Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática
Escuela Académico Profesional de Ingeniería de Sistemas

Plan de desarrollo de software para la elaboración de un sistema de registro de exámenes ecográficos utilizando la metodología RUP: Caso Hospital San José

TESINA

Para optar el Título Profesional de Ingeniero de Sistemas

AUTOR

Elvis Joel CARRILLO MONTALVO

ASESOR

Pablo Jesús ROMERO NAUPARI

Lima, Perú

2008

Referencia bibliográfica

Carrillo, E. (2008). *Plan de desarrollo de software para la elaboración de un sistema de registro de exámenes ecográficos utilizando la metodología RUP: Caso Hospital San José*. Tesina para optar el título profesional de Ingeniero de Sistemas. Escuela Académico Profesional de Ingeniería de Sistemas, Facultad de Ingeniería de Sistemas e Informática, Universidad Nacional Mayor de San Marcos, Lima, Perú.

Este trabajo está dedicado a mi familia, quienes con su apoyo incondicional me dieron la fuerza necesaria para salir adelante.

AGRADECIMIENTOS

A mi asesor Pablo Romero Naupari, por su orientación, paciencia y dedicación para que este trabajo cumpla con los objetivos trazados.

A mis colegas y amigos de nuestra facultad por sus observaciones y porque en todo momento me incentivaron para que culmine este trabajo.

A aquellas personas que indirectamente me ayudaron para culminar este trabajo y que muchas veces constituyen un invalorable apoyo. Especialmente aquellas que el destino nos separo, pero que a la vez nos aferro a una promesa de culminar nuestras metas.

Y por encima de todo doy gracias a Dios.

Plan de Desarrollo de Software para la elaboración de un Sistema de Registro de Exámenes Ecográficos utilizando la metodología RUP: Caso Hospital San José

RESUMEN

Este proyecto se orienta hacia el desarrollo de software, utilizando una metodología y un lenguaje de modelado, el cual nos guiará por todo el camino del desarrollo antes de empezar la implementación del software, con lo cual se asegura la calidad del producto final y el cumplimiento en la entrega en el tiempo establecido.

Elegir tanto la metodología como la herramienta de implementación adecuada es de vital importancia para el éxito de la elaboración del software, es por ello que se compara 4 metodologías de desarrollo: RUP, XP, ICONIX, SCRUM, eligiendo entre ellas a la metodología RUP basada en el lenguaje de modelado UML

Este trabajo contemplara hasta la fase de elaboración del sistema de Información para el registro de exámenes del área de Diagnóstico por Imágenes del Hospital San Jose, con la finalidad de que sirva de guía para futuros trabajos sobre desarrollo de software. La herramienta CASE utilizada es el Rational Rose 2000, que hace posible elaborar las distintas fases del RUP.

Palabras claves: UML, RUP, sistema de información, metodología, software

Software Development Plan for the Development of a Registration System Examinations Ecography using the methodology RUP: San Jose Hospital Case

ABSTRACT

This project is geared toward software development, using a methodology and language modeling, which will guide us throughout the path of development before beginning the implementation of the software, thereby ensuring final product quality and compliance in delivery at the time.

Choosing both the methodology and the proper implementation tool is vital to the success of the development software, which is why compared 4 development methodologies: RUP, XP, ICONIX, scrum, including choosing the methodology based RUP in the language modeling UML

This work contemplated until the stage of development of information systems for registering examinations of the area Diagnostic Imaging Hospital San Jose, with the aim of serving as a guide for future work on software development. The tool is the case use Rational Rose 2000, which makes it possible to develop the various phases of the RUP.

Key words: UML, RUP, system of information, methodology, software.

INDICE

Lista de Figuras xii

Lista de Tablas xv

CAPITULO 1: INTRODUCCION……..... 1

1.1 Antecedentes……. 1

1.2 Definición del problema.....……... 2

1.3 Objetivos…….... 2

1.3.1 Objetivo Principal……..…….. 2

1.3.2 Objetivos Secundarios…….…… 2

1.4 Justificación ……... 3

1.5 Propuesta …… 3

1.6 Organización de la tesina…..…….…. 4

CAPITULO 2: MARCO TEORICO....…… 5

2.1 UML…… 5

2.1.1 Elementos del UML…….. 6

2.1.1.1 Elementos Estructurales……... 7

2.1.1.2 Elementos de Comportamiento…….…… 9

2.1.1.3 Elementos de Agrupación……..…… 10

2.1.1.4 Elementos de Anotación…….. 11

2.1.2 Relaciones entre elementos……..…….……. 12

2.1.2.1 Dependencia…….…… 12

2.1.2.2 Generalización…… ……..…… ... 13

2.1.2.3 Asociación…….……..…… ... …. 14

2.1.2.4 Realización……..... 16

2.1.3 Diagramas de Elementos…….. 17

2.1.3.1 Diagramas de Clases…… 17

2.1.3.2 Diagrama de Objetos…… 18

2.1.3.3 Diagrama de Interacción……. 19

2.1.3.4 Diagrama de Colaboración……..……... 19

2.1.3.5 Diagrama de Secuencia…….. 21

2.1.3.6 Diagrama de Casos de Uso……. 22

2.1.3.7 Diagrama de Estados…….. 23

2.1.3.8 Diagrama de Actividades…… 24

2.1.3.9 Diagrama de Componentes……. 25

2.1.3.10 Diagrama de Despliegue……..… 26

2.2 UML y su Relación con los procesos de desarrollo……. 27

2.3 Proceso Unificado de Desarrollo de Software……. 28

2.3.1 Proceso dirigido por caso de uso …….. 28

2.3.2 Proceso centrado en la arquitectura…… 29

2.3.3 Desarrollo iterativo e incremental…..…… 31

2.3.4 Estructura dinámica del proceso. Fases e iteraciones…… 33

2.3.4.1 Concepción……. 34

2.3.4.2 Elaboración……. 35

2.3.4.3 Construcción…… 36

2.3.4.4 Transición…… 37

2.3.4.5 Estructura Estática del proceso. …….. 38

2.3.4.5.1 Roles……. 39

2.3.4.5.2 Actividades…...…… 41

2.3.4.5.3 Artefactos……. 41

2.3.4.5.4 Flujos de trabajo……. 41

2.3.4.5.5 Modelado de negocio……41

2.3.4.5.6 Requisitos……. 42

2.3.4.5.7 Análisis y diseño ……. 42

2.3.4.5.8 Implementación……. 43

2.3.4.5.9 Pruebas…..…… 44

2.3.4.5.10 Despliegue……44

2.3.4.5.11 Gestión de proyecto…… 45

2.3.4.5.12 Configuración y control de cambios…..……. 45

2.3.4.5.13 Entorno…… 45

2.4 Herramienta Case……. 46
2.4.1 Concepto…… 46
2.4.2 Historia…….. 46
2.4.3 Tecnología CASE…… 47
2.4.4 Componentes…….. 47
2.4.5 Clasificación…….. 48
2.4.6 Estado Actual……. 48
2.4.7 Integración de las Herramientas CASE en el futuro…….. 49
2.4.8 Factores Asociados a la Implantación de las Herramientas CASE…….. 49
2.5 El Modelo Constructivo de Costo(COCOMO)……..... 49
CAPITULO 3: ESTADO DEL ARTE…….. 52
3.1 Introducción…….. 52
3.2 Metodologías y Propuestas……... 53
3.2.1 Rational Unified Process (RUP)……. 53
3.2.1.1 Fases……. 53
3.2.1.2 Ventajas……..... 54
3.2.1.3 Desventajas…….. 54
3.2.2 Extreme Programming (XP)……... 54
3.2.3 SCRUM…….. 56
3.2.4 ICONIX……. 57
3.3 Herramientas Case para UML…….. 58
3.3.1 Rational Rose……... 59
3.3.2 ArgoUML……. 59
3.3.3 Poseidon…… 60
3.4 Aplicativos…… 60
3.4.1 SofyMed Protocolos Ecográficos……... 60
3.4.2 DPI Software de GBSystems…… 62
3.4.3 GalenHos(Sistema Integral de Gestión Hospitalaria)……... 64

3.4.3.1 Características…….. 64

3.4.3.2 Implementación…… 64

CAPITULO 4: RESOLUCION DEL PROBLEMA APLICANDO LA TECNICA SELECCIONADA…… 65

4.1 Análisis Situacional del Hospital San José…….. 65

4.1.1 Reseña Histórica…….. 65

4.2 Infraestructura…….. 66

4.2.1 Organigrama…….. 67

4.2.2 Misión y Visión del Hospital San José del Callao……. 67

4.2.3 Objetivo…..…… 67

4.3 Evaluación y selección de la técnica usada…….. 68

4.3.1 Metodologías para el desarrollo de software…… 68

4.3.1.1. Análisis de criterios de comparación……. 71

4.3.1.1.1 Vista del sistema como algo cambiante…… 71

4.3.1.1.2 Colaboración entre los miembros del equipo…….. 71

4.3.1.1.3. Características metodológicas (CM)…… 71

4.3.1.1.4. Cuadro comparativo de los criterios establecidos…… 72

4.3.2 Herramientas Case…… 73

4.3.2.1 Selección de los criterios de evaluación…… 74

4.3.2.1.1 Enfoque procedimental…….. 74

4.3.2.1.2 Soporte de modelo arquitectónico……. 77

4.3.2.1.3 Apoyo al repositorio…….. 79

4.3.2.1.4 Enfoque funcional……. 82

4.4 Análisis de costo/beneficio…… 84

4.5 Plan de desarrollo de software……87

4.5.1 Introducción…….……... 86

4.5.1.1 Propósito……... 87

4.5.1.2 Alcance……87

4.5.1.3 Resumen…….. 87
4.5.2. Vista general del proyecto…… 88
4.5.2.1 Propósito, alcance y objetivo…… 88
4.5.2.2 Suposiciones y Restricciones…… 89
4.5.2.3 Entregables del proyecto…… 89
4.5.2.4 Evolución del Plan de Desarrollo del Software…… 89
4.5.3. Organización del proyecto…… 90
4.5.4 Gestión del proyecto…….. 91
4.5.4.1 Estimación del proyecto……. 91
4.5.4.2 Plan del proyecto…… 91
4.5.4.2.1 Plan de fases…… 91
4.5.4.2.2 Calendario del proyecto……. 93
4.5.4.3 Seguimiento y control del proyecto…… 95
CAPITULO 5: FASE DE ANALISIS Y DISEÑO - DESCRIPCION DE LA SOLUCION TECNOLOGICA…… 97
5.1 Situación Actual del Proceso de Recepción de Consulta y Entrega de Resultados (ecografías)…… 97
5.2 Modelo de Negocio…….. 99
5.3 Modelo de Caso de Uso…… 99
5.4 Especificaciones de caso de uso……... 101
5.5 Prototipo de Interfases de Usuario……. 103
5.6 Modelo de Análisis y Diseño……. 105
5.7 Modelo de Datos …….. 106
5.8 Modelo dinámico…….. 108
5.8.1 Diagrama de Secuencia……108
5.8.2 Diagrama de Colaboración…….. 110
5.9 Modelo de despliegue…….111
CAPITULO 6: CONCLUSIONES Y FUTUROS TRABAJOS…… 112
REFERENCIA BIBLIOGRAFICA…… 113

Índice de Figuras

	Página
Figura 1: Historia del UML.	**5**
Figura 2: Colaboración.	**9**
Figura 3: Nodo.	**9**
Figura 4: Paquete.	**10**
Figura 5: Nota.	**11**
Figura 6: Relación de dependencia entre dos clases.	**12**
Figura 7: Diagrama de Casos de Uso.	**13**
Figura 8: Realización de generalización.	**13**
Figura 9: Realización de la asociación.	**14**
Figura 10: Roles y multiplicidad en una asociación.	**14**
Figura 11: Asociación unidireccional.	**15**
Figura 12: Relación de agregación.	**15**
Figura 13: Relación de composición.	**16**
Figura 14: Relación de agregación.	**16**
Figura 15: Relación de interfase.	**16**
Figura 16: Relación de agregación.	**18**
Figura 17: Diagrama de Colaboración.	**19**
Figura 18: Diagrama de Secuencia.	**21**
Figura 19: Diagrama de Casos de Uso.	**23**
Figura 20: Diagrama de estado.	**24**
Figura 21: Diagrama de actividades.	**25**
Figura 22: Diagrama de componentes.	**26**
Figura 23: Diagrama de despliegue.	**27**
Figura 24: UML y el proceso de desarrollo.	**28**
Figura 25: Los Casos de Uso integran el trabajo	**29**
Figura 26: Trazabilidad a partir de los Casos de Uso	**29**

Figura 27: Evolución de la arquitectura del sistemas **30**

Figura 28: Los modelos se completan, la arquitectura no cambia drásticamente. **31**

Figura 29: Ciclos iterativos de desarrollo **32**

Figura 30: Desarrollo iterativo **33**

Figura 31: Fases e hitos en RUP **33**

Figura 32: Fases en el desarrollo iterativo incremental **38**

Figura 33: Relación entre roles, actividades, artefactos **39**

Figura 34: Detalle de un workflow mediante roles, actividades y artefactos **39**

Figura 35: Diagrama de XP **54**

Figura 36: Esquema de trabajo SCRUM **57**

Figura 37: Esquema de trabajo ICONIX. **58**

Figura 38: Diagrama UML de secuencia creado con ArgoUML. **59**

Figura 39: Diagrama UML de secuencia creado con Poseidón. **60**

Figura 40: Pantalla de inicio. **61**

Figura 41: Pantalla de examen. **61**

Figura 42: Interface de DPI Software. **63**

Figura 43: Organigrama del Hospital San José. **67**

Figura 44: Diagrama de módulos del sistema. **98**

Figura 45: Diagrama de Casos de uso del negocio **98**

Figura 46: Diagrama de Clases **104**

Figura 47: Modelo físico de base de datos **106**

Figura 48: Diagrama de Secuencia Registrar_comprobante. **107**

Figura 49: Diagrama de Secuencia Validar_usuario. **107**

Figura 50: Diagrama de Secuencia Consultar_datos. **108**

Figura 51: Diagrama de Secuencia Registrar_cita. **108**

Figura 52: Diagrama de Secuencia Registrar_examen. **109**

Figura 53: Diagrama de Colaboración Registrar_boleta. **109**

Figura 54: Diagrama de Colaboración Validar_usuario. **110**

Figura 55: Diagrama de Colaboración Registrar_examen **110**
Figura 56: Diagrama de Despliegue **111**

Índice de Tablas

	Página
Tabla N° 1: Diferencias entre Metodologías Tradicional y Ágiles	**68**
Tabla N° 2: Diferencias por Etapas y Enfoque Metodológico	**69**
Tabla Nº 3: Según características del proyecto.	**69**
Tabla N° 4: Según la curva de aprendizaje.	**70**
Tabla N° 5: Cuadro Comparativo de Metodologías	**72**
Tabla N° 6 Cuadro Comparativo General	**73**
Tabla N° 7: Cuadro Comparativo – Enfoque Procedimental.	**77**
Tabla N° 8: Cuadro Comparativo – Soporte Arquitectónico.	**79**
Tabla N° 9: Cuadro Comparativo – Apoyo al Repositorio.	**81**
Tabla N° 10: Cuadro Comparativo – Enfoque Funcional.	**83**
Tabla N° 11: Cuadro Comparativo – Soporte Arquitectónico.	**83**
Tabla N°12: Cuadro de costo de Hardware.	**84**
Tabla N°13: Cuadro de costo de Software.	**84**
Tabla N°14 Cuadro de costo de personal.	**85**
Tabla N°15 Cuadro de costos varios.	**85**
Tabla N°16: Calculo del VAN (valor actual neto) y TIR.	**86**
Tabla N°17: Cuadro de responsables.	**91**
Tabla N°18: Cuadro de Fases.	**92**
Tabla N°19: Cuadro de descripción Hitos.	**92**
Tabla N°20: Cuadro de duración – fase de inicio	**93**
Tabla N°21: Cuadro de duración – fase de elaboración	**94**

Capitulo 1: Introducción

El presente trabajo de investigación nace con la necesidad de implementar un Sistema de Información que permita registrar, almacenar y consultar los resultados de los exámenes ecográficos por el Área de Diagnóstico por Imágenes; y consultar desde cualquier estación de trabajo por los especialistas médicos dentro de la red de la institución con las políticas de seguridad adecuadas, con una interfaz amigable al usuario.

Para desarrollar un sistema de información primero debemos elaborar un plan de desarrollo de software orientado a una metodología, que en nuestro caso será el RUP (Rational Unified Processing) basada en UML (Unified Modeling Language), el cual nos guiara por todo el camino hasta la implementación del sistema de información.

Para ello se ha investigado sobre las metodologías más comunes en el desarrollo de software, el lenguaje estándar de modelado UML y las herramientas case mas utilizadas en el mercado. Se desarrolla la mayoría de diagramas UML, con el fin de servir de guía para futuros desarrollo de software.

Este proyecto servirá para fomentar el desarrollo de sistemas de información siguiendo un plan de desarrollo basado en una metodología particular, con el fin de optimizar la elaboración del software

1.1 Antecedentes

En el año 2000, el proceso desde el pago del examen hasta la entrega de los resultados a sido manualmente, utilizando un sistema en la cual se llenaba las fichas con un formato establecido, luego en el año 2004 se implementaron sistemas para diversas áreas, principalmente el área de CAJA, la recepción de los pacientes y la generación de citas para los diversos exámenes siguió siendo manual (cuaderno de control de pacientes y control de citas), pero la entrega de los resultados se realizó en un procesador de texto (MS Word) el cual se imprimía en un formato establecido, luego en el año 2006 se creó el sistema de admisión, sistema de emergencia y el sistema de historias clínicas (registro y búsqueda), con lo cual nace la necesidad de cruzar información, sobre datos de los pacientes (historia clínica vs boleta de pago), así como de almacenar en una base de datos los diferentes resultados de los exámenes ecográficos y ser consultados posteriormente por los médicos especialistas; también la generación de citas para los diversos exámenes, ya que los pacientes se atienden días posteriores a su pago debido a la gran demanda que hay por dichos exámenes o por que el examen necesita de algunos requisitos (preparación del paciente) que involucra días de tratamiento antes de realizar el examen.

Tal es el caso de estudio del Hospital de Apoyo San José - Callao, que ve reflejado este incremento en la cantidad de exámenes realizados en el Área de Diagnóstico por Imágenes.

En la actualidad, el Sector Salud está creciendo día a día y son más la cantidad de pacientes que se atienden en hospitales, clínicas, centros de salud, postas; esto genera un incremento de las necesidades de atención, como es el caso de los exámenes ecográficos (Diagnóstico por Imágenes). Estos exámenes complementarios son cada vez más utilizados por los médicos a descartar enfermedades peligrosas.

1.2 Definición del problema

El problema de no contar con un sistema de información que permita la automatización de registro de los exámenes ecográficos almacenadas en una base de datos e integrada con los demás sistemas del hospital, imposibilita el aprovechamiento total de la información.

Actualmente se desarrolla software, pero no se ha definido un proceso o actividades que apoyen a los desarrolladores en cada una de las etapas. Los procesos actuales que están siendo utilizados resultan inadecuados para el desarrollo de software. Es por eso que este trabajo servirá de base para futuros desarrolladores de software de la institución.

1.3 Objetivos

1.3.1 Principal

Elaboración de un plan de desarrollo de software para automatizar el proceso de registro de resultados de exámenes ecográficos en el área de diagnostico por imágenes utilizando la metodología RUP basada en UML.

1.3.2 Secundarios

- Definir la metodología RUP y el lenguaje UML
- Desarrollar el modelamiento de los procesos principales.
- Implementación del plan de desarrollo, a manera de ejemplo, para la automatización de los procesos de registro de resultado de los exámenes ecográficos para el Área de Diagnóstico por Imágenes del Hospital San José – Callao

1.4 Justificación

El trabajo central de esta tesina se orienta, precisamente, hacia el desarrollo de un Sistema de Información en el Área de Diagnóstico por Imágenes, con una problemática recurrente de pérdida de información y gran demanda. Esto supone una aportación valiosa para las operaciones desarrolladas en el centro hospitalario.

Para ello, se elabora un plan de desarrollo de software utilizando la metodología de desarrollo, el RUP, esto debido que la mayoría de proyectos de software fracasan, por

no adoptar procedimientos, metodologías y herramientas que permitan una estandarización de software.

El RUP nos ofrece un "plano" de todo el sistema analizado, mejorando la planeación y control del proyecto, incrementando la calidad de desarrollo y un mejor entendimiento del riesgo del proyecto antes de construir el sistema.

Respecto a los exámenes realizados por el área de diagnóstico por imágenes, estos están siendo utilizados por una amplia variedad de áreas médicas (cardiología, neurología, emergencia, medicina física, etc.); ultrasonidos, tomografías, radiografías, resonancias, ecografías; son la base fundamental de cualquier análisis solicitado por los médicos en el mundo. La ventaja de esta afirmación es la posibilidad de desarrollar este tipo de diagnóstico por imágenes con herramientas de desarrollo de software que no impliquen grandes gastos y que permiten una amplia consistencia e integridad de la información. Asimismo, los resultados manejados en una herramienta tecnológica implican una mejor distribución y reutilización de la información.

Esta aplicación es de vital importancia en la sala de emergencia donde se evitará el tener que volver a tomar una ecografía, en caso de alto riesgo para el paciente y el ahorro en tiempo puede ser muy significativo ya que no será necesario tomarlas nuevamente.

1.5 Propuesta

Lo que proponemos con el presente trabajo es identificar la secuencia ordenada de actividades y los artefactos para cada etapa o fase del desarrollo de un proyecto de software que responda a la problemática actual. Basándonos en casos de usos, orientado a la arquitectura, y con un proceso iterativo incremental que definen a nuestra metodología de desarrollo de software a utilizar, en este caso el RUP.

Se plantea la utilización del lenguaje de modelamiento UML (Lenguaje estándar para el modelamiento de sistemas) para poder establecer las distintas fases del ciclo de vida del desarrollo del sistema de información de la solución planteada en el proyecto (fase de análisis y diseño) basado en la metodología RUP, para lo cual se hizo un estudio comparativo de todas las metodologías más conocidas en el desarrollo de sistemas, tomando criterios generales para su elección; también se realizo un estudio comparativo para las Herramientas CASE que utilizan como lenguaje, el UML, siendo la herramienta elegida el Rational Rose; por ser más completa y basada en UML en sus dos versiones y orientada al RUP. La propuesta planteada implica realizar el plan de desarrollo de software hasta la fase de elaboración, como modelo de ejemplo, para lo cual desarrollaremos las distintas actividades, abarcando la mayor cantidad de modelos UML, basándonos en la problemática de un sistema de información para la automatización de registros de exámenes ecográficos.

1.6 Organización de la tesina

Este trabajo está organizado en 6 capítulos.

En el capítulo 2, se presenta el Marco Teórico, en el se describe los temas más importantes de esta tesina: el lenguaje de modelado UML, la metodología RUP y el modelo constructivo de costo.

En el capítulo 3, se presenta el Estado del Arte, en el cual se detalla la metodologías existentes, las herramientas case que desarrollan dichas metodologías, también se nombran algunos aplicativos existentes en el mercado.

En el capítulo 4, se presenta la resolución del problema. Se describe a la empresa, en este caso el hospital San José, luego se evalúa las metodologías existentes, luego se compara las herramientas case, Luego se hace un análisis de costo / beneficio del proyecto del software y finalmente se elabora el plan de desarrollo del software

En el capítulo 5, se presenta la descripción de la solución tecnológica, en el cual detallamos los distintos diagramas UML: diagrama de caso de uso, diagrama de clases, etc., y el modelo de la base de datos, y una vista del prototipo del aplicativo a desarrollar, todo esto en el orden que nuestro plan describe.

En el capítulo 6, se presenta las Conclusiones y Futuros Trabajos, en el cual se describe los beneficios que nos brinda la solución, los futuros trabajos que se pueden realizar tomando como base este trabajo, y algunos que no se desarrollaron por no ser exclusivos del tema principal.

Capitulo 2.- Marco Teórico

2.1. UML

El Lenguaje Unificado de Modelado, o UML (*Unified Modeling Languaje*), es un lenguaje gráfico de modelado que provee de una sintaxis para describir los elementos importantes (llamadas *artefactos* en UML) de un sistema de software. UML, en cuanto a notación de modelado, es el sucesor de la oleada de métodos de análisis y diseño orientado a objetos que aparecieron a principios de los 90. De hecho, como se dijo en su momento, unifica la notación de los métodos de Booch, Rumbaugh (OMT) y Jacobson (OOSE), y además, es considerado un estándar OMG [OMG03]

El UML es un estándar para construir modelos orientados a objetos. Nació en 1994 por iniciativa de Grady Booch y Jim Rumbaugh para combinar sus dos famosos métodos: el de Booch y el OMT (Object Modeling Technique, Técnica de Modelado de Objetos). Más tarde se les unió Iván Jacobson, creador del método OOSE (Object-Oriented Software Engineering, Ingeniería de Software Orientada a Objetos). En respuesta a una petición del OMG (Object Managament Group, Grupo de Administración de Objetos) para definir un lenguaje y una notación estándar del lenguaje de construcción de modelos, en 1997 propusieron el UML como candidato [Larman00+] ver figura

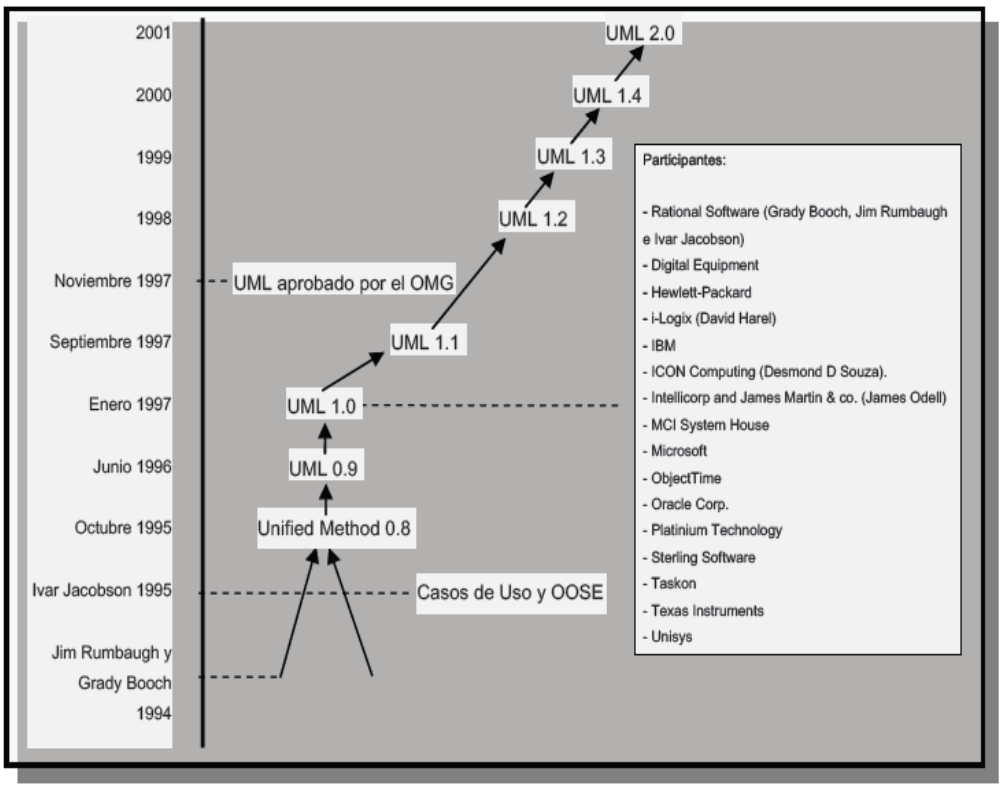


Figura 1: Historia del UML [Letelier02]

Sin embargo, UML no es un método. Un método consiste, en principio, en un lenguaje de modelado y un proceso. UML aporta la parte de lenguaje de modelado, pero es necesario un proceso que utilice dicho lenguaje. El Proceso Unificado de Desarrollo de Software, creado por los mismos autores de UML, aporta la parte de proceso en el que se emplea este lenguaje de modelado.

Booch afirma que el 80% de la mayoría de problemas puede modelarse con el 20% de UML. Los elementos estructurales básicos, tales como clases, atributos, operaciones, casos de uso, componentes y paquetes, junto a las relaciones estructurales básicas, tales como dependencia, generalización y asociación, son suficientes para crear modelos estáticos para muchos tipos de dominios de problemas. Si a esta lista se añaden los elementos de comportamientos básicos, tales como las maquinas de estados simples y las interacciones, se pueden modelar muchos aspectos útiles de la dinámica de un sistema. Solo hará falta utilizar las características más avanzadas de UML cuando se empiece a modelar aspectos relacionados con situaciones más complejas como cuando se debe tratar con mecanismo de concurrencia y de distribución.

El vocabulario de UML, incluye tres grandes bloques de construcción [Booch00+; OMG03]; elementos, relaciones y diagramas. Los elementos son abstracciones fundamentales de un modelo; las relaciones ligan estos elementos entre si; los diagramas agrupan colecciones de elementos.

2.1.1. Elementos en UML

En UML hay cuatro tipos de elementos, los cuales constituyen los bloques básicos de construcción orientados a objetos de este lenguaje de modelado [Booch99]. Por un lado, los elementos estructurales representan las partes estáticas de un modelo, los "materiales de construcción" propios del lenguaje. En total hay siete tipos de elementos estructurales: clases, interfaces, colaboraciones, casos de uso, clases activas, componentes y nodos. Los elementos *de comportamiento* son las partes dinámicas de los modelos UML, son los verbos de un modelo, y representan comportamiento en el tiempo y el espacio. Hay dos tipos principales de elementos de comportamiento: interacciones y maquinas de estados. Los elementos de agrupación son las partes organizativas de los modelos UML. El elemento de agrupación principal son los paquetes. Por ultimo, existen elementos de anotación, que representan las partes explicativas de los modelos. Son comentarios que se pueden aplicar para describir, clarificar y hacer observaciones sobre cualquier elemento del modelo. Hay un tipo principal de elementos de anotación llamado nota.

.

2.1.1.1. Elementos Estructurales

Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. La notación grafica y función de una clase se ha mostrado en anteriores apartados, diferenciándose entre clases de análisis y clase de diseño. Una clase de análisis, aunque tenga una representación grafica propia asociada a su función (clase de interfaz, clase de entidad y clase de control), puede también adoptar la forma de una clase de diseño.

Al modelar clases, un buen comienzo consiste en especificar las responsabilidades de los elementos del vocabulario. Técnicas como las tarjetas CRC y el análisis basado en casos de uso son especialmente útiles aquí. Al ir refinando los modelos, se proporcionan a los atributos y operaciones que mejor satisfagan esas responsabilidades de la clase.

En una clase, es importante especificar la visibilidad de los atributos y operaciones que contenga. Normalmente, los atributos suelen ser privados (símbolo -), solo accesibles desde los objetos propios de la clase que los contiene, y las operaciones públicas (símbolo +), accesibles por otras clases a través de una referencia a un objeto de la clase que las implementa; sin embargo, tanto unos como otros pueden ser declarados públicos o privados. Además, también se pueden proteger (símbolo #) estas propiedades y funciones de la clase, permitiendo su acceso únicamente desde la propia clase y clases derivadas. Finalmente, se puede indicar que el elemento es visible únicamente para las clases que pertenecen al mismo paquete de agrupación (no se añaden símbolo alguno al nombre del atributo u operación).

Además de las opciones de visibilidad, es posible especificar el alcance de un atributo. Puede tener alcance de instancia (normalmente será así), en cuyo caso cada instancia de la clase tiene su propio valor para la característica, o bien, alcance de clase (el atributo aparece subrayado), indicando que sólo hay un valor de la característica para toda las instancias de la clase, es decir, que cualquiera de las instancias de esa clase puede modificar el valor de dicho atributo. El alcance de clase de un atributo se corresponde con lo que en lenguajes como C++ o Java se conoce como atributo estático (static). Por su parte, una operación estática es implementada por un método que puede ser llamado aunque no se haya creado ningún objeto de la clase que contiene.

Una instancia de una clase es conocida como un *objeto;* dicho de otra manera, si una clase es una abstracción, un objeto es una manifestación concreta de esa abstracción. Esta división común entre clase y objeto se modela de manera distinta. UML distingue un objeto utilizando el mismo símbolo de la clase y subrayado el nombre del objeto.

Casi todos los bloques de construcción de UML presentan este tipo de dicotomía abstracción/instancia. Por ejemplo, se pueden tener casos de uso e instancias de casos de uso, componentes e instancias de componentes, nodos e instancias de nodos, etc.

Concretamente, una instancia es una manifestación concreta de una abstracción a la que se puede aplicar un conjunto de operaciones y que posee un estado que almacena el efecto de estas operaciones. La mayoría de las instancias que se modelan en UML son instancias de clase.

Una *clase activa* es una clase cuyos objetos tienen uno o más procesos o hilos de ejecución, dando origen a actividades de control. Los objetos de una clase activa representan elementos cuyo comportamiento es concurrente con otros elementos. Gráficamente, una clase activa se representa como una clase, pero con líneas más gruesas.

Una interfaz es una colección de operaciones que especifican un servicio de una clase o componente. Define un conjunto de especificaciones de operaciones (o sea, sus signaturas), que serán implementadas por las clases o componentes que usen dichos servicios. Gráficamente una interfaz se representa como un círculo junto con su nombre aunque también puede adoptar la forma rectangular de una clase (forma expandida); además, raramente se encuentra aislada, sino que suele estar conectada a la clase o componente que la realiza.

Como se dijo en su momento, las interfaces son similares a las clases abstractas (por ejemplo, ninguna de las dos puede tener instancias directas), pero difieren lo bastante como para merecer ser elementos de modelado diferenciados. Una clase abstracta puede tener atributos, pero una interfaz no. Además, las interfaces cruzan los límites del modelo; la misma interfaz, como se ha comentado en el párrafo anterior, puede ser realizada tanto por una clase (una abstracción lógica) como por un componente (una abstracción física que proporciona una manifestación de la clase).

En UML, las interfaces se emplean para modelar las líneas de separación de un sistema. Al declarar una interfaz, se puede enunciar el comportamiento deseado de una abstracción independientemente de una implementación de ella. Además, las interfaces no solo son importantes para separar la especificación y la implementación de una clase o componente, sino que al pasar a sistemas más grandes, se pueden usar estas interfaces para especificar la vista externa de un paquete o subsistema.

Un caso de uso (elipse con borde continuo) es una descripción de un conjunto de secuencias de acciones que un sistema ejecuta y que produce un resultado observable de interés para un actor particular.

Una colaboración define una interacción y es una sociedad de roles y otros elementos que colaboran para proporcionar un comportamiento cooperativo mayor que la suma de los comportamientos de sus elementos. En este sentido, las colaboraciones tienen una dimensión tanto estructural como de comportamiento (pueden ser realizadas, por

ejemplo mediante diagramas de clases y diagramas de interacción). Gráficamente, una colaboración se representa como una elipse con borde discontinuo:

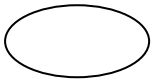


Figura 2: Colaboración

Un caso de uso se especifica normalmente mediante una colaboración, que representa la realización de dicho caso de uso.

Un componente es una parte física y reemplazable de un sistema que conforma con un conjunto de interfaces y proporciona la implementación de dicho conjunto. Un componente representa típicamente el empaquetamiento físico de diferentes elementos lógicos., como clases, interfaces y colaboraciones. Gráficamente, un componente se representa como un rectángulo con pestañas.

Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional (elemento hardware), que por lo general dispone de algo de memoria y, con frecuencia, capacidad de procesamiento. Un conjunto de componentes puede residir en un nodo y puede también migrar de un nodo a otro. Gráficamente, un nodo se representa como un cubo, incluyendo normalmente su nombre (por ejemplo, servidor, cliente, impresora, módem, etc.)

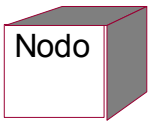


Figura 3: Nodo

Los nodos son empleados en los diagramas de despliegues, que muestran la configuración de los nodos de procesamiento y los componentes que residen en ellos. En este sentido, los nodos representan el hardware sobre el que se despliegan y ejecutan esos componentes.

2.1.1.2. Elementos de comportamiento

Una interacción es un comportamiento que corresponde un conjunto de mensajes intercambiados entre un conjunto de objetos, dentro de un contexto particular, para alcanzar un propósito especifico.

Una maquina de estados es un comportamiento que especifica las secuencias de estados por las que pasa un objeto o una interacción durante su vida en respuesta a eventos,

junto con sus reacciones a estos eventos. El comportamiento de una clase individual o una colaboración de clases puede especificarse con una maquina de estados.

2.1.1.3. Elementos de agrupación

Un paquete es un mecanismo de propósito general para organizar elementos en grupos. Los elementos estructurales, los elementos de comportamiento, e incluso otros elementos de agrupación pueden incluirse en un paquete. Al contrario que los componentes (que existen en tiempo de ejecución), un paquete es puramente conceptual (sólo existe en tiempo de desarrollo). Gráficamente se visualiza como una carpeta:

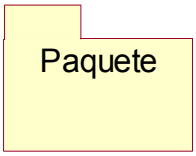


Figura 4: Paquete

Los paquetes bien diseñados agrupan elementos cercanos semánticamente y que suelen cambiar juntos. Por tanto, los paquetes bien estructurados son cohesivos y poco acoplados, estando muy controlado el acceso a su contenido. En este sentido, se puede controlar la visibilidad de los elementos contenidos en un paquete del mismo modo que se puede controlar la visibilidad de los atributos y operaciones de una clase (utilizando para ello la misma notación). Normalmente, un elemento contenido en un paquete es público, es decir, es visible a los contenidos de cualquier paquete que importe (*import,* en lenguajes como Java) el paquete contenedor del elemento. Por el contrario, los elementos protegidos sólo pueden ser vistos por los paquetes derivados (que heredan de otro paquete), y los elementos privados no son visibles fuera del paquete en el que se declaran. El conjunto de las partes públicas de un paquete constituye la interfaz del paquete (elementos exportables).

Cuando un sistema consta de pocas clases que se conocen entre ellas, estamos ante un sistema trivial que no necesita ningún tipo de empaquetamiento. Ahora bien, si son cientos de clases las que se conocen entre ellas, no hay límites para la intrincada red de relaciones que se puede establecer. Además, no hay forma de comprender un grupo de clases tan grande y desorganizado. Este es un problema real para grandes sistemas [Booch00+], puesto que el acceso simple y no restringido no permite el crecimiento. Para estas situaciones se necesita algún tipo de empaquetamiento controlado para organizar las abstracciones.

Booch, plantea un ejemplo simple en el que se comprueba las ventajas de organizar los elementos del sistema en paquetes. Dadas dos clases A y B, ambas públicas (se conocen mutuamente) y que están en un mismo paquete, A puede ver a B y B puede ver a A. Sin

embargo, si A y B pertenecen a paquetes distintos, aunque sean públicas, ninguna puede acceder a la otra porque sus paquetes contenedores forman un muro opaco. Sin embargo, si el paquete de A importa el paquete de B (relación de dependencia). A puede ver a B (si B es publica), aunque B no puede ver a A (aunque A sea publica). La importación concede un permiso de un solo sentido para que los elementos de un paquete accedan a los elementos de otro. Por tanto, al empaquetar las abstracciones en bloques significativos y luego controlar los accesos mediante la importación, se puede controlar la complejidad de tener un gran número de abstracciones.

La generalización (a la que se hará referencia en el siguiente subapartado) es otro tipo de relación entre paquetes, que permite especificar familias de paquetes (en el mismo sentido que las familias de clases). De hecho, la generalización entre paquetes es muy parecida a la generalización entre clases. En este sentido, al igual que en la herencia de clases, los paquetes pueden reemplazar (redefinir) a los elementos mas generales que han sido heredados (elementos públicos y protegidos) y añadir otros nuevos.

Finalmente, dentro de este contexto, hay que hacer mención al concepto de *subsistema.* Los subsistemas son similares a los paquetes, pero tienen identidad, es decir, son considerados *clasificadores* (pueden tener instancias, atributos y operaciones, casos de uso, maquinas de estado y colaboraciones, los cuales especifican el comportamiento de ese subsistema). Un subsistema es simplemente una parte del sistema, y se utiliza para descomponer un sistema complejo en partes casi independientes. Un subsistema se representa (en UML) igual que un paquete, añadiendo a su nombre el estereotipo «subsystem». De la misma manera, un sistema también se representa como un paquete estereotipado (con el estereotipo «system».). Sin embargo, a pesar de esta diferencia conceptual establecida por UML entre paquete y subsistema, hay que tener en cuenta, a nivel práctico, que los lenguajes de programación utilizados en la implementación del sistema utilizan un único concepto para referirse a la agrupación de ficheros en compartimientos diferenciados (por ejemplo, los *packages* en Java).

2.1.1.4. Elementos de anotación

Una nota es simplemente un símbolo para mostrar restricciones y comentarios junto a un elemento o una colección de elementos. Gráficamente, una nota se representa como un rectángulo con una esquina doblada, junto con su comentario.

Figura 5: Nota

2.1.2. Relaciones entre elementos

En los diagramas UML expuestos a lo largo de este trabajo se han utilizado unas u otras relaciones. La intención de este subapartado es la de detallar cada una de estas relaciones, junto a sus posibles variantes.

[Booch99+] Booch, plantea un símil extraído del contexto de la construcción inmobiliaria para concretar en forma de imágenes las relaciones más importantes en el modelado orientado a objetos (dependencias, generalizaciones y asociaciones). Las dependencias son relaciones de uso. Por ejemplos, las tuberías dependen del calentador para calentar el agua que conducen. Las generalizaciones conectan clases generales con otras más especializadas. Por ejemplo, una ventana de patio (clase especializada) es un tipo de ventana (clase general) con hojas de cristal que se abren de lado a lado. Las asociaciones son relaciones estructurales entre instancias. Por ejemplo, las habitaciones constan de paredes y otros elementos; las paredes a su vez pueden contener puertas y ventanas; las tuberías pueden atravesar las paredes. Estos tres tipos de relaciones cubren la mayoría de las formas importantes en que colaboran unos elementos con otros.

2.1.2.1. Dependencia

Una dependencia es una relación semántica entre dos elementos, en la que un cambio en un elemento (el elemento independiente) puede afectar a la semántica del otro elemento (el elemento dependiente). Gráficamente, una dependencia se representa como una línea discontinua dirigida hacia el elemento del cual se depende

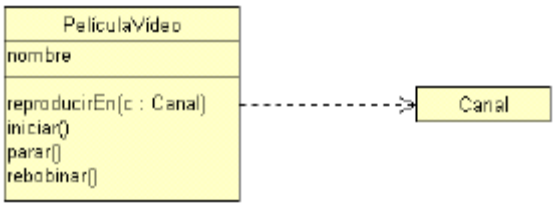


Figura 6: Relación de dependencia entre dos clases

La mayoría de las veces, las dependencias se utilizan en el contexto de las clases, para indicar que una clase utiliza a otra como argumento en la signatura de una operación, tal como se muestra en la figura anterior.

Si se quieren indicar ciertos matices en las relaciones de dependencia, UML define varios estereotipos (extensión del vocabulario específico a un problema determinado). En concreto, en el contexto de los casos de uso, se aplican dos estereotipos a las relaciones de dependencia, el estereotipo *extend* (especifica que el caso de uso destino extiende el comportamiento del caso de uso origen) y el estereotipo *incluye* (especifica que el caso de uso origen incorpora explícitamente el comportamiento de otro caso de uso). Un ejemplo de uso de estos dos estereotipos lo encontramos en la siguiente figura.

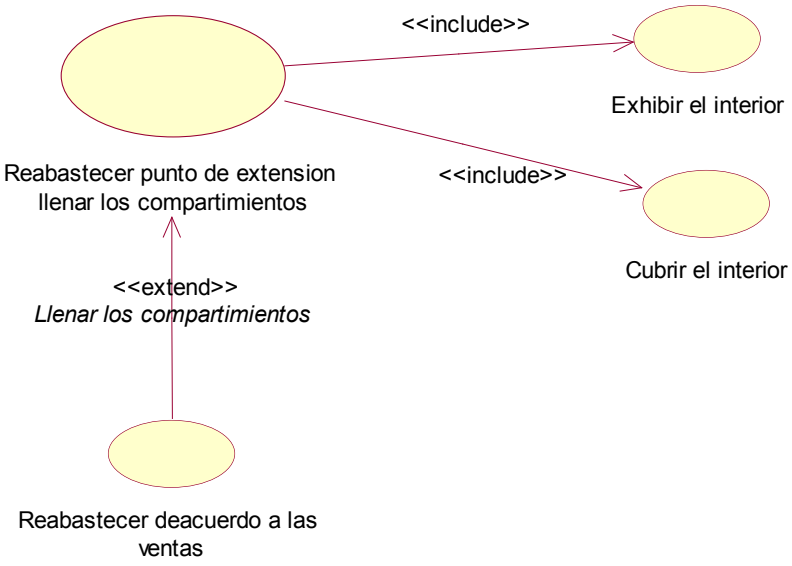


Figura 7: Diagrama de caso de uso

2.1.2.2. Generalización

Una generalización es una relación entre un elemento general (llamado superclase o padre) y un caso mas especifico de ese elemento (llamado subclase o hijo). La generalización se llama a veces relación «es-un-tipo-de»: un elemento (como la clase *Rectángulo*) es un tipo de un elemento más general (por ejemplo, la clase *Figura*). La clase más especifica (subclase) comparte la estructura y el comportamiento de la clase más general (superclase), estableciéndose en definitiva una relación de herencia unidireccional (la clase hija hereda de la clase padre). Gráficamente, la generalización o herencia se representa como una línea dirigida continua con una punta de flecha vacía ha apuntado al padre.

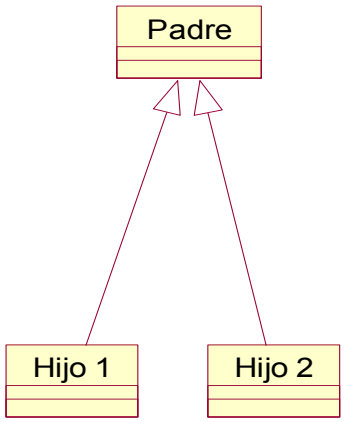


Figura 8: Realización de generalización

2.1.2.3. **Asociación**

Una asociación es una relación estructural que describe un conjunto de enlaces, los cuales son conexiones entre objetos. Por defecto, es posible navegar de los objetos de una clase a los de otra, de manera bidireccional. En este sentido, cuando se modela con relaciones de asociación, se están modelando clases que son del mismo nivel; dada una asociación entre dos clases, ambas dependen de la otra de alguna forma, y se puede navegar en ambas direcciones. En cambio, al modelar con relaciones de dependencia o generalización, se están modelando clases que representan diferentes niveles de importancia o diferentes niveles de abstracción, estableciéndose únicamente relaciones unilaterales.

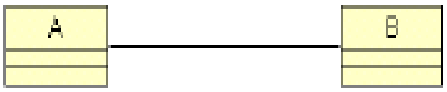


Figura 9: Realización de la asociación

Cuando una asociación conecta dos clases, cada una de las clases puede enviar mensajes a la otra en un diagrama de secuencia o en un diagrama de colaboración. Además, también se permite que ambos extremos de la asociación puedan estar conectados a la misma clase (asociación reflexiva), dando a entender que una instancia de la clase está relacionada con otras instancias de la misma clase.

Aparte de la forma básica de la asociación, hay ciertos elementos que se le pueden añadir. Por ejemplo, puede tener un nombre, que se utiliza para describir la naturaleza de la relación. Además, cuando una clase participa en una asociación puede jugar un rol específico, es decir, presentar un papel ante la clase del otro extremo; este rol se puede nombrar explícitamente y está relacionado con la semántica de las interfaces. Por ejemplo, según el contexto, una instancia de la clase Persona puede jugar el papel de *Madre, Cliente, Directivo, Empleado, Cantante,* etc.; cada rol tendrá asociado un conjunto de operaciones distintas (recogidas en una interfaz). En la figura 10, se puede observar que la clase *Persona* adopta el rol e, cuyo tipo es *Empleado* – una interfaz cuya definición incluye una serie de operaciones: *obtenerHistorialEmpleado( ), obtenerBeneficios( ),* etc. Se podría haber añadido a la asociación el nombre o etiqueta *Trabaja para,* aunque el uso de roles, como se puede comprobar, normalmente hace innecesaria la inclusión de este tipo de elemento.

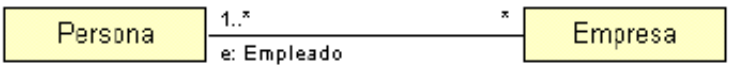


Figura 10: Roles y multiplicidad en una asociación

En esta misma figura aparecen otros descriptores, que hacen referencia a la multiplicidad del rol en la asociación. La multiplicidad indica cuántas instancias de una clase están relacionadas con una única instancia de otra clase; en otras palabras, se especifica que para cada objeto de la clase en el extremo opuesto debe haber tantos objetos en este extremo. En este caso, se observa que una instancia de una empresa puede tener una o más personas empleadas (símbolo 1… *); a su vez, un empleado puede ser contratado por muchas empresas (símbolo *).

Como se ha dicho antes, la navegación a través de una asociación es bidireccional. Sin embargo, puede haber circunstancias en las que se desee limitar la navegación a una sola dirección. Por ejemplo, se encuentra una asociación entre objetos *Usuario y Clave*. Dado un usuario, se pueden encontrar las correspondientes claves; pero dada una clave, no se podrá identificar al usuario correspondiente. Se puede representar de forma explícita la dirección de la navegación con una flecha que apunte en la dirección del recorrido.

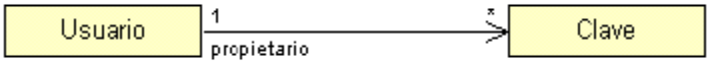


Figura 11: Asociación unidireccional

Una variante de la asociación es la relación de agregación entre clases. Si bien una asociación normal entre dos clases representa una relación estructural entre iguales, esta variante permite asociar clases que conceptualmente se encuentran en distinto nivel. Una clase representa el "todo", que consta de elementos (clases) más pequeños (las "partes"); en términos concretos, un objeto del todo tiene objetos de cada una de las partes asociadas (figura 12). En otros palabras, se puede decir que la agregación representa una relación del tipo «tiene-un».

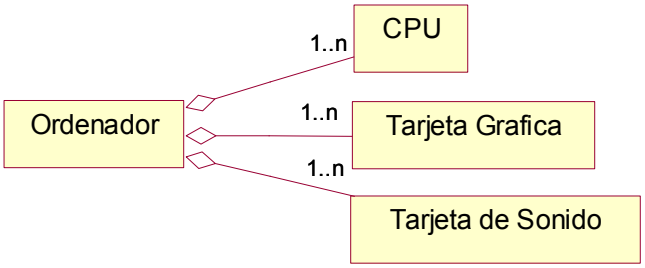


Figura 12: Relación de agregación

La agregación se especifica, como se observa en la figura, añadiendo a una asociación normal un rombo vacío en la parte del todo.

Existe una variante de la agregación (agregación simple), conocida como *composición* (agregación compuesta). Si bien la agregación simple no hace más que distinguir un "todo" de una "parte", la composición establece una fuerte relación de pertenencia, en el sentido que esta relación implica que el "todo" no puede existir sin sus "partes". La composición se especifica añadiendo a una asociación normal un rombo relleno en la parte del elemento compuesto (figura 13).

Figura13: Relación de composición

En el ejemplo de agregación (figura 12), si eliminamos la tarjeta de sonido, el ordenador sigue siendo un ordenador. Sin embargo, un libro no es un libro sin sus páginas, esto es, un libro está compuesto de páginas, y precisamente la parte compuesta (extremo con rombo) es responsable de la distribución de las partes (extremo sin rombo), lo que significa que debe gestionar la creación y destrucción de las mismas. En este sentido, los objetos que representan el "todo" y los objetos que representan sus "partes" son creados y destruidos al mismo tiempo.

2.1.2.4. Realización

Una realización es una relación semántica entre clasificadores, en donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Normalmente, una realización se representa como una línea dirigida discontinua, con punta de flecha vacía que apunta al clasificador que especifica el contrato (notación canoníca):

Figura 14: Relación de agregación

Otra manera de representar una realización es mediante una línea continua no dirigida conectada a un círculo (que representa una interfaz):

Figura 15: Relación de interfase

Normalmente, esta última notación se suele emplear en los diagramas de componentes, para unir un determinado componente con la interfaz que le ofrece unos determinados servicios. Por otro lado, la notación canoníca es la que se suele emplear para representar la realización de una interfaz por parte de una clase. Además, en este caso, la interfaz

suele mostrarse en forma expandida (misma notación gráfica que la clase, y con el estereotipo «interface»), detallando los servicios que ofrece a la clase.

Por último, también se utiliza la realización para especificar la relación entre un caso de uso y una colaboración que realiza ese caso de uso, empleando para ello la notación canoníca. Sin embargo, como la mayoría de veces un caso de uso es realizado por una colaboración, normalmente no se muestra de forma explícita esta relación.

2.1.3. Diagramas de elementos

Un diagrama es la representación grafica de un conjunto de elementos y sus relaciones. Los diagramas se utilizan para visualizar un sistema desde diferentes perspectivas. En el contexto del software hay cinco vistas complementarias, que son las más importantes para visualizar, especificar, construir y documentar una arquitectura software: la vista de casos de uso, la vista de diseño, la vista de procesos, la vista de implementación y la vista de despliegue. Cada una de estas vistas involucra modelado estructural (aspectos estáticos del sistema) y modelado de comportamiento (aspectos dinámicos del sistema). Individualmente, cada una de estas vistas permite centrar la atención en una perspectiva del sistema para poder razonar con claridad sobre las decisiones.

Cuando se ve un sistema software desde cualquier perspectiva mediante UML, se usan los diagramas para organizar los elementos de interés. En teoría, un diagrama puede contener cualquier combinación de elementos y sus relaciones. En la práctica, sin embargo, sólo surge un pequeño número de combinaciones, las cuales son consistentes con las cinco vistas mencionadas. En este sentido, UML incluye nueve tipos de diagramas diferenciados [Booch99+] diagrama de clases, diagrama de objetos, diagrama de casos de uso, diagrama de secuencia, diagrama de colaboración, diagrama de estados, diagrama de actividades, diagrama de componentes y diagrama de despliegue.

2.1.3.1. Diagrama de clases

Un diagrama *de clases* presenta un conjunto de clases e interfaces y las relaciones entre ellas. Son los diagramas más comunes en el modelado de sistemas orientados a objetos y se utilizan para describir la vista de diseño estática de un sistema. Los diagramas de clases que incluyen clases activas se utilizan para cubrir la vista de procesos estática de un sistema.

Los diagramas de clases son la base para un par de diagramas relacionados: los diagramas de componentes y los diagramas de despliegue. Además, los diagramas de clases son importantes no sólo para visualizar, especificar y documentar modelos estructurales, sino también para construir sistemas ejecutables, aplicando ingeniería directa e inversa.

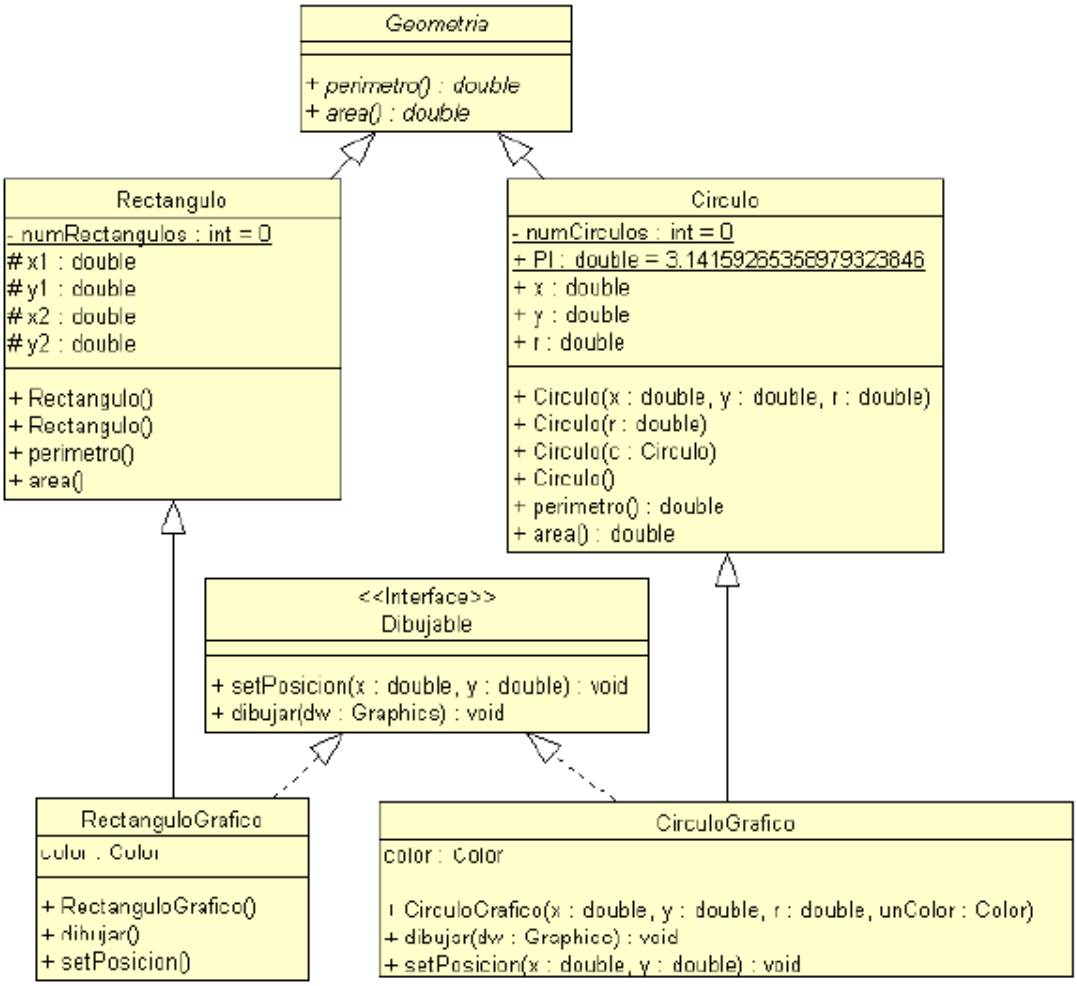


Figura 16: Relación de agregación

2.1.3.2. Diagrama de objetos

Un *diagrama de objetos* representa una instantánea de las instancias de los elementos encontrados en un diagrama de clases. Los diagramas de objetos cubren la vista de diseño estática o la vista de procesos estática de un sistema como lo hacen los diagramas de clases, pero desde la perspectiva de casos reales o prototípicos.

Por tanto, un diagrama de objetos congela un instante en el tiempo, expresando la parte estática de una interacción, que consiste en los objetos que colaboran pero sin ninguno de los mensajes enviados entre ellos. Es en la parte dinámica de esta interacción donde se muestran estos mensajes, vista que se recoge en los diagramas de interacción (diagramas de colaboración y diagramas de secuencia).

Los objetos se conectan mediante enlaces, que representan instancias de las relaciones de asociación que se dan entre las clases de dichos objetos. Estos enlaces permiten introducir en los diagramas de interacción una secuencia dinámica de mensajes.

En los diagramas de objetos (y en los diagramas de interacción), el nombre del objeto va subrayado y asociado a la clase a la que se pertenece, siguiendo la notación nombre:Clase (también es válido incluir objetos anónimos, es decir, sin nombre).

Por otro lado, los objetos pueden mostrar el valor de sus atributos cuando se considere necesario. Los tipos de estos atributos son definidos en la clase a la que pertenece el objeto.

2.1.3.3. Diagrama de interacción

Un diagrama de interacción muestra una interacción, que consta de objetos y sus relaciones, incluyendo los mensajes que pueden ser enviados entre ellos. Los diagramas de interacción cubren la vista dinámica de un sistema. Tanto los diagramas de colaboración como los diagramas de secuencia son un tipo de diagramas de interacción. Un diagrama de colaboración es un diagrama de interacción que resalta la organización estructural de los objetos que envían y reciben mensajes. Un diagrama de secuencia, en cambio, resalta la ordenación temporal de estos mensajes. Se detallan a continuación.

2.1.3.4 Diagrama de colaboración

Un diagrama de colaboración destaca la organización de los objetos que participan en una interacción (figura 17). Este diagrama se constituye colocando en primer lugar los objetos que participan en la colaboración como nodos de un grafo, a continuación se representan los enlaces que conecta esos objetos como arcos del grafo, y por último, estos enlaces se adornan con los mensajes que envían y reciben los objetos.

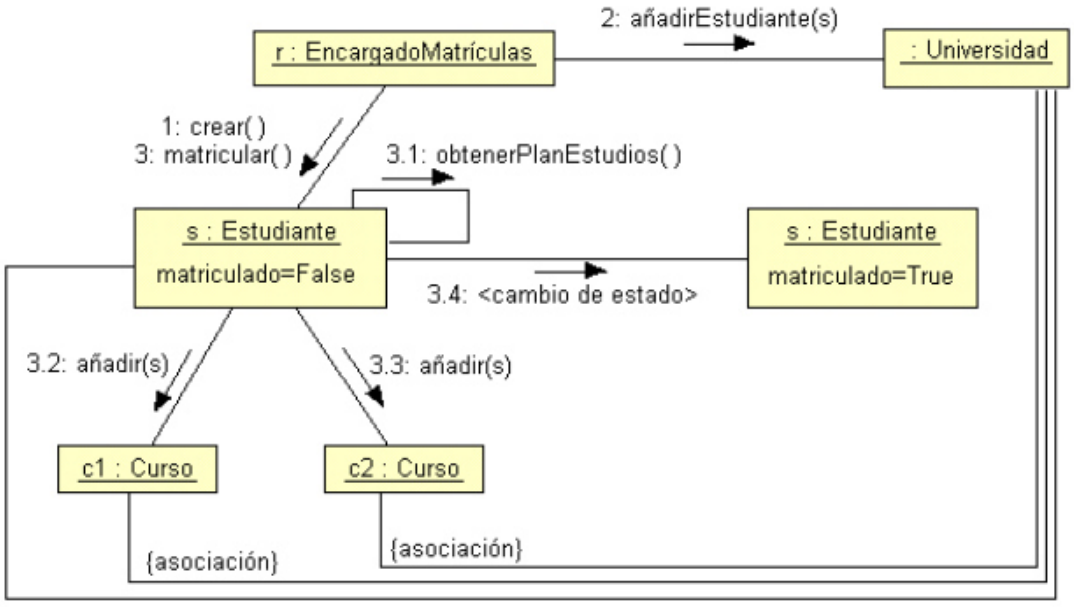


Figura 17: Diagrama de colaboración (Fuente: [Booch99+])

Precisamente, este último paso se obvia en la representación de los diagramas de objetos (al tratar de diagramas estáticos).

Como se observa en la figura 17, para indicar la ordenación temporal de un mensaje éste viene precedido de un número (comenzando con el mensaje número 1), que se incrementa secuencialmente por cada nuevo mensaje en el flujo de control (2, 3, etc.). Para representar el anidamiento, se utiliza la numeración decimal de Dewey (1 es el primer mensaje; 1.1 es el primer mensaje dentro del mensaje 1; 1.2 es el segundo mensaje dentro del mensaje 1, etc.). Además, a través de un mismo enlace se pueden mostrar varios mensajes (en la misma o distinta dirección), y cada uno tendrá un número de secuencia único. Otra opción contemplada en un diagrama de interacción es que un objeto dado se envíe un mensaje a si mismo (mensaje 3.1 en la figura), a través de un enlace que empiece y acabe en él.

En este ejemplo de diagrama de colaboración (figura x), se especifica un flujo de control para matricular un nuevo estudiante en un universidad, destacando las relaciones estructurales entre los objetos. La acción comienza cuando el objeto r (de la clase EncargadoMatriculas) crea un objeto s (de la clase Estudiante), añade el estudiante a la universidad (mensaje añadirEstudiante), y a continuación dice al objeto Estudiante que se matricule (mensaje número 3). El objeto Estudiante (denominado s) invoca la operación obtenerPlanEstudios (mensaje 3.1) sobre sí mismo, de donde presumiblemente obtiene los objetos Curso (c1 y c2) en los que se debe matricular. Después, el objeto Estudiante se añade a sí mismo a cada objeto Curso. El flujo acaba con ser representado de nuevo, mostrando que ha actualizado el valor de su atributo matriculado.

La ingeniería directa (creación de código a partir de un modelo) es posible tanto para los diagramas de secuencia como los de colaboración, especialmente si el contexto del diagrama es una operación. Por ejemplo, con el diagrama de colaboración anterior, una herramienta de ingeniería directa podría generar el siguiente código java para la operación matricular(), asociada a la claseEstudiante:

```
Public void matricular ( ) {
        ColeccionDeCursos c = obtenerPlanEstudios( );
        For (int i = 0; i < c.size( ); i++)
                c.item(i).añadir(this);
        this.matriculado =true
```

2.1.3.5 Diagrama de secuencia

Un diagrama de secuencia, como se ha dicho, es un diagrama de interacción que destaca la ordenación temporal de los mensajes. Gráficamente, un diagrama de secuencia (figura 18) es una tabla que representa objetos, dispuestos a los largo del eje X, y mensajes, ordenados según se suceden en el tiempo, a lo largo del eje Y.

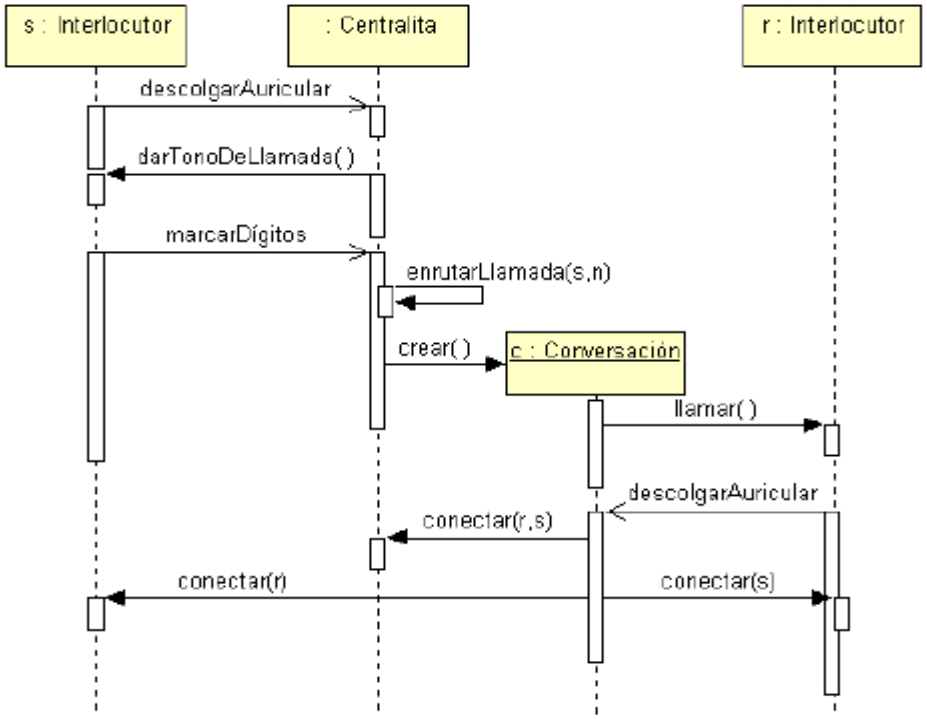


Figura 18: Diagrama de secuencia (Fuente [Booch99+])

Este ejemplo, representa un diagrama de secuencia que especifica el flujo de control para iniciar una simple llamada telefónica entre dos partes. La secuencia comienza cuando un Interlocutor (objeto s) emite una señal (descolgar Auricular) al objeto Centralita (objeto anónimo). A su vez, la Centralita llama a la operación darTonoLlamada sobre este Interlocutor, y el Interlocutor itera sobre el mensaje marcarDigitos. El objeto Centralita se llama a si mismo con el mensaje enrutarLLamada. A continuación, crea un objeto Conservación (objeto c), al cual delega el resto del trabajo. El objeto c llama a (otro Interlocutor), el cual envía de manera asíncrona el mensaje descolgarAuricular. Entonces, el objeto Conversación indica a la Centralita que debe conectar la llamada y luego indica a los dos objetos Interlocutor que pueden conectar, tras lo cual pueden intercambiar información.

Siguiendo con el ejemplo, en este diagrama se pueden distinguir dos tipos de mensajes, en función del tipo de evento asociado al mensaje. Un evento es la especificación de un

acontecimiento que ocupa un lugar en el tiempo y en el espacio; es este sentido, la recepción de un mensaje por parte de un objeto puede considerarse un evento. Se observan en este contexto dos tipos de eventos: eventos de señal y eventos de llamada.

Un evento de señal representa la ocurrencia de una señal que es enviada de manera asíncrona por un objeto y es recibido por otro (como es el caso de la señal descolgarAuricular, que envía el objeto Centralita). Si el mensaje es una señal, por tanto, el emisor y el receptor no se sincronizan el emisor envía la señal pero no espera una respuesta del receptor.

Por otra parte, un evento representa la innovación de una operación y es, en general, síncrono. Ante un evento de llamada síncrono, el emisor y el receptor están sincronizados durante la duración de la operación de manera que el flujo de control del emisor se bloquea con el flujo de control del receptor, hasta que se lleva a cabo la actividad de la operación. En este sentido, cuando un objeto invoca una operación sobre otro objeto que tiene una máquina de estados, el control pasa del emisor al receptor, el evento dispara la transición, la operación acaba, el receptor pasa a un evento y el control regresa al emisor.

Cuando se pasa un mensaje a un objeto, la acción resultante es una instrucción ejecutable que constituye una abstracción de un procedimiento computacional y que puede provocar un cambio de estado en ese objeto. El tipo más común de mensaje que se modela en un diagrama de interacción es la llamada (representada con una línea continua y punta de flecha rellena), que como se ha dicho invoca una operación de otro objeto (o de él mismo).

2.1.3.6 Diagrama de casos de uso

Un diagrama de casos de uso permite modelar la interacción entre el sistema y los usuarios del mismo. Este diagrama muestra un conjunto de casos de uso, actores y sus relaciones, y se utiliza para describir la vista de casos de uso estática de un sistema. Los diagramas de casos de uso son especialmente importantes para organizar y modelar el comportamiento de un sistema, y como se vio en un momento, se utilizan durante la etapa de captura de requisitos del sistema.

Un diagrama de casos de uso permite ver el sistema entero como una caja negra; se puede ver qué hay fuera del sistema y cómo reacciona a los elementos externos, pero no se puede ver cómo funciona por dentro; son otras vistas del sistema las que proporciona esa información de funcionamiento interno.

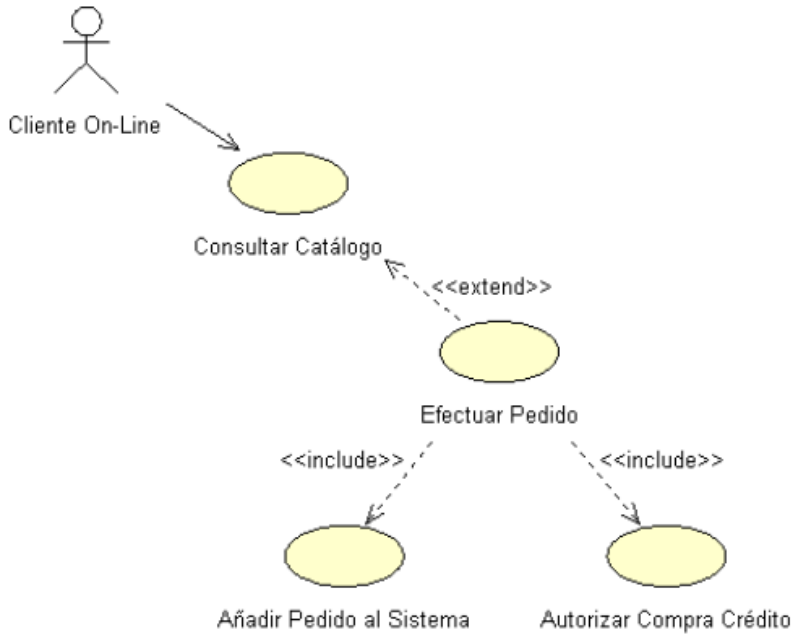


Figura 19: Diagrama de Caso de Uso

2.1.3.7 Diagrama de estados

Un diagrama de estados muestra una máquina de estados, que consta de estados, transiciones, eventos y actividades. Los diagramas de estados cubren la vista dinámica de un sistema, destacando el flujo de control entre estados, es decir, los estados potenciales de los objetos y las transiciones entre esos estados. Gráficamente, un estado se representa como un rectángulo con las esquinas redondeadas y una transición como una línea continua dirigida (véase figura 20). Estos diagramas son especialmente importantes en el modelado del comportamiento de una interfaz, una clase o una colaboración (realización de un caso de uso), y resaltan el comportamiento dirigido por eventos de un objeto.

Por tanto, mientras que una interacción modela una sociedad de objetos que colaboran para llevar a cabo alguna acción, una máquina de estados modela la vida de un único objeto, bien sea una instancia de una clase, un caso de uso o un sistema completo.

Una máquina de estados, como se definió en su momento, especifica las secuencias de estados por las que pasa un objeto a lo largo de su vida en respuesta a eventos, junto con sus respuestas a esos eventos. En este sentido, un objeto puede estar expuesto a varios tipos de eventos, tales como una señal, la invocación de una operación (llamada), la creación o distribución del objeto, al paso del tiempo o el cambio en alguna condición. Como respuesta a estos eventos, el objeto reacciona con alguna acción, que produce un cambio en el estado del objeto o devuelve algún valor.

Un estado es una condición o situación en la vida de un objeto la cual satisface alguna condición, realiza alguna actividad o espera algún evento. Un evento es la especificación de un acontecimiento significativo que ocupa un lugar en el tiempo y en el espacio. En el contexto de las máquinas de estados, un evento es la aparición de un estímulo que puede activar una transición de estado. Una transición es una relación entre dos estados que indica que un objeto que esté en el primer estado realizará ciertas acciones y entrará en el segundo estado cuando ocurra un evento especificado y se satisfagan unas condiciones determinadas. Una actividad es una ejecución en curso, dentro de una máquina de estados.

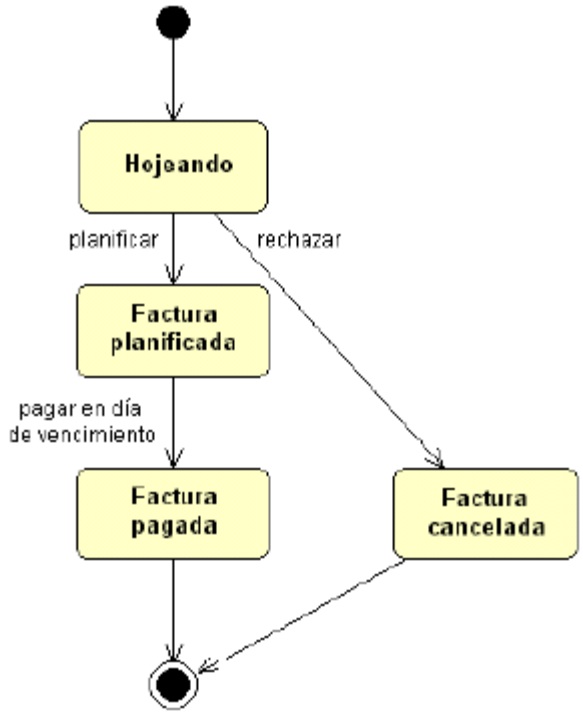


Figura 20: Diagrama de Estado

2.1.3.8 Diagrama de actividades

Un diagrama de actividades es un tipo especial de diagrama de estados que muestra el flujo de actividades dentro de un sistema (figura 21).

Concretando aún más, se puede decir que un diagrama de actividades es un caso especial de máquina de estados en la que los diferentes estados reflejan estados de actividad, actividades que tienen lugar dentro del objeto (una instancia de una clase, un caso de uso, o un sistema completo). Las actividades producen alguna acción, compuesta de computaciones ejecutables que producen un cambio en el estado del sistema o el retorno de un valor.

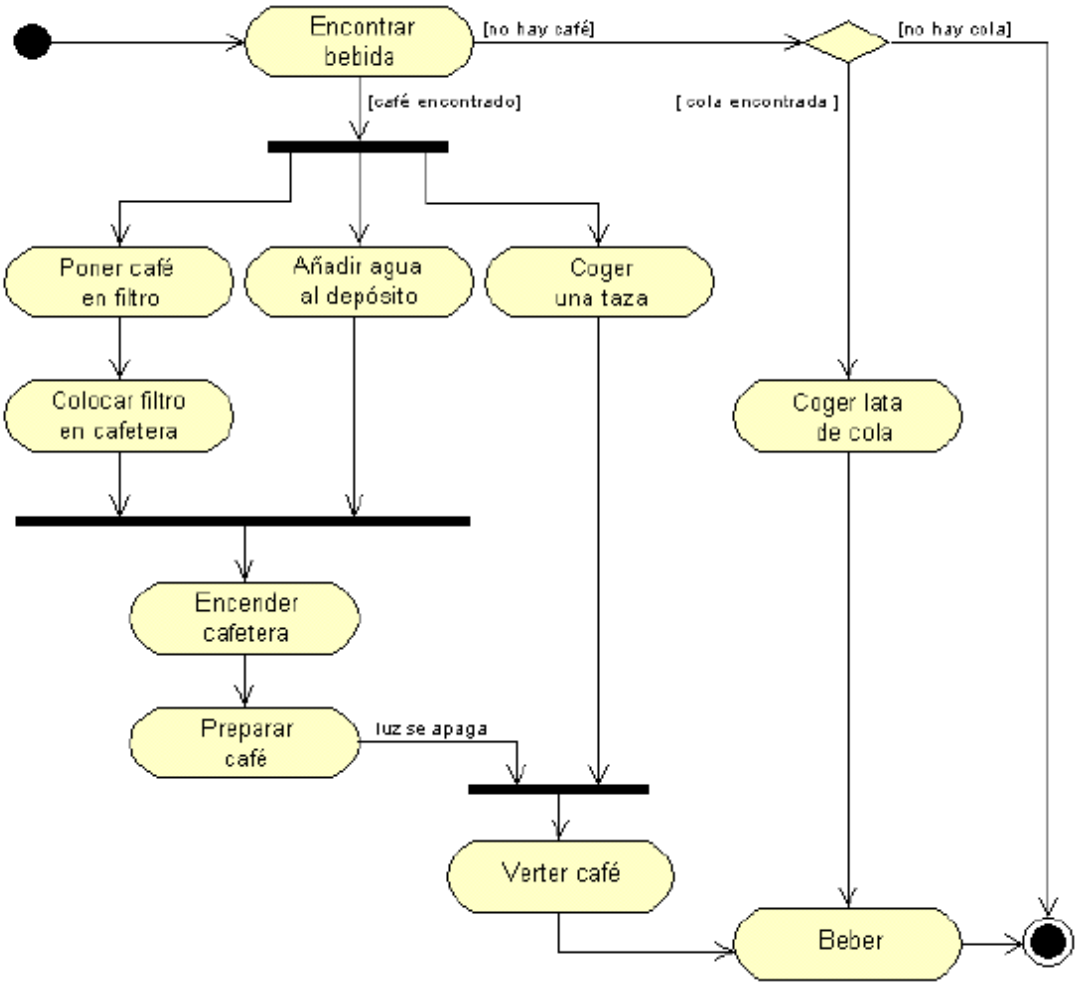


Figura 21: Diagrama de actividades [OMG03]

2.1.3.9 Diagrama de componentes

Un diagrama de componentes muestra la organización y las dependencias entre un conjunto de componentes. Es un diagrama que permite representar la estructura de ciertos aspectos físicos de los sistemas orientados a objetos, concretamente, la organización y dependencias entre los electos físicos que residen en un nodo, es decir, ficheros que representan el empaquetamiento físico de clases, interfaces y colaboraciones.

Los diagramas de componentes se utilizan para modelar la vista de implementación estática de un sistema, teniendo en cuenta que un diagrama de componentes individual es normalmente sólo una presentación particular de dicha vista. Esto significa que un único diagrama de componentes no necesita capturarlo todo sobre la vista de implementación del sistema. Un ejemplo de diagrama de componentes se muestra (Figura 22).

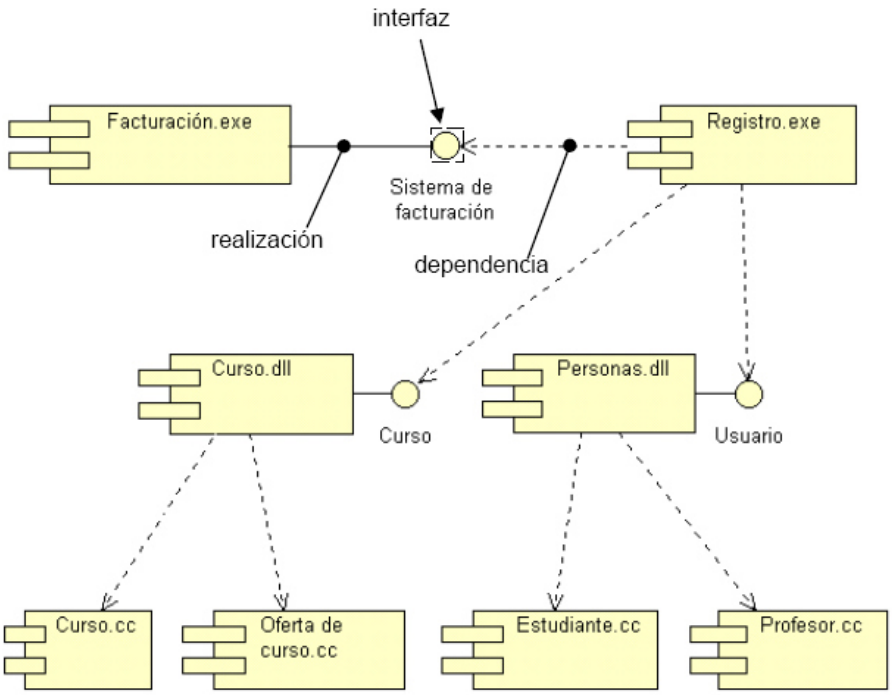


Figura 22: Diagrama de componentes [OMG03]

2.1.3.10 Diagrama de despliegue

Un Diagrama de despliegue muestra la configuración de nodos de procesamiento en tiempo de ejecución y los componentes que residen en ellos. Se relacionan con los diagramas de componentes en que un nodo incluye, por lo común, uno o más componentes. Los nodos representan el hardware sobre el que se despliegan y ejecutan esos componentes.

Los diagramas de despliegue se utilizan para modelar la vista de despliegue estática de un sistema, que implica modelar la topología de hardware sobre el que se ejecuta el sistema. En este sentido, si se desarrolla un software que reside en una máquina e interactúa sólo con dispositivos estándar en esa máquina, que ya son gestionados por el sistema operativo (por ejemplo, el teclado, la pantalla y el módem de un ordenador personal), se pueden ignorar los diagramas de despliegue. Por otro lado, si se desarrolla un software que interactúa con dispositivos que normalmente no gestiona el sistema operativo o si el sistema está distribuido físicamente por varios procesadores, entonces la utilización de los diagramas de despliegue ayuda a razonar sobre la correspondencia entre el software y el hardware del sistema.

Los distintos nodos implicados en un diagrama de despliegue se conectan a partir de relaciones de asociación. En este contexto, las conexiones representan enlaces físicos

(normalmente bidireccionales), como es el caso de una conexión directa mediante cable o indirecta por vía satélite.

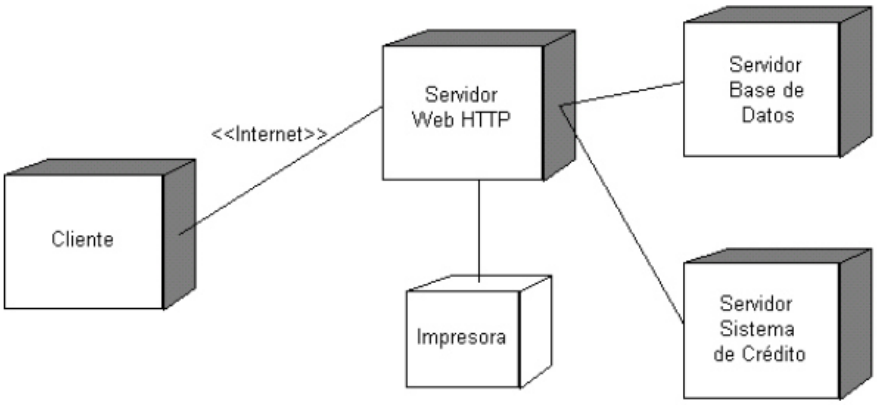


Figura 23: Diagrama de Despliegue

Un diagrama de despliegue también puede contener componentes, cada uno de los cuales debe residir en algún nodo. En ese caso, también se incluyen en el diagrama las relaciones de dependencia que existen entre dichos componentes [OMG01].

2.2 UML y su relación con los procesos de desarrollo de software

Un proceso de desarrollo de software es un método de organizar las actividades relacionadas con la creación, presentación y mantenimiento de los sistemas de software. El lenguaje UML, no define un proceso oficial de desarrollo, en realidad UML se combina con un proceso de desarrollo para obtener un producto final (ver Figura 24). Craig Larman [Larman99] da dos razones importantes que explican esto:

- Aumentar la posibilidad de una aceptación generalizada de la notación estándar del modelado sin la obligación de adoptar un proceso oficial.
- La esencia de un proceso apropiado admite mucha variación y depende de las habilidades del personal, de la razón investigación-desarrollo, de la naturaleza del problema y de las herramientas.

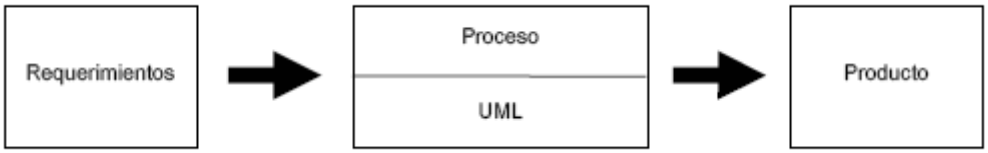


Figura 24: UML y el proceso de desarrollo [Larman99]

2.3 Proceso Unificado de Desarrollo de Software

Uno de los procesos más ocupados y recomendados para trabajar con UML es el Proceso Unificado de Desarrollo de Software (The Unified Software Development Process). Este proceso fue elaborado por los creadores del UML (Jacobson, Booch y Rumbaugh).

Sus características principales son [Booch00+]:

- Es dirigido por los casos de uso; acciones realizadas (interacción) entre los usuarios y el sistema.
- Se centra en el diseño de una arquitectura central, la cual guía el proceso de construcción de software.
- Es un proceso que utiliza un desarrollo iterativo e incremental;
    - Las iteraciones son controladas sobre los diferentes pasos del proyecto.
    - Es incremental porque en cada iteración el software se va ampliando y mejorando.

2.3.1 Proceso dirigido por caso de uso

Según [Kruchten00], los Casos de Uso son una técnica de captura de requisitos que fuerza a pensar en términos de importancia para el usuario y no sólo en términos de funciones que sería bueno contemplar. Se define un Caso de Uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los Casos de Uso representan los requisitos funcionales del sistema.

En RUP los Casos de Uso no son sólo una herramienta para especificar los requisitos del sistema. También guían su diseño, implementación y prueba. Los Casos de Uso constituyen un elemento integrador y una guía del trabajo como se muestra en la Figura 25

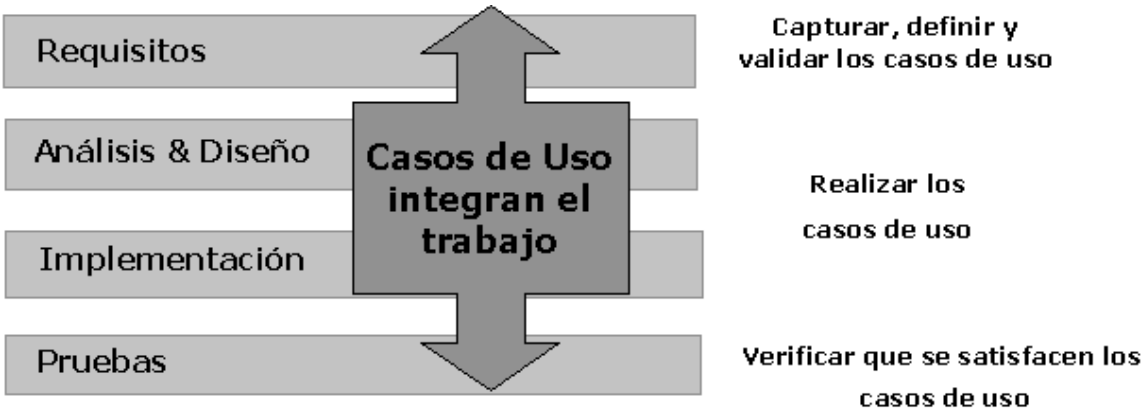


Figura 25: Los Casos de Uso integran el trabajo

Los Casos de Uso no sólo inician el proceso de desarrollo sino que proporcionan un hilo conductor, permitiendo establecer trazabilidad entre los artefactos que son generados en las diferentes actividades del proceso de desarrollo.

Como se muestra en la Figura 26, basándose en los Casos de Uso se crean los modelos de análisis y diseño, luego la implementación que los lleva a cabo, y se verifica que efectivamente el producto implemente adecuadamente cada Caso de Uso. Todos los modelos deben estar sincronizados con el modelo de Casos de Uso.

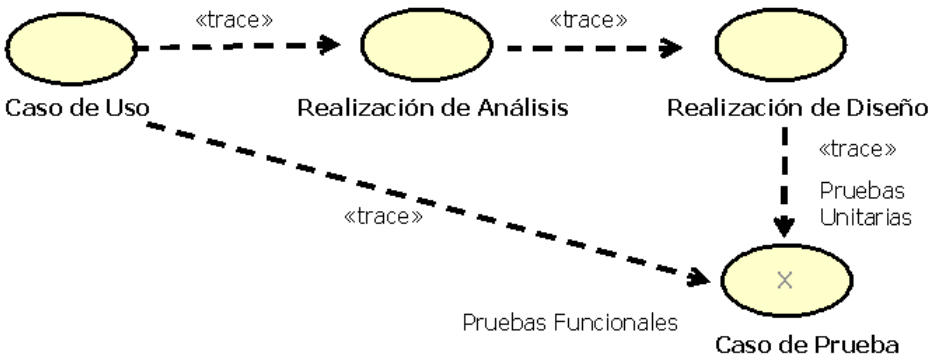


Figura 26: Trazabilidad a partir de los Casos de Uso

2.3.2 Proceso centrado en la arquitectura

La arquitectura de un sistema es la organización o estructura de sus partes más relevantes, lo que permite tener una visión común entre todos los involucrados (desarrolladores y usuarios) y una perspectiva clara del sistema completo, necesaria para controlar el desarrollo [Kruchten00].

La arquitectura involucra los aspectos estáticos y dinámicos más significativos del sistema, está relacionada con la toma de decisiones que indican cómo tiene que ser

construido el sistema y ayuda a determinar en qué orden. Además la definición de la arquitectura debe tomar en consideración elementos de calidad del sistema, rendimiento, reutilización y capacidad de evolución por lo que debe ser flexible durante todo el proceso de desarrollo. La arquitectura se ve influenciada por la plataforma software, sistema operativo, gestor de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados. Muchas de estas restricciones constituyen requisitos no funcionales del sistema.

En el caso de RUP además de utilizar los Casos de Uso para guiar el proceso se presta especial atención al establecimiento temprano de una buena arquitectura que no se vea fuertemente impactada ante cambios posteriores durante la construcción y el mantenimiento.

Cada producto tiene tanto una función como una forma. La función corresponde a la funcionalidad reflejada en los Casos de Uso y la forma la proporciona la arquitectura. Existe una interacción entre los Casos de Uso y la arquitectura, los Casos de Uso deben encajar en la arquitectura cuando se llevan a cabo y la arquitectura debe permitir el desarrollo de todos los Casos de Uso requeridos, actualmente y en el futuro. Esto provoca que tanto arquitectura como Casos de Uso deban evolucionar en paralelo durante todo el proceso de desarrollo de software.

En la Figura 27 se ilustra la evolución de la arquitectura durante las fases de RUP. Se tiene una arquitectura más robusta en las fases finales del proyecto. En las fases iniciales lo que se hace es ir consolidando la arquitectura por medio de baselines y se va modificando dependiendo de las necesidades del proyecto.

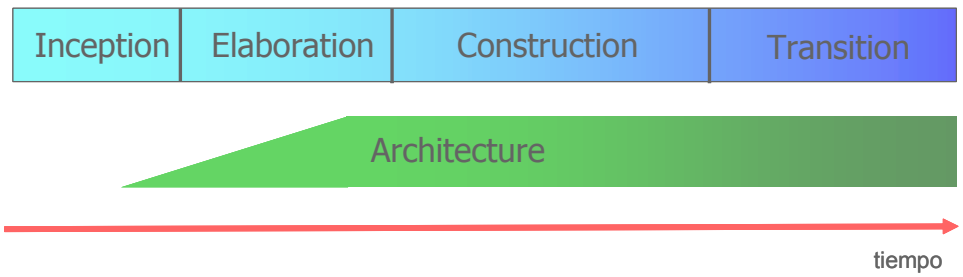


Figura 27: Evolución de la arquitectura del sistema

Es conveniente ver el sistema desde diferentes perspectivas para comprender mejor el diseño por lo que la arquitectura se representa mediante varias vistas que se centran en aspectos concretos del sistema, abstrayéndose de los demás. Para RUP, todas las vistas juntas forman el llamado modelo 4+1 de la arquitectura [Kruchten95], el cual recibe este nombre porque lo forman las vistas lógica, de implementación, de proceso y de despliegue, más la de Casos de Uso que es la que da cohesión a todas.

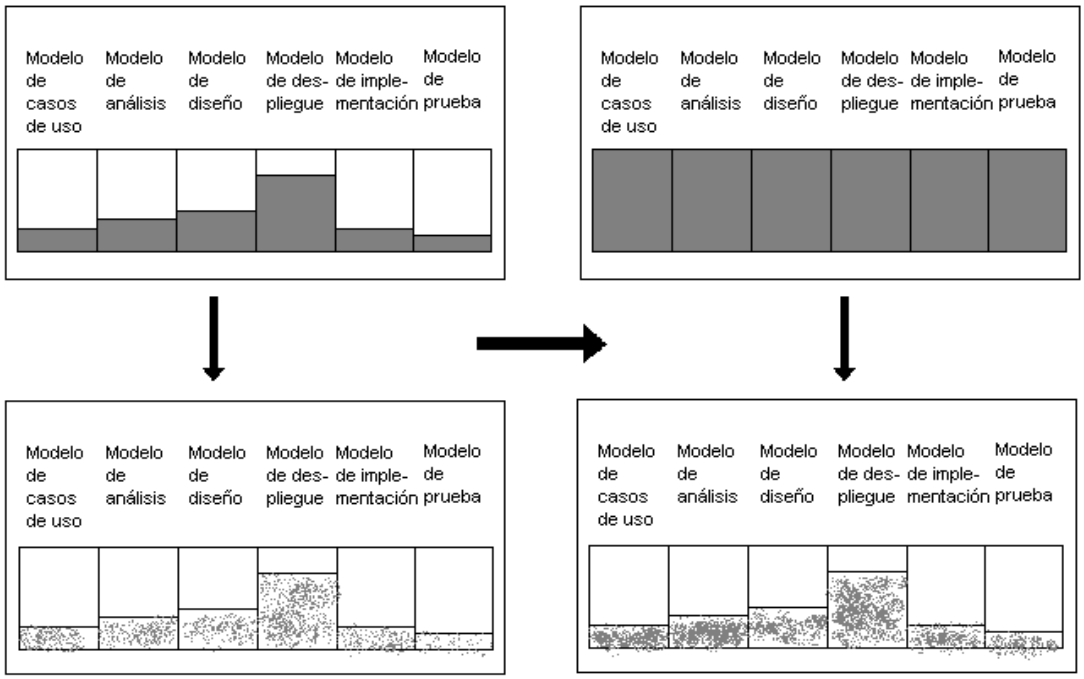


Figura 28: Los modelos se completan, la arquitectura no cambia drásticamente

Al final de la fase de elaboración se obtiene una baseline1 de la arquitectura donde fueron seleccionados una serie de Casos de Uso arquitectónicamente relevantes (aquellos que ayudan a mitigar los riesgos más importantes, aquellos que son los más importantes para el usuario y aquellos que cubran las funcionalidades significativas)

Como se observa en la Figura 28, durante la construcción los diversos modelos van desarrollándose hasta completarse (según se muestra con las formas rellenas en la esquina superior derecha). La descripción de la arquitectura sin embargo, no debería cambiar significativamente (abajo a la derecha) debido a que la mayor parte de la arquitectura se decidió durante la elaboración. Se incorporan pocos cambios a la arquitectura (indicados con mayor densidad de puntos en la figura inferior derecha) [JBR00].

2.3.3 Desarrollo iterativo e incremental

Un ciclo de vida iterativo se basa en el agrandamiento y perfeccionamiento secuencial de un sistema a través de múltiples ciclos de desarrollo, análisis, diseño, implementación y pruebas. El sistema crece al incorporar nuevas funciones en cada ciclo de desarrollo. “En cada ciclo se aborda un conjunto relativamente pequeño de requerimientos, pasando por el análisis, el diseño, la construcción y las pruebas. El sistema va creciendo con cada ciclo que concluye” [Larman99]. (Ver figura 29)

Según [JBR00] el equilibrio correcto entre los Casos de Uso y la arquitectura es algo muy parecido al equilibrio de la forma y la función en el desarrollo del producto, lo cual se consigue con el tiempo. Para esto, la estrategia que se propone en RUP es tener un proceso iterativo e incremental en donde el trabajo se divide en partes más pequeñas o mini proyectos.

Ventajas del desarrollo iterativo e incremental

- Reducción de los riesgos basándose en una retroalimentación temprana.
- Mayor flexibilidad para manejar cambios nuevos o modificaciones a los mismos.
- La complejidad nunca resulta abrumadora.
- Se produce retroalimentación en una etapa temprana, porque la implementación se efectúa rápidamente con una parte pequeña del sistema.

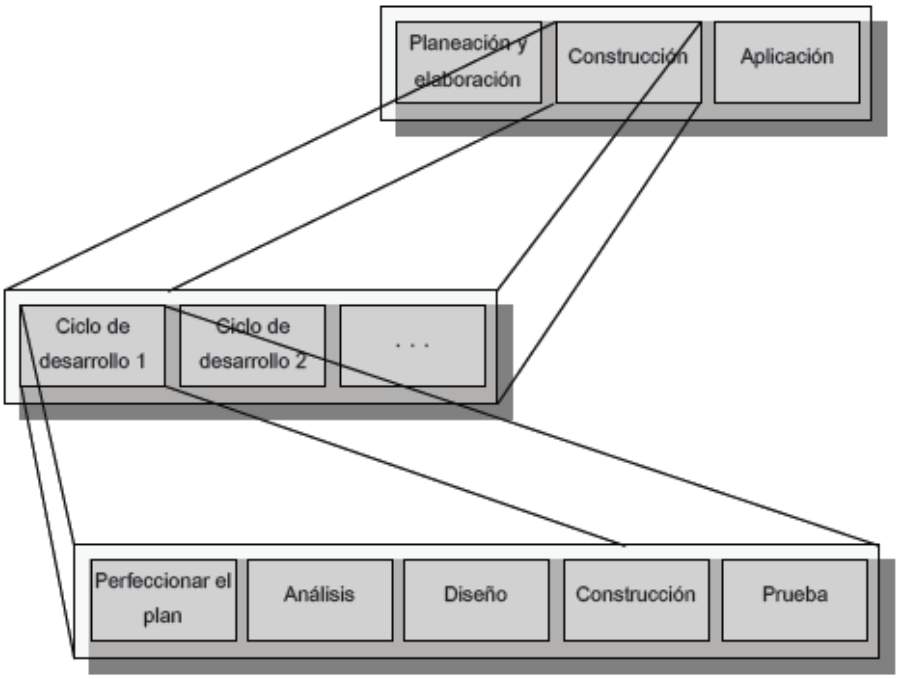


Figura 29: Ciclos iterativos de desarrollo [Larman99]

Cada iteración es un ciclo de desarrollo que termina en la liberación de una versión parcial del producto final y cada iteración pasa a través de todos los aspectos del desarrollo de software (ver figura 30)

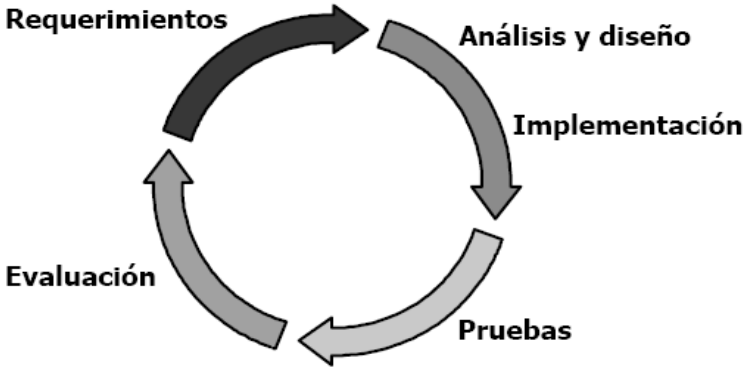


Figura 30: Desarrollo iterativo [Letelier02]

2.3.4 Estructura dinámica del proceso. Fases e iteraciones

En el Proceso Unificado de Desarrollo de Software existen cuatro fases por las que se debe pasar en el desarrollo iterativo (ver figura 31).

Cada fase se concluye con un hito bien definido, un punto en el tiempo en el cual se deben tomar ciertas decisiones críticas y alcanzar las metas clave antes de pasar a la siguiente fase, ese hito principal de cada fase se compone de hitos menores que podrían ser los criterios aplicables a cada iteración. Los hitos para cada una de las fases son: Inicio - Lifecycle Objectives, Elaboración - Lifecycle Architecture, Construcción - Initial Operational Capability, Transición - Product Release. Las fases y sus respectivos hitos se ilustran en la Figura 31.

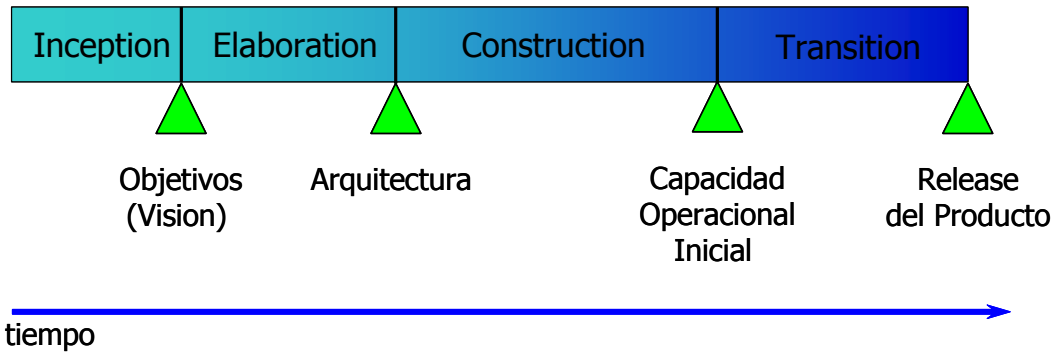


Figura 31: Fases e hitos en RUP

La duración y esfuerzo dedicado en cada fase es variable dependiendo de las características del proyecto. Sin embargo, la tabla 1 ilustra porcentajes frecuentes al respecto. Consecuente con el esfuerzo señalado.

	Inicio	Elaboración	Construcción	Transición
Esfuerzo	**5 %**	**20 %**	**65 %**	**10%**
Tiempo Dedicado	**10 %**	**30 %**	**50 %**	**10%**

Tabla 1: Distribución típicas de esfuerzo y tiempo

Estas fases son [Booch00+].

2.3.4.1 Concepción:

Durante la fase de inicio se define el modelo del negocio y el alcance del proyecto. Se identifican todos los actores y Casos de Uso, y se diseñan los Casos de Uso más esenciales (aproximadamente el 20% del modelo completo). Se desarrolla, un plan de negocio para determinar que recursos deben ser asignados al proyecto.

Los objetivos de estáa fase son [Kruchten00]:

- Establecer el ámbito del proyecto y sus límites.
- Encontrar los Casos de Uso críticos del sistema, los escenarios básicos que definen la funcionalidad.
- Mostrar al menos una arquitectura candidata para los escenarios principales.
- Estimar el coste en recursos y tiempo de todo el proyecto.
- Estimar los riesgos, las fuentes de incertidumbre.

Los resultados de la fase de inicio deben ser [Rational98]:

- Un documento de visión: Una visión general de los requerimientos del proyecto, características clave y restricciones principales.
- Modelo inicial de Casos de Uso (10-20% completado).
- Un glosario inicial: Terminología clave del dominio.
- El caso de negocio.
- Lista de riesgos y plan de contingencia.
- Plan del proyecto, mostrando fases e iteraciones.
- Modelo de negocio, si es necesario
- Prototipos exploratorios para probar conceptos o la arquitectura candidata.

Al terminar la fase de inicio se deben comprobar los criterios de evaluación para continuar:

- Todos los interesados en el proyecto coinciden en la definición del ámbito del sistema y las estimaciones de agenda.
- Entendimiento de los requisitos, como evidencia de la fidelidad de los Casos de Uso principales.
- Las estimaciones de tiempo, coste y riesgo son creíbles.
- Comprensión total de cualquier prototipo de la arquitectura desarrollado.
- Los gastos hasta el momento se asemejan a los planeados.

Si el proyecto no pasa estos criterios hay que plantearse abandonarlo o repensarlo profundamente.

2.3.4.2 **Elaboración:** En esta fase se especifican en detalle la mayoría de los casos de uso y se diseña la arquitectura central. Al final de esta fase, el desarrollador está en la posición de planear las actividades y estimar los recursos necesarios para completar el proyecto. Los casos de uso, la arquitectura y los planes deben estar lo suficientemente estables y los riesgos deben estar bien controlados.

Los objetivos de esta fase son [Rational98]:

- Definir, validar y cimentar la arquitectura.
- Completar la visión.
- Crear un plan fiable para la fase de construcción. Este plan puede evolucionar en sucesivas iteraciones. Debe incluir los costes si procede.
- Demostrar que la arquitectura propuesta soportará la visión con un coste razonable y en un tiempo razonable.

Al terminar deben obtenerse los siguientes resultados [Rational98]:

- Un modelo de Casos de Uso completa al menos hasta el 80%: todos los casos y actores identificados, la mayoría de los casos desarrollados.
- Requisitos adicionales que capturan los requisitos no funcionales y cualquier requisito no asociado con un Caso de Uso específico.
- Descripción de la arquitectura software.
- Un prototipo ejecutable de la arquitectura.
- Lista de riesgos y caso de negocio revisados.
- Plan de desarrollo para el proyecto.
- Un caso de desarrollo actualizado que especifica el proceso a seguir.
- Un manual de usuario preliminar (opcional).

En esta fase se debe tratar de abarcar todo el proyecto con la profundidad mínima. Sólo se profundiza en los puntos críticos de la arquitectura o riesgos importantes.

En la fase de elaboración se actualizan todos los productos de la fase de inicio.
Los criterios de evaluación de esta fase son los siguientes:

- La visión del producto es estable.
- La arquitectura es estable.
- Se ha demostrado mediante la ejecución del prototipo que los principales elementos de riesgo han sido abordados y resueltos.
- El plan para la fase de construcción es detallado y preciso. Las estimaciones son creíbles.
- Todos los interesados coinciden en que la visión actual será alcanzada si se siguen los planes actuales en el contexto de la arquitectura actual.
- Los gastos hasta ahora son aceptables, comparados con los previstos.
  Si no se superan los criterios de evaluación quizá sea necesario abandonar el proyecto o replanteárselo considerablemente.

2.3.4.3 **Construcción:** se empieza a construir cada parte del sistema siguiendo la arquitectura central antes diseñada. En esta fase, el sistema debe crecer lo suficiente para ponerlo a disposición de los usuarios para obtener opiniones y hacer los cambios pertinentes. La arquitectura debe ser estable, pero es posible que durante esta fase se descubran formas mejores de estructurarla; por lo que puede haber cambios menores. El sistema debe resolver satisfactoriamente todos los casos de uso que se hayan planeado al principio del proyecto; aunque puede haber algunos defectos que se resolverán en la fase de transición. La pregunta es: ¿Cumple el sistema con la mayoría de las necesidades de los usuarios para liberar una primera versión?

Los objetivos concretos según [Kruchten00] incluyen:

- Minimizar los costes de desarrollo mediante la optimización de recursos y evitando el tener que rehacer un trabajo o incluso desecharlo.
- Conseguir una calidad adecuada tan rápido como sea práctico.
- Conseguir versiones funcionales (alfa, beta, y otras versiones de prueba) tan rápido como sea práctico.

Los resultados de la fase de construcción deben ser [Rational98]:□□

- Modelos Completos (Casos de Uso, Análisis, Diseño, Despliegue e Implementación)
- Arquitectura íntegra (mantenida y mínimamente actualizada)
- Riesgos Presentados Mitigados

- Plan del Proyecto para la fase de Transición.
- Manual Inicial de Usuario (con suficiente detalle)
- Prototipo Operacional – beta
- Caso del Negocio Actualizado

Los criterios de evaluación de esta fase son los siguientes:

- El producto es estable y maduro como para ser entregado a la comunidad de usuario para ser probado.
- Todos los usuarios expertos están listos para la transición en la comunidad de usuarios.
- Son aceptables los gastos actuales versus los gastos planeados.

2.3.4.4 **Transición**: Esta fase cubre el periodo durante el cual el sistema se convierte en una versión beta. Durante esta fase, un cierto número de usuarios experimentados prueban el sistema y reportan los defectos y deficiencias encontradas. Los desarrolladores corrigen los problemas reportados e incorporan algunas mejoras sugeridas.

En [Kruchten00] se citan algunas de las cosas que puede incluir esta fase:

- Prueba de la versión Beta para validar el nuevo sistema frente a las expectativas de los usuarios
- Funcionamiento paralelo con los sistemas legados que están siendo sustituidos por nuestro proyecto.
- Conversión de las bases de datos operacionales.
- Entrenamiento de los usuarios y técnicos de mantenimiento.
- Traspaso del producto a los equipos de marketing, distribución y venta.

Los principales objetivos de esta fase son:

- Conseguir que el usuario se valga por si mismo.
- Un producto final que cumpla los requisitos esperados, que funcione y satisfaga suficientemente al usuario.

Los resultados de la fase de transición son [Rational98]:

- Prototipo Operacional
- Documentos Legales
- Caso del Negocio Completo
- Línea de Base del Producto completa y corregida que incluye todos los modelos del sistema

- Descripción de la Arquitectura completa y corregida
- Las iteraciones de esta fase irán dirigidas normalmente a conseguir una nueva versión.

Los criterios de evaluación de esta fase son los siguientes:

- El usuario se encuentra satisfecho.
- Son aceptables los gastos actuales versus los gastos planificados.

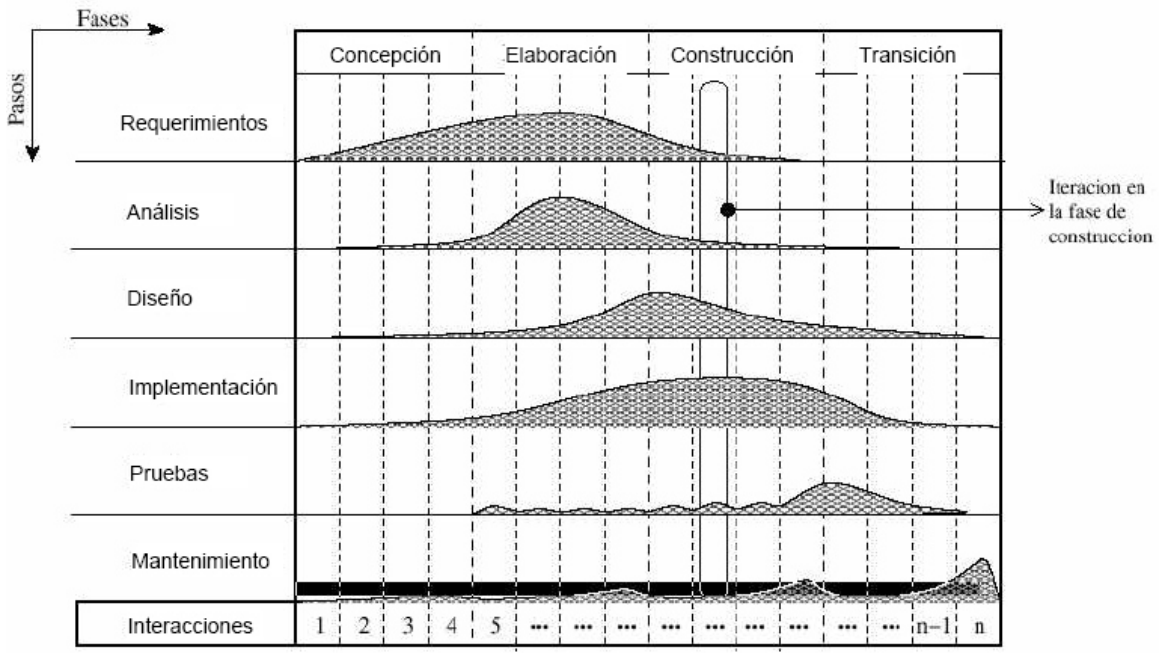


Figura 32: Fases en el desarrollo iterativo incremental [Booch99]

2.3.4.5 Estructura Estática del proceso. Roles, actividades, artefactos y flujos de trabajo

Un proceso de desarrollo de software define quién hace qué, cómo y cuándo. RUP define cuatro elementos los roles, que responden a la pregunta ¿Quién?, las actividades que responden a la pregunta ¿Cómo?, los productos, que responden a la pregunta ¿Qué? y los flujos de trabajo de las disciplinas que responde a la pregunta ¿Cuándo? (ver Figura 33 y 34) [Rational98].

Figura 33: Relación entre roles, actividades, artefactos

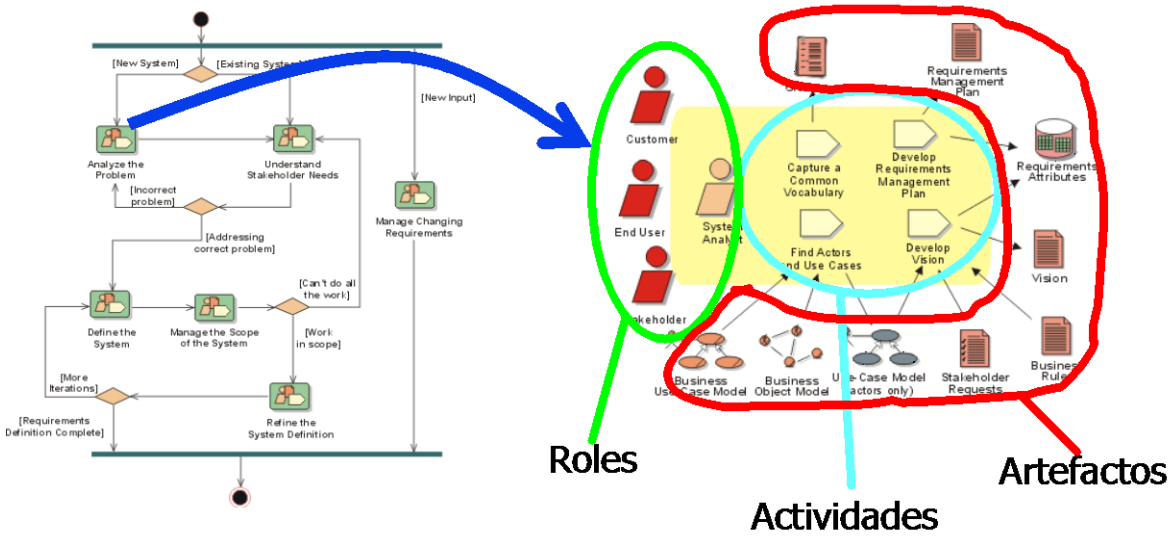


Figura 34: Detalle de un workflow mediante roles, actividades y artefactos.

2.3.4.5.1 Roles

Un rol define el comportamiento y responsabilidades de un individuo, o de un grupo de individuos trabajando juntos como un equipo. Una persona puede desempeñar diversos roles, así como un mismo rol puede ser representado por varias personas.

Las responsabilidades de un rol son tanto el llevar a cabo un conjunto de actividades como el ser el dueño de un conjunto de artefactos.

RUP define grupos de roles, agrupados por participación en actividades relacionadas. Estos grupos son [RSC02]:

Analistas:

- Analista de procesos de negocio.
- Diseñador del negocio.
- Analista de sistema.
- Especificador de requisitos.

Desarrolladores:

- Arquitecto de software.
- Diseñador
- Diseñador de interfaz de usuario
- Diseñador de cápsulas.
- Diseñador de base de datos.
- Implementador.
- Integrador.

Gestores:

- Jefe de proyecto
- Jefe de control de cambios.
- Jefe de configuración.
- Jefe de pruebas
- Jefe de despliegue
- Ingeniero de procesos
- Revisor de gestión del proyecto
- Gestor de pruebas.

Apoyo:

- Documentador técnico
- Administrador de sistema
- Especialista en herramientas
- Desarrollador de cursos
- Artista gráfico

Especialista en pruebas:

- Especialista en Pruebas (*tester*)
- Analista de pruebas
- Diseñador de pruebas

Otros roles:

- *Stakeholders*.
- Revisor
- Coordinación de revisiones

- Revisor técnico
- Cualquier rol

2.3.4.5.2 Actividades

Una actividad en concreto es una unidad de trabajo que una persona que desempeñe un rol puede ser solicitado a que realice. Las actividades tienen un objetivo concreto, normalmente expresado en términos de crear o actualizar algún producto.

2.3.4.5.3 Artefactos

Un producto o artefacto es un trozo de información que es producido, modificado o usado durante el proceso de desarrollo de software. Los productos son los resultados tangibles del proyecto, las cosas que va creando y usando hasta obtener el producto final [MMA].

- Un artefacto puede ser cualquiera de los siguientes [RSC02]:
- Un documento, como el documento de la arquitectura del software.
- Un modelo, como el modelo de Casos de Uso o el modelo de diseño.
- Un elemento del modelo, un elemento que pertenece a un modelo como una clase, un Caso de Uso o un subsistema.

2.3.4.5.4 Flujos de trabajo

Con la enumeración de roles, actividades y artefactos no se define un proceso, necesitamos contar con una secuencia de actividades realizadas por los diferentes roles, así como la relación entre los mismos. Un flujo de trabajo es una relación de actividades que nos producen unos resultados observables. A continuación se dará una explicación de cada flujo de trabajo.

2.3.4.5.5 Modelado del negocio

Con este flujo de trabajo pretendemos llegar a un mejor entendimiento de la organización donde se va a implantar el producto.

Los objetivos del modelado de negocio son [RSC02]:

- Entender la estructura y la dinámica de la organización para la cual el sistema va ser desarrollado (organización objetivo).
- Entender el problema actual en la organización objetivo e identificar potenciales mejoras.
- Asegurar que clientes, usuarios finales y desarrolladores tengan un entendimiento común de la organización objetivo.
- Derivar los requisitos del sistema necesarios para apoyar a la organización objetivo.

Para lograr estos objetivos, el modelo de negocio describe como desarrollar una visión de la nueva organización, basado en esta visión se definen procesos, roles y responsabilidades de la organización por medio de un modelo de Casos de Uso del

negocio y un Modelo de Objetos del Negocio. Complementario a estos modelos, se desarrollan otras especificaciones tales como un Glosario.

2.3.4.5.6 Requisitos

Este es uno de los flujos de trabajo más importantes, porque en él se establece qué tiene que hacer exactamente el sistema que construyamos. En esta línea los requisitos son el contrato que se debe cumplir, de modo que los usuarios finales tienen que comprender y aceptar los requisitos que especifiquemos.

Los objetivos del flujo de datos Requisitos es [RSC02]:

- Establecer y mantener un acuerdo entre clientes y otros *stakeholders* sobre lo que el sistema podría hacer.
- Proveer a los desarrolladores un mejor entendimiento de los requisitos del sistema.
- Definir el ámbito del sistema.
- Proveer una base para la planeación de los contenidos técnicos de las iteraciones.
- Proveer una base para estimar costos y tiempo de desarrollo del sistema.
- Definir una interfaz de usuarios para el sistema, enfocada a las necesidades y metas del usuario.

Los requisitos se dividen en dos grupos. Los requisitos funcionales representan la funcionalidad del sistema. Se modelan mediante diagramas de Casos de Uso. Los requisitos no funcionales representan aquellos atributos que debe exhibir el sistema, pero que no son una funcionalidad específica. Por ejemplo requisitos de facilidad de uso, fiabilidad, eficiencia, portabilidad, etc.

Para capturar los requisitos es preciso entrevistar a todos los interesados en el proyecto, no sólo a los usuarios finales, y anotar todas sus peticiones. A partir de ellas hay que descubrir lo que necesitan y expresarlo en forma de requisitos.

En este flujo de trabajo, y como parte de los requisitos de facilidad de uso, se diseña la interfaz gráfica de usuario. Para ello habitualmente se construyen prototipos de la interfaz gráfica de usuario que se contrastan con el usuario final.

2.3.4.5.7 Análisis y Diseño

El objetivo de este flujo de trabajo es traducir los requisitos a una especificación que describe cómo implementar el sistema.

Los objetivos del análisis y diseño son [RSC02]:

- Transformar los requisitos al diseño del futuro sistema.
- Desarrollar una arquitectura para el sistema.
- Adaptar el diseño para que sea consistente con el entorno de implementación, diseñando para el rendimiento.

El análisis consiste en obtener una visión del sistema que se preocupa de ver qué hace, de modo que sólo se interesa por los requisitos funcionales. Por otro lado el diseño es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales, en definitiva cómo cumple el sistema sus objetivos.

Al principio de la fase de elaboración hay que definir una arquitectura candidata: crear un esquema inicial de la arquitectura del sistema, identificar clases de análisis y actualizar las realizaciones de los Casos de Uso con las interacciones de las clases de análisis. Durante la fase de elaboración se va refinando esta arquitectura hasta llegar a su forma definitiva. En cada iteración hay que analizar el comportamiento para diseñar componentes. Además si el sistema usará una base de datos, habrá que diseñarla también, obteniendo un modelo de datos.

El resultado final más importante de este flujo de trabajo será el modelo de diseño. Consiste en colaboraciones de clases, que pueden ser agregadas en paquetes y subsistemas.

Otro producto importante de este flujo es la documentación de la arquitectura de software, que captura varias vistas arquitectónicas del sistema.

2.3.4.5.8 Implementación

En este flujo de trabajo se implementan las clases y objetos en ficheros fuente, binarios, ejecutables y demás. Además se deben hacer las pruebas de unidad: cada implementador es responsable de probar las unidades que produzca. El resultado final de este flujo de trabajo es un sistema ejecutable.

En cada iteración habrá que hacer lo siguiente:

- Planificar qué subsistemas deben ser implementados y en que orden deben ser integrados, formando el Plan de Integración.
- Cada implementador decide en que orden implementa los elementos del subsistema.
- Si encuentra errores de diseño, los notifica.
- Se prueban los subsistemas individualmente.
- Se integra el sistema siguiendo el plan.

La estructura de todos los elementos implementados forma el modelo de implementación. La integración debe ser incremental, es decir, en cada momento sólo se añade un elemento. De este modo es más fácil localizar fallos y los componentes se prueban más a fondo. En fases tempranas del proceso se pueden implementar prototipos para reducir el riesgo. Su utilidad puede ir desde ver si el sistema es viable desde el principio, probar tecnologías o diseñar la interfaz de

usuario. Los prototipos pueden ser exploratorios (desechables) o evolutivos. Estos últimos llegan a transformarse en el sistema final.

2.3.4.5.9 Pruebas

Este flujo de trabajo es el encargado de evaluar la calidad del producto que estamos desarrollando, pero no para aceptar o rechazar el producto al final del proceso de desarrollo, sino que debe ir integrado en todo el ciclo de vida.

Esta disciplina brinda soporte a las otras disciplinas. Sus objetivos son [RSC02]:

- Encontrar y documentar defectos en la calidad del software.
- Generalmente asesora sobre la calidad del software percibida.
- Provee la validación de los supuestos realizados en el diseño y especificación de requisitos por medio de demostraciones concretas.
- Verificar las funciones del producto de software según lo diseñado.
- Verificar que los requisitos tengan su apropiada implementación.

Las actividades de este flujo comienzan pronto en el proyecto con el plan de prueba (el cual contiene información sobre los objetivos generales y específicos de las prueba en el proyecto, así como las estrategias y recursos con que se dotará a esta tarea), o incluso antes con alguna evaluación durante la fase de inicio, y continuará durante todo el proyecto.

El desarrollo del flujo de trabajo consistirá en planificar que es lo que hay que probar, diseñar cómo se va a hacer, implementar lo necesario para llevarlos a cabo, ejecutarlos en los niveles necesarios y obtener los resultados, de forma que la información obtenida nos sirva para ir refinando el producto a desarrollar.

2.3.4.5.10 Despliegue

El objetivo de este flujo de trabajo es producir con éxito distribuciones del producto y distribuirlo a los usuarios. Las actividades implicadas incluyen:

- Probar el producto en su entorno de ejecución final.
- Empaquetar el software para su distribución.
- Distribuir el software.
- Instalar el software.
- Proveer asistencia y ayuda a los usuarios.
- Formar a los usuarios y al cuerpo de ventas.
- Migrar el software existente o convertir bases de datos.

Este flujo de trabajo se desarrolla con mayor intensidad en la fase de transición, ya que el propósito del flujo es asegurar una aceptación y adaptación sin complicaciones del software por parte de los usuarios. Su ejecución inicia en fases

anteriores, para preparar el camino, sobre todo con actividades de planificación, en la elaboración del manual de usuario y tutoriales.

2.3.4.5.11 Gestión del proyecto

La Gestión del proyecto es el arte de lograr un balance al gestionar objetivos, riesgos y restricciones para desarrollar un producto que sea acorde a los requisitos de los clientes y los usuarios.

Los objetivos de este flujo de trabajo son:

- Proveer un marco de trabajo para la gestión de proyectos de software intensivos.
- Proveer guías prácticas realizar planeación, contratar personal, ejecutar y monitorear el proyecto.
- Proveer un marco de trabajo para gestionar riesgos.

La planeación de un proyecto posee dos niveles de abstracción: un plan para las fases y un plan para cada iteración.

2.3.4.5.12 Configuración y control de cambios

La finalidad de este flujo de trabajo es mantener la integridad de todos los artefactos que se crean en el proceso, así como de mantener información del proceso evolutivo que han seguido.

2.3.4.5.13 Entorno

La finalidad de este flujo de trabajo es dar soporte al proyecto con las adecuadas herramientas, procesos y métodos. Brinda una especificación de las herramientas que se van a necesitar en cada momento, así como definir la instancia concreta del proceso que se va a seguir.

En concreto las responsabilidades de este flujo de trabajo incluyen:

- Selección y adquisición de herramientas
- Establecer y configurar las herramientas para que se ajusten a la organización.
- Configuración del proceso.
- Mejora del proceso.
- Servicios técnicos.

El principal artefacto que se usa en este flujo de trabajo es el *caso de desarrollo* que especifica para el proyecto actual en concreto, como se aplicará el proceso, que productos se van a utilizar y como van a ser utilizados. Además se tendrán que definir las guías para los distintos aspectos del proceso, como pueden ser el modelado del negocio y los Casos de Uso, para la interfaz de usuario, el diseño, la programación, el manual de usuario.

2.4 Herramienta Case.

2.4.1 Concepto.

La ingeniería de sistemas asistida por ordenador es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su objetivo es acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas [Kendall05].

Según, es la automatización de metodologías paso a paso para el desarrollo de software y de sistemas para reducir la cantidad de trabajo repetitivo que el diseñador necesita hacer. Su adopción puede dejar libre al diseñador para tareas más creativas de solución de problemas. También facilitan la creación de documentación estructurada y la coordinación de los esfuerzos de desarrollo del equipo [Laudon04].

Cuando se hace la planificación de la base de datos, la primera etapa del ciclo de vida de las aplicaciones de bases de datos, también se puede escoger una herramienta CASE (Computer-Aided Software Engineering) que permita llevar a cabo el resto de tareas del modo más eficiente y efectivo posible. Una herramienta CASE suele incluir:

- Un diccionario de datos para almacenar información sobre los datos de la aplicación de bases de datos.
- Herramientas de diseño para dar apoyo al análisis de datos.
- Herramientas que permitan desarrollar el modelo de datos corporativo, así como los esquemas conceptual y lógico.
- Herramientas para desarrollar los prototipos de las aplicaciones.

El uso de las herramientas CASE puede mejorar la productividad en el desarrollo de una aplicación de bases de datos [Kendall05].

2.4.2 Historia.

El primer antecesor de las herramientas CASE aparece en los años sesenta. En el proyecto ISDOS se desarrolla un lenguaje denominado PSL (Problem Statement Language; Lenguaje de sentencias de problemas) y PSA (Problem Statement Analizer; Analizador de sentencias de problemas). El primer lenguaje permitía a los usuarios definir problemas y el segundo asociaba los problemas de los usuarios con posibles soluciones. La finalidad del proyecto ISDOS consistía en crear un diccionario computarizado.

Hasta el año 1984 no encontramos la primera herramienta CASE tal como la conocemos hoy en día. Se trata de “Excelator” y estaba programada para PC [Garzon03+].

2.4.3 Tecnología Case.

La tecnología CASE supone la automatización del desarrollo del software, contribuyendo a mejorar la calidad y la productividad en el desarrollo de sistemas de información y se plantean los siguientes objetivos:

- Permitir la aplicación práctica de metodologías estructuradas, las cuales al ser realizadas con una herramienta se consigue agilizar el trabajo.
- Facilitar la realización de prototipos y el desarrollo conjunto de aplicaciones.
- Simplificar el mantenimiento de los programas.
- Mejorar y estandarizar la documentación.
- Aumentar la portabilidad de las aplicaciones.
- Facilitar la reutilización de componentes software.
- Permitir un desarrollo y un refinamiento visual de las aplicaciones, mediante la utilización de gráficos.
- Automatizar:
    - El desarrollo del software.
    - La documentación.
    - La generación del código.
    - El chequeo de errores.
    - La gestión del proyecto.
- Permitir:
    - La reutilización del software.
    - La portabilidad del software.
    - La estandarización de la documentación.

2.4.4 Componentes

De una forma esquemática podemos decir que una herramienta CASE se compone de los siguientes elementos:

- Repositorio (diccionario) donde se almacenan los elementos definidos o creados por la herramienta, y cuya gestión se realiza mediante el apoyo de un SGBD o de un sistema de gestión de ficheros.
- Meta modelo (no siempre visible), que constituye el marco para la definición de las técnicas y metodologías soportadas por la herramienta.
- Carga o descarga de datos, son facilidades que permiten cargar el repertorio de la herramienta CASE con datos provenientes de otros sistemas, o bien generar a partir de la propia herramienta esquemas de base de datos, programas, etc. que pueden, a su vez,

alimentar otros sistemas. Este elemento proporciona así un medio de comunicación con otras herramientas.

- Comprobación de errores, facilidades que permiten llevar a cabo un análisis de la exactitud, integridad y consistencia de los esquemas generados por la herramienta.
- Interfaz de usuario, que constará de editores de texto y herramientas de diseño gráfico que permitan, mediante la utilización de un sistema de ventanas, iconos y menús, con la ayuda del ratón, definir los diagramas, matrices, etc. que incluyen las distintas metodologías [Rob02].

2.4.5 Clasificación

Se han clasificado muchas herramientas CASE en cuanto a si apoyan las actividades de interfaz o de servicios de apoyo del proceso de desarrollo de sistemas. Las herramientas de interfaz CASE se enfocan en la captura de información de análisis y diseño en las primeras etapas del desarrollo de sistemas, en tanto que las herramientas de servicios de apoyo CASE se encarga de las actividades de codificación, prueba y mantenimiento. Las herramientas de servicios de apoyo ayudan a convertir automáticamente las especificaciones en código de programa.

Las herramientas CASE enlazan automáticamente los elementos de datos con los procesos en que se usan. Si un diagrama de flujo de datos se cambia de un proceso a otro, los elementos en el diccionario de datos se alterarían automáticamente para reflejar el cambio en el diagrama [Laudon04].

2.4.6 Estado actual.

En las últimas décadas se ha trabajado en el área de desarrollo de sistemas para encontrar técnicas que permitan incrementar la productividad y el control de calidad en cualquier proceso de elaboración de software, y hoy en día la tecnología CASE (Computer Aided Software Engineering) reemplaza al papel y al lápiz por el ordenador para transformar la actividad de desarrollar software en un proceso automatizado.

La tecnología CASE supone la informatización de la informática, es decir la automatización del desarrollo del software, contribuyendo así a elevar la productividad y la calidad de en el desarrollo de los sistemas de información de forma análoga a lo que suponen las técnicas CAD/CAM en el área de fabricación.

En este nuevo enfoque que persigue mejorar la calidad del software e incrementar la productividad en el proceso de desarrollo del mismo, se plantean los siguientes objetivos:

- Permitir la aplicación práctica de metodologías, lo que resulta muy difícil sin emplear herramientas.
- Facilitar la realización de prototipos y el desarrollo conjunto de aplicaciones.

- Simplificar el mantenimiento del software.
- Mejorar y estandarizar la documentación.
- Aumentar la portabilidad de las aplicaciones.
- Facilitar la reutilización de componentes de software.
- Permitir un desarrollo y un refinamiento (visual) de las aplicaciones, mediante la utilización de controles gráficos (piezas de código reutilizables).

2.4.7 Integración de las Herramientas Case en el futuro

Las herramientas CASE evolucionan hacia tres tipos de integración [Kendall05]:

- La integración de datos permite disponer de herramientas CASE con diferentes estructuras de diccionarios locales para el intercambio de datos.
- La integración de presentación confiere a todas las herramientas CASE el mismo aspecto.
- La integración de herramientas permite disponer de herramientas CASE capaces de invocar a otras CASE de forma automática.

2.4.8 Factores asociados a la implantación de las Herramientas Case.

La difusión de las innovaciones en esta área ha comenzado a estudiarse a partir de los años 1940. Por ello, existen estudios teóricos al respecto, realizándose evaluaciones, adopción e implementación tecnológica.

Existe un amplio cuerpo de investigaciones disponibles sobre la adopción de innovaciones. Muchos de los estudios sobre innovación se han analizado bajo dos perspectivas: adopción y difusión (Kimberly, 1981). Mientras unos estudios usan la perspectiva de la adopción para evaluar la receptividad y los cambios de la organización o sociedad por la innovación, otros usan la perspectiva de la difusión para intentar entender por qué y cómo se difunde y qué características generales o principales de la innovación son aceptadas.

2.5 El Modelo Constructivo de Costo (COCOMO)

Barry Boehm [Pressman93], en su libro sobre "Economía de la Ingeniería del Software", menciona una escala de modelos de estimación de software con el nombre de COCOMO, por COnstrucive COst MOdel (MOdelo COnstructivo de COsto). La escala de modelos de Boehm incluye:

- Modelo 1. El modelo COCOMO *básico* calcula el esfuerzo (y el costo) del desarrollo de software en función del tamaño del programa, expresado en las líneas estimadas de código (LDC).
- Modelo 2, El modelo COCOMO *intermedio* calcula el esfuerzo del desarrollo de software en función del tamaño del programa y de un conjunto de "conductores de

costo" que incluyen la evaluación subjetiva del producto, del hardware, del personal y de los atributos del proyecto.

- Modelo 3, El modelo COCOMO *avanzado* incorpora todas las características de la versión intermedia y lleva a cabo una evaluación del impacto de los conductores de costo en cada fase (análisis, diseño, etc.) del transcurso de ingeniería del software.

Los modelos COCOMO están establecidos para tres prototipos de proyectos de software que empleando la terminología de Boehm son: (1) *modo orgánico*: aquellos proyectos de software que son respectivamente pequeños y sencillos en donde trabajan pequeños equipos que poseen buena experiencia en la aplicación, sobre un conjunto de requisitos poco rígidos; (2) *modo semiacoplado*: son los proyectos de software intermedios hablando de tamaño y complejidad, en donde los equipos tienen diversos niveles de experiencia, y además deben satisfacer requerimientos poco o medio rígidos; (3) *modo empotrado*: son proyectos de software que deben ser desarrollados en un conjunto de hardware, software y restricciones operativas muy restringido.

Las ecuaciones del COCOMO básico tienen la siguiente forma: [Fenton91]

$$E = a_b \, KLDC^{b}{}_{b}$$

$$D = C_b \, E^{d}{}_{b}$$

donde *E* es el esfuerzo aplicado en personas-mes, *D* es el tiempo de desarrollo en meses cronológicos y KLDC es el número estimado de líneas de código distribuidas (en miles) para el proyecto. Los coeficientes a_b y C_b y los exponentes d_b y b_b, con valores constantes se muestran en la Tabla 2 [Fenton91].

Tabla 2: Modelo COCOMO básico, valores constantes [Fenton91]

Proyecto de Software	a_b	$^{b}{}_{b}$	c_b	$^{d}{}_{b}$
Orgánico	2.4	1.05	2.5	0.38
Semiacoplado	3.0	1.12	2.5	0.35
Empotrado	3.6	1.20	2.5	0.32

La ecuación del modelo COCOMO intermedio toma la forma:

$$E = a_i KLDC^{bi} * FAE$$

donde *E* es el esfuerzo aplicado en personas-mes y LDC es el número estimado de líneas de código distribuidas para el proyecto. El coeficiente a_i y el exponente b_i se muestran en la Tabla 4

Tabla 3: Modelo COCOMO intermedio, valores constantes [Fenton91]

Proyecto de Software	a_i	b_i
Orgánico	3.2	1.05
Semiacoplado	3.0	1.12
Empotrado	3.8	1.20

COCOMO es el modelo empírico más completo para la estimación del software publicado hasta la fecha. Sin embargo, deben tenerse en cuenta los propios comentarios de Boehm [Pressman98] sobre COCOMO (y por extensión, sobre todos los modelos):
Hoy en día un modelo de estimación de costos de software está bien fundado si puede evaluar tanto los costos de desarrollo de software en un 20 por ciento de los costos reales, así como un 70 por ciento del tiempo y ello en su propio terreno (o sea dentro de la clase de proyectos para los cuales ha sido calibrado), en realidad ésta no es la exactitud que aspiramos, pero es más que suficiente para facilitar el análisis económico de la ingeniería del software y también en la toma de decisiones.

Capitulo 3.- Estado del Arte

3.1. Introducción

Desde los inicios de la informática se han estado utilizando distintas maneras de representar los diseños de los sistemas de una manera más bien personal o con algún modelo gráfico, La falta de estandarización en la representación gráfica de un modelo impedía que los diseños gráficos realizados se pudieran compartir fácilmente entre distintos diseñadores, con este objetivo se creo el Lenguaje Unificado de Modelado (UML).

Tras la aceptación del paradigma orientado a objetos (OO) como el más adecuado para producir software de calidad, a principios de los noventa emergieron un buen número de métodos de desarrollo de software OO. En julio de 1993, Jacobson criticó en [Jacobson93] lo que el denominaba *guerra de métodos* y planteo la necesidad de llegar a una notación estándar de modelado, para evitar la confusión reinante y favorecer el uso de los métodos de software OO. El lenguaje UML comenzó a gestarse en octubre de 1994 [Booch99], cuando Rumbaugh se unió a la compañía *Racional* fundada por Booch (dos reputados investigadores en el área de metodología del software). EL objetivo era unificar dos métodos que habían desarrollado: el método Booch y el OMT (Object Modelling Tool). El primer borrador apareció en octubre de 1995. En esa misma época, Jacobson, se unió a *Rational* y se incluyeron los tres principales métodos: *Booch, Rumbaugh y Jacobson*. Estas tres personas son conocidas como los "tres amigos". El lenguaje Unificado de Modelado (UML, *Unified Modeling Language*) es el resultado de esa colaboración y de las aportaciones de las principales empresas de software.

UML fue adoptado en noviembre de 1997 por OMG (*Object Management Group*) como una de sus especificaciones y desde entonces se ha convertido en un estándar de facto para visualizar, especificar y documentar los modelos que se crean durante la aplicación de un proceso software [OMG01]. UML ha ejercido un gran impacto en la comunidad software, tanto a nivel de desarrollo como de investigación.

UML es ante todo un lenguaje. Un lenguaje proporciona un vocabulario y unas reglas para permitir una comunicación. En este caso, este lenguaje se centra en la representación gráfica de un sistema. Esto último es el objetivo de las metodologías de desarrollo.

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones:

- Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.

- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Este trabajo se centra en un Sistema de Registro de Exámenes ecográficos. Se han usados varios de los diagramas de UML, de modo que se muestre el uso de los mismos, enfocado desde una perspectiva práctica.

UML es solamente un lenguaje, por si solo no representa una metodología o proceso, es decir, se requiere de una guía de desarrollo. UML proporciona un vocabulario y reglas para la representación física y conceptual de un sistema [Booch99+], pero no define qué modelos deben ser creados y cuándo deben ser realizados. Estas especificaciones las realiza el proceso de desarrollo [Booch99+], un proceso deberá guiar al desarrollador en la identificación de qué artefactos producir, cuales actividades realizar y cuándo realizarlos, para que el resultado final de todas estas especificaciones coincidan en el desarrollo del modelado de un sistema.

3.2. Metodologías Propuestas.

EL UML es independiente de la fase, lo cual significa que el mismo lenguaje genérico y los mismos diagramas son utilizados para modelar cosas diferentes en diferentes fases; el lenguaje de modelaje solamente nos proporciona la habilidad de crear modelos de una manera expresiva y consistente; este trabajo debería ser gobernado por un método o un proceso que subraye los diferentes pasos a tomar y como son implementados esos paso. Es así que listamos una serie de metodologías propuesta que utilizan UML.

3.2.1. Rational Unified Process (RUP)

RUP es un proceso formal: Provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales (respetando cronograma y presupuesto). Fue desarrollado por Rational Software, y está integrado con toda la suite Rational de herramientas. Puede ser adaptado y extendido para satisfacer las necesidades de la organización que lo adopte. (Customización). Es guiado por casos de uso y centrado en la arquitectura, y utiliza UML como lenguaje de notación.

3.2.1.1. Fases

Las cuatro fases del ciclo de vida son:

- Concepción
- Elaboración

- Construcción
- Transición

3.2.1.2. Ventajas

- Evaluación en cada fase que permite cambios de objetivos
- Funciona bien en proyectos de innovación.
- Es sencillo, ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software.
- Seguimiento detallado en cada una de las fases.

3.2.1.3. Desventajas

- La evaluación de riesgos es compleja
- Excesiva flexibilidad para algunos proyectos
- Estamos poniendo a nuestro cliente en una situación que puede ser muy incómoda para él.
- Nuestro cliente deberá ser capaz de describir y entender a un gran nivel de detalle para poder acordar un alcance del proyecto con él.

3.2.2. Extreme Programming (XP)

Figura 35: Diagrama de XP

Es la más destacada de los procesos ágiles de desarrollo de software formulada por Kent Beck[Beck00]. La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.

Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los

requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Las características fundamentales del método son:

- Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- Programación por parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que la mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata.
- Frecuente interacción del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- Corrección de todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
- Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que en más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

  Ventajas

  - Apropiado para entornos volátiles
  - Estar preparados para el cambio, significa reducir su coste.
  - Planificación más transparente para nuestros clientes, conocen las fechas de entrega de funcionalidades. Vital para su negocio
  - Permitirá definir en cada iteración cuales son los objetivos de la siguiente
  - Permite tener realimentación de los usuarios muy útil.
  - La presión esta a lo largo de todo el proyecto y no en una entrega final

Desventajas

- Delimitar el alcance del proyecto con nuestro cliente

Para mitigar esta desventaja se plantea definir un alcance a alto nivel basado en la experiencia.

3.2.3. SCRUM

SCRUM es un proceso ágil y liviano que sirve para administrar y controlar el desarrollo de software. El desarrollo se realiza en forma iterativa e incremental (una iteración es un ciclo corto de construcción repetitivo). Cada ciclo o iteración termina con una pieza de software ejecutable que incorpora nueva funcionalidad. Las iteraciones en general tienen una duración entre 2 y 4 semanas. SCRUM se utiliza como marco para otras prácticas de ingeniería de software como RUP o Extreme Programming. Uno de los análisis más importantes de la metodología desemboco en un libro escrito por dos de sus creadores, Ken Schwaber y Mike Beedle [Schwaber 01].

Scrum se focaliza en priorizar el trabajo en función del valor que tenga para el negocio, maximizando la utilidad de lo que se construye y el retorno de inversión. Está diseñado especialmente para adaptarse a los cambios en los requerimientos, por ejemplo en un mercado de alta competitividad. Los requerimientos y las prioridades se revisan y ajustan durante el proyecto en intervalos muy cortos y regulares. De esta forma se puede adaptar en tiempo real el producto que se está construyendo a las necesidades del cliente. Se busca entregar software que realmente resuelva las necesidades, aumentando la satisfacción del cliente.

En SCRUM, el equipo se focaliza en una única cosa: construir software de calidad. Por el otro lado, la gestión de un proyecto SCRUM se focaliza en definir cuáles son las características que debe tener el producto a construir (qué construir, qué no y en qué orden) y en remover cualquier obstáculo que pudiera entorpecer la tarea del equipo de desarrollo. Se busca que los equipos sean lo más efectivos y productivos posible.

SCRUM tiene un conjunto de reglas muy pequeño y muy simple y está basado en los principios de inspección continua, adaptación, auto-gestión e innovación. El cliente se entusiasma y se compromete con el proyecto dado que ve crecer el producto iteración a iteración y encuentra las herramientas para alinear el desarrollo con los objetivos de negocio de su empresa.

Por otro lado, los desarrolladores encuentran un ámbito propicio para desarrollar sus capacidades profesionales y esto resulta en un incremento en la motivación de los integrantes del equipo.

La intención de SCRUM es la de maximizar la realimentación sobre el desarrollo pudiendo corregir problemas y mitigar riesgos de forma temprana. Su uso se está extendiendo cada vez más dentro de la comunidad de metodologías agiles, siendo combinado con otras – como XP – para completar sus carencias. Cabe mencionar que SCRUM no propone el uso de ninguna práctica de desarrollo en particular, sin embargo, es habitual emplearlo como un framework ágil de administración de proyectos que puede ser combinado con cualquiera de las metodologías mencionadas.

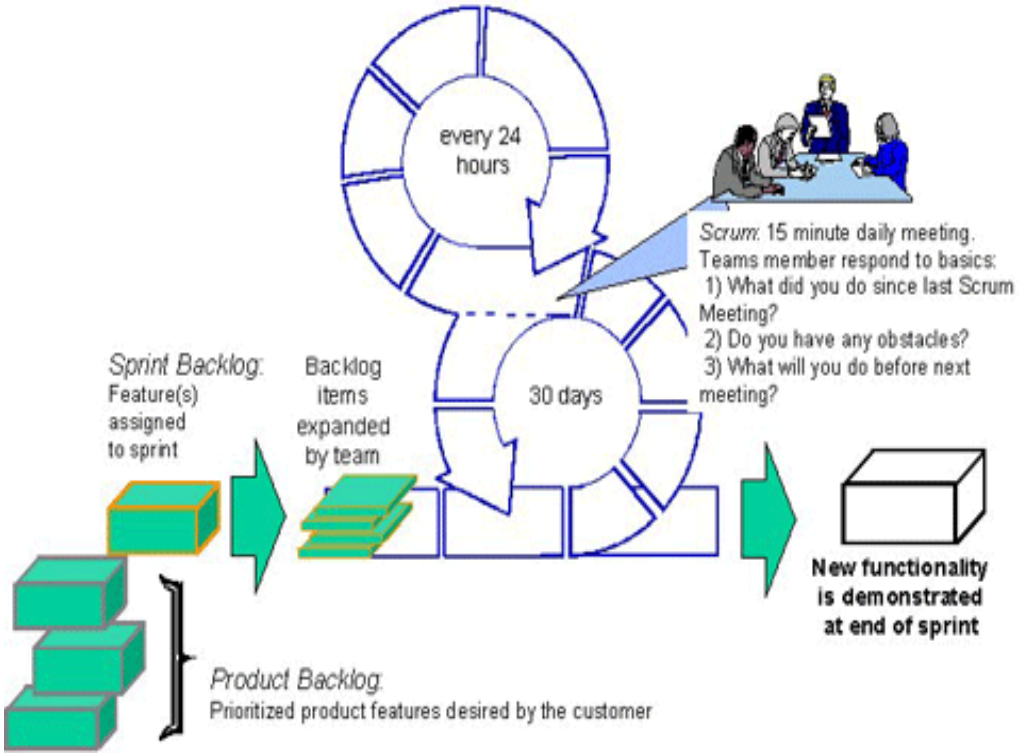


Figura 36: Esquema de trabajo SCRUM

3.2.4. ICONIX

El proceso de ICONIX maneja casos de uso, como el RUP, pero le falta mucho para llegar al nivel del RUP. También es relativamente pequeño y firme, como XP, pero no desecha el análisis y diseño que hace XP. Este proceso también hace uso aerodinámico del UML mientras guarda un enfoque afilado en el seguimiento de requisitos [Conallen99].

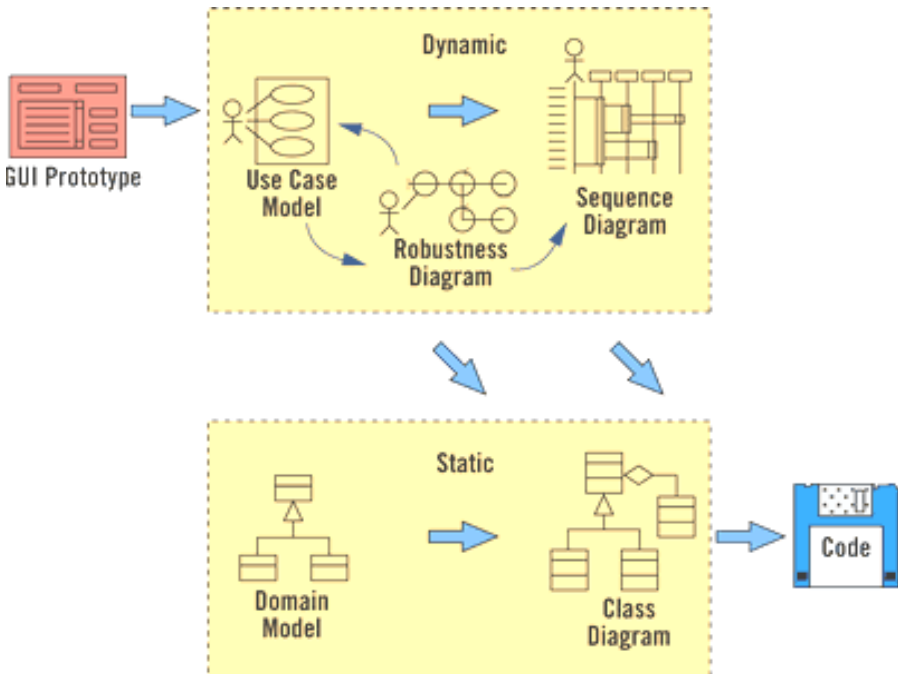


Figura 37: Esquema de trabajo ICONIX

Y, el proceso se queda igual a la visión original de Jacobson del manejo de casos de uso, esto produce un resultado concreto, específico y casos de uso fácilmente entendible, que un equipo de un proyecto puede usar para conducir el esfuerzo hacia un desarrollo real. La Figura 6 muestra el cuadro del proceso. El diagrama retrata la esencia del enfoque aerodinámico al desarrollo del software, que incluye un juego mínimo de diagramas de UML y algunas valiosas técnicas que se toman de los casos del uso para codificar rápida y eficazmente. El enfoque es flexible y abierto; siempre se puede seleccionar de los otros aspectos del UML para complementar los materiales básicos.

3.3. Herramientas Case para UML

La popularidad que ha ido adquiriendo el modelado de sistemas software con diagramas UML se encuentra vinculada a la existencia de herramientas CASE para UML que faciliten su gestión.

Una característica que beneficia a los modeladores, UML también hace más fácil escoger una herramienta de modelado. Hace tiempo, el modelador primero tenía que seleccionar una notación de metodología, y después estaba limitado a seleccionar una herramienta que la soportara. Ahora con UML como estándar, la elección de notación ya se ha hecho para el modelador. Y con todas las herramientas de modelado soportando UML, el modelador puede seleccionar la herramienta basada en las áreas claves de

funcionalidad soportadas que permiten resolver los problemas y documentar las soluciones.

Hoy en día se pueden encontrar disponibles muchas herramientas que permiten modelar procesos con UML. Sin embargo, en este trabajo se presentan únicamente las herramientas más conocidas y las más utilizadas actualmente.

3.3.1. Rational Rose

Rational Rose de Rational Software [RATIONAL] es la herramienta de modelado con UML más conocida actualmente. Rational Rose maneja todos los diagramas definidos por la especificación de UML. Permite generar código fuente a partir de diagramas y exportar estos diagramas a documentos XMI por medio de componentes (plug-ins). Una característica importante de Rational Rose es que utiliza archivos de texto para almacenar sus diagramas lo cual permite utilizar implementaciones externas para extraer información de los diagramas contenidos en un archivo de Rational Rose.

3.3.2. ArgoUML

ArgoUML es una herramienta de distribución gratuita [ArgoUML]. ArgoUML es un proyecto que actualmente está en desarrollo y que cuenta con varias versiones disponibles que se pueden utilizar para modelar en UML. ArgoUML maneja varios diagramas UML, también permite exportar sus diagramas a documentos XMI y SVG. Desafortunadamente ArgoUML usa un formato propietario en codificación binaria de sus diagramas, por lo cual es difícil leer y dar interpretación a la información.

En la Figura 33 se presenta un ejemplo de un diagrama UML de secuencia creado con ArgoUML. ArgoUML no cuenta con componentes relacionados con servicios Web, únicamente se utiliza para modelar en UML.

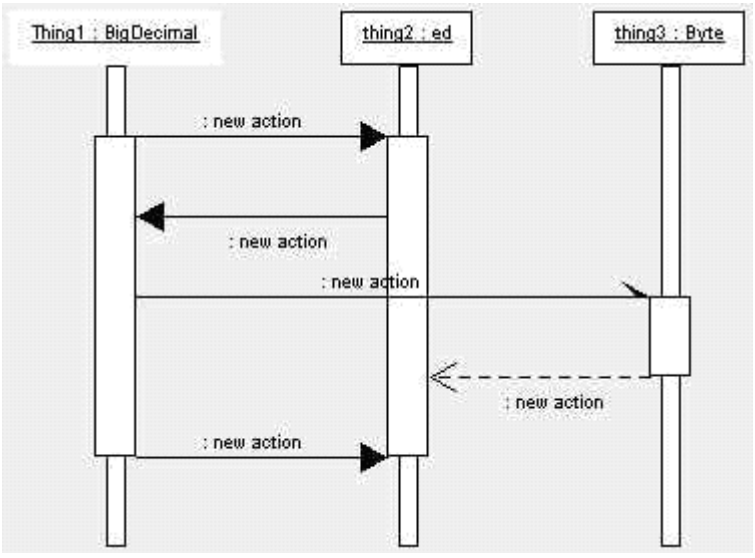


Figura 38: Diagrama UML de secuencia creado con ArgoUML

3.3.3. Poseidón

Poseidón de Gentleware es la versión comercial de ArgoUML [POSEIDON], aunque cuenta con algunas otras características y mejoras. Poseidón permite modelar procesos empleando varios tipos de diagramas UML. Al igual que ArgoUML, Poseidon utiliza un formato propietario en codificación binaria de sus diagramas que dificulta interpretar el contenido de los diagramas almacenados.

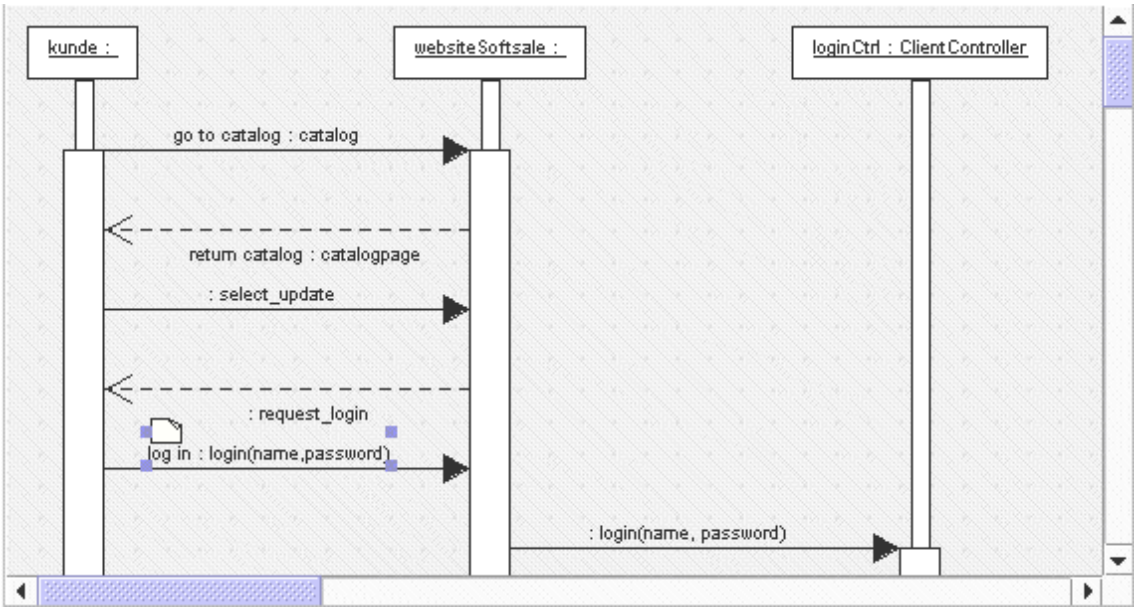


Figura 39: Diagrama UML de secuencia creado con Poseidón

En la Figura 34 se muestra un ejemplo de un diagrama UML de secuencia creado con Poseidón. El cual no tiene relación alguna con servicios Web ya que no cuenta con componentes enfocados con estos servicios. Esta herramienta únicamente se utiliza para diagramar con UML.

3.4. Aplicativos (Software)

3.4.1. SofyMed Protocolos Ecográficos

Potente herramienta de diagnóstico por imágenes utilizado en la república de Argentina que permite la emisión de informes gráficos, estadísticas, banco de imágenes, entre otras características. Estas permiten al personal de salud mantener una automatización del proceso de atención desde que el paciente ingresa al centro hasta que recibe sus resultados. Su funcionalidad y rendimiento; así como sus mínimos requisitos de hardware lo hacen ideal para el mercado de las tecnologías médicas del diagnóstico por imágenes.

Características:

- Elaboración de Informes.- Obtenga informes completos, excelentemente diseñados y redactados, con rapidez y precisión, y controles cruzados entre todos los datos ingresados para evitar errores. Y téngalos a mano siempre.
- Gráficos.- Sin más trabajo que haber ingresado los datos biométricos habituales, agregue a sus informes de Obstetricia y Doppler una excelente hoja de gráficos.
- Ecografía Tocoginecológica y General, Doppler, Estudios Varios.- Más de 30 plantillas inteligentes predefinidas para Ginecología, Obstetricia, Doppler, General y estudios varios, y la posibilidad de definir nuevas plantillas para cualquier tipo de estudio y especialidad.
- Estadísticas.- Completas estadísticas de los estudios realizados, por institución, profesional, médico solicitante, etc.
- Facturación.- Dispondrá de la valorización automática de todos los estudios realizados, emitiendo informes por institución, profesional, o cobertura médica, o exportando la información hacia Excel u otros sistemas.
- Banco de Imágenes.- Adjunte a cada estudio una ilimitada cantidad de imágenes. Con una simple y económica placa de captura de TV, podrá tomar imágenes de cualquier ecógrafo (mediante la salida de video), guardarlas junto al estudio y elegir las que desee para imprimirlas usando una impresora común, con excelente calidad y muy bajo costo.

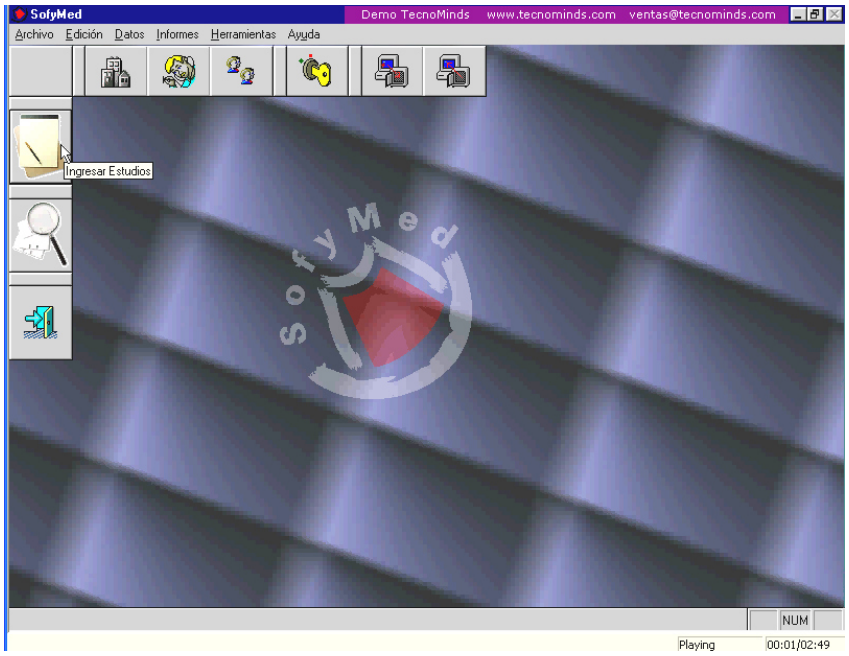


Figura 40: Pantalla de Inicio

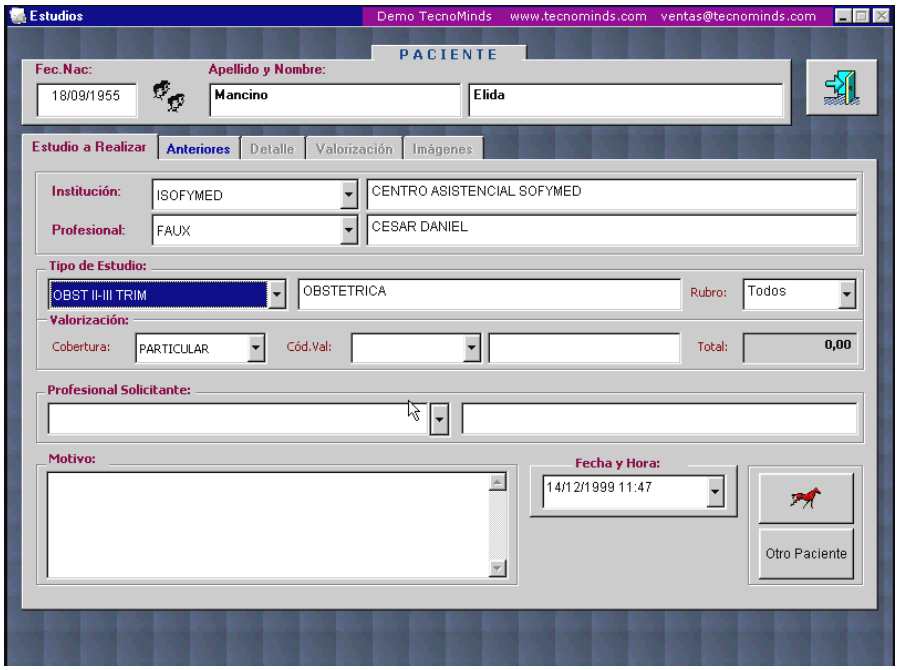


Figura 41: Pantalla de Examen

3.4.2 DPI Software de gbSystems

Completo sistema que maneja historias clínicas, registro de los estudios realizados, entre muchas otras funcionalidades. El manejo de imágenes en pantalla permitirá al personal de salud un mejor manejo de la información, automatización del área y mejores prestaciones en el diagnóstico de las enfermedades a través de imágenes.

Características

- Un número ilimitado de doctores puede utilizar el sistema, ya que todas las funciones del mismo pueden ser restringidas a un doctor en especial (esto es muy útil si varios profesionales trabajan en el mismo consultorio).
- Todas las prestaciones que brinda (informes, historias clínicas, agendas, etc.) pueden ser impresas en papel, tal como aparecen en pantalla.
- Manejo de números grandes para países donde la moneda tiene varios dígitos.
- Actualizaciones permanentes.
- Manejo de historias clínicas a través de formularios predefinidos (con datos personales y una amplia lista de antecedentes médicos) y/o de las fichas tradicionales. A la Historia Clínica de cada paciente se puede adosar una cantidad ilimitada de imágenes del mismo.
- Si bien el sistema incluye algunos protocolos predefinidos, el usuario puede modificarlos o definir los propios.
- La idea fundamental es que al ingresar el resultado de un estudio simplemente haya que indicarle al sistema qué protocolo debe asignar al informe.
- Para ingresar el resultado del estudio, simplemente debe ingresar los valores de los parámetros indicados previamente y DPI generará automáticamente el informe, pudiéndose modificar o agregar observaciones muy fácilmente.
- Existen protocolos que requieren parámetros, por ejemplo, una ecografía ginecológica normal, se deben ingresar la longitud, el diámetro anteroposterior y transverso del útero entre otros parámetros. En estos casos, simplemente se deben ingresar los valores de estos parámetros y el informe se armará automáticamente.
- Para que este mecanismo funcione correctamente, se encuentran definidos los parámetros que debe incluir cada protocolo y la posición en el mismo donde se

Figura 42: Interface de DPI Software

3.4.3 GalenHos (Sistema Integrado de Gestión Hospitalaria)

3.4.3.1 Características.

- Favorece la obtención de información para la toma de decisiones (reportes gerenciales y operativos, construcción de balanced scorecard).
- Soporta los requerimientos de reporte de información de niveles de gobierno regional y nacional (tablas exportables según requerimientos).
- Gestiona áreas críticas de manera integrada: consulta externa, hospitalización, emergencia, archivo clínico y facturación.
- Permite disminuir significativamente tiempos de espera para los pacientes (evita registro múltiple).
- Contribuye a mejorar la calidad de registro clínico (validación para registro de diagnósticos y procedimientos).
- Favorece un registro adecuado mediante la utilización de estándares del MINSA (Catálogo de servicios, Códigos de establecimientos) y estándares nacionales (DNI, UbiGeo) e internacionales.
- Vincula producción y facturación de servicios reduciendo problemas de sub facturación.
- Incrementa estándares de seguridad favoreciendo controles y auditorias.

3.4.3.2 Implementación.

Proyecto PHRplus (Partners for Health Reform plus), al Hospital Belen de Trujillo, Peru. Precisamente, la decisión y el esfuerzo de sus autoridades han permitido y permiten lograr este desarrollo e implementación actual.

El proceso de su construcción e implementación ha abarcado meses intensos de interacción con todos los equipos institucionales: médicos, enfermeras, químico-farmacéuticos, obstétricos, personal administrativo, informático, estadístico y directivo. Igualmente, los entes especializados del gobierno nacional (Ministerio de Salud) y el gobierno regional (Dirección Regional de Salud de La Libertad) se han integrado en el esfuerzo común de contar con esta experiencia modelo ofrecida a instituciones públicas o privadas.

Capitulo 4.- Resolución del problema aplicando la técnica seleccionada

4.1 Análisis Situacional del Hospital San José

4.1.1 Reseña Histórica.

El Hospital San José del Callao, fue creado en Junio de 1970 como Hospital de Campaña a causa del terremoto del 31 de Mayo de ese año producido en Huaraz – Ancash; gracias al apoyo de la Fundación Inglesa "St. Joseph's Hospice Association Liverpool". En el año 1973, fue transferido al Ministerio de Salud y administrado por el Complejo Hospitalario Daniel Alcides Carrión, encontrándose ubicado en la cuarta cuadra de la Av. Elmer Faucett, entre la Av. Morales Duarez y la Av. Argentina, en el Distrito de Carmen de la Legua - Reynoso. Inicialmente funcionó como Hospital Materno Infantil, priorizando su atención en las madres, además de servir como reposo de adultos mayores. Contaba con personal profesional de origen inglés; así como, obstetras y técnicas peruanas, algunas de las cuales se encuentran laborando en nuestra institución. Gradualmente el Hospital fue mejorando su cobertura de atención, ya en 1983 se contaba con 87 médicos y se inició la atención de Emergencia las 24 horas del día. En el año 1984 contaba con sólo cuatro (4) Licenciadas en Enfermería, las cuales hacían esfuerzos dignos de resaltar al atender pacientes de Emergencia, Hospitalización, Programas de Salud, etc. Este lento crecimiento en servicios se acentúa en gran medida a partir del año 1996, en que logramos ser Unidad Ejecutora. Es desde esa fecha y con el apoyo del Ministerio de la Presidencia que logramos un Equipamiento Médico, se ampliaron los Servicios y los Programas Preventivo Promociónales, mejora de los Sistemas y Unidades Administrativas, el campo de la Comunicación é Informática, logrando cubrir en gran medida las necesidades de la creciente demanda.

En Agosto de 1997 es reconocido como Hospital Amigo de la Madre, en 1998 se le otorga el "Tumi de Oro" por la Voz del Callao, en Junio de 1999 tiene el reconocimiento como Hospital que cumple los "Diez pasos para un Parto Seguro" por el Programa de Salud Materno Perinatal del Ministerio de Salud, también en este año logramos ser considerados dentro del Plan Piloto del Sistema de Información de Costos e Ingresos – SICI, realizado por la Dirección General de Planificación del MINSA y el Proyecto 2000. En Diciembre del 2001 se inició el reto impuesto por el MINSA en el Sistema de Gestión de la Calidad en Salud, aprobado por Resolución Ministerial N° 768-2001 y en mérito a ello es uno de los cinco hospitales suscriptores del Pacto de la Calidad.

En Marzo del 2003 se firma el Convenio de Administración por Resultados con la Dirección Nacional de Presupuesto Público, en este año también se firmó el Convenio de Cooperación Interinstitucional entre el Instituto de Enfermedades Neoplásicas y el Hospital San José inaugurándose en este último la Unidad de Displasias, se inaugura así mismo, el Centro Coordinador del Conocimiento del Callao. En Enero del 2004 es reconocido por lograr que sus instalaciones sean ambientes libres de humo de tabaco; en el mes de Abril se firmó por segunda vez el Convenio de Administración por Resultados, entre el Hospital San José, Ministerio de Salud y Dirección Nacional de Presupuesto Público del Ministerio de Economía y Finanzas; en el mes de Noviembre del mismo año, obtiene el Segundo lugar en la Categoría Institutos Especializados y Hospitales en el "I° Encuentro Nacional de Experiencias en Mejoramiento de la Calidad en Salud". En el año 2,005 y por segundo año consecutivo, el Hospital San José Callao, recibió el premio a la calidad en el concurso organizado por el Ministerio de Salud, obteniendo el tercer puesto en la convocatoria hecha por la Dirección de Calidad entre 1,500 trabajos presentados a nivel nacional, con el tema "Fortalecimiento del Sistema de Quejas en el Hospital San José".

En el año 2006 continuamos cosechando premios al haber obtenido el tercer lugar en el III Encuentro Nacional de Experiencias en Mejoramiento de la Calidad en Salud con el trabajo "Sistematización de los Procesos Previos a Consulta Externa" presentado por la Unidad de Estadística é Informática. En el año 2007 y por cuarto año consecutivo el Hospital San José se hace acreedor al tercer puesto en la convocatoria hecha por la Dirección de Calidad del Ministerio de Salud, siendo ganador el trabajo presentado por el Servicio de Farmacia con el tema "Mejorando la adherencia de los pacientes en tratamiento con antirretrovirales". [HSJ07]

4.2 Infraestructura

El Hospital de Apoyo San José se encuentra ubicado en la calle Las Magnolias N° 475 Altura de la Cuadra 4 de la Avenida Elmer Faucett Distrito de Carmen de la Legua – Callao.

Área Total del Terreno	:	4,491.37 m^2
Área Ocupada	:	2,895.85 m^2
Área Construida	:	2,895.85 m^2

El material de construcción de este nosocomio es predominantemente prefabricado existe acondicionamiento en casi todos los ambientes para hacer operativa la atención al ritmo de la modernidad [HSJ07].

4.2.1 Organigrama

El Hospital San José del Callao es una Unidad Ejecutora del Ministerio de Salud, sin embargo mantiene relaciones de dependencia técnica con la Dirección de Salud I Callao (DISA).

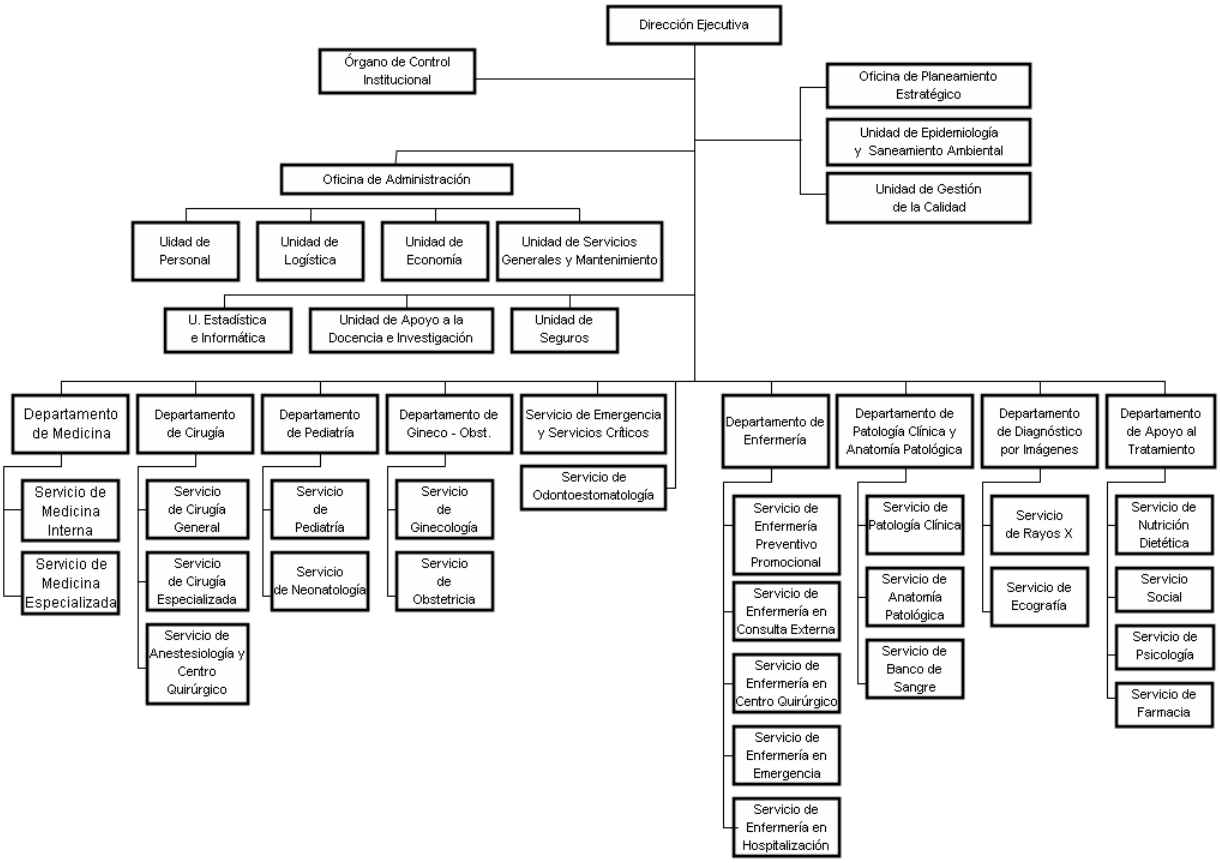


Figura 43: Organigrama del Hospital San Jose.

4.2.2 Misión y visión del Hospital San José del Callao

- Somos un hospital del sector salud que brinda atenciones preventivo-promociónales, recuperativas y de rehabilitación a las personas, la familia y la población en un ambiente saludable, contribuyendo al desarrollo sostenido del país.
- "Ser una organización del sector salud exitosa y competitiva en los próximos 5 años que contribuya a tener una población sana con fácil acceso al establecimiento, con servicios eficientes y equitativos, con personal comprometido en una cultura organizacional de excelencia y calidad".

4.2.3 Objetivo.

Brindar atención de salud especializada al usuario del ámbito regional con calidad.

4.3 Evaluación y selección de la técnica usada

4.3.1 Metodologías para el desarrollo de software

Entre las metodologías a utilizar para la solución de nuestro problema y basados en el Lenguaje Unificado de Modelamiento tenemos:

- RUP
- ICONIX
- XP
- SCRUM
-

Tabla N°1: Diferencias entre metodología tradicional y ágiles

Metodologías Tradicionales (RUP)	**Metodologías Agiles (ICONIX, SCRUM, XP)**
Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo	Basadas en heurísticas provenientes de prácticas de producción de código
Cierta resistencia a los cambios	Especialmente preparados para cambios durante el proyecto
Impuestas externamente	Impuestas internamente (por el equipo)
Proceso mucho más controlado, con numerosas políticas/normas	Proceso menos controlado, con pocos principios.
El cliente interactúa con el equipo de desarrollo mediante reuniones	El cliente es parte del equipo de desarrollo
Más artefactos	Pocos artefactos
Más roles	Pocos roles
Grupos grandes y posiblemente distribuidos	Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio
La arquitectura del software es esencial y se expresa mediante modelos. Existe un contrato prefijado	Menos énfasis en la arquitectura del software No existe contrato tradicional o al menos es bastante flexible

Tabla N°2: Diferencias por etapas y enfoque metodológico

MODELOS RIGUROSOS (RUP)	ETAPA	MODELOS AGILES (ICONIX, XP , SCRUM)
Planificación predictiva y "aislada"	Análisis de requerimientos	Planificación adaptativa: Entregas frecuentes + colaboración del cliente
	Planificación	
Diseño flexible y Extensible + modelos + Documentación exhaustiva	Diseño	Diseño Simple: Documentación Mínima + Focalizado en la comunicación
Desarrollo individual con Roles y responsabilidades estrictas	Codificación	Transferencia de conocimiento: Programación en pares + conocimiento colectivo
Actividades de control: Orientado a los hitos + Gestión miniproyectos	Pruebas	Liderazgo-Colaboración: empoderamiento +auto-organización
	Puesta en Producción	

Además presentamos un cuadro de comparación por cada aspecto analizado:

Tabla N° 3 Según características del proyecto

Modelo de Proceso	Tamaño del Proceso	Tamaño del Equipo	Complejidad del Problema
RUP	Medio / Extenso	Medio / Extenso	Medio / Alto
ICONIX	Pequeño / Medio	Pequeño / Medio	Pequeño / Medio
XP	Pequeño / Medio	Pequeño	Medio / Alto
SCRUM	Pequeño / Medio	Pequeño	Medio / Alto

En este cuadro se presenta una comparativa de las modelos de proceso en cuanto a las características del proyecto, analizamos el tamaño del proceso, del equipo y la complejidad del problema para cada uno de los modelos. Podemos resaltar que: con un pequeño equipo de desarrollo se puede realizar grandes proyectos, de alta complejidad; es el caso de XP y SCRUM.

Tabla N°4 Según la curva de aprendizaje

Modelo de Proceso	Curva de aprendizaje	Herramienta de integración	Soporte Externo
RUP	Lenta	Alto Soporte	Alto Soporte
ICONIX	Rápida	Algún Soporte Disponible	Algún Soporte Disponible
XP	Rápida	No mencionado	Algún Soporte Disponible
SCRUM	Rápida	No mencionado	Algún Soporte Disponible

Con respecto a la curva de aprendizaje, vemos que los modelos ágiles, nos ofrecen una mayor ventaja pero con ciertas limitaciones, ya que aun no hay sido explotadas a gran escala como lo es RUP que posee alto soporte y herramientas integrales que nos guían a través del mismo, facilitando aplicar con mayor efectividad esta metodología, permitiendo aprovecharla al máximo

Conclusiones.

- El retrasar las decisiones en un proyecto de software permite potenciar el valor del producto tanto para el cliente como al equipo o empresa que desarrolla
- Para que un grupo de desarrollo adopte una metodología ágil debe poseer experiencia trabajando con metodologías tradicionales, ya que la experiencia es la que predomina en los momentos cruciales del proyecto, además debe tener la capacidad de ser equipos auto-gestionados, altamente motivados y con gran innovación
- Las metodologías ágiles permiten disminuir costos y brindar flexibilidad a los proyectos de software donde la incertidumbre está presente
- El uso de metodologías tradicionales es esencial al inicio en un equipo de desarrollo de software
- Las metodologías ágiles se deberían aplicar en proyectos donde exista mucha incertidumbre donde el entorno es volátil, donde los requisitos no se conocen con exactitud, mientras que las metodologías tradicionales obligan al cliente a tomar las decisiones al inicio del proyecto.

4.3.1.1 Análisis de criterios de comparación.

Para establecer los criterios adecuados de evaluación de desarrollo del software nos basaremos en la propuesta de Highsmith 2002[Highsmith02] la cual compara las distintas aproximaciones en base a tres parámetros: vista del sistema como algo cambiante, tener en cuenta la colaboración entre los miembros del equipo y características más especificas de la propia metodología como son la simplicidad, excelencia técnica, resultados, adaptabilidad, etc.

4.3.1.1.1 Vista del sistema como algo cambiante.

Este principio es una actitud que deben adoptar los miembros del equipo de desarrollo. Los cambios en los requisitos deben verse como algo positivo. Les va a permitir aprender más, a la vez que logran una mayor satisfacción del cliente. Este principio implica además que la estructura del software debe ser flexible para poder incorporar los cambios sin demasiado coste añadido.

4.3.1.1.2 Colaboración entre los miembros del equipo

La gente es el principal factor de éxito, todos los demás (proceso, entorno, gestión, etc.) queda en segundo plano. Si cualquiera de ellos tiene un efecto negativo sobre los individuos debe ser cambiado.

El dialogo cara a cara es el método más eficiente y efectivo para comunicar dentro de un equipo de desarrollo. Los miembros de equipo deben hablar entre ellos, este es el principal modo de comunicación. Se puede crear documentos pero no todo estará en ellos, no es lo que el equipo espera.

4.3.1.1.3 Características metodológicas (CM)

Resultados: verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones.

También realiza el seguimiento del progreso de cada iteración y evalúa si los objetivos son alcanzables con las restricciones de tiempo y recursos presentes.

Simplicidad: consiste en tomar los caminos más simples que sean consistentes con los objetivos perseguidos. Si el código producido es simple y de alta calidad será más sencillo adaptarlo a los cambios que puedan surgir.

Adaptabilidad: la habilidad de responder a los cambo s que puedan surgir a lo largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta puesto que hay muchas variables en juego, debe de ser flexible para poder adaptarse a los cambios que puedan surgir.

Excelencia técnica: las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos. Todo el equipo es informado de las

responsabilidades y estas recaen sobre todo sus miembros de acuerdo a los objetivos que se persigan.

Practicas de colaboración: se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure el éxito.

4.3.1.2 Cuadro comparativo con respecto al puntaje de criterios establecidos

En el siguiente cuadro se lista los valores por puntaje de acuerdo a cada criterio utilizado para la comparación y se señala los puntajes que corresponde por cada valor.

Respecto a los criterios de: vista del sistema como algo cambiante, colaboración entre los miembros del equipo y características metodológicas (CM), el modelo permite medirlos en puntaje de 1 a 5.

Tabla Nª5: Cuadro comparativo de metodologías

	RUP	XP	ICONIX	SCRUM
Sistema como algo cambiante	5	5	3	5
Colaboración	5	5	5	5
Características Metodológicas(CM)				
- Resultados	5	5	4	5
- Simplicidad	4	5	5	5
- Adaptabilidad	5	3	4	4
- Excelencia técnica	5	4	4	3
- Practicas de colaboración	5	5	4	4
Media CM	**4.8**	**4.4**	**4.2**	**4.2**
Media Total	**4.9**	**4.8**	**4.0**	**4.7**

El resultado encontrado, respecto a los parámetros de medición, da un buen puntaje a la metodología RUP. Es por ello que nuestro de desarrollo de software utilizara como estándar esta metodología.

4.3.2 Herramientas Case

Vamos a considerar 3 herramientas que son líderes en el mercado del modelamiento con lenguaje UML, entre ellas tenemos unas características básicas de cada una de ellas:

- **ArgoUML.-** Una herramienta desarrollada en Java y de código libre, que soporta mucha de las características de UML 2.0 por lo cual resulta bastante práctica para llevar el control del diseño de la aplicación a nivel base de datos, así como también a nivel de casos de uso, diagramas de estado, y clases, aparte ArgoUML entre sus características es multiplataforma por lo que puede ejecutarse tanto en Windows o Linux.
- **Rational Rose**.- Plataforma creada por Booch, Rumbaugh y Jacobson, miembros de la OMG (Object Management Group). Es una de las más poderosas herramientas de modelado visual para cubrir las etapas del desarrollo de ciclo de vida de software.
- **Poseidon**.- Herramienta de Gentleware que se concibe como la versión comercial mejorada de ArgoUML.

A continuación se hace uso de cuadros que permiten ver las ventajas y desventajas de su uso y él porque he escogido como alternativa de modelador para este trabajo a Rational Rose de IBM.

Tabla N°6 Comparativo General

Propiedades	ArgoUML	Rational Rose	Poseidón
ASPECTOS GENERALES			
Observaciones Generales	Navega de acuerdo con diferentes perspectivas de modelado y de metodologías de desarrollo.	Provee productos de UML para los lenguajes comunes de la industria para especificación, visualización, construcción y documentación de los artefactos de los SI.	Herramienta de Gentleware que genera código Java, usando plantillas. Utiliza ingeniería reversa y sincronización de código en Java.
Ámbito de Utilización	Sus características se ajustan a proyectos de investigación y académicos.	Se enmarca dentro del desarrollo de modelado para fines académicos, investigativos.	Sus características se apoyan en el desarrollo orientado a objetos para fines académicos, investigativos
Plataforma	Independiente de la plataforma. Requiere JVM (Maquina Virtual de Java) 1.3 o superior.	Sistemas Operativos Win 95, 98, ME, NT 4.0, XP, 2000, 2003.	Independiente de la plataforma. Requiere JVM (Maquina Virtual de Java) 1.3 o superior.
Precio	Software Libre	Depende de la versión	Software Libre

4.3.2.1 Selección de los criterios de evaluación.

Aquí lo que se busca es brindar las características de las herramientas que permiten realizar modelos de UML, siguiendo cuatro puntos definidos para su comparación: aspectos procedimentales, apoyo al repositorio, arquitectura y funcionalidad. A continuación un detalle de cada uno de ellos:

4.3.2.1.1 Enfoque procedimental

Se refiere a la forma como la herramienta utiliza la metodología para guiar al usuario a través del proceso de desarrollo de sistemas.

- Apoyo metodológico

  Una herramienta UML debe permitir al usuario el uso de una o varias metodologías orientadas a objetos para el desarrollo de sistemas de información (RUP, Métrica, OMT) entre otras; permitiendo garantizar la unicidad y coherencia de los diferentes modelos del sistema que representan vistas o modelos que representan diferentes niveles de abstracción. Esto para permitir guiar al usuario a través de etapas o fases permitiendo una trazabilidad entre diagramas de alto nivel que representen aspectos del dominio del problema y diagramas detallados que representan detalles de implementación [Shaw03].

  Para conseguir tal propósito, es importante que el navegador de la herramienta, muestre las etapas de la metodología o los modelos que propone la misma para cada etapa, fase o arquitectura, y de ser necesario que le ayude al usuario a evolucionar o refinar un modelo al momento de pasar de una etapa a otra o de un aspecto o contexto arquitectónico a otro. [SHAW2003, DEWAYNE2003, Garlan2003]

- Soporte completo UML

  Se refiere a la capacidad de las herramientas de construir todos los diagramas que propone el lenguaje UML, o al menos la grana mayoría de ellas: Casos de Uso (representan la funcionalidad o alcance del sistema), diagrama de clases (describe la estructura de objetos y sus relaciones), diagrama de interacción (secuencia o colaboración) y en algunos casos el diagrama de estados (visualiza el ciclo de vida de un objeto) [OBD2003]

  También hay que considerar la versión que soporta de UML y como la herramienta cubre los conceptos tratados en ella.

- Facilidad de extensión

  Se pueden presentar situaciones muy particulares en el ámbito del desarrollo que quizás no se logren satisfacer. Es por ello que UML ofrece mecanismos de extensibilidad como restricciones, estereotipos y valores etiquetados [OMG2003], que se deben contemplar e implementar por las herramientas

CASE. Además es deseable que dichas herramientas contemplen la posibilidad de edición de símbolos gráficos que permitan representar aspectos particulares de un dominio no contemplados por UML [OBD2003]

- Modelo de datos

  El modelamiento de la información persistente constituye uno de los actores importantes cuando se está construyendo un sistema de información. Es decir, estará soportada por Base de datos relacionales [ARCE2003]. Es por ello que las herramientas de modelado con UML le deben permitir al usuario facilidades para definir el esquema relacional, bien sea a través de una interfaz para la construcción de diagramas entidad-relación o modelos de datos, o a través de la extensión de UML, modelando clases persistentes que se puedan implementar en motores relacionales [Dorssey99].

- Autogeneración

  Es una característica que podría mirarse desde dos puntos de vista, generar código a partir de modelos y generar modelos a partir de código.

  Varias de las herramientas disponen de la generación de la definición estática (propiedades y signatura de los métodos) de las clases de los diferentes lenguajes de programación y para diferentes arquitecturas (CORBA, J2EE, etc.).

  También es recomendable que la herramienta genere el código que implementa la funcionalidad de mapeo (materialización y desmaterialización) entre el modelo de objetos y el modelo relacional, generalmente conocido como esquema de persistencia [Larman03].

  La generación de código puede presentar dos tipos de inconvenientes [Gomez03]:

  - Cuando el código ya se ha generado y se desean realizar cambios sobre este, la herramienta tiende a escribir el código que el desarrollador introdujo inicialmente.
  - Una vez generado el código, la herramienta CASE coloca un numero de líneas de comentario llamadas polución. En la actualidad si colapsara la línea de comentario automáticamente los editores de lenguajes de programación o IDE's mostraran solo lo que usuario necesite ver para su desarrollo en la herramienta.
  - Ingeniería inversa

  Mientras que un proceso clásico de ingeniería de software, conduce al usuario a través de unas etapas donde a partir de modelos llegue a un producto de

software, la ingeniería inversa parte del producto y lo transforma en modelos [Sanchez03].

La generación de diagramas a partir de código puede tener dos inicios: Generación a partir de lenguajes orientado a objetos y a partir de lenguajes estructurales.

Para aplicaciones construidas en lenguajes no orientados a objetos, como es nuestro caso de estudio, se realiza ingeniería inversa, para migración de datos y/o identificación de aspectos estructurales difíciles de leer desde el código.

- Métricas

  Son generalmente presentadas en reportes con información como: el número de atributos y métodos a utilizar por clases, índice de métodos públicos, privados y protegidos, líneas de código pro método, grado de acoplamiento y cohesión [OBD2003].

- Apoyo a lenguajes formales

  Lenguaje formal es la técnica de base matemática para describir las propiedades de un sistema, con el propósito de evitar la ambigüedad y minimizar la interpretación subjetiva de la especificación de un sistema.

  OCL es el lenguaje adoptado alrededor de UML; utilizado para especificar las variantes, pre y post condiciones; así como de otorgar mayor formalidad a las especificaciones y aspectos visuales de UML [Moreno03].

  Integrar un lenguaje formal en una herramienta CASE, es una de las características que maslo distinguen por considerarse una herramienta robusta; ya que dota a los diagramas con una semántica precisa que permite que la generación de código pueda hacerse de manera completa.

- Actualización

  El mejoramiento de la herramienta UML a través de versiones de actualización que provean de nuevas funcionalidades, mayor rendimiento y soporte de nuevas tecnologías es fundamental para estandarizar y promover su uso.

Tabla N°7 Cuadro Comparativo – Enfoque Procedimental

Propiedades	ArgoUML	Rational Rose	Poseidón
ENFOQUE PROCEDIMENTAL			
Apoyo Metodológico	No apoya una metodología especifica	UML, Booch y OOSE; todas apoyadas en RUP, Soporta MVC	No apoya una metodología específica. A través del buscador(browser) sugiere cierta organización del proyecto
Soporte Completo UML	UML 1.3	UML 1.3 y 2.0	UML 2.0
Ingeniería Reversa	Módulos de Framework para Java y .jar	C++, Visual Basic, and VB.NET, COM, código ADA, J2EE, CORBA/IDL,MIDL	C#, Visual Basic.NET, PHP y JAVA
Métricas	No utiliza métricas	Apoya utilización de métricas	No hay métricas
Apoyo a lenguajes formales	OCL	OCL	OCL
Actualización	Posee servicio de actualización	Posee servicio de actualización	Posee servicio de actualización

4.3.2.1.2 Soporte de modelo arquitectónico

Con la revisión de este aspecto se pretende evaluar la capacidad que tienen las herramientas de modelado UML para apoyar la definición de la arquitectura de un sistema. La arquitectura es el elemento clave de diseño que permite establecer acuerdos e alto nivel referentes a la forma del sistema y sus relaciones [Garlan03].

- Soporte a ADL

  El lenguaje de definición de arquitectura proporciona notaciones para descomponer un sistema en componentes y conectores y especificar cómo se combinan estos elementos para formar cierta configuración teniendo la ventaja de ofrecer una notación descriptiva directa de las principales abstracciones arquitectónicas. De este modo, los arquitectos pueden definir la estructura de sus sistemas evaluando posibles cambios, y guiar su implementación evitando la ambigüedad. Entre los más importantes están: WRIGHT, Aesop, Darwin, Rapide y Unicon [Georgas03]. El lenguaje UML en su versión 2.0 presenta soporte para lenguaje ADL.

- Apoyo al modelado de capas

  El patrón de arquitectura más utilizado es el que especifica un sistema estableciendo una separación de la funcionalidad en capas o niveles. Ya que establece como mínimo la definición de tres capas: [Clements03]

  Presentación.- Ofrece una interface al entorno

  Lógica del negocio.- determina el comportamiento de la aplicación

  Acceso a datos.- administra la información persistente que maneja el sistema

  Gran importancia tiene esta arquitectura por los beneficios que proporciona: escalabilidad, confiabilidad, flexibilidad y soporte a la reutilización. Asimismo, permita la visualización del sistema a través de paquetes y relaciones de dependencia; proveyendo de soporte especializado en cada una de las etapas de la arquitectura como a continuación se detalla:

  - Capa de presentación

    Son componentes que especifican la interacción del sistema con el entorno (usuarios, dispositivos, etc.), cuyos elementos deben poderse representar haciendo uso del estereotipo, con su respectivo icono asociado establecido por UML para especificar el tipo de frontera (<<boundary>>). Es posible que algunas herramientas provean el soporte a las extensiones de UML para manejar los componentes de la capa de presentación; por ejemplo, para modelar aplicaciones web [Conallenn99].

  - Lógica del Negocio

    Las reglas del negocio deben normalizarse de la misma manera como lo hacen los datos, para disminuir costos y mantenimiento, incrementando la productividad y reduciendo esfuerzos. Es por ello, que el proceso de administración de las reglas del negocio, por parte de las herramientas CASE, debería caracterizarse por facilitar la unicidad, consistencia e independencia en la implementación de reglas, que apoyen el sostenimiento de la integridad del sistema [Clarion03].

  - Acceso a datos

    Administra la información que debe perdurar en el sistema, y para ello los servicios prestados por las herramientas de modelado deben permitir al usuario construir diagramas de bases de datos o extender el UML, de tal manera que se puedan representar las clases persistentes, haciendo uso de estereotipos.

Tabla N° 8 Cuadro Comparativo – Soporte Arquitectónico

Propiedades	ArgoUML	Rational Rose	Poseidón
APOYO AL SOPORTE ARQUITECTONICO			
Soporte ADL	A través de UML	A través de UML	A través de UML
Servicios de Presentación	Extensiones de UML	Extensiones de UML	Extensiones de UML
Servicios de Lógica del Negocio	Soporte de edición de restricciones con OCL	Chequeo de la sintaxis UML	Edición de restricciones usando OCL. Control de validez de sintaxis de los modelos a partir de las reglas de UML.
Servicios de Persistencia	A través de la definición de estereotipos, no genera SQL	A través de la definición de estereotipos	A través de la definición de estereotipos, no genera SQL

4.3.2.1.3 Apoyo al repositorio

Los repositorios son herramientas que permiten centralizar, administrar y gestionar las versiones o estados de un proyecto en el que se requieren revisiones frecuentes. En el desarrollo de sistemas usando herramientas CASE, se hace relevante la presencia de un repositorio , no solo si la herramienta es multiusuario, sino en general para la organización de los cambios reflejados en el refinamiento de los modelos, en el avance de las etapas o en la colaboración entre usuarios[Macpro03].

- Robustez

  Capacidad de los sistemas de reaccionar apropiadamente ante condiciones excepcionales[Meyer99], como casos no previstos en la especificación del sistema(por ejemplo, fallas eléctricas, entradas erróneas de datos, corrupción por virus); y la capacidad de recuperarse ante tales situaciones, protegiendo de forma segura la información almacenada en un repositorio.

- Herramientas de administración

  Se clasifica en cuatro categorías:

  - De Repositorio

    El administrador de la plataforma Case necesita realizar tareas como la creación, actualización y borrado del repositorio, mantenimiento de usuarios, copias de seguridad, recolección de estadísticas y manejo de las

extensiones que usa los usuarios en los archivos, al momento de la autogeneración. [Dorsey97]

- De Ejecución de Proyectos

  El administrador del proyecto necesita realizar tareas de identificación de riesgos, planeación del proyecto, asignación de tareas a personas, asignación de fechas y recursos a tareas, seguimiento de tareas o fases de un proyecto.

- De Áreas de Trabajo

  Las áreas de trabajo (workspace) son la manera en que la herramienta facilita la administración de todos los objetos sobre los que puede trabajar el usuario. La administración de las áreas de trabajo las realiza cada usuario a través de un navegador de objetos del repositorio. [Oracle02]

- De Aplicaciones

  Aquí encontramos tareas de administración de versiones, manejo de dependencias y comparación entre modelos. Aquí se presentan dos variantes:

- Reconciliación.- Los modelos se actualizan con base en los cambios realizados en la versión implementada.
- Sincronización.- Consiste en pasar los cambios realizados en un modelo, a la implementación.

- Colaboración entre usuarios

  Una colaboración representa la relación de interacción entre usuarios para realizar las tareas de cada uno con el sistema. Estas pueden asistir al establecimiento de dichas relaciones entre usuarios a partir del apoyo metodológico y de la especificación de las responsabilidades de los usuarios [Homero03].

  Las herramientas que cuentan con este tipo de soporte a usuarios deben tener funciones de: definición de roles, otorgar y revocar privilegios para cada rol establecido, y permitir compartir objetos del repositorio.

- Intercambio

  Posibilidad de que la información de los proyectos, guardada por una herramienta en un formato determinado pueda ser leída por otra herramienta CASE, que acepte dicho formato, para usarse en otros desarrollos. Si soporta UML la herramienta entonces es posible el alcance de dicha meta, ya que todo lenguaje persigue la estandarización sintáctica y semántica. Es por ello que, la OMG definió un estándar de intercambio llamado XMI (extensible Markup

Language Interchange), que basado en el metamodelo de UML, sirve para el intercambio de metadatos utilizando XML.

Tabla N° 9 Cuadro Comparativo – Apoyo al repositorio

Propiedades		ArgoUML	Rational Rose	Poseidón
APOYO AL REPOSITORIO				
Robustez		Mediana. No realiza autoguardado	Depende de la acción propia del usuario para hacer el guardado.	Control sobre acciones inesperadas, fallos, autoguardado y errores.
Herramientas para Administrar	Repositorio	Posee un sistema de archivos usual. Almacenamiento local	Administración del repositorio con acceso multiusuario.	Administración multiusuario para la versión empresarial. Apoyo de persistencia y concurrencia en repositorio común por proyectos.
	Proyectos	Administración del diseño y aplicación en el desarrollo	Evaluación y control de edición de diagramas y proyectos. Administración de proyectos.	Configuración de las propiedades del proyecto. Apoyo de documentación del proyecto.
	Área de Trabajo	Corrección automática de errores sobre el área de diagramación.	Generación de reportes por uso, instancias, accesos de violación y documentación. Integración, documentación y archivos vía URL.	Barra de herramientas y menú, panel de navegación, con tipos diferentes de vistas estructurales y arquitectura.
	Aplicación	Representan alternativa del diseño de manera objetual, código de origen y textual	Consistencia de modelos del sistema software.	Actualización de diagrama para ingeniería reversa.
Colaboración entre usuarios		Es una herramienta StandAlone , por lo que no aplica la colaboración entre usuarios	Soporte multiusuario, modelos compartidos y proyectos para desarrollo individual.	Configuración de propiedades de usuarios de repositorio, por proyecto, para versión empresarial.
Intercambio		XMI como mecanismo estándar	XMI como mecanismo estándar	XMI como mecanismo estándar, compatible con proyectos Rational 1.6

4.3.2.1.4 Enfoque funcional

Aquí veremos las utilidades adicionales que permitirán al usuario desarrollar un sistema de información con mayor o menor facilidad.

- Versionamiento

  Una propuesta ideal para una herramienta de modelamiento es la de permitir construir versiones de software de tipo incremental hasta llegar a la madurez del producto. Por ello al usuario le permitirá guardar versiones previas que una vez actualizada la herramienta la versión previa se encuentre disponible.

- Navegación

  El uso de un navegador de objetos incluido en la herramienta que permita al usuario revisar y acceder directamente a la información de una clase o método.

- Manejo de diagramas

  Existen tres funciones que resultan de utilidad al usuario: Visualización a través de Zoom, Refrescar y Ajustar; Impresión y Exportación de datos.

- Edición de código

  Deberá disponer de editores de código integrado, que faciliten al usuario la edición de código fuente de las clases, métodos y demás artefactos que involucran la solución implementada.

Tabla N°10 Cuadro Comparativo – Enfoque Funcional

Propiedades		ArgoUML	Rational Rose	Poseidón
ENFOQUE FUNCIONAL				
Autogeneración		Java, C++, PHP	Java, J2EE, ANSI C++, Visual C++, VB, VB.NET, CORBA IDL, MIDL y XML	Java, HTML, C++ y XML, plugins para C#, CORBA IDL.
Versionamiento		No aplica	Rose ClearCase, Rose Clearcase LT	Administración propia, propiedades del proyecto, versión y actualización.
Navegación		Tres vistas	Proyectos, Diagramas	Modelos
Manejo de Diagramas	Visualización	Zoom In/Out Refresh Ajuste Autolayout	Discriminación de colores en diagramas Zoom In/Out	Zoom In/Out Show/Hide Autolayout Copy/Cut/Paste
	Impresión	Configuración de pagina	Funcionalidad WYSIWYG con vista previa	Permite impresión de diagramas solo con versión empresarial
	Exportación	GIF, PS, Encapsulated PS, PGML, SVG.	GIF, JPEG, BMP y otros	JPG, JPEG, BMP, WMF, GIF y SVG
Edición de código		Java	Visual Basic, Java	OCL para JAVA

4.3.2.2 Cuadro Comparativo respecto a los criterios.

Tabla Nª 11: Cuadro comparativo de Herramientas CASE

	ArgoUML	Rational Rose	Poseidón
Enfoque procedimental	4	5	3
Soporte de modelo arquitectónico	3	4	3
Apoyo al repositorio	4	4	4
Enfoque funcional	3	4	4
Media Total	**3.5**	**4.25**	**3.5**

El resultado encontrado, respecto a los parámetros de medición, da un buen puntaje a la Herramienta CASE Rational Rose. Es por ello que nuestro desarrollo de software utilizara como esta herramienta.

4.4 Análisis de costo beneficio

Nos mostrara los distintos costos que implicara realizar el proyecto de desarrollo del sistema de información, tanto hardware, software y costo de recursos humanos.

Tabla N°12 Cuadro de costo de Hardware

Descripción	Cantidad	Precio unitario	Total
Pc Pentium IV - Desktop	4	S/. 1829.00	S/. 7316.00
Pc Pentium IV - Servidor	1	S/. 22000.00	S/. 22000.00
		Total	**S/. 29316.00**

Tabla N°13 Cuadro de costo de Software

Descripción	Cantidad	Precio unitario	Total
Licencia Visual Studio 6.0	1	S/. 1929.23	S/. 1929.23
Licencia de MS SQL Server 2000 Standard Edition	1	S/. 2396.87	S/. 2396.87
Licencia de Microsoft Windows 2003 R2 Standard Edition	1	S/. 2586.84	S/. 2586.84
CAL de SQL Server 2000	4	S/. 505.15	S/. 2020.60
Licencia de Microsoft Windows XP Profesional SP2	4	S/. 840.00	S/. 3360.00
Microsoft Office 2007 Standard	4	S/. 930.00	S/. 3720.00
Rational Rose 2000	1	S/. 13596.46	S/. 13596.46
Total			**S/. 29610.36**

Tabla N°14 Cuadro de costo de personal

Descripción	Cantidad	Horas de trabajo diario	Días	Total horas en el proyecto	Costo por hora	Costo total x proyecto
Programadores	1	10 h	60 d	448	S/. 3.30	S/. 1478.4
Analista de Sistemas	1	10 h	75 d	560	S/. 4.60	S/. 2576.00
Ingeniero de Software	1	10 h	90 d	672	S/. 5.60	S/. 3763.20
Total						**S/. 7,817.60**

Tabla N°15 Cuadro de costos varios

Descripción	Costo total
Útiles de escritorio	S/. 92.00
Insumo de impresora laser(tóner)	S/. 50.00
Refrigerios	S/. 238.00
Total	**S/. 380.00**

Tabla N°16: Calculo del VAN (valor actual neto) y TIR (tasa de retorno de inversión)

AÑO	0	1	2	3	4	5
Beneficio	40000.00	20000.00	20000.00	20000.00	20000.00	20000.00
Costo por Hardware	29316.00	4397.4	4397.4	4397.4	4397.4	4397.4
Costo por Software	29610.36	4441.5	4441.5	4441.5	4441.5	4441.5
Costo del Personal	7,817.60	-	-	-	-	-
Costos Adicionales	380.00	-	-	-	-	-
Total del Costo	67123.96	8838.9	8838.9	8838.9	8838.9	8838.9
Beneficio Neto	27123.96	11161.1	11161.1	11161.1	11161.1	11161.1
Beneficio Neto Acumulado	27123.96	15962.86	4801.76	6359.34	17520.44	28681.54
VAN 30%	11861.148	11161.1	23970.07	18438.52	14183.48	10910.37
Beneficio actual neto	11861.148	700.048	23270.022	41708.542	55892.022	66802.392
TIR	30.11%					

El costo de mantenimiento del sistema está estimado en un 10% de su costo; asimismo el mantenimiento del servidor estima un costo del 15% del precio de compra.

Se deduce que el VAN > 0 por lo tanto el proyecto es económicamente viable; asimismo, al tener una Tasa Interna de Retorno con porcentaje mayor a la tasa de interés entonces el inversionista realizara el proyecto.

Debe entenderse que se necesita cuando menos 5 años para poder tener el retorno de inversión deseado.

Cuando la TIR es mayor que la tasa de interés, el rendimiento que obtendría el inversionista realizando la inversión es mayor que el que obtendría en la mejor inversión alternativa, por lo tanto, conviene realizar la inversión.

Si la TIR es menor que la tasa de interés, el proyecto debe rechazarse.

TIR > i => realizar el proyecto

TIR < i => no realizar el proyecto

TIR = i => el inversionista es indiferente entre realizar el proyecto o no.

Explicaciones de las cifras:

En el año 0 el proyecto recibe un aporte de la institución de S/.40,000.00 producto de ingresos provenientes del estado para adquisición de equipos de computo; asimismo, anualmente se recibe un aporte de S/.20,000.00 (como parte de los ingresos provenientes del área de diagnostico por imágenes y presupuesto en promedio) durante el año que se desarrollo el proyecto (gastos que incluyen el mantenimiento del software y de los equipos de computo).

4.5 Plan de desarrollo de software

4.5.1 Introducción

Este Plan de Desarrollo del Software es una versión preliminar preparada para ser incluida en la propuesta elaborada como respuesta al proyecto basado en una metodología de Rational Unified Process en la que únicamente se procederá a cumplir con las dos primeras fases que marca la metodología. Se incluirá el detalle para las fases de Inicio y Elaboración y adicionalmente se esbozarán las fases posteriores de Construcción y Transición para dar una visión global de todo proceso.

El enfoque desarrollo propuesto constituye una configuración del proceso RUP de acuerdo a las características del proyecto, seleccionando los roles de los participantes, las actividades a realizar y los artefactos (entregables) que serán generados. Este documento es a su vez uno de los artefactos de RUP.

4.5.1.1 Propósito

El propósito del Plan de Desarrollo de Software es proporcionar la información necesaria para controlar el proyecto. En él se describe el enfoque de desarrollo del software.

Los usuarios del Plan de Desarrollo del Software son:

- El jefe del proyecto lo utiliza para organizar la agenda y necesidades de recursos, y para realizar su seguimiento.
- Los miembros del equipo de desarrollo lo usan para entender lo qué deben hacer, cuándo deben hacerlo y qué otras actividades dependen de ello**.**

4.5.1.2 Alcance

El Plan de Desarrollo del Software describe el plan global usado para el desarrollo del "Sistema de información de ingreso de resultados de exámenes ecográficos". El detalle de las iteraciones individuales se describe en los planes de cada iteración, documentos que se aportan en forma separada. Durante el proceso de desarrollo en el artefacto "Visión" se definen las características del producto a desarrollar, lo cual constituye la base para la planificación de las iteraciones. Para la versión 1.0 del Plan de Desarrollo del Software, nos hemos basado en la captura de requisitos por medio del stakeholder representante del área para hacer una estimación aproximada, una vez comenzado el proyecto Posteriormente, el avance del proyecto y el seguimiento en cada una de las iteraciones ocasionará el ajuste de este documento produciendo nuevas versiones actualizadas.

4.5.1.3 Resumen

Después de esta introducción, el resto del documento está organizado en las siguientes secciones:

Vista General del Proyecto — proporciona una descripción del propósito, alcance y objetivos del proyecto, estableciendo los artefactos que serán producidos y utilizados durante el proyecto.

Organización del Proyecto — describe la estructura organizacional del equipo de desarrollo.

Gestión del Proceso — explica los costos y planificación estimada, define las fases e hitos del proyecto y describe cómo se realizará su seguimiento.

Planes y Guías de aplicación — proporciona una vista global del proceso de desarrollo de software, incluyendo métodos, herramientas y técnicas que serán utilizadas.

4.5.2 Vista General del Proyecto

4.5.2.1 Propósito, Alcance y Objetivos

La información que a continuación se incluye ha sido extraída de las diferentes reuniones que se han celebrado con el stakeholder del Hospital San José.

El Hospital San José, por la gran demanda y por mejorar la atención al paciente se plantea el desarrollo de un sistema de información de registro de resultados de exámenes ecográficos que formara parte del sistemas de Historia clínicas informatizadas; por tanto los solicitantes demandan una atención más rápida, automática y segura de las entregas de los resultados, reportes, estadística de todos los distintos exámenes ecográficos.

El proyecto debe proporcionar una propuesta para el desarrollo de todos los subsistemas implicados en dicho sistema de información. Estos subsistemas se pueden diferenciar en cuatro grandes bloques:

a) Subsistema de ingreso de paciente, incluyendo:
   - Procedimiento registro de paciente externos.
   - Procedimiento de registro de pacientes internos.
   - Procedimiento de registro de pacientes por emergencia.

b) Subsistemas de ingreso de citas, incluyendo:
   - Gestión de citas programadas.
   - Gestión de Reserva de citas.
   - Ingreso de roles de atención de médicos.
   - Cancelación de citas.
   - Modificación de citas programadas.
   - Gestión de consultas de citas.

c) Subsistema de ingreso de exámenes, incluyendo:
   - Ingreso de exámenes
   - Cancelación de exámenes
   - Modificación de exámenes
   - Consultas de exámenes

d) Subsistema de ingreso de resultados, incluyendo:
   - Ingreso de resultados de exámenes.
   - Consulta de resultado de exámenes.
   - Modificación de resultado de exámenes.

e) Subsistema de reportes.

4.5.2.2 Suposiciones y Restricciones

Las suposiciones y restricciones respecto del sistema, y que se derivan directamente de las entrevistas con el stakeholder de la empresa son:

- Sistemas seguros: protección de información, seguridad en las trasmisiones de datos.
- Gestión de flujos de trabajo, seguridad de transacciones e intercambio de información
- Adaptación a la normativa de Protección de Datos.

Como es natural, la lista de suposiciones y restricciones se incrementará durante el desarrollo del proyecto, particularmente una vez establecido el artefacto "Visión".

4.5.2.3 Entregables del proyecto

A continuación se indican y describen cada uno de los artefactos que serán generados y utilizados por el proyecto y que constituyen los entregables.

1) Plan de Desarrollo del Software
2) Modelo de Casos de Uso del Negocio
3) 7Modelo de Objetos del Negocio
4) Glosario
5) Modelo de Casos de Uso
6) Visión
7) Especificaciones de Casos de Uso
8) Especificaciones Adicionales
9) Prototipos de Interfaces de Usuario
10) Modelo de Análisis y Diseño
11) Modelo de Datos
12) Modelo de Implementación
13) Modelo de Despliegue
14) Casos de Prueba
15) Solicitud de Cambio
16) Plan de Iteración
17) Evaluación de Iteración
18) Lista de Riesgos
19) Manual de Instalación
20) Material de Apoyo al Usuario Final
21) Producto

4.5.2.4 Evolución del Plan de Desarrollo del Software

El Plan de Desarrollo del Software se revisará semanalmente y se refinará antes del comienzo de cada iteración.

4.5.3 Organización del Proyecto

Participantes en el Proyecto

Está formado por los siguientes puestos de trabajo y personal asociado:

Jefe de Proyecto. Con experiencia en metodologías de desarrollo, herramientas CASE y notaciones, en particular la notación UML y el proceso de desarrollo RUP.

Analista de Sistemas. El perfil establecido es: Ingeniero en Informática con conocimientos de UML, uno de ellos al menos con experiencia en sistemas afines a la línea del proyecto.

Analistas - Programadores. Con experiencia en el entorno de desarrollo del proyecto, con el fin de que los prototipos puedan ser lo más cercanos posibles al producto final.

Ingeniero de Software. El perfil establecido es: Ingeniero en Informática titulado que participará realizando labores de gestión de requisitos, gestión de configuración, documentación y diseño de datos. Encargada de las pruebas funcionales del sistema, realizará la labor de Tester.

Los Hojas de vida del personal del proyecto que ya ha comprometido su participación se adjuntan por separado.

Interfaces Externas

La Unidad de Diagnostico por Imagen designara a los participantes del proyecto que proporcionarán los requisitos del sistema, y entre ellos quiénes serán los encargados de evaluar los artefactos de acuerdo a cada subsistema y según el plan establecido.

El equipo de desarrollo interactuará activamente con los participantes de dicha Unidad, para la especificación y la validación de los artefactos generados.

Roles y Responsabilidades

A continuación se describen las principales responsabilidades de cada uno de los puestos en el equipo de desarrollo durante las fases de Inicio y Elaboración, de acuerdo con los roles que desempeñan en RUP.

Tabla N°17: Cuadro de responsables

Puesto	Responsabilidad
Jefe de Proyecto	El jefe de proyecto asigna los recursos, gestiona las prioridades, coordina las interacciones con los clientes y usuarios, y mantiene al equipo del proyecto enfocado en los objetivos. El jefe de proyecto también establece un conjunto de prácticas que aseguran la integridad y calidad de los artefactos del proyecto. Además, el jefe de proyecto se encargará de supervisar el establecimiento de la arquitectura del sistema. Gestión de riesgos. Planificación y control del proyecto.
Analista de Sistemas	Captura, especificación y validación de requisitos, interactuando con el cliente y los usuarios mediante entrevistas. Elaboración del Modelo de Análisis y Diseño. Colaboración en la elaboración de las pruebas funcionales y el modelo de datos.
Programador	Construcción de prototipos. Colaboración en la elaboración de las pruebas funcionales, modelo de datos y en las validaciones con el usuario
Ingeniero de Software	Gestión de requisitos, gestión de configuración y cambios, elaboración del modelo de datos, preparación de las pruebas funcionales, elaboración de la documentación. Elaborar modelos de implementación y despliegue.

4.5.4 Gestión del Proceso

4.5.4.1 Estimaciones del Proyecto

Esfuerzo realizado = $a\ KLDC^{\ b}$ *(persona x mes)* = 18.52 personas-mes

Tiempo estimado= = $c\ Esfuerzo^{\ d}$ *(meses)* = 6.8 mes

Nº de personas para desarrollar el proyecto = E/T= 18.52 /6.8 ≈ 3 personas

4.5.4.2 Plan del Proyecto

4.5.4.2.1 Plan de las Fases

Tabla N°18: Cuadro de Fases

Fase	Nro. Iteraciones	Duración
Fase de Inicio	1	4 semanas
Fase de Elaboración	2	8 semanas
Fase de Construcción	2	10 semanas
Fase de Transición	1	4 semanas

Los hitos que marcan el final de cada fase se describen en la siguiente tabla.

Tabla N°19: Cuadro de descripción Hitos

Descripción	Hito
Fase de Inicio	En esta fase desarrollará los requisitos del producto desde la perspectiva del usuario, los cuales serán establecidos en el artefacto Visión. Los principales casos de uso serán identificados y se hará un refinamiento del Plan de Desarrollo del Proyecto. La aceptación del cliente / usuario del artefacto Visión y el Plan de Desarrollo marcan el final de esta fase.
Fase de Elaboración	En esta fase se analizan los requisitos y se desarrolla un prototipo de arquitectura Al final de esta fase, todos los casos de uso correspondientes a requisitos que serán implementados en la primera release de la fase de Construcción deben estar analizados y diseñados (en el Modelo de Análisis / Diseño). La revisión y aceptación del prototipo de la arquitectura del sistema marca el final de esta fase. En nuestro caso particular, por no incluirse las fases siguientes, la revisión y entrega de todos los artefactos hasta este punto de desarrollo también se incluye como hito. La primera iteración tendrá como objetivo la identificación y especificación de los principales casos de uso, así como su realización preliminar en el Modelo de Análisis / Diseño.
Fase de Construcción	Durante la fase de construcción se terminan de analizar y diseñar todos los casos de uso, refinando el Modelo de Análisis / Diseño. El producto se construye en base a 2 iteraciones, cada una produciendo una release a la cual se le aplican las pruebas y se valida con el cliente / usuario. Se comienza la elaboración de material de apoyo al usuario. El hito que marca el fin de esta fase es la versión de la release 3.0, con la capacidad operacional parcial del producto que se haya considerado como crítica, lista para ser entregada a los usuarios para pruebas beta.
Fase de Transición	En esta fase se prepararán dos releases para distribución, asegurando una implantación y cambio del sistema previo de manera adecuada, incluyendo el entrenamiento de los usuarios. El hito que marca el fin de esta fase incluye, la entrega de toda la documentación del proyecto con los manuales de instalación y todo el material de apoyo al usuario, la finalización del entrenamiento de los usuarios y el empaquetamiento del producto.

4.5.4.2.2 Calendario del proyecto

Tabla N°20: Cuadro de duración – fase de inicio

Disciplinas / Artefactos generados o modificados durante la Fase de Inicio	**Comienzo**	**Aprobación**
Modelado del Negocio		
Modelo de Casos de Uso del Negocio.	Semana 1	Semana 3
Requisitos		
Glosario.	Semana 1	Semana 3
Visión.	Semana 2	Semana 3
Modelo de Casos de Uso.	Semana 3	siguiente fase
Especificación de Casos de Uso.	Semana 3	siguiente fase
Análisis / Diseño		
Modelo de Análisis / Diseño.	Semana 2	siguiente fase
Modelo de Datos.	Semana 2	siguiente fase
Implementación		
Prototipos de Interfaces de Usuario.	Semana 3	siguiente fase
Modelo de Implementación.	Semana 3	siguiente fase
Pruebas		
Casos de Pruebas Funcionales.	Semana 3	siguiente fase
Despliegue		
Modelo de Despliegue.	Semana 4	siguiente fase
Gestión de Cambios y Configuración	Durante todo el proyecto	
Gestión del proyecto		
Plan de Desarrollo del Software en su versión 1.0 y planes de las Iteraciones.	Semana 1	Semana 4
Ambiente	Durante todo el proyecto	

Tabla N°21: Cuadro de duración – fase de elaboración

Disciplinas / Artefactos generados o modificados durante la Fase de Elaboración	Comienzo	Aprobación
Modelado del Negocio		
Modelo de Casos de Uso del Negocio.	Semana 1	aprobado
Requisitos		
Glosario.	Semana 1	aprobado
Visión.	Semana 2	aprobado
Modelo de Casos de Uso.	Semana 4	Semana 5
Especificación de Casos de Uso.	Semana 4	Semana 6
Especificaciones Adicionales.	Semana 4	Semana 6
Análisis / Diseño		
Modelo de Análisis / Diseño.	Semana 2	Revisar en cada iteración
Modelo de Datos.	Semana 2	Revisar en cada iteración
Implementación		
Prototipos de Interfaces de Usuario.	Semana 5	Revisar en cada iteración
Modelo de Implementación.	Semana 5	Revisar en cada iteración
Pruebas		
Casos de Pruebas Funcionales.	Semana 5	Revisar en cada iteración
Despliegue		
Modelo de Despliegue.	Semana 6	Revisar en cada iteración
Gestión de Cambios y Configuración	Durante todo el proyecto	
Gestión del proyecto		
Plan de Desarrollo del Software en su versión 2.0 y planes de las Iteraciones.	Semana 4	Revisar en cada iteración
Ambiente	Durante todo el proyecto	

4.5.4.3 Seguimiento y Control del Proyecto

Gestión de Requisitos

Los requisitos del sistema son especificados en el artefacto Visión. Cada requisito tendrá una serie de atributos tales como importancia, estado, iteración donde se implementa, etc. Estos atributos permitirán realizar un efectivo seguimiento de cada requisito. Los cambios en los requisitos serán gestionados mediante una Solicitud de Cambio, las cuales serán evaluadas y distribuidas para asegurar la integridad del sistema y el correcto proceso de gestión de configuración y cambios.

Control de Plazos

El calendario del proyecto tendrá un seguimiento y evaluación semanal por el jefe de proyecto y por el Comité de Seguimiento y Control.

Control de Calidad

Los defectos detectados en las revisiones y formalizados también en una Solicitud de Cambio tendrán un seguimiento para asegurar la conformidad respecto de la solución de dichas deficiencias Para la revisión de cada artefacto y su correspondiente garantía de calidad se utilizarán las guías de revisión y checklist (listas de verificación) incluidas en RUP.

Gestión de Riesgos

A partir de la fase de Inicio se mantendrá una lista de riesgos asociados al proyecto y de las acciones establecidas como estrategia para mitigarlos o acciones de contingencia. Esta lista será evaluada al menos una vez en cada iteración.

Gestión de Configuración

Se realizará una gestión de configuración para llevar un registro de los artefactos generados y sus versiones. También se incluirá la gestión de las Solicitudes de Cambio y de las modificaciones que éstas produzcan, informando y publicando dichos cambios para que sean accesibles a todo los participantes en el proyecto. Al final de cada iteración se establecerá una baseline (un registro del estado de cada artefacto, estableciendo una versión), la cual podrá ser modificada sólo por una Solicitud de Cambio aprobada.

Capitulo 5.- Fase de Análisis y Diseño - Descripción de la Solución Tecnológica

5.1. Situación Actual del Proceso de Recepción de Consulta y Entrega de Resultados (Ecografías)

- El médico del consultorio genera la consulta respectiva para que se pueda realizar dicho examen, excepto cuando viene de otro centro de salud (particulares).
- El paciente paga en caja, y se dirige a ecografía.
- Recepcionan la boleta de pago y su solicitud de consulta, dicha solicitud se le engrampa con la copia de su boleta de pago (en caso que la solicitud tuviera también examen de rayos X, se le pedirá al paciente sacar una copia para que lo presente para rayos X) solo si encuentra cupo, se le pide su carné de consulta siempre y cuando no sea por el S.I.S (Sistema Integral de Salud). y se llenan los datos en un cuaderno de control.
- Los pacientes que vienen de emergencia y pacientes que vienen con un diagnóstico reservado se le dan la prioridad del caso (pe. pacientes que presenta un cuadro de sangrado), no requieren previa cita.
- Los datos que llenan en el cuaderno de control son los siguientes:
  - Edad del paciente
  - Numero de orden diario (1, 2, 3, 4,…., etc.) del turno, el cual se reiniciara para el turno siguiente (mañana y tarde).
  - Numero correlativo general el cual es el código de todas las consultas por turno; es decir, el turno mañana y tarde son códigos distintos.
  - Apellidos y nombres del paciente(en ese orden)
  - Tipo de ecografías, solo una breve descripción del examen (pe. TV = transvaginal).
  - Llenan con una "x" la columna que pertenece al consultorio que hizo el requerimiento de dicho examen (1.- Ginecología, 2.- Pediatría, 3.-Medicina, 4.- Cirugía, 5.- Particular, 6.- Emergencia, 7.-Hospitalización, 8.- S.I.S.)
  - Nombre del médico que solicita dicho examen.
- Distrito de vivienda del paciente
- Una vez ingresado los datos del paciente, se llena en la solicitud de consulta los datos faltantes: Historia clínica, edad y distrito; después se le devuelve la boleta azul, indispensable para recoger los resultados. Luego la técnica la orientara como debe venir o que requisitos debe cumplir para que puedan realizarse el examen respectivo, ya sea en ayunas, ropa ligera, ingesta de líquidos, etc.

- Hay exámenes que sólo se realizaran en la mañana, como el caso de la ecografía abdominal, ya que comprende en sus requisitos que se venga en ayunas.
- La cantidad de registros depende de la cantidad de exámenes que se realizara la persona. Si un paciente se va a realizar 3 exámenes, entonces tendrá 3 registros, cada uno indicando un solo examen.
- Con respecto a los cupos, cada médico especialista de turno que realiza dichos exámenes, solicitará una cantidad de cupos a realizar (mañana y tarde). Tratando de prever los exámenes por emergencia y los de alto riesgo.
- Si el paciente no encuentra cupo (con excepción de pacientes que vienen por emergencia), se le citara para un día y una hora especifica, que se le escribirá en la boleta de pago; pero no se le registra; solo se contabiliza en el cuaderno de citas (mañana y tarde) y se guarda la hoja o solicitud de consulta engrampado con la copia de la boleta de pago. Las citas comprenden de las 8 AM., 9 AM, 10 AM. Y 2PM
- Los citados, cuando llega el día de la cita, recién le registran en el cuaderno ya que en la cita se quedaron con la hoja de la solicitud en donde apuntan todos los datos faltantes.
- Todas las solicitudes registradas en el cuaderno de control, son llevadas al doctor especialista para el llamado de los pacientes.
- Con respecto a los resultados del examen, el doctor escribe el resultado en el reverso de la solicitud de examen de diagnóstico por imágenes; también puede ser dictado por el médico, o puede hacer una grabación de voz; la cual es ingresada en un procesador de texto (pe. Word) por la secretaria para luego proceder a su impresión.
- Una vez impreso son engrampados junto con las imágenes de los resultados y entregados al médico responsable que procederá a firmar los resultados, con lo cual se llamara a los pacientes, para realizar la entrega de los resultados, previa presentación de su boleta de pago, se le sellará el sobre y la boleta con la etiqueta "ENTREGADO", con la respectiva fecha de entrega.
- Hay caso en donde el doctor hace todo el proceso, ya sea por que el personal se ha retirado, y tiene que atender una emergencia; en ese caso, llena una fichas pre impresas y los entrega con los resultados.

5.2. Modelo de Negocio

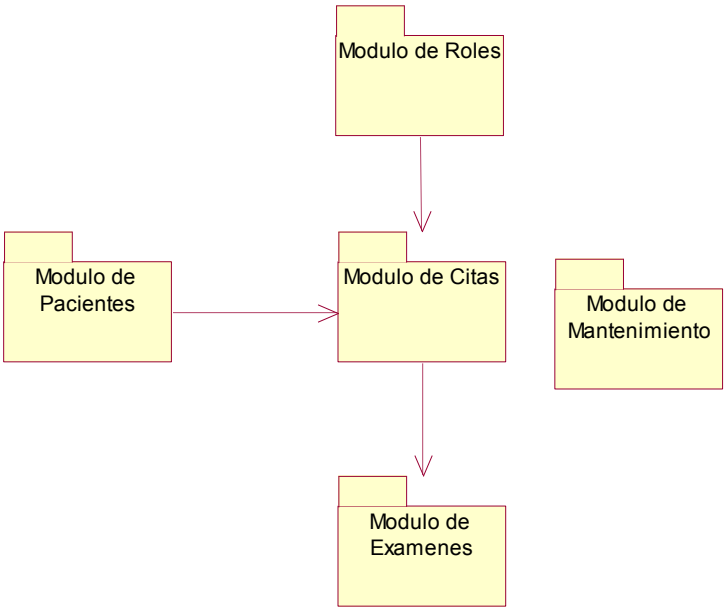


Figura 44: Diagrama de módulos del sistema

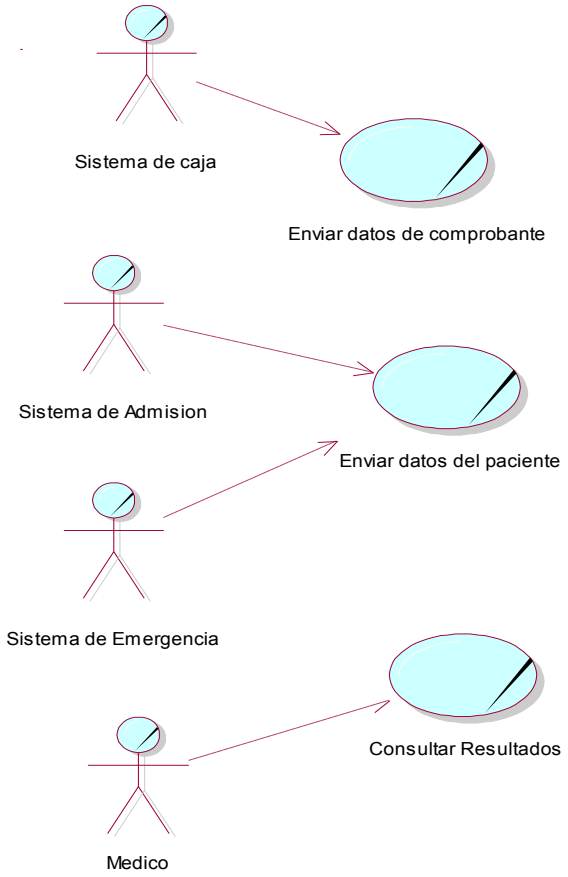
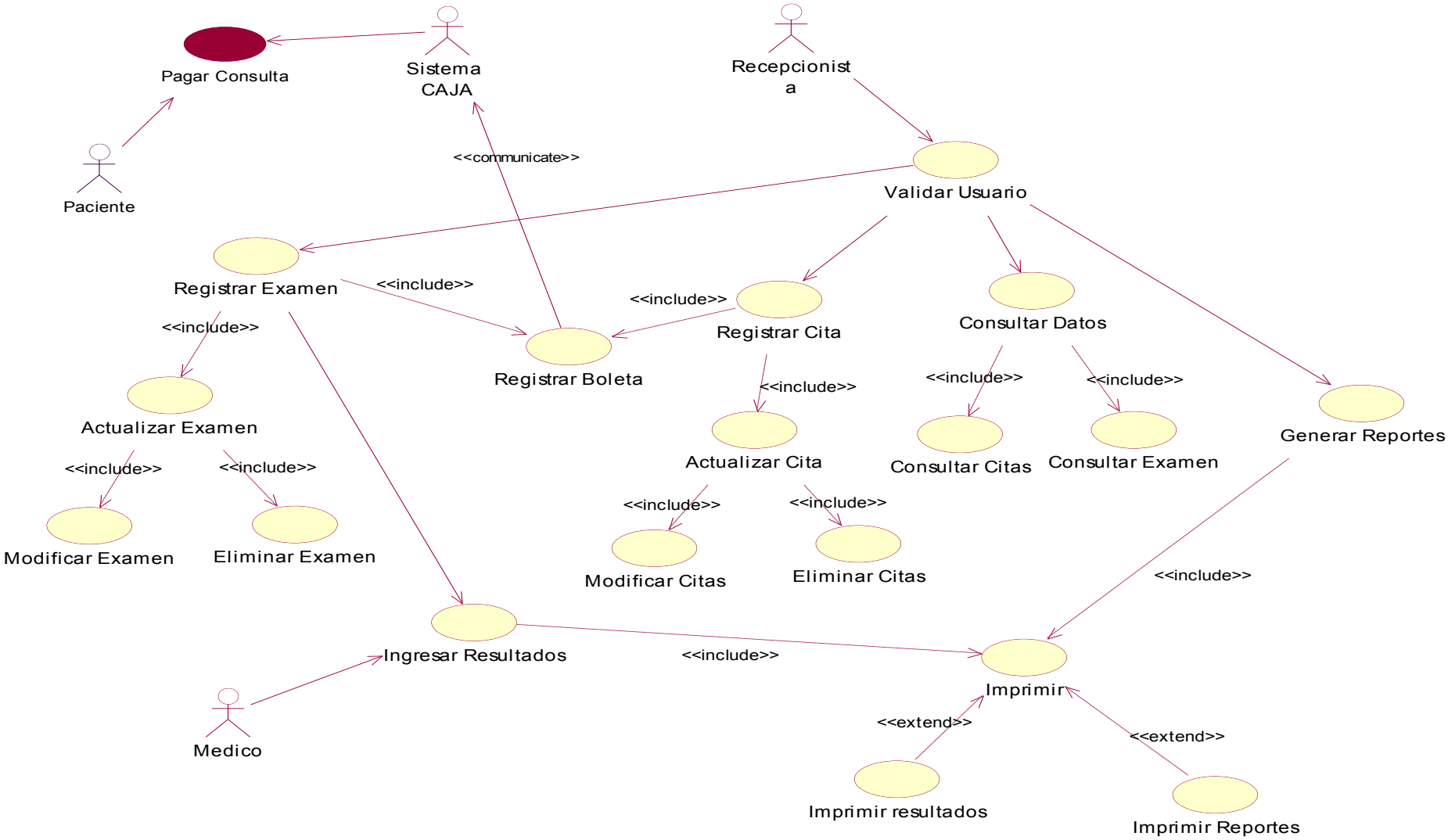


Figura 45: Modelo de casos de uso del negocio

5.3. Modelo de Caso de Uso



5.4. Especificaciones de caso de uso.

Caso de Uso	Registrar Boleta
Actores	Caja, Recepcionista.
Propósito	Registrar al paciente con su respectiva boleta de pago.
Resumen	Este caso es iniciado por la recepcionista, se valida el numero y el estado de la boleta de pago, para su registro.
Precondiciones	La boleta debe a ver sido cancelado, y la recepcionista se debe haber logeado en el sistema.
Flujo Principal	Este caso comienza cuando se inicia el registro una cita, en el cual debe presentar su boleta de pago, para proseguir el registro.

Caso de Uso	Registrar Cita
Actores	Recepcionista.
Propósito	Registrar al paciente para una cita.
Resumen	Este caso es iniciado por la recepcionista, para registrar o separar una cita.
Precondiciones	La boleta previamente debe a ver sido registrada y el paciente debe poseer su hoja de solicitud de consulta en donde especifique el examen.
Flujo Principal	Este caso comienza cuando se inicia el registro una cita en el sistema, previo registro de la boleta, luego el sistema proporciona los datos del paciente si cuenta con su historia clínica, y si es externo, se visualizara los datos que fue registrado en la boleta de pago, con la posibilidad de cambiarlos solo en esta etapa.

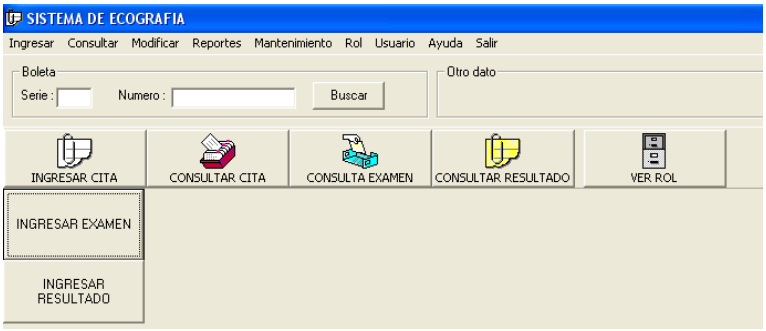
Caso de Uso	Registrar Examen
Actores	Recepcionista.
Propósito	Registrar un tipo de examen al paciente.
Resumen	Este caso es iniciado por la recepcionista, la cual registra un examen ecográfico al paciente.
Precondiciones	La fecha de su cita debe coincidir con la fecha de registro de su examen, y cumplir con los requisitos previos al examen (p.e. ayunas, etc.).
Flujo Principal	El paciente debe acercarse a ventanilla, para registrar su asistencia a la cita convocada, con el cual se registrara el tipo de examen solicitado y esperara su turno para hacer atendido.

Caso de Uso	Ingresar Resultados
Actores	Recepcionista, Medico.
Propósito	Registrar los resultados del examen.
Resumen	Este caso es iniciado por el medico de turno, el cual registra o entrega un borrador, para que la recepcionista o la técnica de turno lo ingrese al sistema.
Precondiciones	Previamente a ver sido citado, estar registrado para el examen y ser atendido por el medico.
Flujo Principal	La auxiliar en enfermería o la recepcionista (operador) ingresan los resultados emitidos por el doctor, en formato de audio y/o texto, al sistema.

5.5. Prototipo de Interfaces de Usuario.

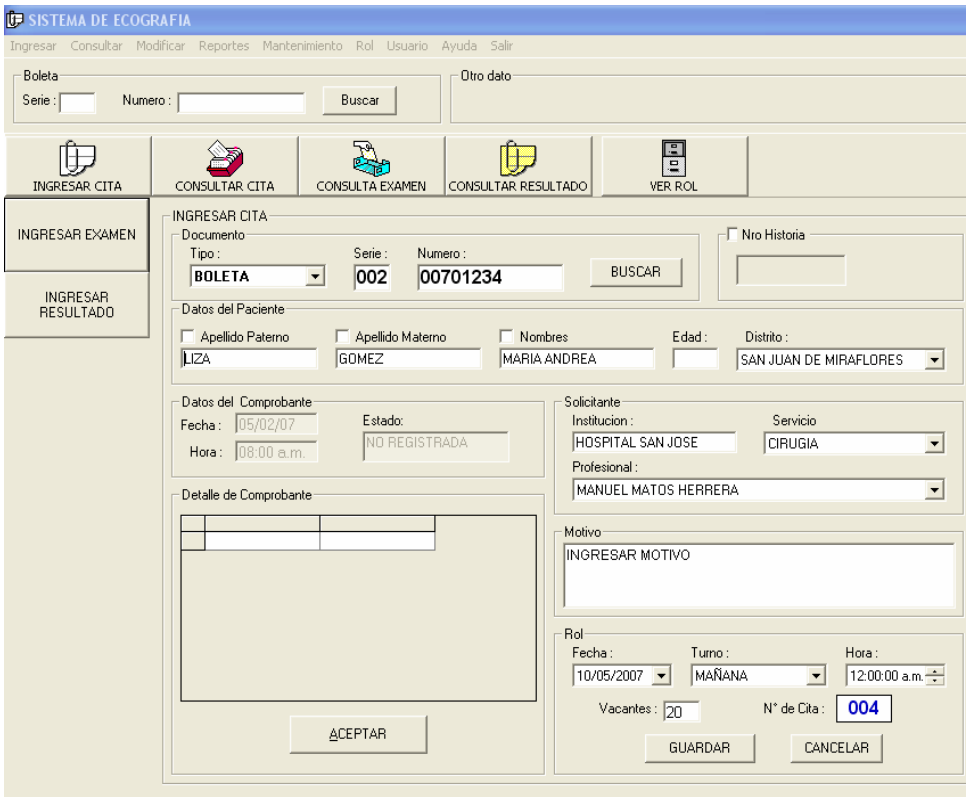
Menú principal

Esta ventana me presenta las principales funciones que realizara el sistema



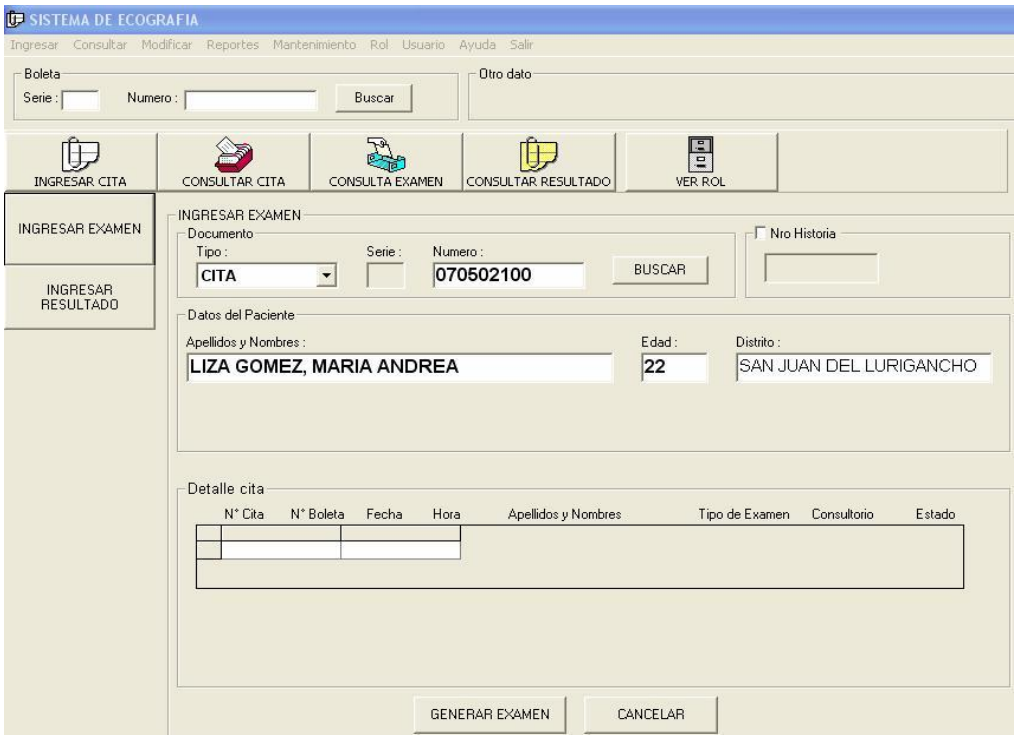
Ingresar cita

Esta ventana me muestra el ingreso de una cita, previa verificación de la boleta de pago del paciente.



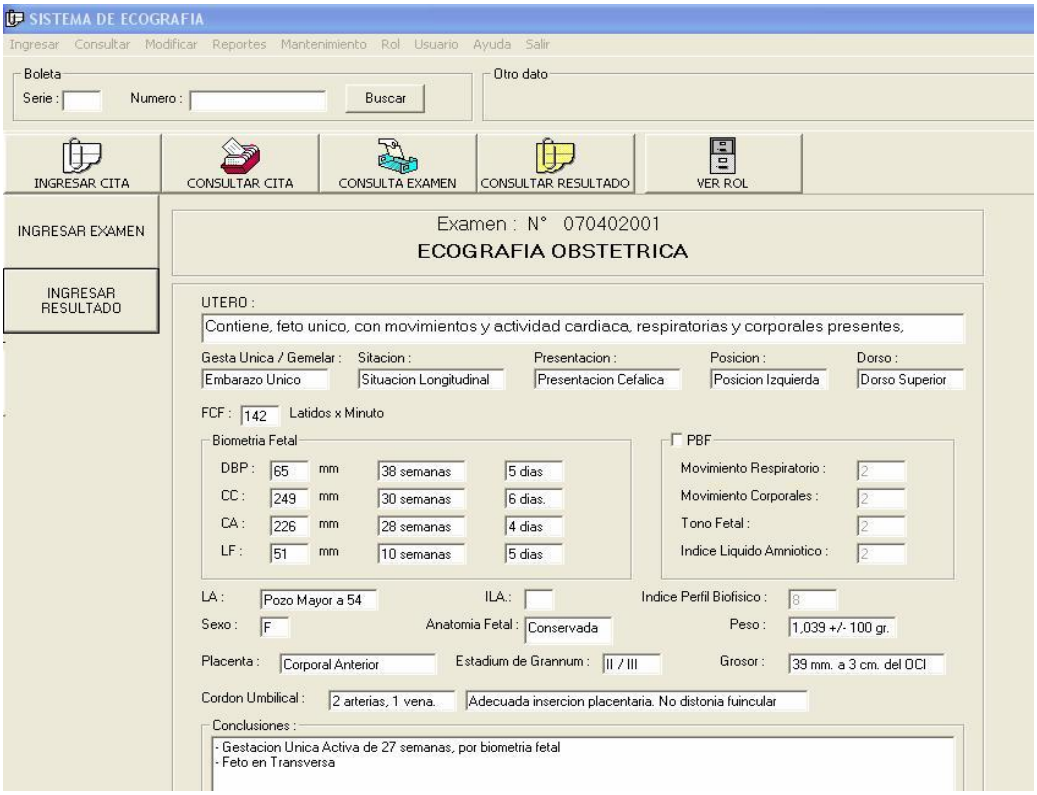
Ingresar Examen

Esta ventana nos muestra el ingreso de exámenes, previo registro de la cita



Ingresar resultado (Ecografía Obstétrica)

Esta ventana nos muestra el ingreso del resultado de un examen ecográfico, en este caso una ecografía obstétrica.



5.6. Modelo de Análisis y Diseño

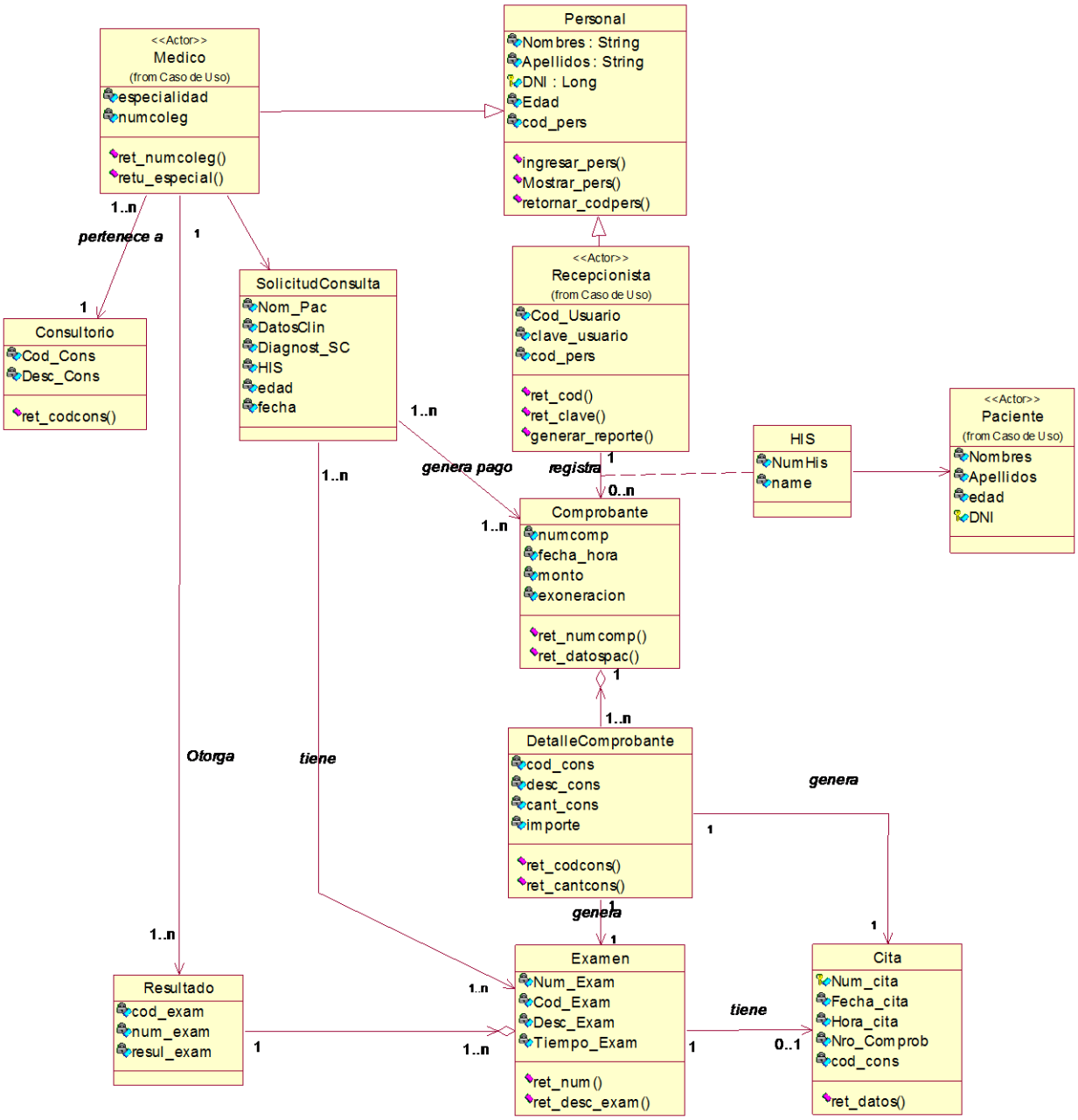
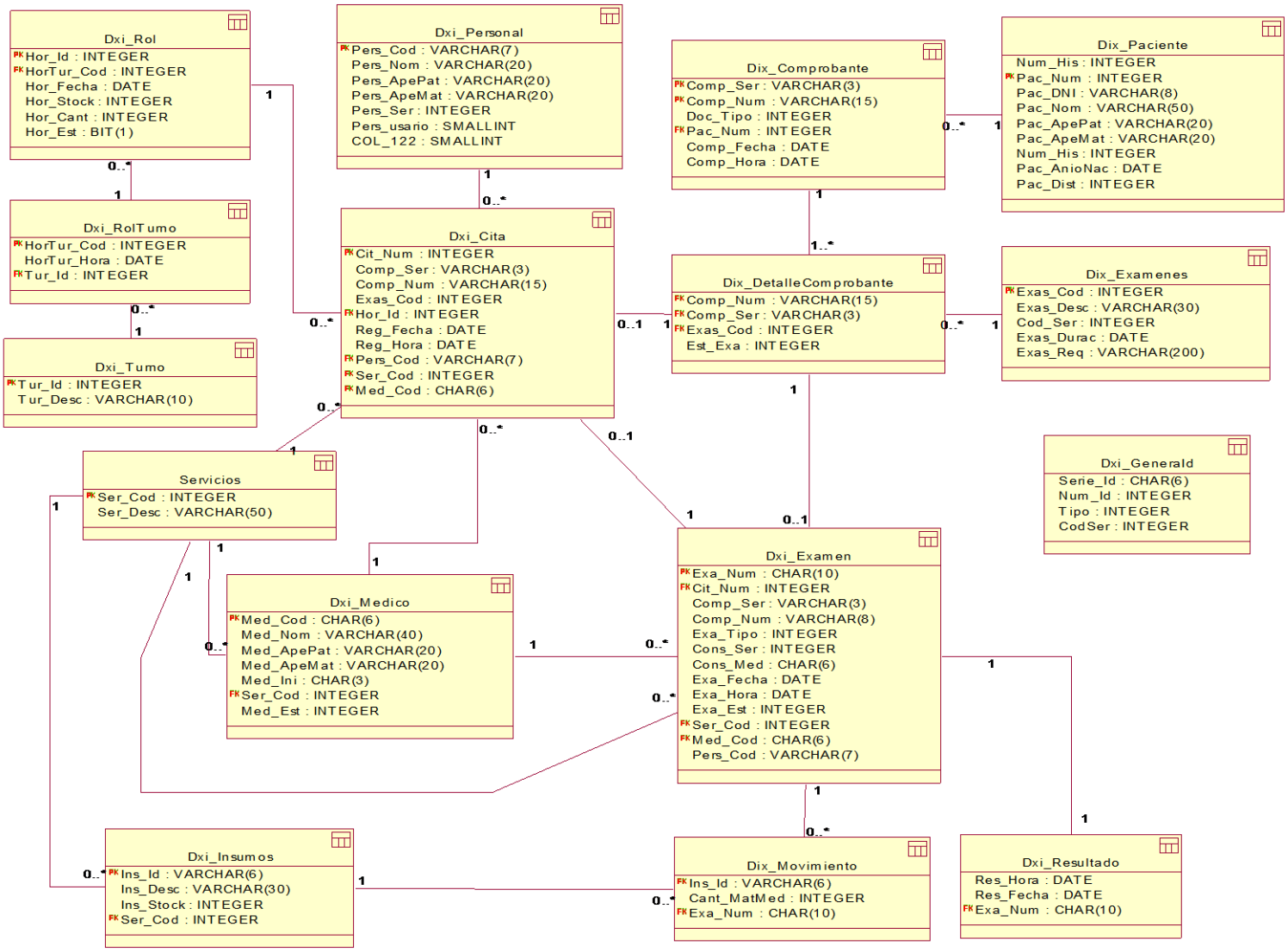


Figura 46: Diagrama de Clases

5.7. Modelo de dato



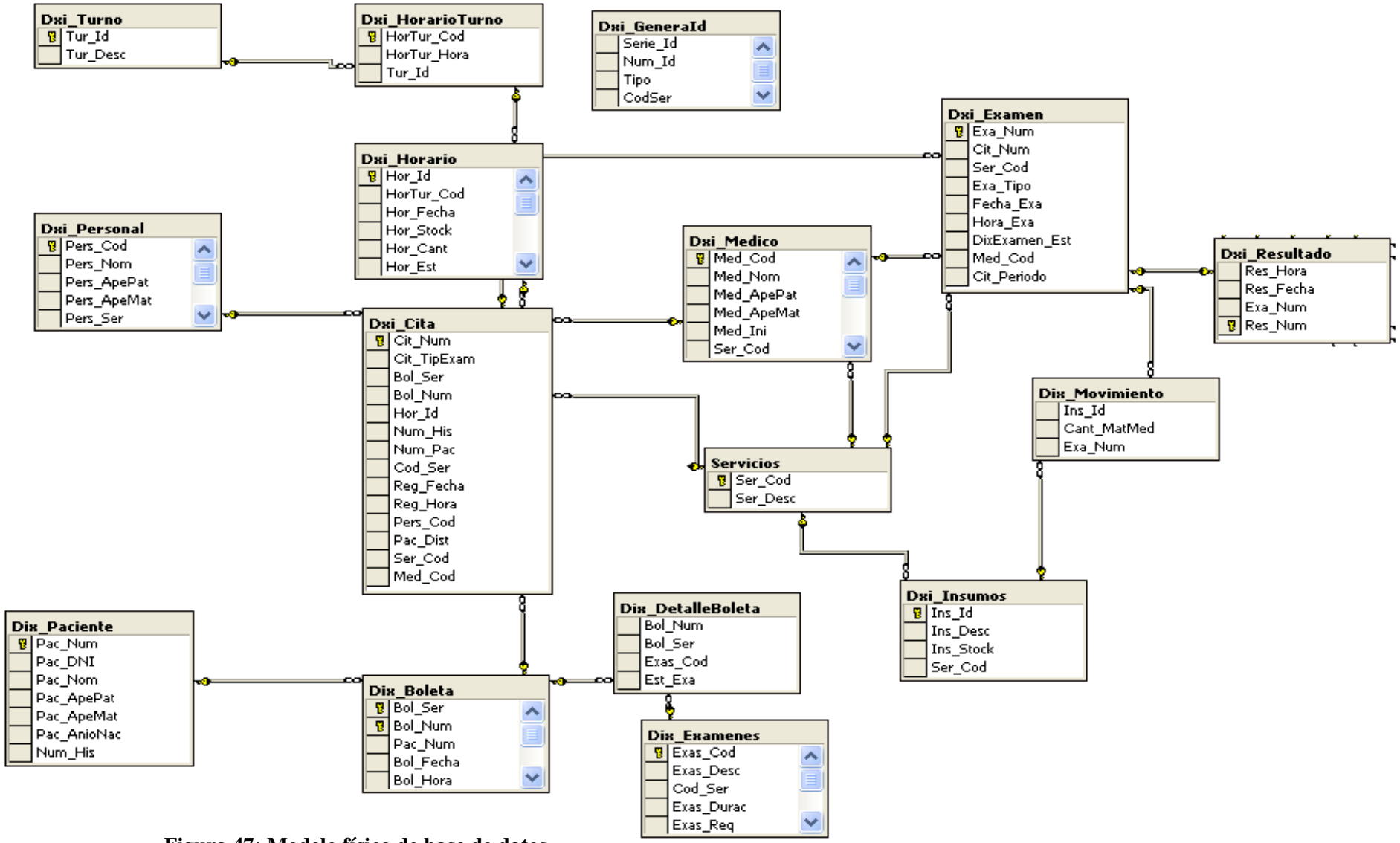


Figura 47: Modelo físico de base de datos

5.8. Modelo dinámico.

5.8.1. Diagrama de Secuencia

Registrar Comprobante

Nos muestra cómo interactúan los objetos a la hora de registrar un comprobante (boleta, factura, Cta. Cte.)

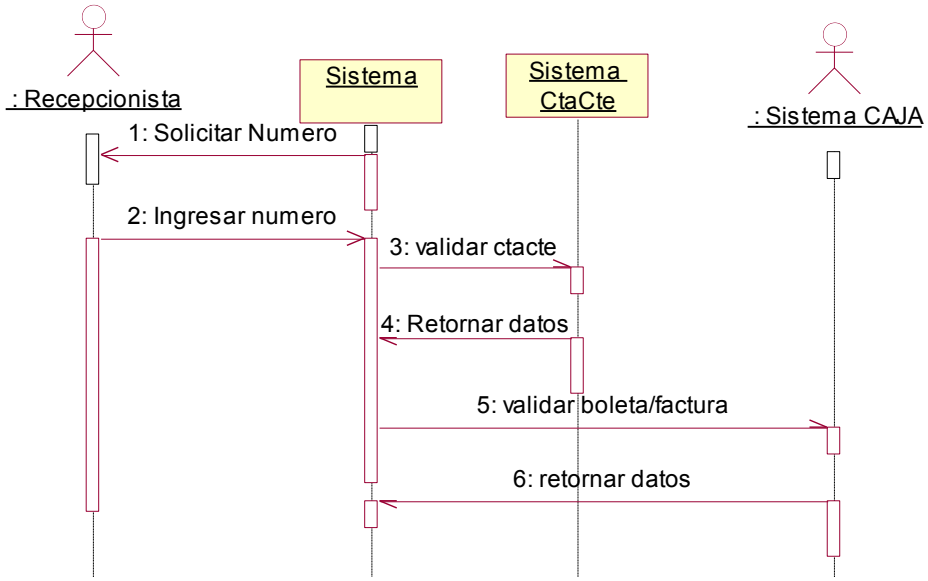


Figura 48: Diagrama de Secuencia Registrar_Comprobante

Validar usuario

Nos muestra que objetos interactúan a la hora de validar usuarios

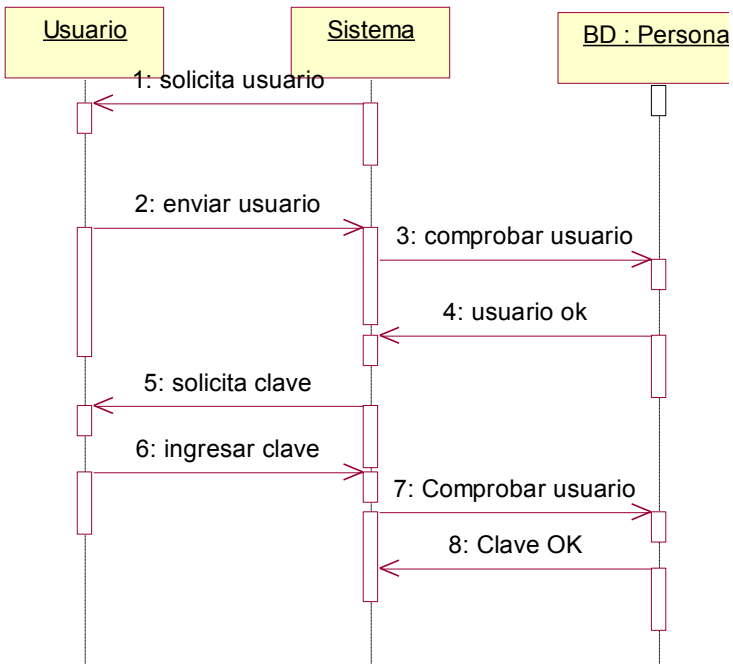


Figura 49: Diagrama de Secuencia Validar_Usuario

Consultar Datos

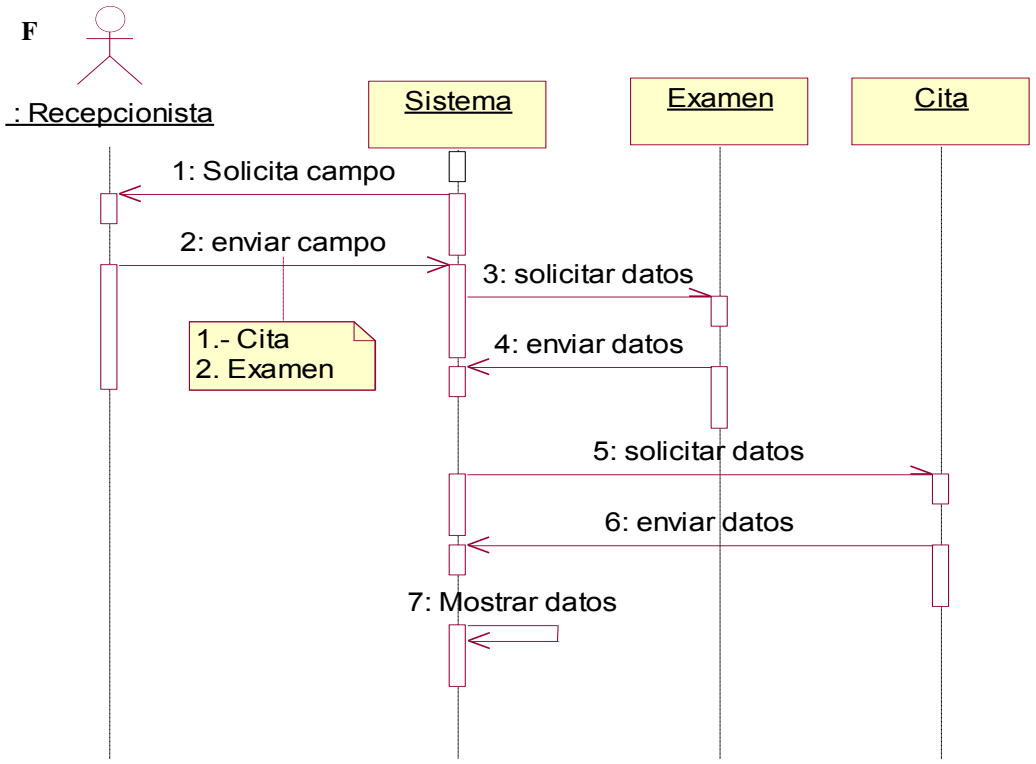


Figura 50: Diagrama de Secuencia Consultar_Datos

Registrar Cita

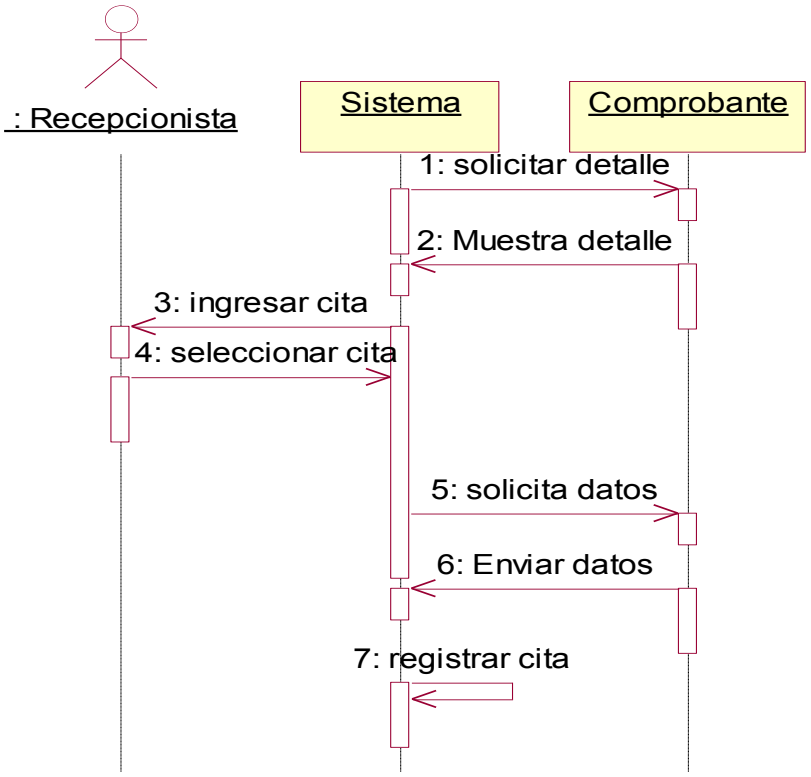


Figura 51: Diagrama de Secuencia Registrar_Cita

Registrar Examen

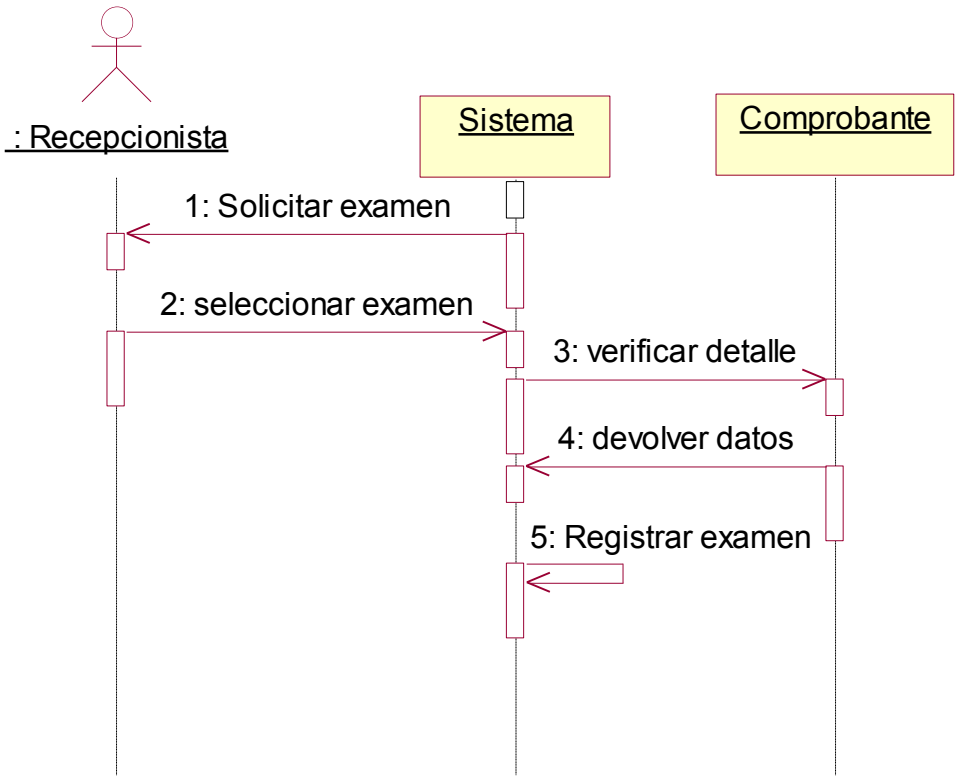


Figura 52: Diagrama de Secuencia Registrar_Examen

5.8.2. Diagrama de Colaboración

Registrar boleta

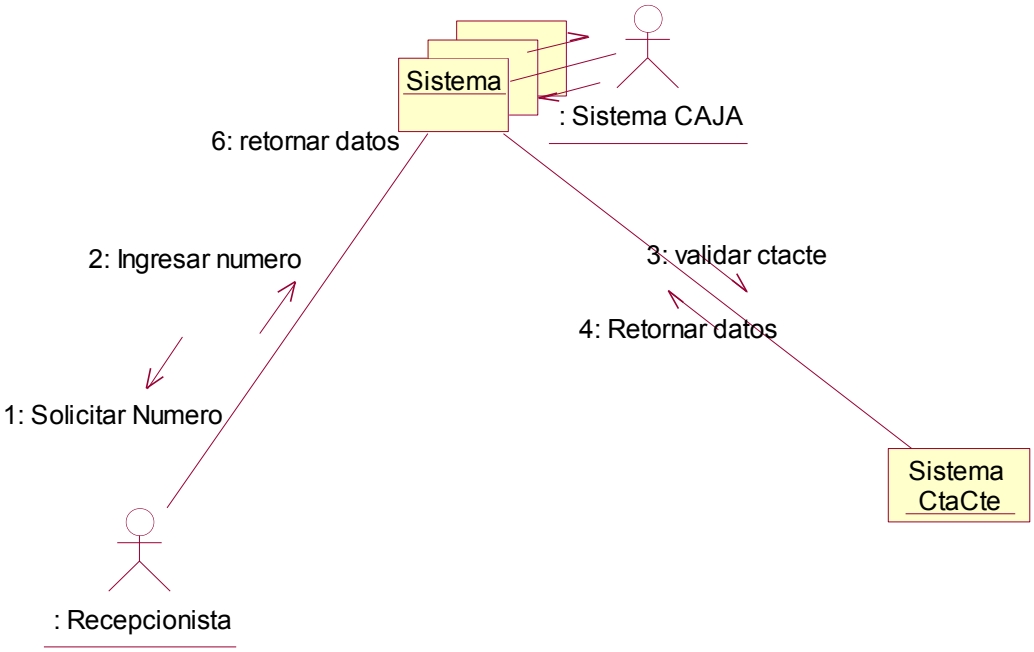


Figura 53: Diagrama de Colaboración Registrar_Boleta

Validar usuario

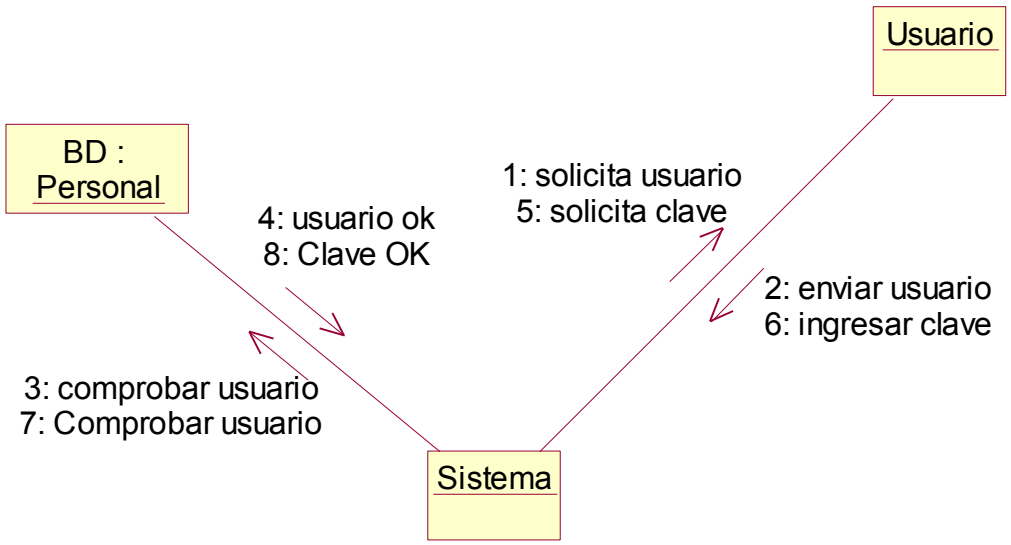


Figura 54: Diagrama de Colaboración Validar_Usuario

Registrar Examen

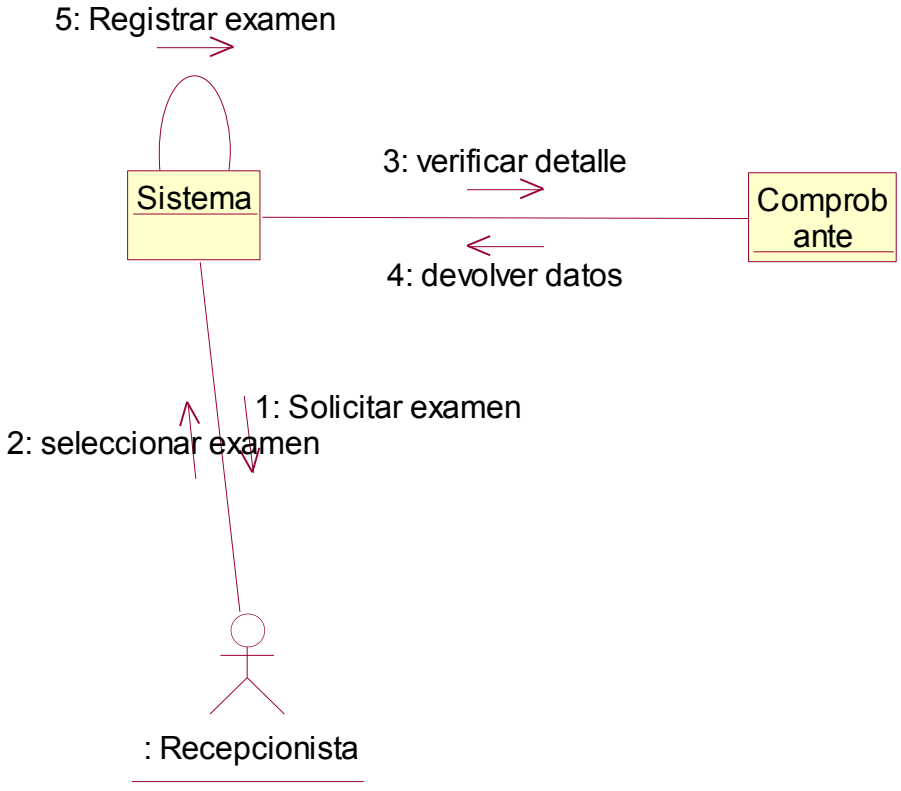


Figura 55: Diagrama de Colaboración Registrar_Examen

5.10 Modelo de despliegue

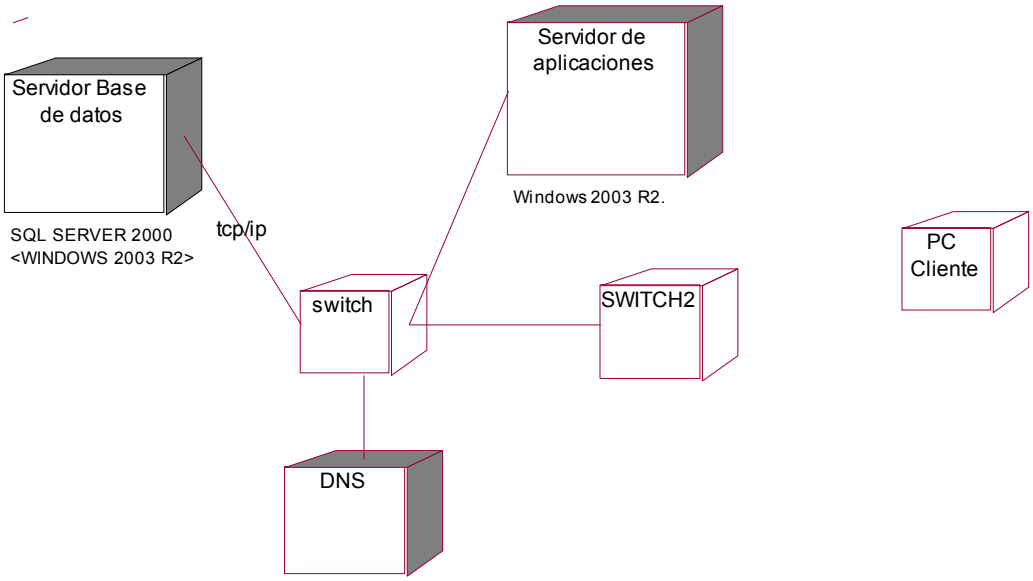


Figura 56: Diagrama de Despliegue

Capitulo 6.- Conclusiones y futuros trabajos

Una vez aplicado el lenguaje UML a la solución de la problemática de la institución y obteniendo la estructura del desarrollo del sistema de información, se obtuvieron los siguientes resultados permitiendo presentar el siguiente conjunto de conclusiones.

- En lo referido al plan de desarrollo de software para la implementación del sistema de información en el área de diagnostico por imágenes, se puede concluir que se elaboro el plan de desarrollo de software.
- En lo referido a la descripción de la metodología RUP y el lenguaje UML, se puede concluir que se realizo, mostrando las principales características y artefactos de esta metodología y modelos realizados con UML.
- En lo referido a la Implementación del plan de desarrollo, a manera de ejemplo, para la automatización de los procesos de registro de resultado de los exámenes ecográficos , se puede decir que se logro cubrir las fases de análisis y diseño del ciclo de vida del desarrollo del software; así como, un prototipo de las interfaces de usuario del sistema.

Como parte de los trabajos a ser realizados a futuro, podríamos citar:

- Implementar en su totalidad el plan de desarrollo de software propuesto, concluyendo en un producto final,
- Un plan de desarrollo para la Integración de los sistemas de información dentro del hospital, basado en una estandarización de los datos.

REFERENCIAS BIBLIOGRÁFICAS

[ArgoUML] Tigris Open Source Engineering, http://argouml.tigris.org/

[Asenjo07] Asenjo Sebastián Miguel Angel, Gestión diaria del hospital, Elsevier, (2007), España.

[Arce03] Rafael Arce M, Diseño y Explotación de las bases de datos relacionales. Escuela de geografía, Universidad de Costa Rica. (2003)

[Beck00] Beck Kent, Extreme Programming Explained, Addison-Wesley, (2000)

[Blanco06+] Blanco Solsona Antonio, Jose Manuel Huidobro Moya, J. Jordan Calero, Redes de Área Local: Administración de sistemas, 332 pág., Edit. Thompson Paraninfo, (2006), Madrid – España.

[Booch00+] Booch G., Jacobson, I. y Rumbaugh, J., El Proceso Unificado de Desarrollo de Software, Pearson Educación, (2000), Madrid - España.

[Booch99+] Booch G., Rumbaugh J., Jacobson I.; "The Unified Modeling Language User Guide", Addison-Wesley, (1999), Boston - EEUU.

[Campoli03] Campoli Marcela, Sistemas de información en el Sector Salud: "Utopia o realidad", Fundación Universitaria Dr. René Favaloro, (2003), Buenos Aires - Argentina.

[Clark88] Clark David D., The design philosophy of the internet protocols: SIGCOMM", organization ACM, (1988), CA – EEUU.

[Clarion03] Clarión Software, Clario Business Rules Manager Application Guide.(2003)

[Clements03] Clements Paul; Northrop Linda, Software Architecture: An executive Overview. Technical report CMU/SEI-96-TR-003.

[Conallen99] Conallen Jim, Building Web Applications with UML. 2da Edicion España Addison Wesley. (1999)

[Conallen99] Conallen, J., Modeling web application architectures with UML, Comm. ACM, (1999).

[Date01] Date J.C., Introducción a los Sistemas de Base de Datos, Alhambra mexicana, (2001), México.

[Dewayne03] Dewayne E. Perry Foundations for the Study of Software Architecture. ACM SIFSOFT, (2003)

[Dorsey97] Dorsey Paul, manual de Oracle Designer España Mc Graw Hill, (1997)

[Fenton91] Fenton E. Norman. Software Metrics A rigorous approach, Chapman & Hall, (1991), Londres – Inglaterra.

[Garlan03+] Garlan David, Shaw Mary, An Introduction to Software Architecture. School of Computer Science. Carnegie Mellon University, (2003), Pittsburgh

[Garzon03+] Garzon Villar María Luisa, Informática, Mad-Eduforma, (2003), Sevilla – España.

[Georgas03] Georgas John, recommendations for Architecture-Cntric Software Support self-adaptive behavior,(2003).

[Gomez03] Gómez del Moral Martin, tendencia de futuro de las herramientas UML, Universidad Politécnica de Valencia, (2003), Valencia – España.

[Iconix02] www.spinec.org/wp-content/ICONIX.pdf

[Laudon04] Laudon Jane, Sistemas de información gerencial: Administración de la empresa, Pearson Education, (2004), 564 pag. Madrid – España.

[Hesselbach06] Hesselbach Serra Xavier, Análisis de redes y sistemas de comunicaciones, Ediciones UPC, (2006), 195 pág., Barcelona – España.

[Highsmith02] Highsmith, Jim, Agile Software Development ecosystems, Addison Wesley, 2003, Madrid – España.

[HSJ07] Hospital San José, Análisis de la situación de salud 2007 del Hospital San Jose - Callao, (2007), Callao - Perú

[Huidobro06] Huidobro Moya Jose Manuel, Redes y servicios de comunicaciones, Edit. Thompson Paraninfo, 477 pág. (2006), Madrid – España.

[Jacobson93] I. Jacobson, “Time for a cease-fire in the methods war”, Journal Object-Oriented Programming, (1993).

[Jacobson00] Jacobson, I., Booch, G., Rumbaugh J., El Proceso Unificado de Desarrollo de Software, (2000) Addison Wesley, Madrid – España.

[Kendall05] Kendall Kenneth, Análisis y diseño de sistemas, Pearson Education, (2005), Madrid – España.

[Kenneth 04] Kenneth C. Lourden, Lenguajes de programación, principios y prácticas, Cengage Learning Editores, 2da edición, (2004), 633 pág., Madrid - España.

[Korth90] Korth Henry, Levy Eliezer, A Formal Aprroach to Recovery by Compensating Transactions, Conf. on very large data bases, (1990), Brisbane – Australia.

[Kruchten04] Kruchten Philippe, The Rational Unified Process, Addison-Wesley, (2004), Londres – Inglaterra.

[Kruchten00] Kruchten, Philippe., The Rational Unified Process: An Introduction, (2000) Addison Wesley, Londres – Inglaterra.

[Lain79] Lain Entralgo, El diagnóstico médico, Ed. Salvat, (1979), Lisboa- Portugal.

[Larman99] Larman Craig, UML y Patrones: Introducción al análisis y diseño orientado a objetos, Prentice Hall, (1999), México.

[Larman2003] Larman Craig, UML y Patrones: Una Introducción al Análisis y al Diseño Orientado a Objetos y al Proceso unificado. Madrid Prentice Hall.

[Marimon99] Marimon Suñol Santiago, La sanidad en la sociedad de la información, Sistemas y tecnologías de la información para la gestión y la reforma de los servicios de salud, Ediciones Díaz de Santos, (1999), Madrid - España.

[Marquez01] Márquez Andrés, SQL y desarrollo de aplicaciones en Oracle 8, Universitat Jaume I, (2001), Castelló - España

[Meyer1999] Meyer Bertrand, Construcción de Software Orientado a Objetos. España Prentice Hall

[Microsoft98] Microsoft Corporation, Manual del programador Microsoft Visual Basic 6.0, Mc Graw Hill, (1998), Madrid – España,

[Moreno03] Moreno García María, Técnicas Formales de especificación. Universidad de Salamanca, (2003), Salamanca - España.

[Norman05] Norman Jeremy, From Gutemberg to the Internet, A Sourcebook on the history of the Information Technology, Norman Publishing (2005), California – EEUU.

[OMG01] Object Management Group. *Unified Modeling Language Specification.* Version 1.4. September 2001 (http://www.omg.org/).

[Oracle2002] Oracle Corporation, Design Editor: View Menu.

[Pressman93] Pressman S. Roger. A Manager´s Guide to Software Engineering. Mc Graw Hill, (1993), New York – EEUU.

[Poseidon] GentleWare, http://www.gentleware.com/.

[Rational] *Rational Rose*. IBM Rational Software, http://www.rational.com/.

[Sanchez03] Sánchez de la Cruz María Mar, Del Casal Tenorio Mar Sol, Reingeniería de Bases de Datos Relacionales. Universidad de Castilla, La Mancha.

[Shaw03] Shaw Mary, The Comming-of-Age of Software Architecture Research. Institute for Software Research International, Carnegie Mellon University, Pittsburgh.

[Stair+00] Stair Ralph M, Reynolds George W., Principios de sistemas de información, Cengage Learning Editores, (2000), Madrid – España.

[Rob02] Rob Peter, Sistemas de Base de Datos, Cengage Learning Editores, (2002), 838 pág., Madrid – España.

[VISIO] Microsoft, http://www.microsoft.com/