistema entero incluyendo su arquitectura.

El resultado del flujo de trabajo del análisis es el modelo de análisis, que es un modelo de objetos conceptual que analiza los requisitos mediante su refinamiento y estructuración.

El modelo del análisis incluye los siguientes elementos:

- Paquetes del análisis y paquetes de servicio, y sus dependencias y contenidos. Los paquetes del análisis pueden aislar los cambios de un proceso del negocio, el comportamiento de un actor, o en un conjunto de casos de uso estrechamente relacionados.

  Los paquetes de servicio aislarán los cambios en determinados servicios ofrecidos por el sistema, y constituyen un elemento esencial para construir pensando en la reutilización durante el análisis.

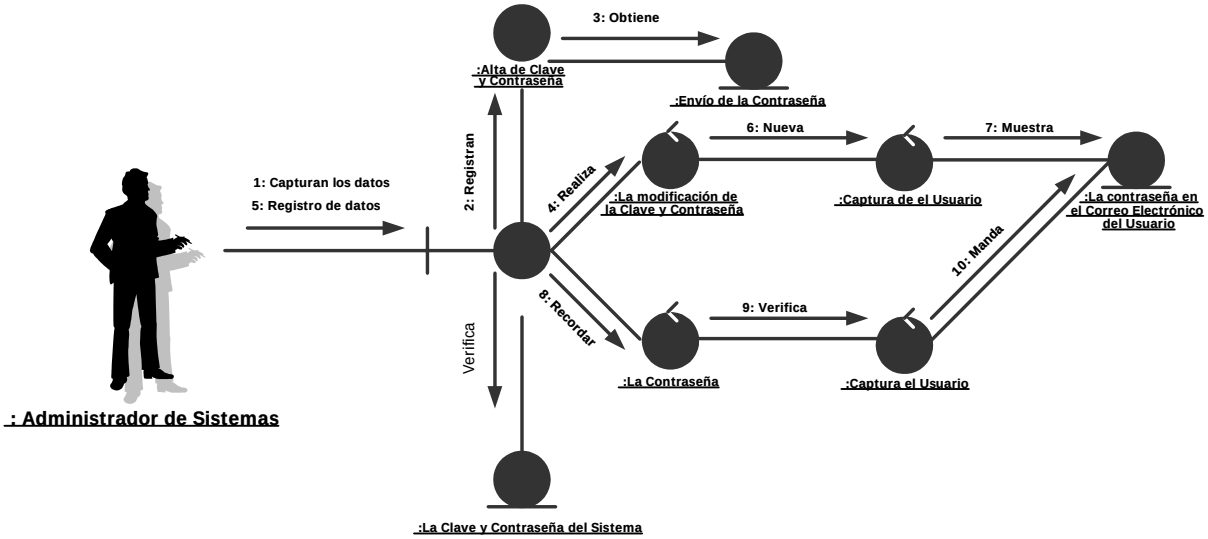
- Clases de análisis, sus responsabilidades, atributos, relaciones, y requisitos especiales. Cada una de las clases de control, entidad e interfaz aislarán los cambios al comportamiento y a la información que representa.

Realizaciones de casos de uso-análisis, que describen como se refinan los casos de uso en términos de colaboraciones dentro del modelo de análisis y de sus requisitos especiales.

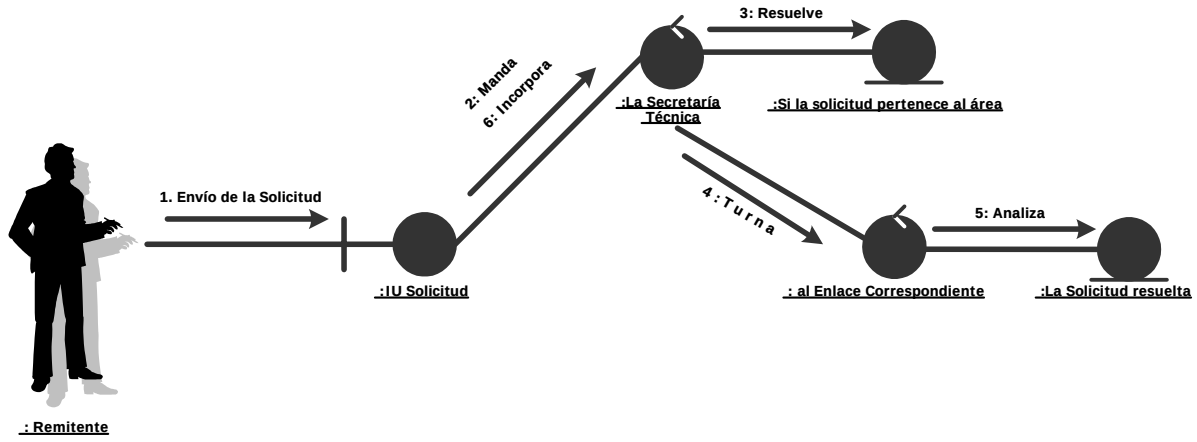
4.8.2 Diagrama de Interacción y/o Colaboración

Un diagrama de interacción que enfatiza la organización estructural de los objetos que envían y reciben mensajes; un diagrama que muestra las interacciones organizadas alrededor de instancias y de los enlaces entre ellas.
En los diagramas de colaboración se muestran las interacciones entre objetos creando enlaces entre ellos y añadiendo mensajes a esos enlaces.

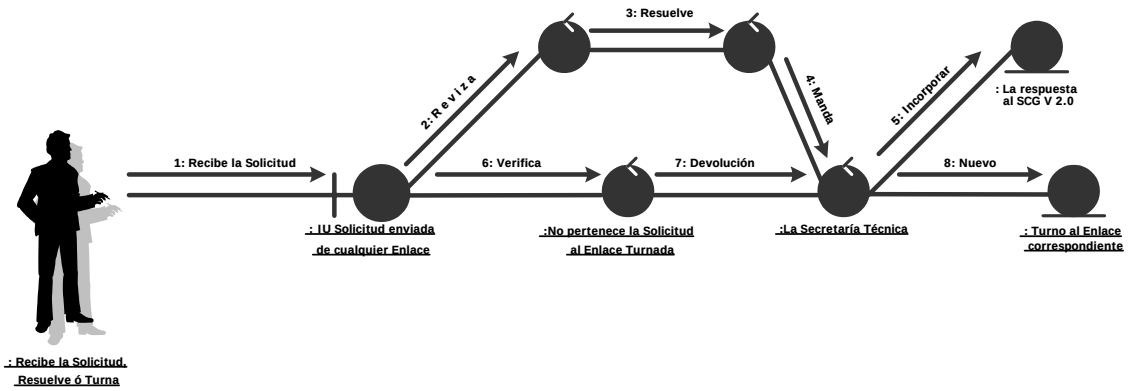
4.8.2.1 Diagrama de Colaboración: Asignación de claves al personal



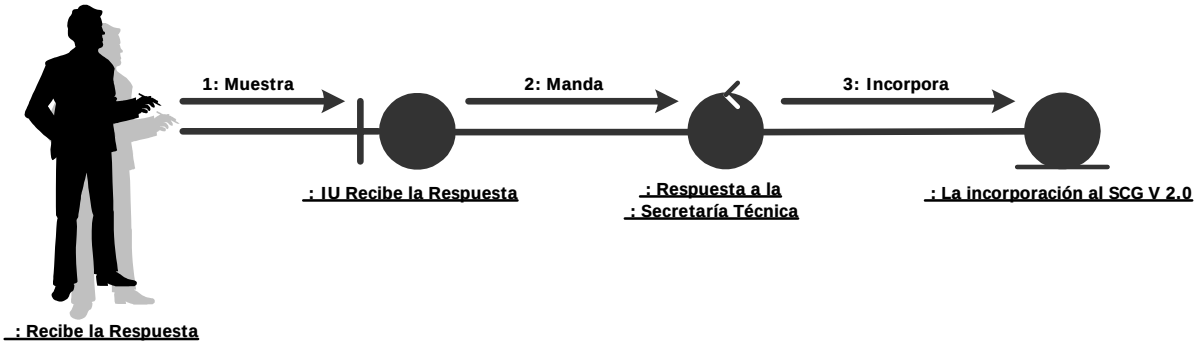
4.8.2.2 Diagrama de Colaboración: Envío de Solicitudes



4.8.2.3 Diagrama de Colaboración: Recibe la Solicitud, Resuelve ó Turna



4.8.2.4 Diagrama de Colaboración: Recibe Respuesta



4.8.3 Paquetes

La forma que tiene UML de agrupar elementos en subsistemas es a través del uso de **Paquetes**, pudiéndose anidar los paquetes formando jerarquías de paquetes.

De hecho un sistema que no tenga necesidad de ser descompuesto en subsistemas se puede considerar como un único paquete que lo abarca todo.

Un mecanismo de propósito general para organizar elementos en grupos. Los paquetes del análisis proporcionan un medio para organizar los artefactos del modelo del análisis en piezas manejables.

Un paquete puede constar de clases de análisis, de realizaciones de caso de uso, y de otros paquetes del análisis. Los paquetes de análisis son cohesivos (es decir, sus contenidos deberían estar fuertemente relacionados), deberían ser débilmente acoplados (es decir, sus dependencias unos de otros deberían de maximizarse).

- Los paquetes de análisis pueden representar una separación de interés de análisis. Es decir, en un sistema grande algunos paquetes de análisis pueden analizarse de manera concurrente por diferentes desarrolladores con diferente conocimiento de dominio.

- Los paquetes de análisis deberían crearse basándose en los requisitos funcionales y en el dominio del problema (es decir, la aplicación o el negocio) y deberían de ser reconocibles por personas con conocimiento del dominio.

Los paquetes del análisis probablemente se convertirán en subsistemas en dos capas de aplicación superiores del modelo de diseña que se distribuirán entre ellos.

Identificación de Paquetes: Vamos a definir una serie de reglas que nos pueden ser de utilidad a la hora de agrupar los diferentes elementos en paquetes.
Conviene agrupar elementos que proporcionen un mismo servicio. Los elementos que se agrupen en un mismo paquete han de presentar un alto grado de cohesión, es decir deben estar muy relacionados. Los elementos que estén en diferentes paquetes deben tener poca relación, es decir debe colaborar lo menos posible.

4.8.4 Paquetes

Fig. 17 Representación de los paquetes del Sistema de Control de Gestión V 2.0

4.9 Diseño

4.9.1 Introducción

El diseño se modela al sistema y se encuentra su forma para que soporte todos los requisitos incluyendo los no funcionales y otras restricciones que se le supone.

Los propósitos de esta etapa son:

Adquirir una compresión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de interfaz de usuario, tecnologías de gestión de transacción, etc..

Crear una entrada apropiada y un conjunto de partida para actividades de implementación subsiguientes capturando los requisitos o subsistemas individuales, interfases y clases.

Ser capaces de descomponer los trabajos de implementación en partes mas manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia.

4.9.2 Diagrama de Despliegue

El diagrama de despliegue es un modelo de objetos que describen la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de computo.

El modelo utiliza como entrada fundamental en las actividades del diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño. Un diagrama que muestra un conjunto de nodos y sus relaciones; un diagrama de despliegue muestra el despliegue de un sistema desde el punto de vista estático. Se puede observar lo siguiente en un modelo de despliegue:

Cada nodo representa un recurso de cómputo, normalmente un procesador o un dispositivo hardware similar.

Los nodos poseen relaciones de comunicación entre ellos, tales como Internet, bus y similares.

El modelo de despliegue puede describir diferentes configuraciones de red, incluidas las configuraciones para pruebas y para simulación.

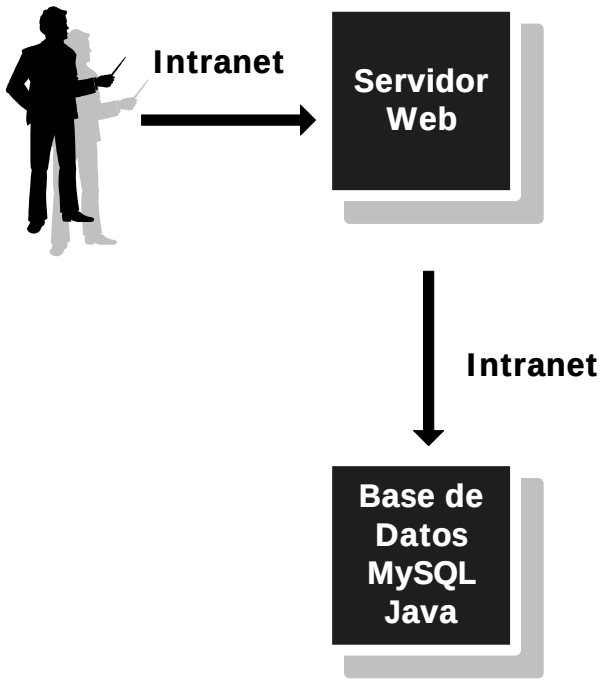


Fig. 18 Diagrama de Despliegue del Sistema de Control de Gestión V 2.0

4.9.3 Diseño de clases del Sistema

Muestra un conjunto de clases, interfases, colaboraciones y las relaciones entre estos; los diagramas de clase muestran el diseño de un sistema desde el punto de vista estadístico; un diagrama que muestra una colección de elementos estadísticos declarativos.

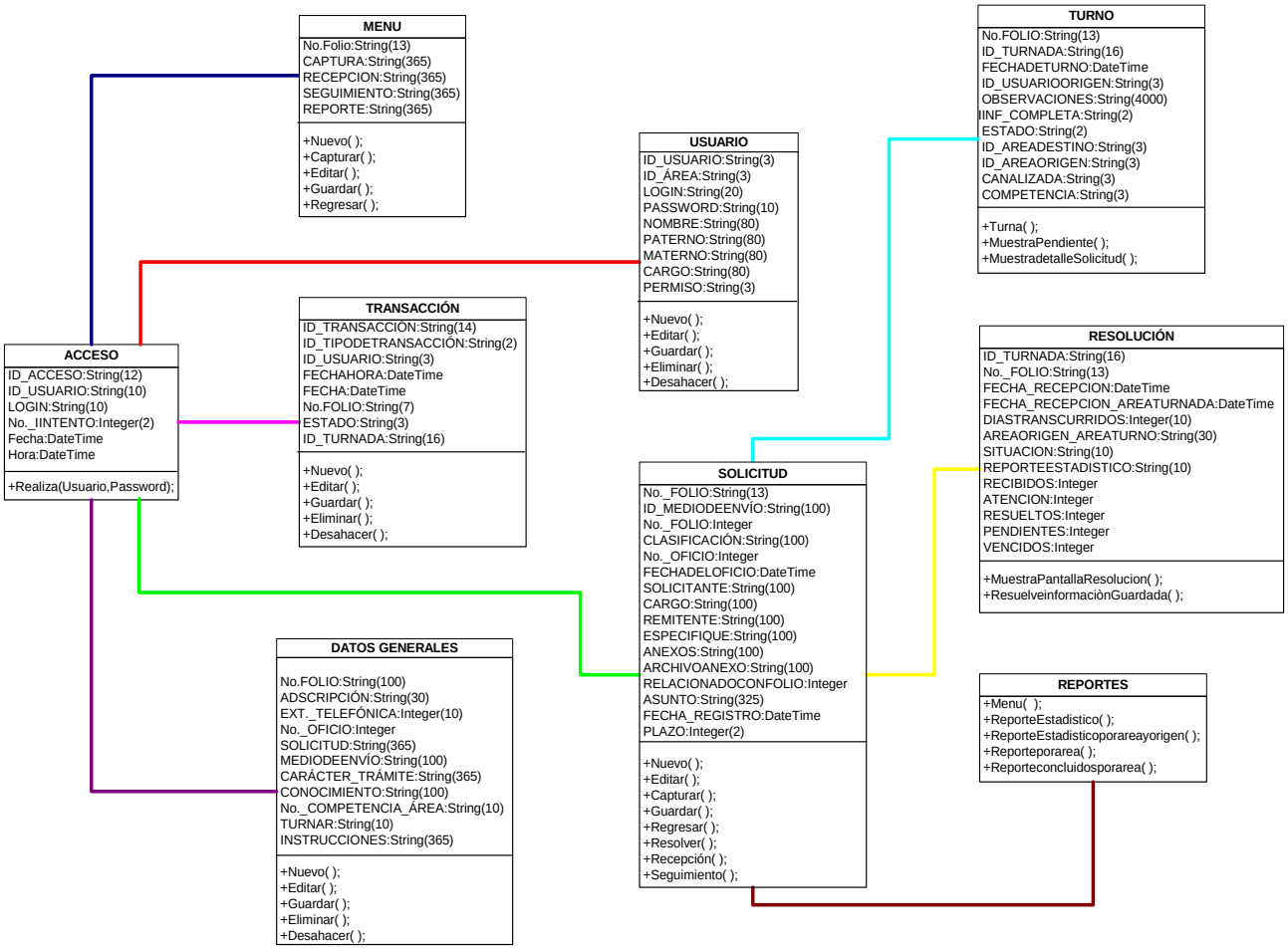


Fig. 19 Diagrama de Clases del Sistema de Control de Gestión V 2.0

4.9.4 Subsistemas del Diseño

Los métodos pueden ser utilizados durante el diseño para especificar como se deben de realizar las operaciones. El método puede ser especificado utilizado lenguaje natural o seudo código si se considera mas apropiado.

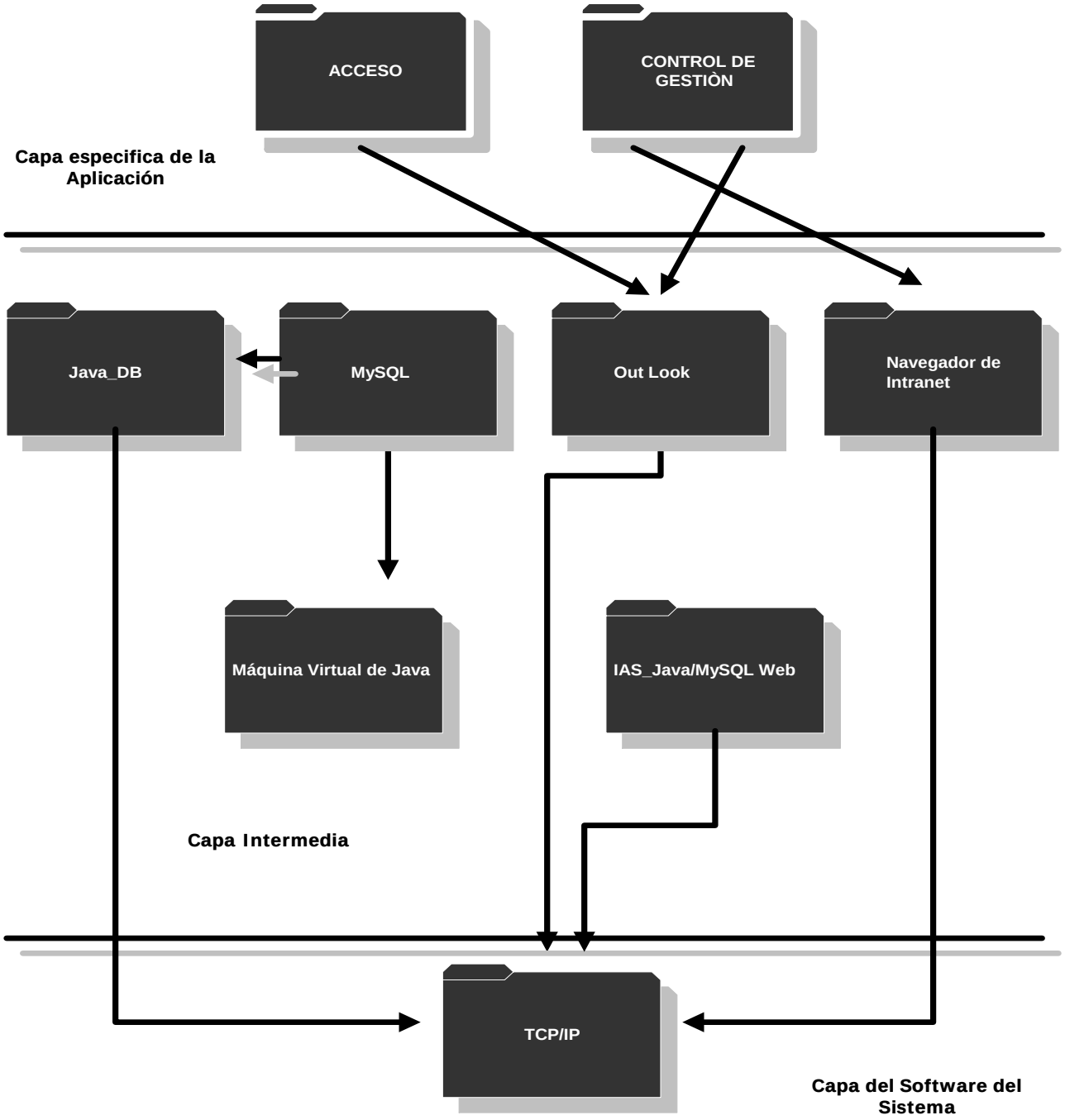


Fig. 20 Capas del sistema de Control de gestión V 2.0

4.9.5 Modelo de Entidad Relación

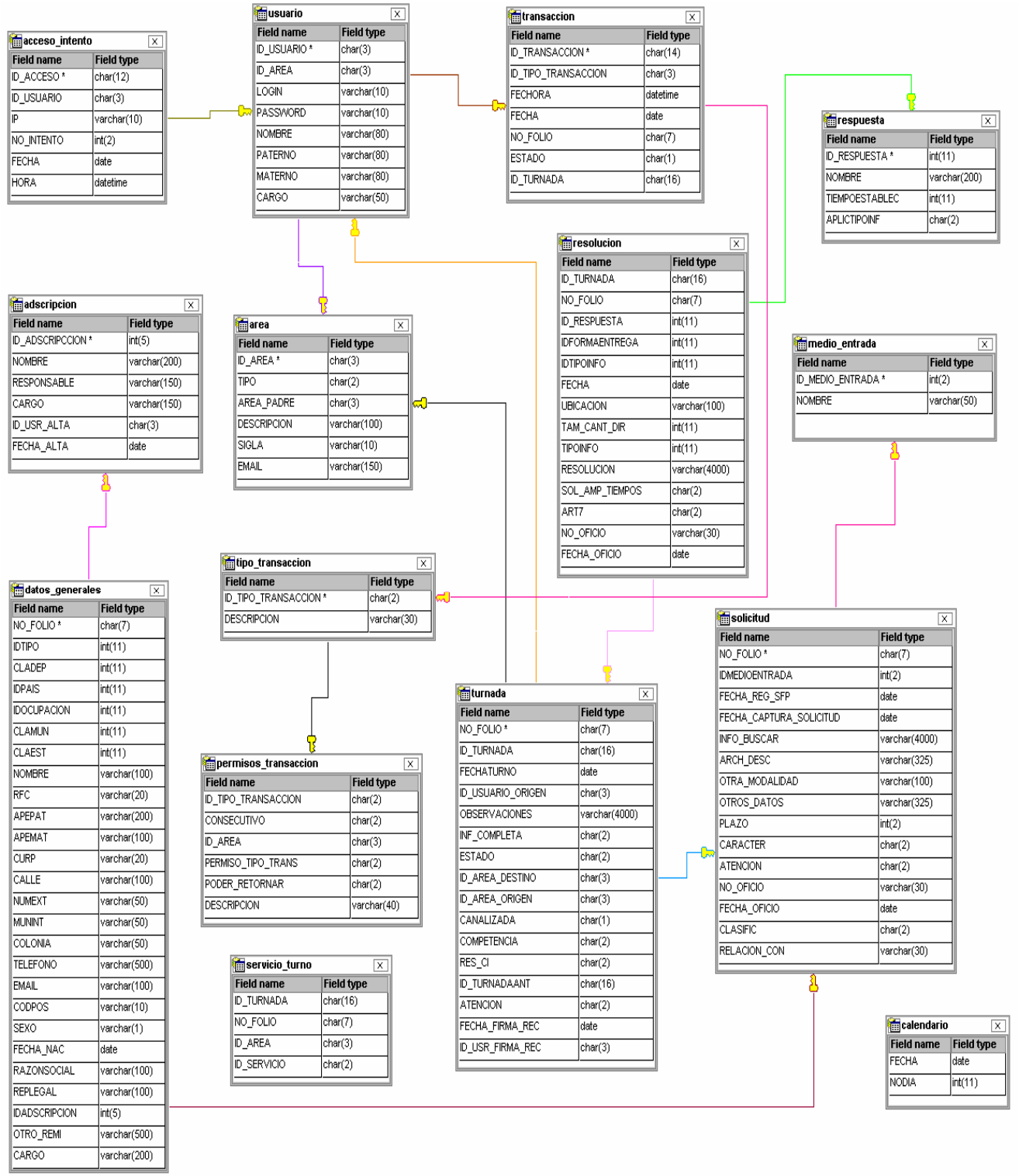


Fig. 21 Modelo de Entidad / Relación del Sistema de Control de Gestión V 2.0

4.10 Implementación

4.10.1 Introducción

La implementación es el resultado del diseño de un sistema en términos de componentes, en pocas palabras en ficheros de código fuente. La principal tarea de la implementación es el establecer el desarrollo de la arquitectura del sistema forma parte de la columna vertebral del sistema, cada vértebra es la planificación, la iteración, la distribución, la implementación de clases y subsistemas definidos en el sistema. La implementación como se menciono anteriormente es el centro de la construcción que crea la línea base ejecutable de la arquitectura, la descripción la organización de los componentes deacuerdo a los mecanismos de estructuración y modularización; así facilita la selección de el lenguaje ó lenguajes a utilizar dependiendo de cada uno de los mismos. La utilización del modelo de implementación es simplemente la representación jerárquica de subsistemas que contiene componentes e interfaces, es decir simple un componente es un empaquetamiento físico de los elementos de librerías, ficheros, tablas y documentos que serán ejecutados para la representación del sistema. Los subsistemas de implementación proporcionan una forma de organización de los artefactos en trozos manejables, un subsistema puede estar conformado por componentes, interfaces he incluso por otros subsistemas. El trabajador durante esta fase es el responsable de la integridad del modelo de implementación y asegura que el modelo como un todo es correcto, completo y legible, como en el análisis y diseño. El modelo es correcto cuando implementa la funcionalidad descrita en el modelo de diseño y en los requisitos adicionales o relevantes para la funcionalidad.

Es el resultado principal de la implementación es el modelo, el cual incluye los siguientes elementos:

- ♣ Subsistemas de implementación y sus dependencias, interfaces y contenidos.
- ♣ Componentes, incluyendo componentes fichero y ejecutable, y las dependencias entre ellos, los componentes son sometidos a prueba de unidad.
- ♣ La vista de la arquitectura del modelo de implementación, incluyendo sus elementos arquitectónicamente significativos.

La implementación también produce como resultado un refinamiento de la vista de la arquitectura del modelo de despliegue, donde los componentes ejecutables son asignados a nodos.

Modelo de Implementación

- Componentes ejecutables con interfaces

Subsistema 1

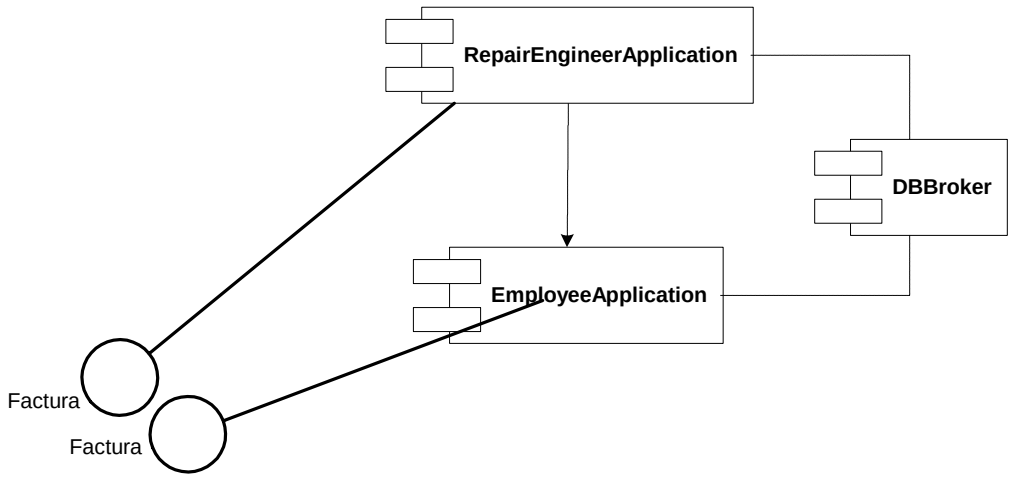


Fig. 22 Modelo de Implementación

4.10.2 Modelo de Implementación del SCG V 2.0

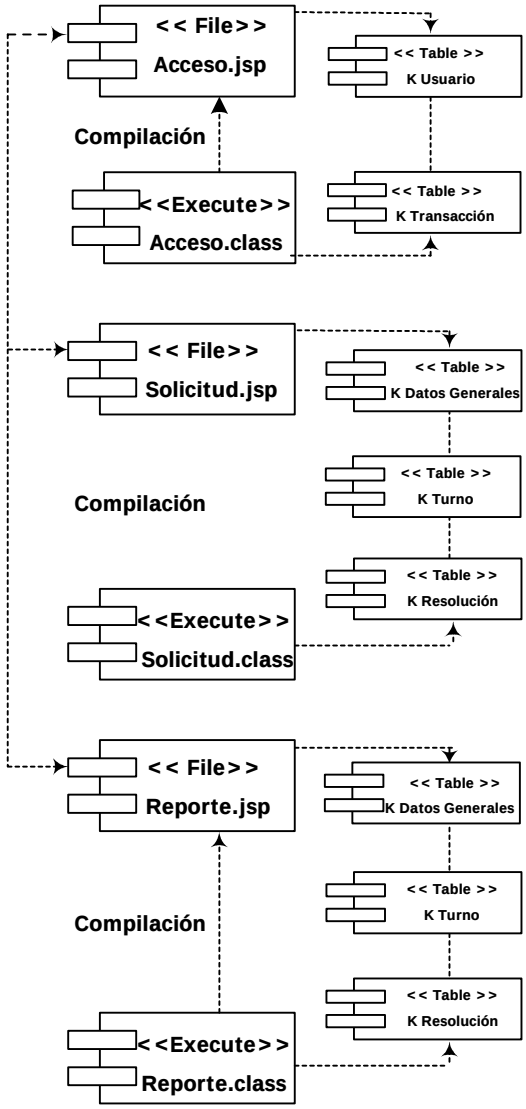


Fig. 23 Modelo de Implementación del Sistema de Control de Gestión V 2.0

4.10.3 Implementación Simplificada

Pantalla de Acceso.jsp

```
String=a=getParameter("Tipo");
        if(a==" ")//Muestra pantalla de acceso
{
        <html>
        <!-- interfaz de acceso(usuario,contraseña,"correcto")
           <form action="Acceso.jsp?Tipo=1"
        -->
        </html>
}else if(a=="1"){//Si tipo=1 Revisa información correcta
        int real=Realiza(Usuario,Password);
           if(real==1){//Si información es correcta entonces muestra menú
        <jsp:forward page="Menu.jsp">
}else {//si la información es incorrecta muestra error y captura nuevamente información
        <html>
        <!-- interfaz de acceso(usuario,contraseña,"incorrecto")
           <form action="acceso.jsp?Tipo=1"
        -->
</html>
           }
}
```

4.10.4 Implementación Simplificada

Pantalla de Menú.jsp

```
String b=getParameter("Tipo");
        if (b==" ")//Muestra pantalla de menú
{
        <html>
<!-- interfaz de menú(frames)
        Encabezado.jsp
        Opciones.jsp
        Contenido.jsp(blanco)
-->
        </html>
}else if(b=="1"){//Si tipo=1, Interfaz Captura
        Capturar( );
        }
}else if(b=="2"){//Si tipo= 2, Recepción
        Recepcion( );
        }
}else if(b=="3"){//Si tipo= 3, Seguimiento
        Seguimiento( );
        }
}else if(b=="4"){//Si tipo= 4, Reportes
        Reporte.jsp;
}
```

4.10.5 Implementación Simplificada

Pantalla de Reportes.jsp

```
1. String=e=getParameter("Topo");
2. if(e==" ")//Muestra pantalla de reportes
3.       {
4.       <html>
5. <!--interfaz de reporte(ReporteEstadistico, ReporteEstadisticoporAreayOrigen, ReporteConcluidoporArea)
6. <form action=Reporte.jsp?Tipo="1"
7. -->
8.       </html>
9. }else if=(e=="1"){//Si tipo=1 Manda pantalla de Reportes para la selección de la fecha solicitada
10.      int real=(ReporteEstadistico, ReporteEstadisticoporAreayOrigen, ReporteConcluidoporArea);
11.      if e=2{web:Reporte_estadistico}
12.      if e=3{web:Reporte por área}
13.      if e=4{web:Reportependientes por área}
14.      if e=5{web:Reporteconcluidos por área}
15.               <jsp:forward page="Reporte.jsp">
16.               }else
17.      <html>
18.      <!-- interfaz de reporte
19.      <form action=Reporte.jsp?Tipo="1,2,3,4"
20.      -->
21.      </html>
22.               }
```

Conclusiones

Se hace incapié en esta investigación que toda organización gubernamental, privada ó civil tendrá como necesidad el recibir y dar a conocer información y comunicación que brinda la posibilidad de realizar estas tareas con eficiencia atendiendo las necesidades de información.

Teniendo por lo tanto la visión integral sobre el mercado y su futuro alcanzable de cambio organizado. Analizando pero sobre todo comunicar de manera fluida y adecuada la creación de nuevos sistemas.

Por lo que la Subsecretaría de Atención Ciudadana y Normatividad de la Secretaría de la Función Pública tuvo la necesidad de crear un nuevo sistema planteando del porque la utilización y la importancia que se tiene.

Se plantea y se diseña la objetividad del nuevo diseño tales como una Metodología denominada como Lenguaje de Modelado Unificado (UML); que permite la especificación conceptual de elementos concretos como clases ó diseño de bases de datos; así como el diseño de interfases Web del Sistema de Control de Gestión V 2.0, que tiene como respuesta el cubrir la necesidad de agilizar el control de los turnos de la información en los documentos enviados a la Secretaría Técnica, la cual se encargara de resolver ó turnar entre las diferentes áreas de la Subsecretaría de Atención Ciudadana y Normatividad identificando el tiempo que ha transcurrido en cada una de ellas, así como la consulta del trámite y la resolución que le ha dado a la misma.

Es cierto iniciar un software no es nada sencillo pero buscamos un marco para analizar un proyecto teniendo presente que cada sistema cuenta con un ciclo de vida, una arquitectura, lenguaje de programación, un sistema de administrador de base de datos, fases de inicio, elaboración, construcción y transición.

Por lo cual únicamente se tiene que tener en claro que el Licenciado de Informático es capaz de crear y utilizar tecnologías de información y comunicación para resolver los problemas de información en las organizaciones.

Vocabulario

-A-

Abstracción: Características esenciales de una entidad que la distinguen de cualquier otra clase de entidades. Una abstracción define un contorno relativo a la perspectiva del observador separando las características que interesan para un modelo de las irrelevantes.

Acoplamiento: Medida de la complejidad de las interfaces de los módulos.

Actividad: Grafico UML que muestra el comportamiento y la participación de las clases en el mismo, útil para flujo de trabajo. También puede verse como la ejecución de una operación por un trabajador.

Actor: Puede ser determinado como persona o sistema que juega un papel en la interacción con un sistema, manejado también como el estereotipo predefinido que denota una entidad externa al sistema que interactúa con casos de uso.

Ada: Un lenguaje de programación de alta nivel concurrente desarrollado por el departamento de defensa de los Estados Unidos.

Administrador de proyecto: Consiste en estructurar el planteamiento, la implementación, el seguimiento de todos los elementos de una solución de sistemas, la iniciación y monitoreo de acciones correctivas.

Agregación: Relación entre dos clases en la que es parte de la otra.

Ámbito: Número de veces que existe el objeto. De instancia una por cada instancia de la clase, archivador una única vez, como las variables static de java.

Análisis: Etapa donde el problema es la comprensión de las necesidades, en donde la parte del proceso de desarrollo de software cuyo propósito principal es realizar un modelo del dominio del problema, haciendo enfoque en que hacer, el diseño y como hacerlo.

Análisis crítico: Actividad que se realiza desde el principio del proyecto para localizar los problemas de alto riesgo.

Arquitectura: Consiste en la estructura organizacional de un sistema. Una arquitectura puede ser descompuesta recurvisamente en partes que interactúan entre si por medio de interfaces, relaciones que conectan las partes y restricciones para ensamblar las partes.

Arquitectura de Referencia: Es un diagrama que permite separar los distintos componentes de una solución de datawarehousing e integrar en una clasificación común los distintos tipos de información necesarios para construir un datawarehousing.

Artefacto: Es la información que es usada o producida mediante un proceso de desarrollo de software, un artefacto puede ser un modelo, una descripción o un software.

Asociación: Una asociación es una relación estructural entre varios elementos. Una relación de asociación implica que los objetos de los distintos elementos de la relación están conectados entre sí y se pueden comunicar. Una relación de asociación se representa gráficamente con una línea continua entre los elementos relacionados.

Atributo: Es la parte especifica de una clase, una propiedad de un tipo identificada mediante un nombre, variable de una clase.

-B-

Base de Datos: Lugar donde se almacena información computacional (Archivo especialmente estructurado para su actualización y consulta electrónica). Permite buscar, ordenar y realizar múltiples operaciones sobre dichos datos.

Body: Define el cuerpo del programa.

Bloques de construcción: Elementos: Son los modelos UML (clases, casos de uso, estados, anotaciones...), Relaciones: Ligan elementos entre sí, establecen la forma en que interactúan y Diagramas: Representación gráfica de un grupo de elementos y sus relaciones.

-C-

Caja blanca: Pruebas que tienen en cuenta la estructura lógica del programa.

Caja negra: Pruebas que tienen en cuenta las especificaciones.

Capa: Una forma especifica de agrupar paquetes en un modelo al mismo nivel de abstracción.

Capa del software del sistema: Capa que contiene el software para la infraestructura de computación y comunicación, se trata de la capa inferior de la jerarquía de capas.

Capa específica de aplicación: Paquetes o subsistemas de un sistema que es específica de la aplicación y no es compartida, esta capa utiliza la capa general de aplicación.

Capa general de aplicación: Paquetes o subsistemas de un sistema que puede ser reutilizada dentro de un negocio o dominio. Esta capa es utilizada la capa es utilizada por la capa especifica de aplicación.

Capa intermedia: Capa que ofrece bloques de construcción reutilizables ya sean paquetes ó subsistemas a marcos de trabajo y servicios independientes de la plataforma.

Cascada: Ciclo de vida lineal.

Case (Computer Aided Software Engineering): Herramienta para automatizar ó facilitar parte ó todo el trabajo mecánico (no creativo) de parte o todo el ciclo de vida.

Caso de uso: Método de análisis para la interacción del sistema con el usuario u otro sistema. En aquello que describe la interacción de los actores con el sistema para lograr un objetivo.

Casos de prueba: Conjunto de entradas y salidas esperadas para esas entradas.

Ciclo de vida: Conjunto de etapas por las que pasa un sistema informático desde su concepción hasta su retirada.

CGI ("Common Gateway Interface"): Interfaz estándar de programas instalados en un servidor de web para ser ejecutados por el servidor y exhibidos con la apariencia de una página web, (Fue el primero pero ya no es el único sistema de programación que cumple esta función).

Carácter: Número, letra o símbolo (introducido a través de un teclado).

Clase: Unidad fundamental de programación en Java, que sirve como plantilla para la creación de objetos, una clase define datos, métodos y es la unidad de organización básica de un programa java.

Clases de objeto: Es aquello que sirve para crear objetos. Una clase es una implementación de un tipo.

Clases, Diagrama: Principal tipo de diagrama en UML que muestre las clases y las relaciones que se tienen.

Cliente: Un clasificador que solicita servicio de otro clasificador.

Cliente / Servidor: Arquitectura de sistemas distribuidos con una parte que funciona como interfaz (cliente) y otra que gestiona recursos y realiza el trabajo (servidor).

Codificación: Etapa del ciclo de vida en el que se escribe el código fuente.

Colaboración, Diagrama: Indica interacciones entre objetos a través de los enlaces que hay entre ellos. Similar al diagrama de secuencia.

Compilador: Programa de software que traduce código fuente en el lenguaje de programación legible por una persona a código máquina interpretable por un ordenador.

Componentes: Software pensado para ser reutilizable, consta de código y datos.

Composición: Es una forma de agregación más fuerte, con una mayor posesión y coincidencia de ciclo de vida entre el todo y las partes. Las partes que no tienen multiplicidad fija pueden ser creadas después del compuesto en sí, pero una vez creadas viven y mueren con él (i.e., comparten su ciclo de vida). Estas partes también pueden ser removidas antes de la muerte del compuesto. La composición puede ser recursiva.

Congruencia: Un método es congruente si tiene el nombre similar a métodos similares, condiciones, orden de argumentos, valor proporcionado y condiciones de error.

Constante: Valor utilizado en el programa de computadoras con una garantía de no cambiar en tiempo de ejecución. La garantía es a menudo reforzada por el compilador. En Java las constantes se declaran como static final.

Construir: Las anteriores características permiten que UML pueda generar código en distintos lenguajes de programación y tablas en una base de datos a partir de modelos UML. Además permite simular el comportamiento de sistemas software.

Correo Electrónico: Sistema de envío y recepción de correspondencia privada (Todos los usuarios tienen una clave de acceso -password- que asegura la confidencialidad). A los mensajes escritos y

enviados con la aplicación correspondiente es posible agregar archivos de otros tipos, que son transmitidos en formatos diferentes.

Crisis de software: La consecuencia de escribir código sin plantear seriamente metodologías para el análisis, diseño y mantenimiento.

-CH-

Char: Determinado como character, carácter es cualquier valor que pueda representarse en el alfabeto de la computadora.

-D-

Datawarehousing: Es un conjunto de datos integrados orientados a una materia, que varían con el tiempo y que no son transitorios, los cuales soportan el proceso de toma de decisiones de una administración.

Dependencia: Una dependencia es una relación de uso entre dos elementos (un elemento utiliza a otro). Una relación de dependencia entre dos elementos implica que los cambios que se produzcan en un elemento pueden afectar al otro pero no necesariamente a la inversa. Las dependencias se representan con una línea dirigida discontinua.

Diagrama de Actividades: Es un caso especial de diagrama de estados en el que todos, o la mayoría, son estados activos y en el que todas, o la mayoría, de las transiciones son disparadas por la finalización de las acciones de los estados.

Diagrama de Casos de Uso: Un diagrama de casos de uso es un diagrama que muestra un conjunto de casos de uso con sus relaciones y los actores implicados. Es un diagrama que sirve para modelar la vista estática de un programa. La vista estática nos permite visualizar el comportamiento externo del programa; de esta forma conseguimos conocer qué es lo que debe hacer el programa independientemente de cómo lo haga y sabremos los elementos que interactúan con el sistema. Los elementos implicados en un diagrama de casos de uso son los casos de uso, las relaciones y los actores. Las relaciones y los casos de uso ya han sido explicados anteriormente y el papel del actor también ha sido comentado pero merece la pena detallarlo más: Un actor es un rol que interactúa con el sistema. Lo definimos como rol porque un actor puede ser tanto un usuario de la aplicación como otro sistema o dispositivos externos.

Diagrama de Clases: Un diagrama de clases es un diagrama que muestra un conjunto de clases y sus colaboraciones y relaciones. Estos diagramas sirven para visualizar las relaciones existentes entre las distintas clases y la forma en que colaboran unas con otras. Las relaciones entre las distintas clases son las relaciones comunes existentes en UML aunque con matices: Una asociación se traduce como que desde los objetos de una clase se puede acceder a los objetos de otra. Una dependencia se puede visualizar como que la clase utilizada es un parámetro de un método de la clase que la utiliza. Una generalización se traduce como una herencia entre clases.

Diagrama de Colaboración: Es un diagrama que muestra interacciones entre objetos organizadas alrededor de los objetos y sus vinculaciones. A diferencia de un diagrama de secuencias, un diagrama de colaboraciones muestra las relaciones entre los objetos. Los diagramas de secuencias y los diagramas de colaboraciones expresan información similar, pero en una forma diferente.

Diagrama de Componentes: El diagrama de componentes ilustra los componentes del software que serán usados para construir el sistema. Estos pueden ser construidos para el modelo de clase y escritos para satisfacer los requisitos del nuevo sistema, o puede ser dada para otros proyectos o vendedores de tercera persona.

Diagrama de Entidad / Relación: Es una descripción conceptual de las estructuras de datos y sus relaciones.

Diagrama de Estado: El Diagrama de Estado de la Máquina captura los ciclos de vida de los objetos, subsistemas y sistemas. Ellos indican qué estado de un objeto puede tener y qué eventos diferentes afectan aquellos estados fuera de tiempo. Éste diagrama podría ser adherido a clases que tienen claramente estados identificables y es gobernado por un comportamiento complejo.

Diagrama de Flujo de Datos: Es una descripción informal del sistema de información.

Diagrama de Interacción: Es un término genérico que se aplica a diversos tipos de diagramas que enfatizan la interacción entre objetos. Incluye: diagrama de colaboraciones, diagrama de secuencias, diagrama de actividades.

Diagrama de Objetos: Es un diagrama que contiene los objetos y sus relaciones en un momento dado del tiempo. Un diagrama de objetos puede ser considerado un caso especial de un diagrama de clases o de un diagrama de colaboraciones.

Diagrama de Secuencia: Un diagrama de secuencia es un diagrama de interacción UML. Estos diagramas muestran la secuencia de mensajes que se van lanzando los objetos implicados en una determinada operación del programa. Dentro del diagrama los objetos se alinean en el eje X respetando su orden de aparición. En el eje Y se van mostrando los mensajes que se envían, también respetando su orden temporal. Cada objeto tiene una línea de vida donde se sitúa su foco de control. El foco de control es un rectángulo que representa el tiempo durante el que un objeto está activo ejecutando una acción. Con este sencillo esquema podemos visualizar la comunicación y sincronización bajo un estricto orden temporal de los objetos implicados en las distintas funcionalidades de un sistema.

Diccionario de Datos: Descripción detallada de los flujos de datos.

Diseño: Es la parte del proceso de desarrollo de software cuyo propósito principal es decidir cómo se construirá el sistema. Durante el diseño se toman decisiones estratégicas y tácticas para alcanzar los requerimientos funcionales y la calidad esperada.

Documentar: Como ya se comentó antes, UML permite especificar los procesos de análisis, diseño y codificación y también permite documentar los mismos, dejando clara la arquitectura del sistema.

Dominio: Es un área de conocimiento o actividad caracterizada por un conjunto de conceptos y términos comprendidos por los practicantes de esa área.

E-

Eficiencia: Es la cantidad de recursos de computadoras y de código requeridos por un programa para llevar a cabo sus funciones.

Eficiencia de Ejecución: Consiste en el rendimiento en tiempo de ejecución de un programa.

Elemento: Es un elemento atómico constitutivo de un modelo.

Encapsulamiento: Propiedad por la que un objeto o módulo proporciona al exterior las funciones necesarias para su manejo y no los detalles internos.

Encapsular: Consiste en crear objetos que funcionan como unidades completas con características y métodos (código y datos) en el mismo tipo de objeto.

Entero: Un número entero, sin parte decimal, positivo o negativo.

Entidad / Relación: Modelo de datos del diseño estructurado.

Entrevistas: Técnica de obtención de requisitos que consiste en hablar con un usuario.

Error: Equivocación de un desarrollador en cualquiera de las fases. Produce uno ó más defectos.

Escenario: Es una instancia de un Caso de Uso.

Especificación: Traducción de los requisitos del análisis a un documento que sirva para empezar el diseño.

Especificación de Procesos: Concretar en pseudo código o en algún lenguaje de programación cómo es un módulo que ya no se puede descomponer más.

Especificar: UML permite especificar los procesos de análisis, diseño y codificación de un sistema software. También permite determinar modelos precisos, sin ambigüedades, detallando las partes esenciales de los mismos.

Estereotipo: Es un nuevo tipo de elemento de modelado, que extiende la semántica del metamodelo. Los estereotipos deben basarse en ciertos tipos o clases existentes en el metamodelo. Pueden extender la semántica, pero no la estructura de los tipos o clases preexistentes. Algunos estereotipos están predefinidos en el UML, otros pueden ser definidos por el usuario. Los estereotipos son uno de los mecanismos de extensión del UML.

-F-

Factor de calidad: Forma de determinar la calidad de un software. Están divididos en función de criterios de operación, revisión y transición. Cada uno se obtiene a partir de varias métricas.

Fallo: Manifestación de un defecto.

Fase de análisis: Una fase en el ciclo de vida del sistema de software que define los requerimientos que especifiquen lo que el sistema propuesto va a lograr.

Fase de diseño: Una fase en el ciclo de vida del sistema de software que define como lograra el sistema definir la fase de análisis.

Fase de Implementación: Una fase en el ciclo de vida del sistema de software en el cual se crean los programas reales.

Fase de pruebas: Una fase del ciclo de vida de un sistema de software en el cual realiza experimentos para probar que un paquete de software puede funcionar.

Flujo de control: Diagrama que informa sobre cuando y en que orden ocurren los sucesos.

Flujo de datos: Representación de datos que se mueven entre procesos o entidades externas.

-G-

Generalización: Una generalización es una relación de especialización. Los elementos especializados (hijos) son elementos que derivan de un elemento general (padre). Los elementos hijos mantienen la estructura y el funcionamiento del elemento padre pero de una forma más especializada. Su representación gráfica es la de una línea dirigida con punta triangular.

GUI (Graphical User Inteface): Interfaz gráfica de usuario.

-H-

Head: Define el encabezado del programa.

Herencia: Mecanismo por el que unas clases mantienen los mismos métodos y propiedades de otras a las cuales amplian.

HTML (HyperText Markup Languaje): Lenguaje que se utiliza para crear páginas Web. Los programas de navegación de la Web muestran estas páginas de acuerdo con un esquema de representación definido por el programa de navegación.

-I-

I- CASE: Es un CASE Integrado que abarca distintas etapas del ciclo de vida de desarrollo del sistema.

Implementación: Es la definición de cómo está construido o compuesto algo. Por ejemplo: una clase es una implementación de un tipo, un método es una implementación de una operación.

Ingeniería de Software: Es una disciplina para el desarrollo de software de alta calidad para sistemas basados en computadora.

Instancia: Es un objeto individual de una clase. Un individuo descrito por una clase o un tipo. Nota de uso: De acuerdo con la interpretación estricta del metamodelo un individuo de un tipo es una instancia y un individuo de una clase es un objeto. Es aceptable, en un contexto menos formal, referirse a un individuo de una clase como un objeto o una instancia.

Int: Conocido también como Integer que significa entero, y el entero es un número sin ninguna parte fraccionaria.

Interacción: Es una especificación de comportamiento cuyo fin es lograr un propósito específico. Abarca un conjunto de intercambios de mensajes entre un conjunto de objetos dentro de un contexto particular. Una interacción puede ilustrarse mediante uno ó más escenarios.

Interfaz: Mecanismo Java para decirle al compilador que un conjunto de métodos serán definidos en futuras clases. (Esas clases estarán definidas para implementar la interfaz).

Interfaz lógica: Presentación las partes lógicas de las interfaces.

Interfaz física: Presentación física de la interfaz lógica.

Iteración: Conjunto de actividades llevadas acabo deacuerdo al plan y unos criterios evolutivos que lleva a producir una versión, ya sea interna o externa.

-J-

Java: Lenguaje de programación que permite utilizar aplicaciones en computadores con diversos sistemas operativos. Especialmente utilizado en combinación con servidores y browsers de World Wide Web.

JSP (Java Server Pages): Programa cuyo contenido es facilitar la labor del programador java y la labor del diseñador web.

JVM (Java Virtual Machine): El intérprete de Java que ejecuta los códigos de byte en una plataforma particular.

-L-

Lenguaje Estructurado: Pseudocódigo para escribir las especificaciones de proceso.

Línea base: Puntos de referencia en la gestión de configuración.

-M-

Mantenimiento: Etapa final del ciclo de vida. Consume la mayor parte de los recursos.

Mensajes: Es el pedido que se le hace al objeto para que ejecute determinadas operaciones. Es una comunicación entre objetos que transmite información con la expectativa de desatar una acción. La recepción de un mensaje es, normalmente, considerada un evento.

Metaclase: Es una clase cuyas instancias son clases. Las metaclases son típicamente utilizadas para construir metamodelos.

Metadatos: Son definiciones acerca de datos.

Meta – metamodelos: Es un modelo que define el lenguaje para expresar el metamodelo. La relación entre meta-metamodelo y metamodelo es análoga a la relación entre metamodelo y modelo.

Metamodelo: Es un modelo que define el lenguaje para poder expresar un modelo.

Metaobjeto: Es un término genérico que se refiere a todas las metaentidades que componen un lenguaje de metamodelado. Por ejemplo: metatipos, metaclases, metaatributos y metaasociaciones.

Método: Es un procedimiento o función asociada a un tipo de objeto declarado dentro de un objeto. Es la implementación del mensaje. La implementación de una operación. El algoritmo o procedimiento que permite llegar al resultado de una operación.

Metodología: Modo sistemático de fabricar un producto.

Modelo: En diseño orientado a objetos, una representación del mundo real en unas abstracciones de software denominadas clases y la relación entre ellas.

Modelo de Casos de Uso: Es un modelo que describe los requerimientos funcionales de un sistema en términos de casos de uso.

Modelo de Dominio: El modelo de Dominio se realiza habitualmente en reuniones organizadas por los analistas de lo que genera la documentación de resultados, el objetivo principal del modelo de dominio es comprender y describir las clases más importantes dentro de los sistemas.

Modelo de Negocios: El modelo de Negocios es el identificar los casos de uso del software y las entidades relevantes que el software debe soportar de forma que podríamos modelar solo lo necesario para comprender el modelo de casos de uso y el modelo de objetos.

Modularidad: Propiedad del software. Un programa es modular si está dividido en partes con tareas definidas.

Módulo: Son los objetos más las operaciones. Es una unidad de manipulación y almacenamiento de un software. Incluyen: módulos de código fuente, módulos de código binario, módulos de código ejecutable.

Multiplicidad: Es una especificación del rango permitido de cardinalidades que puede asumir un conjunto. Es posible especificar la multiplicidad de roles de una asociación, partes de un compuesto, repeticiones, y otros. Una multiplicidad es, esencialmente, un subconjunto (posiblemente infinito) de enteros no negativos.

-N-

Navegador: Software que permite al usuario conectarse a un servidor de Web utilizando Hypertext Transfer Protocol (HTTP). Microsoft Internet Explorer, Netscape Navigator, HotJava de Sun, son populares navegadores de Web.

Notación: Consiste en una serie de diagramas, y en los elementos visuales y textuales que los componen.

Notas: Explicaciones de los elementos de notación de UML.

Null: Valor de Java que significa vacío.

-O-

Objeto: Es un componente del mundo real que tiene una cierta estructura interna y un determinado comportamiento. Una entidad delimitada precisamente y con identidad, que encapsula estado y comportamiento. El estado es representado por sus atributos y relaciones, el comportamiento es representado por sus operaciones y métodos. Un objeto es una instancia de una clase.

-P-

Packages: Es un grupo de elementos del modelo.

Paquete: Serie digital de caracteres, de tamaño variable, que se envía a través de Internet y a la cual son asociados datos acerca de su integridad (dígitos verificadores), de su origen y su destino, de tal modo que llegue adecuadamente al destinatario (Se repite en caso de no llegar íntegramente).

Parámetros: Valores u objetos pasados entre una subrutina y la rutina de llamada.

Password: Contraseña o clave de acceso: palabra o número secreto que permite acceso confidencial a una máquina, una red, un servicio o un conjunto de datos.

Personas: Las personas son realmente los que se encuentran implicados en el desarrollo de un software durante todo un ciclo de vida, financian el producto, lo planifican, lo desarrollan, lo gestionan, lo prueban, lo utilizan y se benefician con el.

Proceso: Instancia de un programa ejecutable. Por ejemplo, si inicia dos copias de un intérprete de Java, tiene dos procesos de la máquina virtual de Java ejecutándose en su computadora.

Portabilidad: Es el esfuerzo requerido para transferir el programa desde un hardware y/o un entorno de sistema de software a otro.

Proceso de Desarrollo: Es un conjunto parcialmente ordenado de pasos, realizados durante el desarrollo de software con el fin de lograr un objetivo dado; por ejemplo, construir modelos o implementar modelos.

Producto: Son los artefactos del desarrollo de software. Por ejemplo: modelos, código, documentación, planes de trabajo.

Programa: Lugar donde se almacena información computacional (Archivo especialmente estructurado para su actualización y consulta electrónica). Permite buscar, ordenar y realizar múltiples operaciones sobre dichos datos.

Prototipo: Son aquellos que permiten diseñar con una adecuada definición lo que ve el usuario, cómo interpreta la interfase con el sistema y qué espera de él a nivel de información.

Proyecto: Un proyecto de desarrollo da como resultado una nueva versión de un producto. El primer proyecto dentro del ciclo de vida de desarrollo a veces llamado proyecto inicial desarrolla y obtiene el sistema o producto inicial.

Proyecto Informático: Conjunto ordenado de tareas realizado por recursos humanos con responsabilidad utilizando recursos técnicos entendiendo su complejidad, que permiten construir un producto de software, que cubre el logro de algún objetivo u objetivos claramente predeterminados por alguien.

Pruebas: Batería de test que el sistema tiene que superar para ser considerado operativo.

Pseudoestado: Es un vértice dentro de una máquina de estados que tiene la forma de un estado pero que no se comporta como tal. Los pseudoestado incluyen el inicial, el final y la historia.

-R-

Referencia: Significa que la arquitectura es independiente de un distribuidor y caracteriza la naturaleza genérica de todos los datawarehousing.

Refinamiento: Es una relación que representa la especificación completa de algo que ya ha sido especificado con un cierto nivel de detalle. Por ejemplo, una clase del diseño es un refinamiento de una clase del análisis.

Reglas: Los bloques de construcción se deben combinar siguiendo las normas que establece UML. UML establece una serie de normas sobre cómo nombrar a los elementos, relaciones y diagramas; la visibilidad y alcance de dichos nombres y sobre su integridad (cómo diseñar relaciones consistentes). Podremos decir que un modelo está bien formado cuando cumpla estas reglas.

Relación: Es una conexión semántica entre elementos del modelo. La asociación y la generalización son ejemplos de relaciones.

Requisitos funcionales: Lo que el sistema hace para el usuario.

Requisitos no funcionales: Características del sistema (fiabilidad, mantenibilidad, etc.). Se tienen en cuenta en la fase de diseño.

Responsabilidad: Descripción de lo que hace una clase.

Restricción: Es una condición u obligación semántica. Algunas restricciones están predefinidas en el UML, otras pueden ser definidas por el usuario. Las restricciones son uno de los tres mecanismos de extensión del UML.

Reutilización: El factor de calidad que trata de facilitar con el cual el software puede utilizarse en otros programas.

Revisión: Las distintas versiones de algunos elementos de software que van surgiendo.

-S-

Seudocódigo: Documentación de diseño que describe el trabajo de un programa en inglés estructurado (o en otro lenguaje) en lugar de un lenguaje de computadora.

Servlets: Módulos que permiten sustituir o utilizar el lenguaje Java en lugar de programas CGI.

Sistema: Conjunto de elementos que cooperan entre si para proporcionar servicios.

SQL (Structured Query Language): Lenguaje para realizar consultas a Bases de Datos relacionales. Lenguaje creado por IBM para el manejo de bases de datos relacionales. Es usado por la mayoría de los programas comerciales de administración de bases de datos.

String: Objeto Java estandarizado en el lenguaje, que representa una cadena de carácteres.

Subclase: Clase descendiente de otra clase de la que hereda métodos y variables.

-T-

Tarea: Una etapa de diseño que se descompone en tareas. Una tarea puede realizarse a lo largo de varias etapas, por ejemplo la documentación.

TCP/IP: Protocolo de control de transferencia de datos establecido en 1982 como estándar de Internet.

Title: Define el título del programa.

Trabajador: Puesto que puede ser asignado a una persona o equipo y que requiere responsabilidades y habilidades como realizar determinadas actividades o desarrollar determinados artefactos.

Transición de datos: Es una relación entre dos estados que indica que un objeto que está en el primer estado realizará una acción especificada y entrará en el segundo estado cuando un evento especificado ocurra y unas condiciones especificadas sean satisfechas. En dicho cambio de estado se dice que la transición es 'disparada'.

-U-

UML ("Unified Modeling Language"): Lenguaje de modelamiento unificado: lenguaje de programación que facilita la reutilización de componentes de software. Facilita además la verificación del proceso de desarrollo desde el comienzo, para evitar la necesidad de corregir fallas después del término del proyecto.

Usuarios Ejecutivos: Son las personas que están concentradas en los usos estratégicos de la información y están interesados desde el punto de vista más global del sistema.

Usuarios Expertos: Son las personas que realmente entienden sobre los sistemas y tecnología.

Usuarios Intermedios: Son aquellos que ya se vieron involucrados en uno o más proyectos.

Usuarios Operacionales: Son las personas que tienen mayor contacto con el sistema.

Usuarios Supervisores: Son empleados con capacidad de supervisión.

-V-

Valor etiquetado: Permite añadir nuevas propiedades a un bloque de construcción.

Variable: Una variable en Java es un identificador que representa una palabra de memoria que contiene información.

Versión: Conjunto de artefactos relativamente completo y consistente que incluye posiblemente una construcción.

Versión externa: Versión expuesta a los clientes y usuarios, externos al proyecto y sus miembros.

Versión interna: Una versión no expuesta a los clientes y usuarios, sino de forma interna, al proyecto y sus miembros.
Vista: Proyección de un modelo, que es visto desde una perspectiva dada o un lugar estratégico y que omite las entidades que no son relevantes para esta perspectiva.

Visualizar: UML permite representar mediante su simbología el contenido y la estructura de un sistema software. La notación UML permite definir modelos que serán claramente comprensibles por otros desarrolladores facilitando así el mantenimiento del sistema que describe la visualización.

Bibliografía

1. El Proceso Unificado de Desarrollo de Software
Ivar Jacobson
Grady Booch
James Rumbaugh
RATIONAL SOFTWARE CORPORATION
Traducción:
Salvador Sánchez
Universidad Pontificia de Salamanca en Madrid
Miguel Ángel Sicilia
Universidad Pontificia de Salamanca en Madrid
Carlos Canal
Universidad de Málaga
Francisco Javier Durán
Universidad de Málaga
ADDISON WESLEY

2. Java Server Pages
Manual de Usuario y Tutorial
Agustín Froufe
Alfaomega Ra-Ma

3. Ingeniería del Software Orientada a Objetos
Bruegge, Bernd y Dutoit, Allen H.
Pearson Educación, México
Prentice Hall

4. Análisis y Diseño Orientado a Objetos
James Martin
James J. Odell
Traducción:
Oscar Alfredo Palmas Velasco
Facultad de Ciencias UNAM
Revisión Técnica:
Dr. Gabriel Guerrero
Doctor en Informática
Prentice Hall

5. Introducción a la Ciencia de la Computación
De la manipulación de datos a la teoría de la Computación
Behroz A. Forouzan
De Anza College
Con colaboración de Shopina Chung Fegan
Thomson

7. Análisis y Diseño de Aplicaciones Informáticas de Gestión
Una perspectiva de Ingeniería del Software
Mario G. Piattini Velthuis
José A. Calvo Manzano Villalón
Joaquín Cervera Bravo
Luís Fernández Sanz
AlfaOmega Ra- Ma

8. Ingeniería de Software
Teoría y Práctica
Sharí Lawrence Pfleeger
Revisión Técnica
Ing. Álvaro Ruiz de Mendarozqueta
Traducción
Ing. Elvira Quiroga
Universidad Tecnológica Nacional
Universidad Nacional de la Matanza
Prentice Hall
Pearson Educación

9. UML and the Unifed Process
Liliana Favre
Universidad Nacional del Centro de la Provincia de Buenos Aires
IRM Press

10. Tecnologías de la información y las comunicaciones
Claudio F. Freijedo, Alicia B. Cortagerena
Ediciones Macchi

11. Real – Time UML
Developing Efficient Objects
For Embedded System
Bruce Powel Douglass
Foreword by Proffessor David Harel
The Werzmann Institute of Science

12. El Lenguaje Unificado de Modelado
G. Booch, J. Rumbaugh, I. Jacobson.
Addison Wesley Iberoamericana

13. Object-Oriented Analysis and Design
G. Booch. Benjamin/Cummings

14. The UML Specification Document
G. Booch, I. Jacobson and J. Rumbaugh. Rational Software

15. Páginas Consultadas

www.rational.com.uml
www.agamenon.unidades.edu.co-pfigueroa/soo/uml
www.tutorialdeuml.com.mx