UNIVERSIDAD AUSTRAL DE CHILE

FACULTAD DE CIENCIAS DE LA INGENIERÍA

ESCUELA DE INGENIERÍA CIVIL EN INFORMÁTICA

METODOLOGÍA DE DESARROLLO DE APLICACIONES SOBRE LA PLATAFORMA MICROSOFT .NET FRAMEWORK SDK

TESIS DE GRADO PARA OPTAR AL TÍTULO PROFESIONAL DE INGENIERO CIVIL EN INFORMÁTICA

PATROCINANTE:
CHRISTIAN HAGEDORN HITSCHFELD
INGENIERO CIVIL EN INFORMÁTICA.

CO-PATROCINANTE:
MAURICIO RUIZ-TAGLE M.
INGENIERO CIVIL EN INFORMÁTICA

JORGE ANDRÉS CORREA PULIDO

VALDIVIA – CHILE

2003

Santiago, 11 de Marzo de 2003

Sra. Miguelina Vega
Directora
Escuela de Ingeniería Civil en Informática
Universidad Austral de Chile
Valdivia
Presente

De mi consideración:

El proyecto de tesis del Sr. Jorge Correa "Metodología de Desarrollo de Aplicaciones sobre Plataforma Microsoft .NET Framework SDK" es un aporte importante para el desarrollo y conocimiento de nuevas metodologías en la Ingeniería del Software enfocada a técnicas modernas como UML, Cocomo II y herramientas de modelamiento CASE Rational XDE. Destaca también el desarrollo de un prototipo funcional de un sistema de Recaudación Electrónica de Minoristas utilizando tecnología WAP, en el cual se pudo aplicar el estudio teórico de la metodología. El Sr. Jorge Correa demostró una actitud altamente profesional, cumpliendo cabalmente los objetivos planteados durante el desarrollo del proyecto. Por lo anterior expuesto califico la tesis con nota 7,0 (siete).

Sin otro particular, le saluda cordialmente

Christian Hagedorn H.
Patrocinante

Universidad Austral de Chile
Instituto de Informática

Valdivia, 21 de marzo de 2003

Sra. Miguelina Vega R.
Directora Escuela Ingeniería Civil en Informática

De mi consideración:

Se me ha solicitado la revisión y evaluación como profesor informante, del Trabajo de Titulación del Sr. Jorge Correa Pulido, titulado -METODOLOGÍA DE DESARROLLO DE APLICACIONES SOBRE LA PLATAFORMA MICROSOFT NET FRAMEWORK SDK"

Estimo que el trabajo de titulación de la Sr. Jorge Correa cumple los objetivos propuestos, planteando sobre la plataforma elegida una metodología rigurosamente descrita, para el desarrollo de aplicaciones.

Adicionalmente, el desarrollo del prototipo del sistema Recaudación Electrónica a Minoristas tiene él mérito adicional de la aplicación de una metodología de desarrollo de software a un problema real del sector productivo.

Cabe destacar finalmente, el cuidado puesto en los aspectos formales de utilización del lenguaje técnico en la redacción del trabajo.

Por todo lo anteriormente expuesto, califico el trabajo de titulación del Sr. Jorge Correa con nota 7.0 (siete coma cero).

Sin otro particular, se despide atte.

UNIVERSIDAD AUSTRAL DE CHILE
INSTITUTO DE INFORMATICA

Mauricio Ruiz-Tagle Molina
Instituto de Informática

General Lagos 2086 · Campus Miraflores · Valdivia · Chile
Casilla 567 · Fono: 56 63 221427 · Fax: 56 63 293115 · instituto@inf.uach.cl · www.inf. uach.cl

Valdivia, 13 de Marzo de 2003

De : Martín Gonzalo Solar Monsalves

A : Directora Escuela Ingeniería Civil en Informática

Ref. : Informe Calificación Trabajo de Titulación

Nombre Trabajo de Titulación:

"METODOLOGÍA DE DESARROLLO DE APLICACIONES SOBRE LA PLATAFORMA MICROSOFT .NET FRAMEWORK SDK".

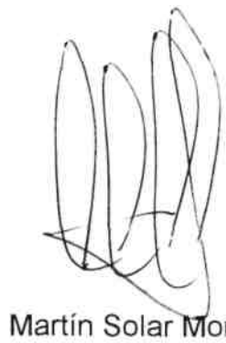
Nombre Alumno:

Jorge Andrés Correa Pulido

Evaluación:

Cumplimiento del objetivo propuesto	7.0
Satisfacción de alguna necesidad	7.0
Aplicación del método científico	7.0
Interpretación de los datos y obtención de conclusiones	7.0
Originalidad	7.0
Aplicación de criterios de análisis y diseño	7.0
Perspectivas del trabajo	7.0
Coherencia y rigurosidad lógica	7.0
Precisión del lenguaje técnico en la exposición, composición, redacción e ilustración	7.0
Nota Final	**7.0**

Sin otro particular, atte.



Martín Solar Monsalves

Agradecimientos:

Quisiera aprovechar esta oportunidad para agradecer a toda la gente que ha hecho posible la realización de esta obra. A Christian Hagedorn, Mauricio Ruiz-Tagle y a don Martín Solar, por su sólido apoyo; sinceramente, gracias. También me gustaría agradecer a todos mis profesores que han acompañado mi formación profesional y personal, en especial a Sra. Miguelina Vega.

También me gustaría agradecer a todos mis compañeros, que de una forma u otra han influido en el desarrollo de mis estudios universitarios, en especial a mi querido grupo de estudio, conformado por: Camilo Ruiz-Tagle y Cristian Porflitt, a ellos sinceramente, gracias.

En forma muy especial, agradezco a mis padres y abuelos por su constante cariño y apoyo durante todos estos años. A mi hermano Rodrigo, gracias por tu brillante ejemplo.

Agradezco a mi esposa, Mónica, sin ti nada de esto habría sido posible.

Finalmente le dedico esta tesis a mi hijo Tomás Benjamín, con todo el amor del mundo.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Título: **METODOLOGÍA DE DESARROLLO DE APLICACIONES SOBRE LA PLATAFORMA MICROSOFT .NET FRAMEWORK SDK.**

Resumen:

Las etapas del desarrollo y el ciclo de vida de un proyecto software por lo general tienden a mezclarse y confundirse, en especial cuando, los tiempos de desarrollo comprometen el buen término de un proyecto, elevando sus costos y perjudicando su calidad.

No obstante la aparición de nuevas tecnologías que ofrecen solución a estos problemas, no existe una clara metodología de desarrollo que pueda ser aplicada por los desarrolladores de software. Esto es evidente cuando se trata de integrar muchas tecnologías distintas que convergen hacia un determinado servicio.

Las nuevas tecnologías de desarrollo de software que tienden a integrar múltiples lenguajes de programación en una sola plataforma de desarrollo aparecen como una solución muy buena para ser cierta.

El desarrollo de software bajo la plataforma de Microsoft .Net Framework SDK, pretende disminuir los costos y tiempos de desarrollo de proyectos software, incrementando la reutilización de código y aumentando su legibilidad. Lo que se logra junto con la integración de potentes herramientas de modelamiento y desarrollo tipo CASE (computer-aided software engineering) tales como Rational XDE Professional v2002 y Microsoft VISIO 2000, que permiten modelar los sistemas mediante UML (Unified Modeling Language) y generar a partir de éste, código fuente y documentación que permiten avanzar los proyectos software considerablemente con sólo haber realizado el modelo.

El objetivo del presente documento es desarrollar una metodología de trabajo para enfrentar todas las etapas de desarrollo de un proyecto software, haciendo uso de estas herramientas.

SUMMARY

This thesis, is the result of a hole year of continuos research in the field of Software Engineering. Its main goal, is to offer a brand new methodology that is build with the best practices on the software development industry. The first part, it's avocated to understand what a methodology is made of, its main components and how they interact with each other. After defining the three elements that have to be included in a methodology: process, notation, and tools. I begin to study every one of them with a high level of detail.

This methodology was made to fit the needs of software development teams in business of almost any size. It helps to understand why is so important to make a preliminary design to get some preliminary estimations. And then, use these estimations to make an efficient development plan.

This methodology was named **EFT-SDM**, that stands for EFT Software Development Methodology. The process used in this methodology was named **EFT-UP (EFT Unified Process)** an is based entirelly on Rational Unified Process (RUP) and other RUP like processes available on the market. The tools used for designing, analyzing and implementing on **EFT-SDM** are high tech tools such as **Visual Studio .Net**, **Rational XDE Professional 2002**, and the **COCOMO II** model for software cost estimates.

At the end of this thesis, I built a pretty good example of how an application can be developed the traditional way, and how should it be done using EFT-SDM as the software development methodology.

Capítulo 1 : Introducción

1.1 Preámbulo

El objetivo final de esta tesis es formular una metodología de desarrollo de software que se ajuste a las necesidades de la empresa EFT Banca S.A., en todas las etapas del ciclo de vida de proyectos software. Dichas etapas incluyen el análisis, diseño, implementación, prueba y mantención de los productos desarrollados.

Una de las características fundamentales de la formulación de la metodología propuesta, es que será concebida para ser usada en conjunto con la nueva plataforma de desarrollo de software Microsoft .NET Framework SDK y su herramienta Visual Studio .NET Architect. Esta plataforma de desarrollo se integrará con herramientas CASE, en especial la herramienta Rational XDE Professional v2002, la cual provee un entorno de modelamiento de sistemas basado en UML[1].

Se incorporarán elementos de estimación de costos de proyectos software, haciendo una descripción del método COCOMO II [2], para su posterior incorporación a la metodología propuesta.

En el **capítulo 1** se expondrá la plataforma de Microsoft .NET Framework SDK, describiendo su funcionamiento en forma detallada, los lenguajes de programación apoyados por ésta, centrando la atención en C#. El **capítulo 2**, expone los aspectos fundamentales de una metodología, así como una breve introducción al UML. En el **capítulo 3**, se expondrá el proceso de desarrollo de software Rational Unified Process (RUP). El **capítulo 4**, corresponde a un breve análisis de la herramienta de desarrollo Visual Studio .NET. El **capítulo 5** consiste en un breve análisis de la herramienta Rational XDE Professional. En el **capítulo 6**, se analizará el modelo COCOMO II, y se obtendrá el método de estimación de esfuerzos, tiempos y costos para un proyecto software, de modo de rescatar los aspectos significativos que puedan incorporarse a la metodología de desarrollo de software.

---

[1] **UML :** lenguaje de modelamiento unificado, del inglés Unified Modeling Language
[2] **COCOMO II:** Constructive Cost Model, desarrollado en USC por Dr. Barry Boehm.

El **capítulo 7**, muestra dos enfoques con los que ha sido construido el sistema prototipo de Recaudación Electrónica a Minoristas. El primero de una forma artesanal, mientras que el segundo empleará la metodología propuesta. El **capítulo 8**, expondrá el diseño de la metodología de desarrollo de aplicaciones sobre la plataforma Microsoft .NET Framework SDK.

El **capítulo 9**, discutirá las conclusiones y mejoras tanto a la metodología como al prototipo implementado con ésta.

Ciertamente, la revisión bibliográfica acerca de administración de proyectos (Project Managment), Ingeniería de Software y Sistemas de Gestión son las áreas que marcan la evolución del presente trabajo.

Esta tesis pretende contribuir como un marco de apoyo a la gestión y desarrollo de proyectos software en la empresa EFT Banca S.A.. Específicamente, aportará al trabajo realizado por los jefes de proyecto, brindándoles una base de conocimiento compartido entre ellos y sus equipos de desarrollo, aumentando así los canales de comunicación y difusión de los modelos a implementar.

1.1.1 ¿Qué es .NET Framework SDK?

.NET Framework SDK es una nueva plataforma que simplifica el desarrollo de aplicaciones en ambientes altamente distribuidos de Internet. .NET Framework ha sido diseñado para satisfacer los siguientes objetivos:

- Proveer de un ambiente consistente de programación orientada a objeto, cuando el código objeto se almacena y ejecuta en forma local, cuando se ejecuta en forma local pero está almacenado en internet de una forma distribuida, o cuando es ejecutado remotamente.
- Proveer de un ambiente de codificación y ejecución que minimice el desarrollo de software y conflictos de versión.

- Proveer de un ambiente de ejecución de código que garantice una ejecución segura, incluyendo el código creado por una tercera empresa participante, que sea desconocida o poco confiable.
- Proveer de un entorno de ejecución de código que elimine los problemas de los entornos de script o interpretados.
- Hacer que la experiencia del desarrollador sea consistente a través de la amplia gamma de aplicaciones, tales como aplicaciones basadas en ventanas y aplicaciones basadas en WEB
- Construir todos los estándares de comunicación en la industria para asegurar que el código escrito en la plataforma .NET Framework se pueda integrar con cualquier otro lenguaje.

Actualmente lo que la mayoría de los usuarios conocen por Internet es, la WWW[3] una red de servidores que almacenan documentos. Los clientes de esta red son aplicaciones relativamente simples, los navegadores como Internet Explorer y Netscape Navegator, que se limitan a solicitar esos documentos y mostrarlos en pantalla.

Uno de los objetivos de la plataforma Microsoft .NET, es transformar esta red de documentos, basada en grandes servidores independientes, en una red en que cualquier computador pueda ofrecer sus servicios a los demás.

El mayor reto es hacer llegar esos servicios a distintos tipos de clientes, con diferentes sistemas operativos, procesadores, lenguajes, etc.

Para conseguir este objetivo, Microsoft ha decidido basarse exclusivamente en estándares, principalmente HTML (HyperText Markup Language), HTTP (HiperText Transfer Protocol) y XML (Extensible Markup Language). La información es procesada, por el servidor o por el propio cliente en forma de documento XML cuando el proceso tiene lugar en el servidor, lo que obtiene el cliente son documentos HTML básicos, pero generados dinámicamente según las necesidades. La comunicación entre los consumidores y suministradores de servicios se efectúa

mediante SOAP (Simple Object Access Protocol), un protocolo RPC (Remote Procedure Call) basado en XML.

Utilizando este conjunto de protocolos y formatos, tanto los consumidores como los proveedores de servicios web pueden, ejecutarse desde cualquier dispositivo y sistema.

La plataforma .NET está formada por un entorno común de ejecución para los programas, independientemente del lenguaje con que se hayan desarrollado.

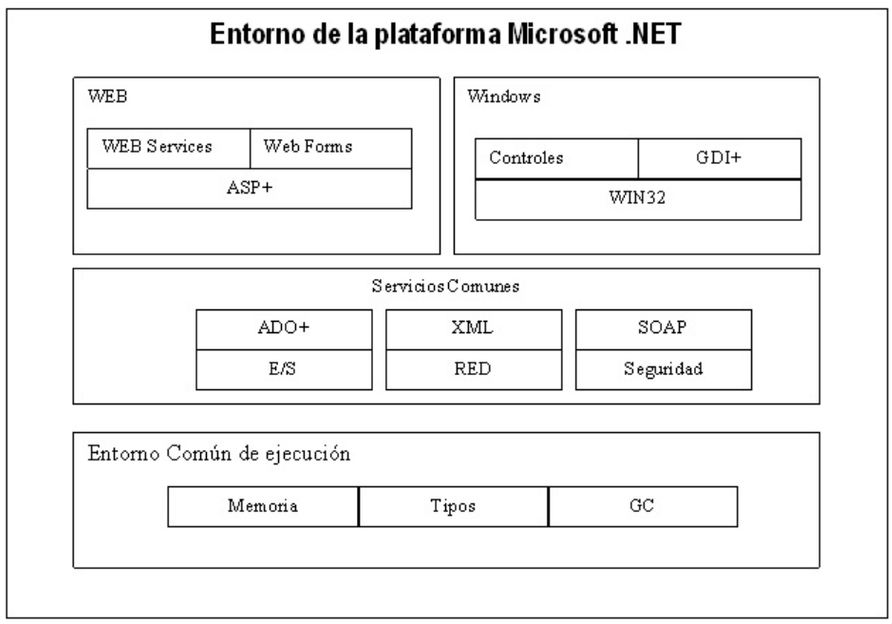


Figura 1.1: Entorno de la plataforma Microsoft .NET

Este entorno común de ejecución se ocupa de tareas tradicionalmente asociadas a los compiladores, como la gestión de memoria, definición de un conjunto estándar de tipos de datos o efectuar la recogida de basura (Garbage Collection). De esta forma, Visual Basic, C#, COBOL, Eifel o cualquier lenguaje que genere código para la plataforma .NET cuenta ya con los mismos tipos de datos y un servicio de recogida de basura común. Esto hace que la generación de código por parte de los compiladores de esos lenguajes sea más simple y el tamaño del código generado sea menor.

[3] **WWW:** World Wide Web

1.1.2 Código Intermedio

Para aprovechar todas las ventajas de la plataforma .NET, es necesario contar con nuevos compiladores. Estos deben ajustarse a una especificación común denominada CLS (Common Language Specification), en la que se indica cuales han de ser las características y formato del código a generar.

Cualquier herramienta que tenga como objetivo la plataforma .NET, debe generar código intermedio, *no código nativo*, además de toda la meta-información adicional que el compilador JIT (Just In Time) necesitará para encontrar todos los elementos de los que dependa.

Dicho código intermedio estará escrito en el denominado lenguaje intermedio (IL: Itermediate Language), que es un leguaje de bajo nivel desarrollado por Microsoft con la ayuda de expertos en el campo del diseño de lenguajes y compiladores.

La figura 1.2 muestra los distintos pasos del proceso de desarrollo.

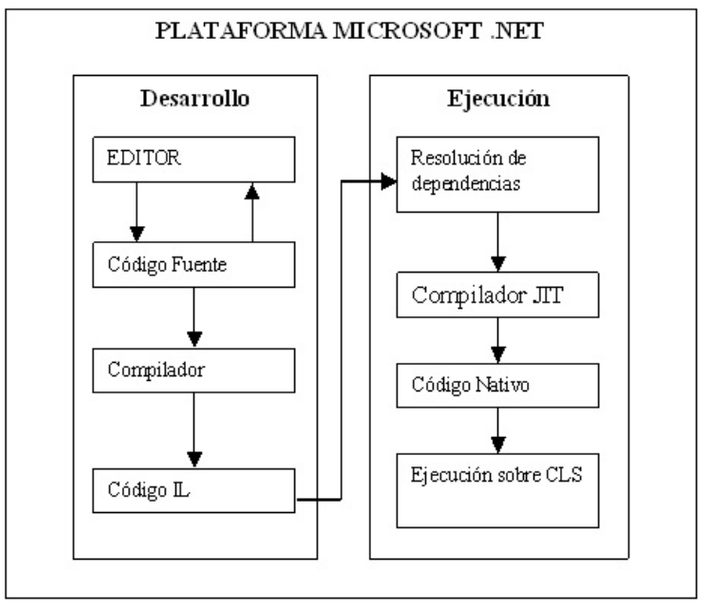


Figura 1.2: El proceso de generación de aplicaciones de la plataforma .NET.

El código fuente es procesado por un compilador, del lenguaje que corresponda, generando un archivo EXE o DLL que no contiene código nativo, sino intermedio. Podríamos comparar este código con el p-code generado tradicionalmente por Visual Basic o bien el bytecode generado por Java. La diferencia, es que no sería necesario trabajar con Visual Basic o con Java, sino que es posible utilizar cualquier lenguaje para el cual exista un compilador .NET.

Al ejecutar un programa sobre la plataforma .NET. Lo primero que se hace es resolver las dependencias que pudiesen existir, tras lo cual se procede a realizar una compilación al vuelo. El código, ya nativo se ejecuta en el contexto del entorno común de ejecución. Haciendo uso del CTS (Common Type System) o sistema común de tipos, el recogedor de basura y la administración de memoria común. Por tanto, la existencia de la plataforma .NET iguala las posibilidades de todos los lenguajes. Independientemente de que usemos C++, Visual Basic, COBOL, el código obtenido siempre será IL, y éste se compilará con la técnica JIT y se ejecutará en el CLR[4]. No existirán por lo tanto diferencias notables de rendimiento entre los lenguajes.

.NET Framework tiene dos componentes principales: la llamada CLR (Common Language Runtime) y la librería de clases de .NET Framework. La CLR corresponde a los cimientos de la plataforma .NET Framework. Se podría considerar a CLR como un agente que administra el código en tiempo de ejecución, provee de los servicios básicos de administración de memoria, administración de hilos y administración remota. Al mismo tiempo refuerza la seguridad de tipos y otras formas de exactitud de código que privilegian la seguridad y robustez. De hecho, el concepto de administración de código es un principio fundamental de CLR. El otro componente de .NET Framework , la librería de clases, es una colección de tipos reusables que se pueden usar para desarrollar aplicaciones que van desde las tradicionales aplicaciones de línea de comando o las aplicaciones con interfaces gráficas de usuario (GUI) a las aplicaciones basadas en las últimas innovaciones provistas por ASP.NET, tales como las Web Forms y los XML Web Services.

1.1.3 Interoperabilidad entre lenguajes

Gracias a la existencia del lenguaje intermedio o IL, la plataforma .NET facilita un nivel de interoperatibilidad entre lenguajes, desconocido hasta ahora, llegando a ser posible una herencia inter-lenguaje. Esto significa que es posible escribir una clase en Visual Basic y, posteriormente

[4] **CLR:** entorno común de ejecución, del inglés : Common Language Runtime

usarla de base para derivar otra en C#, o viceversa. No es necesario utilizar siempre un mismo lenguaje, ni recurrir a modelos de componentes como COM.

1.1.4 Servicios .NET

.NET es un conjunto de protocolos, técnicas, servicios y software difícil de definir de manera rápida y simple.

Por una parte están los protocolos, técnicas y especificaciones, como SOAP para facilitar las llamadas remotas, a través del Web, el uso de XML para representar los conjuntos de datos o la CLS.

Los servicios ocupan un lugar importante en .NET y están accesibles desde la CLR para cualquier aplicación compilada en IL. Estos servicios permitirán a las aplicaciones efectuar operaciones de entrada/salida, trabajar con redes, etc.

También forman parte de .NET las nuevas versiones de ASP (Active Server Page) y ADO (ActiveX Data Objects) llamadas ASP+ y ADO+. Una de las características más interesantes de ASP+ es que los scripts que generan las respuestas son compilados a código IL. Esto significa que no es necesario utilizar código VBScript ni Jscript, siendo posible utilizar por ejemplo simplemente Visual Basic o C#.

En cuanto a ADO+, lo más significativo es el uso de XML para representar los conjuntos de datos y de SOAP para intercambiar dichos conjuntos.

1.1.5 Proceso de estandarización

Microsoft ha sido acusada en reiteradas ocasiones de no respetar los estándares, establecidos o de facto, y de modificarlos según sus intereses particulares.

En su plataforma .NET, Microsoft no sólo se apoya en estándares como XML o HTTP, sino que propone otros, totalmente abiertos, que como SOAP, harán posible la existencia de redes de servicios en Internet comunicándose sobre HTTP utilizando XML. No importa el sistema operativo ni el hardware, SOAP no es sólo para windows.

Lo más interesante es que Microsoft a remitido al ECMA[5], para iniciar su proceso de estandarización, la especificación de su CLI (Common Language Infraestructure) y la del nuevo lenguaje C#. Esto significa que cualquier otra empresa tendrá acceso a dicha información.

La estandarización de CLI es fundamental, ya que sobre ella se apoyan el sistema común de tipos y el entorno común de ejecución. Cuando CLI sea un estándar, cualquiera podrá crear un compilador que se ajuste a esta especificación y, por tanto, pueda utilizarse en la plataforma .NET

1.2 ¿Qué es UML?

El UML (Lenguaje de modelamiento unificado) es el sucesor de una ola de métodos de análisis y diseño orientado a objeto que aparecieron entre fines de los 80 y comienzos de los 90. Unifica directamente los métodos de Booch, Rumbaugh (OMT), y Jacobson, pero su alcance es mucho más amplio. UML pasó un proceso de estandarización en el OMG (Object Management Group) y hoy en día es un estándar OMG.

El UML es un lenguaje de modelamiento, *no es un método*. La mayoría de los métodos consisten, por lo menos en principio, de un lenguaje de modelamiento y un proceso. El lenguaje de modelamiento es la notación (principalmente gráfica) que usan los métodos para expresar los diseños. El proceso, en cambio, son los pasos recomendados a seguir en la realización de un diseño.

El lenguaje de modelamiento es la parte más importante del método. Ciertamente es la clave de la comunicación. Si uno quiere discutir un diseño con alguien, es el lenguaje de modelamiento el cual debe ser conocido por ambos, no así el proceso que se usó para llegar a ese diseño.

Booch, Rumbaugh y Jacobson, también desarrollaron un proceso unificado, el cual llamaron *Rational Unified Process (RUP).* Sin embargo, no es necesario hacer uso del RUP para poder usar UML.

[5] **ECMA:** Instituto Europeo fundado en 1961 dedicado a la creación de estándares

El UML, en su estado actual, define una notación y un meta-modelo. La notación son las gráficas que se ven en los modelos; es la sintaxis del lenguaje de modelamiento. Por ejemplo, la notación de diagramas de clases define como se representan los ítems y conceptos tales como clase, asociación y multiplicidad.

Una forma de mejorar el rigor de los métodos sin sacrificar su utilidad es definir un meta-modelo, que es un diagrama, usualmente un diagrama de clases, el cual define la notación.

La razón fundamental para usar UML es la comunicación. Permite comunicar ciertos conceptos con mayor claridad. El lenguaje natural es muy impreciso y resulta confuso cuando se tratan de comunicar conceptos más complejos. Por otra parte, el código es preciso, pero muy detallado. Por lo que un buen uso del UML sería en situaciones en las que se desea mantener cierto grado de precisión, pero no se quiere perder tiempo en los detalles. Esto no significa que se eviten los detalles, muy por el contrario, se usa UML para destacar los detalles más importantes.

Uno de los más grandes desafíos en el desarrollo es construir el sistema correcto, uno que satisfaga las necesidades del usuario a un costo razonable.

Lograr una buena comunicación, junto con un buen entendimiento del mundo del usuario, es la clave para desarrollar un buen software.

Una de las técnicas para conseguir esto es el **diagrama de casos de uso**, el cual representa una fotografía instantánea de un aspecto del sistema. La suma de todos los diagramas de casos de uso representa una fotografía externa del sistema. Una buena colección de casos de uso es central para el buen entendimiento de lo que el usuario quiere. Los casos de uso también representan un buen vehículo para la planificación de proyectos.

Como *UML es independiente del proceso*, se puede elegir cualquier proceso que sea apropiado dependiendo del tipo de proyecto. Cualquiera sea el proceso que se use, siempre se podrá usar UML para registrar las decisiones de análisis y diseño.

UML define nueve tipos de diagramas:

1. **Diagramas de clases:**

   Son la columna vertebral de casi todos los métodos orientados a objeto, incluyendo UML. Describen la estructura estática de un sistema.

   Diagramas de empaquetamiento: Son un subconjunto de los diagramas de clases, pero los desarrolladores algunas veces los tratan como un tipo de diagrama separado. Organiza a los elementos de un sistema en grupos relacionados para minimizar las dependencias entre paquetes.

2. **Diagramas de Objetos:**

   Describe la estructura estática de un sistema en un tiempo en particular. Pueden ser usados para probar la exactitud de los diagramas de clases.

3. **Diagramas de casos de uso:**

   Modelan la funcionalidad del sistema haciendo uso de actores y casos de uso.

4. **Diagramas de secuencia :**

   Describe interacciones entre clases en términos de intercambio de mensajes en el tiempo.

5. **Diagramas de colaboración:**

   Representan interacciones entre objetos como una serie de mensajes secuenciados. Describen tanto la estructura estática como el comportamiento dinámico del sistema.

6. **Diagramas de estado:**

   Describe el comportamiento dinámico de un sistema en respuesta a estímulos externos. Son especialmente útiles en el modelamiento de objetos reactivos cuyos estados son gatillados por eventos específicos.

7. **Diagramas de actividad:**

   Ilustran la naturaleza dinámica de un sistema, modelando el flujo de control de actividad en actividad. Una actividad representa una operación en alguna clase del sistema que resulta en un cambio en el estado de éste. Son usados generalmente para modelar flujos de trabajo (Workflows) o modelos de negocio y operaciones internas.

8. **Diagramas de componente:**

   Describen la organización de componentes físicos del software, incluyendo código fuente, código binario en ejecución y ejecutables.

9. **Diagramas de desarrollo:**

   Muestra los recursos físicos en un sistema, incluyendo nodos, componentes y conecciones.

1.3 SEI SW-CMM (The Capability Maturity Model for Software)

SW-CMM es un modelo para juzgar la madurez de los procesos software para una organización y para identificar las prácticas clave que se requieren para incrementar la madurez de estos procesos.

El CMM define 5 niveles de madurez, los cuales son :

1. **Inicial:**

   Los procesos software se caracterizan como caóticos. Existen pocos procesos definidos, y el éxito depende de esfuerzos personales y heróicos.

2. **Repetibles:**

   Procesos básicos de administración de proyectos, se establecen para seguir los costos, el horario y la funcionalidad. Se posee la disciplina de procesos necesaria para repetir éxitos anteriores en proyectos con aplicaciones similares.

3. **Definidos:**

   El proceso software, tanto para las actividades de administración e ingeniería, está documentado, estandarizado e integrado dentro de un proceso software estándar para la

organización. Todos los proyectos usan una versión adaptada y aprobada del proceso software estándar de la organización para desarrollo y mantención de software.

4. **Administrado:**

   Se recopilan mediciones detalladas de los procesos software y de la calidad del producto. Tanto el proceso software como el producto son comprendidos y controlados cuantitativamente.

5. **Optimizado:**

   Se posibilita un continuo mejoramiento del proceso gracias a una retroalimentación cuantitativa desde el proceso.

La predictibilidad, efectividad y control de los procesos software de una organización son mejorados gradualmente cuando la organización posee una calificación próxima al nivel 5 CMM.

Excepto por el nivel 1, cada nivel de madurez es descompuesto en varias áreas de procesos clave, éstas indican las áreas en las que una organización debe enfocarse para mejorar su proceso software.

Las áreas de procesos clave del nivel 2, se centran en *establecer un control básico de administración de proyecto*, relacionado con el proyecto software. Las áreas son: Administración de Requisitos, Planificación del proyecto software, Seguimiento y vigilancia del proyecto software, Administración de subcontratos del software, Aseguramiento de la calidad y administración de la configuración del software.

Las áreas de procesos clave en el nivel 3, apuntan a temas de proyectos y organizacionales, mientras la organización establece una infraestructura que institucionaliza efectivamente la ingeniería de software y administra los procesos a través de todos los proyectos. Las áreas son las siguientes: Focalización de los procesos de la organización, definición de los procesos de la organización, programa de entrenamiento, administración de software integrado, ingeniería de productos software, coordinación intergrupal y revisiones.

Las áreas de procesos clave del nivel 4, se centran en establecer un entendimiento cuantitativo tanto de los procesos software como de los productos del trabajo software que están

siendo construidos. Estas áreas son administración de procesos cuantitativos y administración de calidad del software.

Las áreas de procesos clave del nivel 5, cubren los asuntos que deben cumplir tanto la organización como los proyectos para implementar un continuo y medible mejoramiento del proceso software. Estas áreas son : Prevención contra defectos, administración del cambio de tecnología y administración del cambio de procesos.

1.4 COCOMO II

COCOMO II es un modelo que permite estimar el costo, esfuerzo y tiempo cuando se planifica una nueva actividad de desarrollo de software. Está asociado a los ciclos de vida modernos. El modelo original COCOMO ha tenido mucho éxito, pero no puede emplearse con las prácticas de desarrollo de software más recientes tan bien como con las tradicionales.

Los objetivos a la hora de la creación del modelo COCOMO II fueron:

- Desarrollar un modelo de estimación de tiempo y costo del software de acuerdo con los ciclos de vida utilizados en los 90 y en la primera década del 2000.
- Desarrollar bases de datos con costos de software y herramientas de soporte para el continuo mejoramiento del modelo.
- Proporcionar un marco analítico cuantitativo y un conjunto de herramientas y técnicas para la evaluación de los efectos de la mejora tecnológica de software en costo y tiempo del ciclo de vida software.

Para apoyar a los distintos sectores del mercado de software, COCOMO II proporciona una familia de modelos de estimación de costo de software cada vez más detallado y tiene en cuenta las necesidades de cada sector y el tipo de información disponible para sostener la estimación de costo del software. Esta familia de modelos está compuesta por tres submodelos, cada uno de los cuales ofrece mayor fidelidad a medida que uno avanza en la planificación del proyecto y en el proceso de diseño, estos tres submodelos se denominan:

- **Modelo de composición de aplicaciones**

  Indicado para proyectos construidos con herramientas modernas de construcción de interfaces gráficas de usuario.

- **Modelo de diseño anticipado**

  Este modelo puede utilizarse para obtener estimaciones aproximadas del costo de un proyecto antes de que esté determinada por completo su arquitectura. Utiliza un pequeño conjunto de drivers de costo [6] nuevos y nuevas ecuaciones de estimación. Está basado en puntos de función sin ajustar o KSLOC[7] .

- **Modelo Post-arquitectura**

  Este es el modelo COCOMO II más detallado. Se utiliza una vez que se ha desarrollado por completo la arquitectura del proyecto. Posee nuevos drivers de costo, nuevas reglas para el recuento de líneas y nuevas ecuaciones

La primera versión de COCOMO II fue presentada en 1997. USC[8] COCOMO II.1997 se calibró con 83 puntos de datos (proyectos de desarrollo de software históricos), utilizando un enfoque de media ponderada del 10% para combinar datos empíricos con la opinión del experto y calibrar los parámetros del modelo. USC COCOMO II.1998.0 Beta, se creó en octubre de 1998. Esta versión se calibró con 161 puntos de datos y utilizando por primera vez un enfoque bayesiano para realizar la calibración del modelo. USC COCOMO II.1999.0 fue publicada a mediados de 1999 y la versión 2000 ya fue publicada. Cada versión de USC COCOMO II se ha hecho más amigable con el usuario, sin embargo las calibraciones del modelo de los años 1998,1999 y 2000 son las mismas. Es decir, no se han agregado nuevos puntos de datos a la base de datos usada para calibrar las versiones de las herramientas del año 1999 y el año 2000, además de aquellas que aparecieron en la calibración de la base de datos del año 1998.

En las tres implementaciones aparecen los mismos 161 puntos de datos a los que se hacía referencia anteriormente.

---

[6] **Drivers de Costo:** Conductores de costo, variables que influyen en el modelo COCOMO II.
[7] **KSLOC:** Miles de líneas de código fuente.
[8] **USC:** Universidad del Sur de California.

Según los desarrolladores de COCOMO II, la experiencia ha demostrado que si una organización calibra la constante multiplicativa para sus propios datos empíricos, la precisión del modelo puede aumentar significativamente por encima de los resultados de calibración genérica obtenidos con las versiones mencionadas anteriormente.

1.5 NIVEL ACTUAL

Con respecto a las metodologías de desarrollo de software existentes, actualmente la industria del software cuenta con muy buenas herramientas que engloban dichas metodologías, muchas de ellas son propietarias y no están disponibles en forma gratuita, es el caso por ejemplo de Foundation de Anderson Consulting, SUMMIT D de PriceWaterHouseCoopers, Microsoft Solution Framework , SEI Development Capability Maturity Model y Rational Unified Process, entre otros.

De las 38 metodologías encontradas y revisadas se ha optado por describir el Rational Unified Process (RUP), que más que una metodología corresponde a un proceso.

La decisión de considerar sólo esta metodología de desarrollo de software se sustenta en los siguientes puntos:

- Microsoft y Rational sostienen actualmente relaciones de transferencia tecnológica para mantener integrados sus productos.
- Tomando en cuenta que la mayoría de las metodologías sostienen una misma estructura en sus ciclos de vida de proyectos software: Análisis de requisitos, Diseño del Modelo, Implementación del modelo, Prueba del sistema, mantención del sistema.
- Rational Unified Process, se ha escogido por su proximidad al modelamiento en UML.

1.6 MOTIVACION

Las motivaciones principales para realizar este proyecto de tesis son:

- Desarrollar Metodología que aplique técnicas de Ingeniería de Software, en el análisis, diseño, implementación, prueba y mantenimiento de software desarrollado en EFT Banca S.A.
- Construcción de prototipo funcional.

1.7 IMPACTOS

El impacto real de la metodología para el desarrollo de proyectos software debería notarse en la evolución diaria de proyectos, así como en su mantención y escalabilidad.

La información estadística de métricas del software debería ajustar el modelo de estimación de tiempos, costos y esfuerzos para el desarrollo de un determinado producto. La documentación sobre el proyecto mejoraría notablemente, aumentando su legibilidad y portabilidad.

1.8 Objetivos Generales.

Desarrollar una Metodología para el desarrollo de Proyectos que permita disminuir los tiempos de desarrollo, mejorar la calidad de los productos de software e incrementar la modularización, reusabilidad y estandarización del software.

1.9 Objetivos Específicos.

Los objetivos específicos de este proyecto de tesis son:

1. Integración de la herramientas de modelamiento CASE Rational XDE Professional con la plataforma de desarrollo de Software Microsoft .NET Framework SDK y Visual Studio .NET.

2. Modelamiento estándar de los sistemas mediante UML.
3. Integrar herramientas de conceptualización visual de la arquitectura y funcionalidad de las aplicaciones.
4. Captura y comunicación de requerimientos de negocios con herramientas de diseño conceptual, lógico y físico de bases de datos.
5. Metodología para la estimación de desarrollo de proyectos apoyado en modelo COCOMO II (Constructive Cost Model).
6. Definición de Métricas en el desarrollo del software.
7. Definición de documentación estándar.

Capítulo 2 : Metodología

2.1 Preámbulo

Las aplicaciones de las grandes empresas –las que ejecutan aplicaciones de negocio importantes, que las mantienen funcionando– deben ser mucho más que sólo un montón de código. Deben estar estructuradas en una forma que permitan la escalabilidad, seguridad y ejecución robusta bajo condición de stress, y su estructura –frecuentemente conocida como su arquitectura– debe estar claramente definida, lo suficiente como para que los programadores de mantenimiento puedan encontrar y reparar rápidamente los errores que aparecen mucho después, cuando los autores originales de la aplicación ya se han movido a otro proyecto. Por supuesto una arquitectura bien diseñada, beneficia a cualquier programa y no sólo a los más grandes, como se ha dicho. Se mencionó a las aplicaciones más grandes primero, ya que la estructura es una forma de tratar con la complejidad, por lo que los beneficios de la estructura (y del modelamiento y el diseño, como se ha demostrado) aumentan cuando las aplicaciones crecen en tamaño. Otro de los beneficios de la estructura es que permite la reutilización de código: En tiempo de diseño es más fácil estructurar una aplicación como una colección de módulos auto contenidos o componentes.

Eventualmente, las empresas construyen librerías de modelos de componentes, cada una de las cuales representa un almacén de implementación en una librería de módulos de código. Cuando otra aplicación necesita la misma funcionalidad, el diseñador puede rápidamente importar los módulos desde la librería. En tiempo de codificación, el desarrollador puede igualmente importar el modulo de código dentro del ejecutable.

El Modelamiento es el diseño de aplicaciones de software antes de codificar. El modelamiento es una parte esencial de los grandes proyectos software, y útil para proyectos medianos y pequeños. Un modelo juega el rol análogo en desarrollo de software a los planos en la construcción de un edificio.

Usando un modelo, aquellos responsables por el éxito del desarrollo de proyectos software pueden asegurarse que la funcionalidad del negocio está correcta y completa, las necesidades de los usuarios finales han sido satisfechas, y que el diseño del programa apoya los requerimientos de escalabilidad, robustez, seguridad, extensibilidad, y otras características, antes de la implementación y codificación, que hacen que los cambios sean difíciles y costosos de realizar.

Las estadísticas muestran que los grandes proyectos software tienen una alta probabilidad de fallar, de hecho, es más probable que una gran aplicación falle en satisfacer todos los requerimientos de tiempo y presupuesto, a que tenga éxito. Si se está administrando uno de estos proyectos, se necesita hacer todo lo posible para aumentar las probabilidades de éxito, y el modelamiento es la única forma de visualizar el diseño y verificarlo contra los requerimientos antes que los programadores comiencen a codificar.

2.1.1 ¿Qué es una metodología?

"¿Es una metodología una notación, un proceso, ninguna de las anteriores, o ambas?

Muchos profesionales se pierden en la creencia de que ellos están usando una metodología cuando lo único que están usando es su notación. Por otra parte, una notación es un componente importante de una metodología y no puede ser seleccionada en forma arbitraria. Debe ser diseñada para un uso óptimo. Una metodología de ciclo de vida apropiada debe contener todos los componentes siguientes: un proceso de ciclo de vida completo para negocios y asuntos tecnológicos; un conjunto completo de conceptos y modelos los cuales son auto-consistentes; un conjunto de reglas y directivas; una completa descripción de todas las versiones; una notación funcional; que idealmente sea apoyada por herramientas de dibujo de una tercera empresa; un conjunto de técnicas suficientemente probadas; un conjunto de métricas adecuadas; estándares y estrategias probadas; identificación de roles organizacionales, por ejemplo: Analista de negocios, programador; directivas para administración de proyectos y aseguramiento de la calidad; consejos en la administración de librería y reusabilidad. " (B. Henderson-Sellers)

2.1.2 El lenguaje de modelamiento unificado de OMG (UML™)

El lenguaje de modelamiento unificado (UML), nos ayuda a especificar, visualizar y documentar modelos de software, incluyendo su estructura y diseño. Se puede usar UML para el modelamiento de negocios y también el modelamiento de otros sistemas que no sean de software. Se puede usar cualquiera de las herramientas basadas en UML que existen en el mercado, las cuales se incluyen en el **Anexo 1**.

Es posible analizar los futuros requerimientos de la aplicación y diseñar una solución que los cumpla, representándolos mediante los 12 tipos de diagramas estándar de UML.

Se puede modelar cualquier tipo de aplicación, que se ejecute en cualquier tipo o combinación de hardware, sistema operativo, lenguaje de programación y red. Su flexibilidad permite modelar aplicaciones distribuidas que pueden usar cualquier middleware[9] disponible en el mercado. Construir sobre el meta modelo MOF[10], el cual define clases y operaciones como conceptos fundamentales, se ajusta en forma natural a los lenguajes orientados a objeto y entornos tales como C++, Java y, recientemente, C#, pero puede usarse para modelar aplicaciones no orientadas a objeto, por ejemplo en, Fortran, VB, o COBOL. Los perfiles de UML, los cuales son un subconjunto adaptado del UML para un propósito específico, ayudan a modelar sistemas transaccionales, sistemas de tiempo-real, y sistemas de tolerancia a fallas de una forma natural.

También es posible hacer otras cosas útiles con UML: Por ejemplo algunas herramientas analizan código fuente existente y le aplican ingeniería inversa para transformarlo en un conjunto de diagramas UML.

---

[9] **Middleware:** Es una capa de software entre la red y las aplicaciones. Provee servicios tales como identificación, autentificación, autorización, directorios y seguridad.

[10] **MOF:** Meta-Object Facility: Define un meta modelo común para todas las especificaciones de modelamiento OMG, define un repositorio estándar para los meta modelos.

Aunque UML se centra en el diseño en vez de la ejecución, algunas herramientas disponibles en el mercado ejecutan modelos UML, típicamente en una de dos formas: Algunas herramientas ejecutan los modelos interpretativamente de una forma que nos permite confirmar que realmente el modelo hace lo que queremos que haga, pero sin la escalabilidad y velocidad que se necesita en la aplicación desarrollada. Otras herramientas, diseñadas para trabajar sólo con dominios de aplicación registrados tales como telecomunicaciones y finanzas, generan código en lenguaje de programación desde el UML, produciendo una aplicación libre de errores, aplicaciones desplegables que se ejecutan rápidamente si el generador de código incorpora las mejores prácticas de patrones escalables, por ejemplo: operaciones de bases de datos transaccionales u otra tarea común. La última entrada en esta categoría es un número de herramientas en el mercado que generan prueba y verificación a partir de los modelos UML.

2.1.3 Modelos versus Metodologías

El proceso de recolectar y analizar los requerimientos de una aplicación e incorporarlos dentro del diseño de un programa, es complejo, y la industria actualmente apoya muchas metodologías que definen procedimientos formales especificando como hacer las cosas. Una característica de UML –de hecho, la que posibilita el amplio respaldo de la industria del que goza este lenguaje– es que es independiente de la metodología. No importando la metodología que use para el análisis y el diseño, siempre se puede usar UML para expresar los resultados, y utilizando XMI (Intercambio de XML meta data, otro estándar OMG), se puede transferir un modelo UML desde una herramienta a un repositorio, a otra herramienta para refinamiento, o al próximo paso en el proceso de desarrollo escogido. Estos son los beneficios de la estandarización.

2.1.4 ¿Qué se puede modelar con UML?

UML define 12 tipos de diagramas, divididos en tres categorías: cuatro tipos de diagramas representan estructuras de aplicaciones estáticas; cinco tipos de diagramas representan diferentes aspectos del comportamiento dinámico; y tres representan formas en que se pueden organizar y administrar los módulos de las aplicaciones.

Diagramas Estructurales

Incluye el diagrama de clases, el diagrama de objeto, el diagrama de componente, y el diagrama de despliegue.

Diagramas de comportamiento

Incluye el diagrama de casos de uso (utilizado por algunas metodologías durante la recolección de requerimientos); diagrama de secuencias, diagrama de actividad, diagrama de colaboración, y diagrama de estados.

Diagramas de administración de modelos

Incluye paquetes, subsistemas, y modelos.

2.1.5 Primer proyecto de desarrollo basado en UML

¿Qué se necesita hacer para comenzar el primer proyecto de desarrollo basado en UML?

Probablemente tres cosas, pero no necesariamente en este orden

1) Seleccionar una metodología

Una metodología define formalmente el proceso que se va a utilizar para recolectar los requerimientos, analizarlos, y diseñar una aplicación que cumpla con ellos en cada uno de sus puntos. Hay muchas metodologías, cada una difiere de alguna manera o maneras de las otras. Hay muchas razones de por qué una metodología puede ser mejor que otra para un proyecto en particular: Por ejemplo, algunas se acomodan mejor para aplicaciones de grandes empresas, mientras que otras han sido construidas para diseñar pequeños sistemas encapsulados o de seguridad crítica. Por otra parte, algunos métodos apoyan mejor a un gran número de arquitectos y diseñadores trabajando en un mismo proyecto, mientras que otros funcionan mejor cuando son usados por una sola persona o un pequeño grupo.

2) Seleccionar una herramienta de desarrollo UML

Debido a que la mayoría de las herramientas basadas en UML implementan una metodología en particular, aunque no todas, en algunos casos no será práctico elegir una herramienta y luego tratar de usarla con una metodología para la cual no fue construida.

Uno podría encontrar una herramienta que se ajuste a la aplicación que se está construyendo o a la organización, de tal forma, que se podría esperar cambiar de metodología para poder usarla. Sin embargo lo recomendable en tal situación es escoger una metodología primero.

3) Obtener entrenamiento en UML

Los jefes de proyecto y equipos de desarrollo necesitarán entrenamiento en UML. Es mejor obtener un entrenamiento que enseñe como usar la herramienta de modelamiento junto con la metodología escogida.

2.2 Arquitectura de Software y UML

Resumen de los puntos clave de la presentación de Grady Booch en la conferencia mundial de UML en New York, Marzo 10, 1999.

"La casa de una mascota, un perrito por ejemplo, puede ser construida por una sola persona con modelos mínimos, un proceso simple, con herramientas simples. Para ser eficiente, la construcción de una casa real requiere de un equipo, modelamiento, un proceso bien definido y herramientas poderosas. Si se quisiera construir un edificio, poniendo las casitas de perro una sobre otra, lo más probable es que colapsen en poco tiempo."

Obviamente, el contexto de desarrollo debe girar en torno a la **envergadura del proyecto**, y debe determinar la configuración adecuada del proceso y los recursos necesarios. Existen propuestas radicales que promueven procesos y modelos más ligeros, tales como: *Extreme Programming (de Kent Beck)* y *Agile Modeling (de Scott Ambler)*. Sin embargo para proyectos de envergadura es difícil eludir un proceso y modelamiento más riguroso.

Hace unos 30 años atrás, Barry Boehm[11] formuló la teoría de que el costo del cambio del software se incrementaba en forma exponencial a través del tiempo; es decir, si un error se encontraba en tiempo de recolección de requerimientos, su solución costaría US$1, pero para un error que se encuentre durante el desarrollo del software, su solución costaría US$1000.

[11] **Barry Boehm:** Ha contribuido al desarrollo de la ingeniería de software aportando su modelo constructivo de costos (COCOMO) y el modelo espiral del proceso software.

La arquitectura es la visión de lo que uno esta construyendo. También implica trabajar con estándares y modelos precisos. Es insuficiente simplemente hacerlo, se requiere planificación y herramientas poderosas. La arquitectura es una ciencia y un arte. Implica imitación, método e intuición. La importancia relativa de estos elementos es diferente dependiendo si involucra una creación innovadora o tradicional.

2.2.1 Complejidad

Así como con los edificios, la complejidad y desarrollo de los sistemas software se ha incrementado sostenidamente, por eso se requiere de técnicas más sofisticadas. Un buen software debe ser funcional, compatible, realizable, resistente, capaz, seguro a fallas, tolerante a fallas, de costos alcanzables, disponible y adaptable a cambios tecnológicos. En el futuro, el gran desafío será dominar la complejidad.

2.2.2 Arquitectura del software

La arquitectura del software puede ser definida como el conjunto de decisiones importantes concernientes a la organización del sistema que debe ser construido: La selección de los elementos estructurales y sus interfaces, el comportamiento que se expresa por la colaboración entre estos elementos y el estilo que guía a la organización. La arquitectura de software, también involucra uso, funcionalidad, desempeño, elasticidad, reutilización, comprensión, equilibrio y restricciones de naturaleza económica y tecnológica, así también como lo concerniente a la estética. Una buena arquitectura es adaptable, simple y accesible. Los objetivos están claramente separados, las responsabilidades están uniformemente distribuidas y las restricciones económicas y técnicas están bien balanceadas.

2.2.3 Modelos

Los modelos son el lenguaje del diseñador. Son la representación de un sistema a ser construido o el de uno ya existente, son planos. Se dibujan a una escala reducida y muestran aspectos distintos del sistema real. Muchos son los actores involucrados en la construcción de un sistema y tienen diferentes vistas del proyecto. Una vista de arquitectura es una descripción simplificada del sistema (una abstracción) desde una perspectiva particular.

2.2.4 Vistas

La arquitectura de un sistema software puede ser representada por medio de 5 vistas: en el centro se encuentra la vista de casos de uso, rodeada de la vista lógica, la cual concierne al usuario, la vista de proceso la cual concierne al integrador de sistemas, la vista de implementación o realización, la cual concierne al programador, y la vista de despliegue o distribución, la cual concierne al ingeniero de sistema. Es posible agregar la vista de datos y la vista de seguridad. Todas estas vistas no tienen la misma importancia para todos los sistemas.

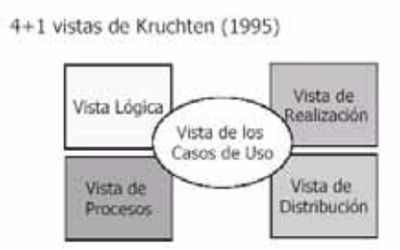


Figura 2.1: Vistas de Kruchten

2.2.5 UML

UML es un lenguaje para visualizar, especificar, construir y documentar los componentes de un sistema software. Es un estándar abierto que cubre todo el ciclo de vida de desarrollo de software, puede ser usado para todas las aplicaciones, esta basado en la experiencia y en las necesidades de la comunidad de usuarios y es apoyado por muchas herramientas. Muchas empresas de la industria de computadores han participado en su elaboración y está basado en el trabajo de los autores más importantes del campo del modelamiento de software orientado a objeto.

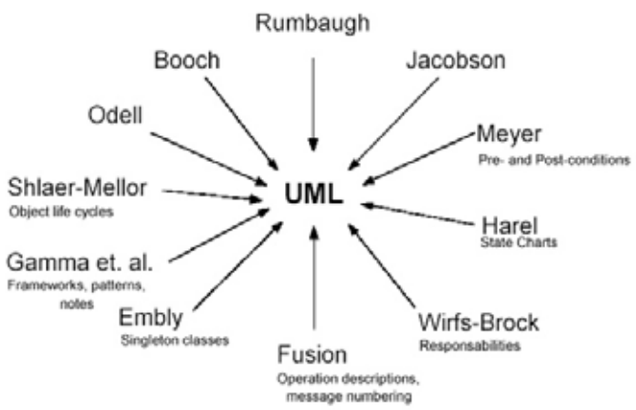


Figura 2.2: Influencias del UML

2.2.6 Elementos de UML

UML incluye elementos estructurales (clases, interfaces, colaboraciones, casos de uso, componentes, nodos); elementos de comportamiento (interacciones, máquinas de estado); elementos de agrupación (paquetes, subsistemas); y otros (notas). Los muchos tipos de relaciones son la dependencia, la asociación, la generalización y la realización.

Los casos de uso capturan la funcionalidad del sistema como es visto por los usuarios. Los diagramas de clase, el vocabulario del sistema. Los diagramas de objetos describen instancias específicas y enlaces. Los diagramas de componente y de despliegue describen la estructura física y la topología del sistema. Los diagramas de secuencia y de colaboración describen el comportamiento dinámico del sistema (orientados al tiempo). Los diagramas de estado muestran el comportamiento del sistema en respuesta a eventos, donde los diagramas de actividad representan los enlaces de actividad.

2.3 Diseño de arquitectura

Diseñar la arquitectura consiste en identificar, seleccionar y validar los elementos de arquitectura significativos. La arquitectura debería estar documentada. Un patrón es una solución recurrente a un problema específico en un contexto determinado. Un sistema bien estructurado está lleno de patrones (idiomas, patrones de diseño y patrones de arquitectura)

Estatutos del equipo de arquitectura de software

- Definir la arquitectura del software
- Mantener la integridad de la arquitectura del software
- Evaluar el riesgo técnico relacionado con el diseño del software
- Proponer el orden y el contenido de las iteraciones sucesivas
- Asistir a Marketing para la definición de futuros productos
- Facilitar la comunicación entre equipos de proyecto

Por lo tanto**, la clave en el desarrollo de sistemas de información se centra en: La notación, las herramientas y el proceso.**

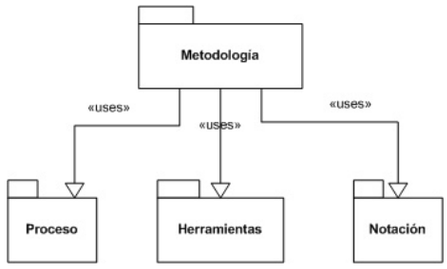


Figura 2.3: Componentes básicos de una metodología

La abstracción se relaciona fuertemente con el modelamiento visual, este modelamiento captura las partes esenciales del sistema. El objetivo real es poder generar una abstracción del proceso de negocios y transformarlo en un sistema computacional que sea equivalente.

Figura 2.4: Abstracción de un proceso de negocios.

La gran idea tras el uso del modelamiento visual es el poder manejar la complejidad, esto debido a que el cerebro humano sólo puede manejar entre 5 y 9 problemas al mismo tiempo.

"... Hay dos formas de construir software: una es hacerlo tan simple que obviamente no existan deficiencias, otra es hacerlo tan complejo que no existan deficiencias obvias" (C.A.R Houre, Turing Award Lecture 1980.)

El modelamiento visual nos permite además definir la arquitectura del software y modelar el sistema independientemente del lenguaje de implementación. Además promueve la reutilización de componentes en múltiples sistemas.

2.3.1 Resumen de técnicas y sus usos

Tabla 2.1: Técnicas de diagramación de UML.

Técnica	Propósito
Diagrama de actividad	Muestra el comportamiento con estructuras de control. Puede mostrar muchos objetos sobre uno o muchos casos de uso, o implementación de métodos. Promueve el comportamiento paralelo.
Diagrama de clases	Muestra la estructura estática de conceptos, tipos y clases. Los conceptos muestran como el usuario ve el mundo; los tipos las interfaces de componentes de software y las clases; La implementación de componentes de software.
Diagrama de despliegue	Muestra un diseño físico de componentes en un nodo de hardware.
Diagramas de interacción	Muestran como varios objetos colaboran en un solo caso de uso.
Diagrama de paquetes	Muestra grupos de clases y las dependencias entre ellas.
Diagrama de estado	Muestra como un objeto se comporta a través de muchos casos de uso.
Casos de uso	Obtiene los requerimientos de usuario en trozos significativos. La planificación de la construcción se realiza liberando algunos casos de uso en cada iteración. Constituye la base para la prueba del sistema.

Capítulo 3 : Rational Unified Process

3.1 ¿Qué es el Rational Unified Process?

El Rational Unified Process es un proceso de Ingeniería de Software. Provee de un método disciplinado de asignar tareas y responsabilidades dentro de una organización de desarrollo. Su meta es asegurar la producción de software de alta calidad que se ajuste a las necesidades de sus usuarios finales, dentro de un tiempo y presupuesto predecibles.

El Rational Unified Process es un producto, desarrollado y mantenido por **Rational® Software**. RUP[12] acentúa la productividad de los equipos de desarrollo de software, dando a cada miembro un fácil acceso a la base de conocimientos con directivas, plantillas y tutores de herramientas para todas las actividades de desarrollo críticas. Permitiendo que todos los miembros del equipo tengan acceso a la misma base de conocimiento, no importando si trabaja con requerimientos, diseño, prueba, administración de proyectos o administración de la configuración. Asegura que todos los miembros del equipo compartan un lenguaje, un proceso y una visión común de cómo desarrollar el software. Las actividades que se realizan en el RUP involucran crear y mantener modelos, en vez de centrarse en la producción de una gran cantidad de documentos en papel. El proceso unificado enfatiza el desarrollo y mantenimiento de modelos que constituyen una rica representación semántica del sistema software en desarrollo.

RUP constituye una guía de cómo usar en forma efectiva el lenguaje de modelamiento unificado (UML). RUP es apoyado por herramientas las que automatizan una gran parte del proceso. Una de las ventajas de RUP es que es un proceso configurable. No existe ningún proceso que se ajuste a todos los desarrollos de software. RUP se ajusta bien a pequeños equipos, así como a grandes organizaciones de desarrollo.

3.2 El proceso de ingeniería de software

El proceso de desarrollar un nuevo sistema o mejorar uno que ya existe implica un conjunto ordenado de pasos con la intención de lograr una meta. El **Rational Unified Process** es iterativo, se centra en la arquitectura, es conducido por casos de uso y confrontado al riesgo.

[12] **RUP:** Rational Unified Process

Consiste en las siguientes fases:

- **Inicio (Inception) :** Define el ámbito del proyecto y desarrolla los casos de negocio.
- **Elaboración (Elaboration):** Planifica el proyecto, especifica sus características y determina la línea base de la arquitectura.
- **Construcción (Construction):** Construye el producto.
- **Transición (Transition):** Muestra el producto al usuario.

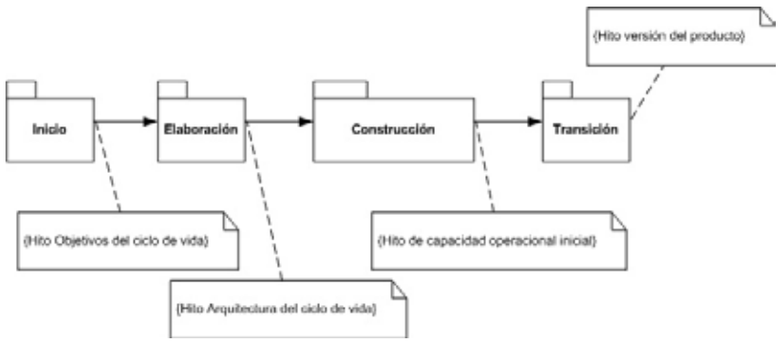


Figura 3.1: Fases del Rational Unified Process (RUP).

Cada fase está compuesta de varias iteraciones, dividida en una secuencia de actividades con un plan establecido y un criterio de evaluación, que resulta en una versión ejecutable. Modelamiento, definición de requerimientos, análisis, diseño, implementación, prueba y otras actividades se dividen entre las fases.

Las fases no son idénticas en términos de duración y esfuerzo. Aunque esto varíe considerablemente dependiendo del proyecto, un ciclo de desarrollo inicial típico para un proyecto de tamaño mediano, debería anticipar la siguiente distribución de esfuerzos y duración:

Tabla 3.1: Distribución de tiempo y esfuerzo para un ciclo de desarrollo inicial típico de un proyecto mediano.

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

Representada gráficamente como sigue:

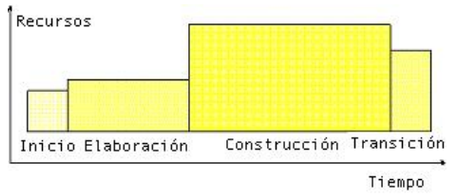


Figura 3.2: Distribución de tiempo y esfuerzo en un ciclo de desarrollo típico (mediano).

Para un ciclo de evolución, las fases de inicio y elaboración (inception y elaboration) serían considerablemente más pequeñas. Las herramientas con las cuales se puede automatizar cierta parte del esfuerzo de la fase de construcción pueden lograr que dicha fase sea mucho más pequeña que las fases de inicio y elaboración juntas.

Una pasada por las cuatro fases es un ciclo de desarrollo; cada pasada a través de las cuatro fases produce una generación del software. A menos que el producto "muera", evolucionará a su próxima generación, repitiendo la misma secuencia de fases de inicio, elaboración, construcción y transición, pero esta vez con un énfasis distinto en cada fase. A esta secuencia de ciclos se les llama ciclos de evolución. Al pasar por varios ciclos, se producen nuevas generaciones.

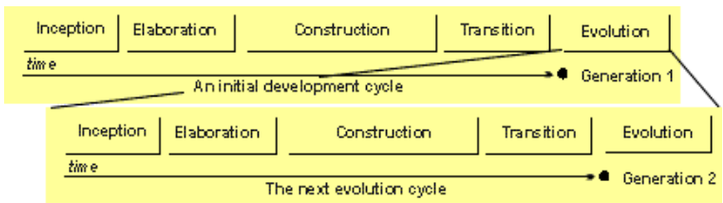


Figura 3.3: Ciclos de Evolución, extraído de Rational Software Inc.

Los ciclos de evolución pueden ser gatillados por mejoramientos sugeridos por usuarios, cambios en el contexto de usuario, cambios en la tecnología subyacente, reacciones hacia la competencia, y así sucesivamente. Los ciclos de evolución por lo general tienen fases de inicio y elaboración más cortas, esto debido a que la definición del producto y su arquitectura han sido ya determinados por ciclos de desarrollo previos. Excepciones a esta regla son los ciclos de evolución en los cuales ocurre una redefinición de la arquitectura del producto.

3.3 Desarrollo efectivo de las seis mejores prácticas

El Rational Unified Process describe como desplegar efectivamente métodos probados comercialmente para el desarrollo de software. Se les llama “las mejores prácticas” no precisamente por que se pueda medir su valor, sino porque se ha observado que son comúnmente usadas en la industria por organizaciones exitosas.

Las siguientes se conocen como las seis mejores prácticas para el desarrollo de software:

1. **Desarrollar el software iterativamente**
2. **Administrar los requerimientos**
3. **Usar arquitecturas basadas en componentes**
4. **Modelar el Software en forma visual**
5. **Verificar la Calidad del Software**
6. **Controlar los cambios del software**

3.3.1 Desarrollar el software iterativamente

Dados los sofisticados sistemas de software de hoy en día, no es posible realizar de una manera secuencial las siguientes etapas: definir el problema por completo, diseñar por completo la solución, construir el software y luego probar el producto al final. Se requiere un método iterativo para incrementar la comprensión del problema a través de refinamientos sucesivos e ir madurando en forma incremental una solución efectiva, después de múltiples iteraciones.

El RUP apoya un método iterativo de desarrollo que apunta a los elementos de mayor riesgo en cada etapa del ciclo de vida, reduciendo significativamente el perfil de riesgo del proyecto. Este método iterativo ayuda a atacar el riesgo a través de progresos demostrables, mediante versiones ejecutables que permiten un contacto y retroalimentación continua con el usuario final. Así cada iteración termina con una versión ejecutable, por lo que el equipo de trabajo se mantiene concentrado en producir resultados. Además, las frecuentes revisiones de estado ayudan a asegurar que el proyecto cumple con los tiempos estimados.

Un método iterativo también permite acomodar con mayor facilidad cambios tácticos en los requerimientos, características o el plan de trabajo.

3.3.2 Administrar los requerimientos

RUP permite describir cómo recabar, organizar y documentar las funcionalidades y restricciones requeridas: seguirle la pista y documentar los cambios y decisiones; capturar y comunicar con facilidad los requerimientos de negocio. Las notaciones de casos de uso y escenarios en el proceso han probado ser una excelente forma de capturar los requerimientos funcionales y asegurar que estos conduzcan al diseño, implementación y prueba del software, aumentando las posibilidades de que el sistema final satisfaga las necesidades del usuario final.

3.3.3 Utilizar arquitecturas basadas en componentes

El proceso se centra en el desarrollo temprano y una línea base para una arquitectura ejecutable robusta, antes de comprometer recursos para un desarrollo a gran escala. Describe como diseñar una arquitectura elástica (flexible), que se adecue a los cambios, sea comprendida en forma intuitiva, y promueva una reutilización más efectiva del software.

Los componentes son módulos no triviales, subsistemas que satisfacen una función clara. RUP provee de un método sistemático para definir una arquitectura usando componentes nuevos y ya existentes. Estos son ensamblados en una arquitectura bien definida, tal como: Internet, CORBA, .COM o .NET.

3.3.4 Modelar el Software en forma visual

El proceso muestra cómo modelar el software visualmente, para capturar la estructura y el comportamiento de las arquitecturas y componentes. Esto permite ocultar los detalles y usar “Bloques de construcción gráficos”. La abstracción visual ayuda a comunicar diferentes aspectos del software; ver cómo se agrupan los elementos del sistema; asegurarse de que los “bloques de construcción” sean consistentes con el código; manteniendo una consistencia entre el diseño y su implementación; y promover una comunicación no ambigua. El estándar UML, son los cimientos de un modelamiento visual exitoso.

3.3.5 Verificar la Calidad del Software

Aplicaciones con un desempeño pobre y baja confiabilidad son factores comunes que reprimen dramáticamente la aceptabilidad del software.

Consecuentemente, la calidad debería ser revisada con respecto a los requerimientos basados en la confiabilidad, funcionalidad, desempeño de la aplicación y desempeño del sistema.

El Rational Unified Process asiste la planificación, el diseño, implementación, ejecución y evaluación de estos tipos de prueba. La evaluación de la calidad se realiza en todas las actividades del proceso e involucra a todos los participantes, haciendo uso de medidas y criterios objetivos, y nunca es tratada como una experiencia tardía o una actividad aparte del proceso.

3.3.6 Controlar los cambios del software

La habilidad de manejar los cambios, asegurándose que cada uno de ellos sea aceptable y, además, poder seguirles la pista, es esencial en un entorno en que estos son inevitables.

El proceso describe cómo controlar, seguir la pista y monitorear los cambios, para permitir el éxito del desarrollo iterativo. También constituye una guía de cómo establecer espacios de trabajo seguros para cada desarrollador, así entonces, provee de un aislamiento de los cambios realizados en otros espacios de trabajo, controlando los cambios de todo el software (Modelos, códigos, documentos, etc.). Permite que un equipo trabaje junto como si fuera una sola unidad.

3.4 Visión general del proceso

3.4.1 Dos Dimensiones

El proceso puede ser descrito en dos dimensiones, o a lo largo de dos ejes. El eje horizontal representa tiempo y representa el aspecto dinámico del proceso cómo se establece, y se expresa en términos de ciclos, fases, iteraciones e hitos. El eje vertical representa el aspecto estático del proceso: cómo es descrito en términos de actividades, objetos, trabajadores y flujos de trabajo (workflows).

El modelo iterativo del gráfico muestra cómo se estructura el proceso en dos dimensiones.

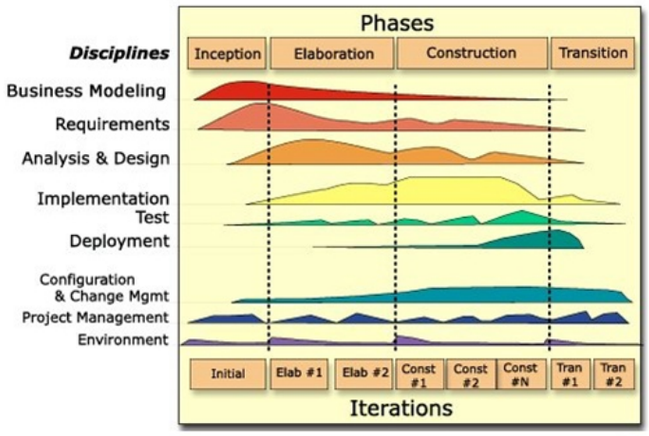


Figura 3.4: Fases y Workflows del RUP, extraído de Rational Software Inc.

Fases e Iteraciones - La dimensión del tiempo

Esta es la organización dinámica del proceso a lo largo del tiempo.

El ciclo de vida del software se divide en ciclos, cada ciclo trabaja en una nueva generación del producto.

El RUP divide un ciclo de desarrollo en cuatro fases consecutivas:

1) **Fase de Inicio (Inception Phase)**
2) **Fase de elaboración (Elaboration Phase)**
3) **Fase de construcción (Construction Phase)**
4) **Fase de transición (Transition Phase)**

Cada fase termina con un hito bien definido, que es un punto en el tiempo, en el cual debe ser tomada una decisión importante, y por consiguiente, se deben haber logrado metas clave.

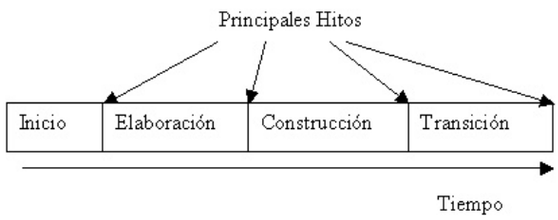


Figura 3.5: Principales Hitos de las fases del proceso.

Cada fase tiene un propósito específico, los cuales serán descritos a continuación.

3.4.1.1 Fase de Inicio (Inception Phase)

Durante la fase de inicio, se establece el caso de negocio para el sistema y se delimita el ámbito del proyecto. Para lograr esto, se deben identificar todas las entidades externas con las cuales el sistema interactuará (actores) y definir la naturaleza de estas interacciones a un nivel mayor. Esto significa encontrar todos los casos de uso y describir los más significativos. El caso de negocios incluye criterios de éxito, evaluación de riesgo, una estimación de los recursos necesarios y una fase de planificación que muestre las fechas de los hitos más importantes.

El resultado de la fase de inicio es:

- **Un documento de visión del proyecto:**

  Es una visión general de los requerimientos principales del proyecto, características principales y restricciones principales.

- **Un modelo de casos de uso inicial:** (Completado en un 10% – 20%)
- **Un glosario inicial del proyecto:**

  Puede ser parcialmente expresado como un modelo.

- **Un caso de negocio inicial:**

  Debe incluir el contexto del negocio, criterios de éxito (proyecciones de renta, reconocimiento del mercado, etc.), y proyección financiera.

- **Una evaluación de riesgo inicial.**
- **Una planificación del proyecto:**

  Que muestre las fases e iteraciones.

- **Un modelo de negocio:** Si es necesario
- **Uno o varios prototipos**

Hito: Objetivo del Ciclo de vida.

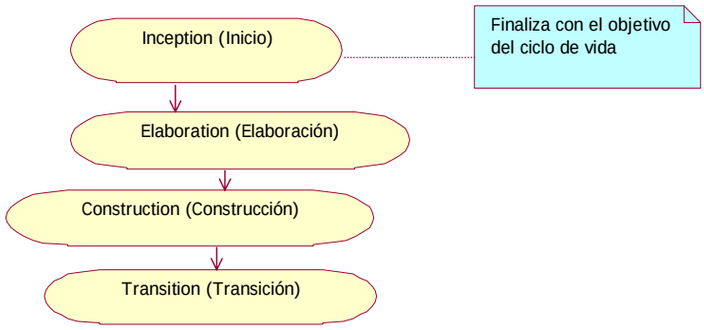


Figura 3.6: Objetivo del ciclo de vida.

Al final de la fase inicial se encuentra el primer hito del proyecto: el hito de objetivos del ciclo de vida (lifecycle objective milestone) .

Los criterios de evaluación para la fase de inicio son:

- **Consentimiento de los inversionistas en el ámbito de definición y costos / tiempos estimados.**
- **Entendimiento de los requerimientos, el cual es evidenciado por la fidelidad del caso de uso principal.**
- **Credibilidad de los costos y tiempos estimados, prioridades, riesgos y procesos de desarrollo.**
- **Profundidad y amplitud de cualquier prototipo de arquitectura que fue desarrollado.**
- **Desembolsos actuales versus desembolsos planificados.**

El proyecto podría ser cancelado o considerablemente modificado si falla pasar este hito.

3.4.1.2 Fase de elaboración (Elaboration Phase)

El propósito de la fase de elaboración es analizar el dominio del problema, establecer una arquitectura base acertada, desarrollar la planificación del proyecto, y eliminar los elementos de mayor riesgo. Para lograr estos objetivos se debe contar con una visión amplia y de profundidad detallada del sistema. Las decisiones de arquitectura deben ser hechas con un entendimiento del sistema como un todo, su visibilidad, funcionalidades principales y requerimientos no funcionales tales como requerimientos de desempeño. Es fácil vislumbrar que la fase de elaboración es la

más crítica de las cuatro fases. Al final de esta fase, el trabajo de ingeniería "Fuerte" se considera completo y el proyecto continúa a su día de recuento más importante: la decisión si se realizan o no las fases de construcción y transición del proyecto. Para la mayoría de los proyectos, este día también significa la transición desde una operación de bajo riesgo, ligera y de bajo costo a ser una operación de alto riesgo, pesada y de alto costo. El proceso debe siempre ajustar los cambios, las actividades de la fase de elaboración aseguran que la arquitectura, los requerimientos y planificación están lo suficientemente estables, y que los riesgos están suficientemente mitigados, por lo que se puede determinar en forma predictiva el costo y tiempo necesarios para completar el desarrollo. Conceptualmente, este nivel de fidelidad correspondería al nivel necesario para que una organización consiga un precio fijo en la fase de construcción.

En la fase de elaboración, se construye un prototipo ejecutable en una o más iteraciones, dependiendo de la visibilidad, tamaño, riesgo y lo novedoso del proyecto. Este esfuerzo debería por lo menos identificar los casos de uso críticos en la fase de inicio, los que por lo general exponen los riesgos técnicos principales del proyecto. Un prototipo evolutivo de calidad, de un componente en producción, será siempre una meta, esto no excluye el hecho de explorar y, si es necesario, desechar el prototipo para suavizar riesgos específicos.

Los resultados de la fase de elaboración son:

- **Un modelo de casos de uso:** Completo al menos en un 80%
- **Todos los actores y casos de uso se han identificado, y la mayoría de las descripciones de los casos de uso se han desarrollado.**

Requerimientos suplementarios:

- **Capturando los requerimientos no funcionales y cualquier requerimiento que no esté asociado con un caso de uso específico.**
- **Una descripción de la arquitectura del software**
- **Un prototipo arquitectónico ejecutable del prototipo.**
- **Una lista de riesgos revisada y un caso de negocios revisado.**
- **Una planificación del desarrollo para el proyecto global:**

- **Incluye una planificación de grano grueso del proyecto, que muestre las interacciones y el criterio de evolución para cada iteración.**

- **Una actualización del evento de desarrollo:** Especificando el proceso a ser usado.
- **Un manual de usuario preliminar (opcional).**

Hito: Arquitectura del Ciclo de vida.

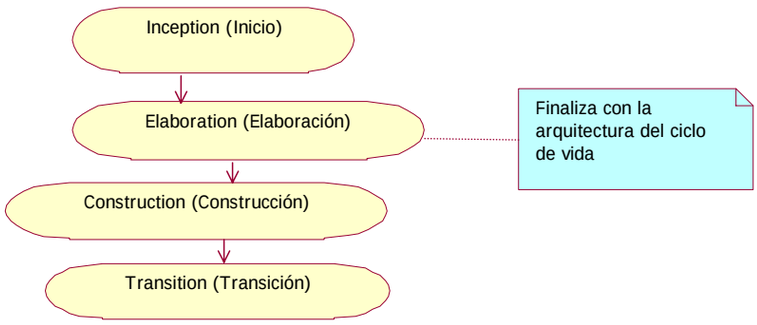


Figura 3.7: Arquitectura del Ciclo de vida.

Al final de la fase de elaboración se encuentra el segundo hito importante del proyecto, el hito ciclo de vida de la arquitectura. En este punto, se examinan en forma detallada los objetivos y ámbito del sistema, la elección de la arquitectura, y la resolución, o nivel de profundidad, de los principales riesgos.

El criterio de evaluación principal para la fase de elaboración involucra la respuesta a las siguientes preguntas:

- **¿Es estable la visión del producto?**
- **¿Es estable la arquitectura?**
- **¿La demostración del ejecutable muestra que los principales elementos de riesgo han sido revisados y resueltos?**
- **¿Es el plan de la fase de construcción lo suficientemente detallado y exacto?, ¿Está respaldado por una base creíble de estimaciones?**
- **¿Están todos los involucrados en el proyecto de acuerdo en que la visión actual que se tiene de éste puede ser lograda si la planificación fuese ejecutada (para desarrollar el sistema por completo, en el contexto de la arquitectura actual)?**

- **¿Es el gasto de recursos actual aceptable en comparación al gasto de recursos planificado?**

El proyecto puede ser abortado o rediseñado si falla pasar por este hito.

3.4.1.3 Fase de construcción (Construction Phase)

Durante la fase de construcción, son desarrollados todos los componentes y características restantes de la aplicación, y son integrados al producto, además, todas estas características se prueban a conciencia. La fase de construcción, descrita en una frase, es un proceso de manufactura, donde se pone énfasis en administrar los recursos y controlar las operaciones para optimizar los costos, tiempos, y la calidad. En este sentido, la administración tiende a experimentar una transición desde el desarrollo de propiedad intelectual, durante las fases de inicio y elaboración, al desarrollo de productos durante las fases de construcción y transición.

Muchos proyectos son lo suficientemente grandes como para necesitar generar incrementos de producción a través de la construcción en paralelo. Estas actividades en paralelo pueden acelerar en forma considerable la disponibilidad de una nueva versión. Pero también pueden incrementar la complejidad de la administración de recursos y la sincronización del flujo de trabajo (workflow).

Una arquitectura robusta se relaciona directamente con una buena planificación (comprensible). En otras palabras, una de las cualidades críticas de la arquitectura es su facilidad de construcción. Esta es una de las razones por las cuales se acentúa un desarrollo balanceado de la arquitectura y planificación durante la fase de elaboración.

El resultado de la fase de construcción es un producto listo para ponerlo en las manos del usuario final. Como mínimo consiste en:

- **El producto software se ha integrado en las plataformas adecuadas.**
- **Los manuales de usuario.**
- **Una descripción de la versión actual.**

Hito: Capacidad operacional inicial

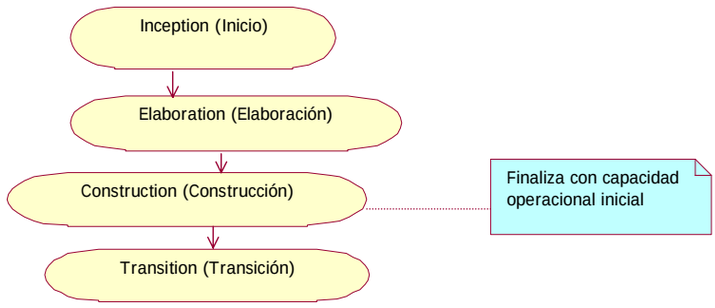


Figura 3.8: Capacidad Operacional Inicial.

En este punto, se decide si el software, los sitios y los usuarios están listos para ir a operación, sin exponer el proyecto a riesgos altos. Esta versión se llama a menudo "versión beta".

El criterio de evaluación para la fase de construcción involucra dar respuesta a las siguientes preguntas:

- **¿Está esta versión del producto estable y lo suficientemente madura para ser liberada a la comunidad de usuarios?**
- **¿Están todos los involucrados en el proyecto listos para la transición a la comunidad de usuarios?**
- **¿Son aún aceptables los gastos de recursos actuales en comparación con los gastos de recursos planificados?**

La transición puede ser pospuesta si el proyecto falla el alcanzar este hito.

3.4.1.4 Fase de Transición (Transition Phase)

El propósito de la fase de transición es traspasar el producto a la comunidad de usuarios. Una vez que el producto se ha entregado a las manos del usuario final, siempre surgen imprevistos que requieren del desarrollo de una nueva versión, corregir algunos errores, o finalizar las características que fueron pospuestas anteriormente.

La fase de transición se completa cuando la línea base está lo suficientemente madura para ser liberada al dominio de los usuarios finales. Esto requiere que alguna parte del sistema, que se pueda usar, haya sido completada con un nivel aceptable de calidad y que la

documentación de usuario esté disponible, de modo que la transición al usuario resulte positiva para todas las partes involucradas. Esto incluye:

- **"Beta Testing"**

  Para validar el nuevo sistema contra las expectativas del usuario.

- **Operación en paralelo con el sistema que se pretende reemplazar.**
- **Conversión de las bases de datos operacionales**
- **Entrenamiento de los usuarios y administradores de mantención.**
- **Primera aparición pública del producto al mercado, distribución y equipos de venta.**

La fase de transición se centra en las actividades requeridas para colocar el software en las manos de los usuarios finales. Esta fase incluye diversas iteraciones, incluyendo versiones beta, versiones de disponibilidad general, tanto como versiones libres de bugs y versiones mejoradas. Se gastan esfuerzos considerables en desarrollar documentación orientada al usuario, entrenar usuarios, apoyar a los usuarios en el uso inicial del producto, y reaccionar a las consultas de los usuarios. En este punto las consultas o sugerencias de los usuarios deberían ser confinadas al afinamiento del producto, configuración, instalación y asuntos de funcionalidad.

Los objetivos principales de la fase de transición son:

- **Lograr que el usuario se auto-apoye en el uso del software**
- **Lograr el consenso de los involucrados en el proyecto de que las líneas base se han completado y son consistentes con el criterio de evaluación de la visión del proyecto.**
- **Lograr la línea base final del producto tan rápidamente como sea posible y a tan bajo costo como se permita.**

Esta fase puede ser muy simple o extremadamente compleja dependiendo del tipo de producto.

Hito : Versión del producto

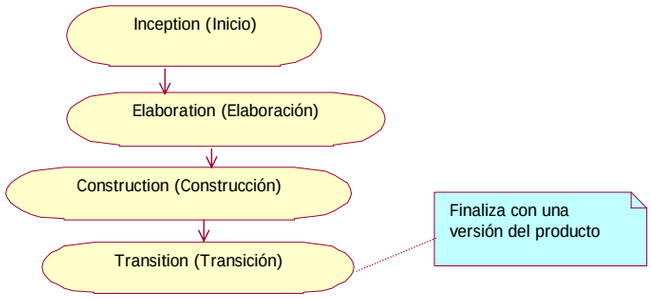


Figura 3.9: Versión del producto.

Al final de la fase de transición se encuentra el cuarto hito importante del proyecto, el hito de versión del producto. En este punto, se decide si se lograron los objetivos, y si se debería comenzar otro ciclo de desarrollo. En algunos casos, este hito, puede coincidir con el fin de la fase de inicio para el próximo ciclo.

El criterio de evaluación principal para la fase de transición involucra dar respuesta a las siguientes preguntas:

- **¿Está satisfecho el usuario?**
- **¿Son aceptables aún los gastos actuales de recursos comparados con los gastos planificados?**

3.4.2 Iteraciones

Cada fase en Rational Unified Process puede ser dividida en iteraciones. Una iteración es un lazo de desarrollo completo que resulta en una versión (interna o externa) de un producto ejecutable, un subconjunto del producto final de desarrollo, el cual crece incrementalmente de iteración a iteración para convertirse en el sistema final.

3.4.2.1 Beneficios de un método iterativo

Comparado con el proceso de cascada tradicional, el proceso iterativo tiene las siguientes ventajas:

- **Los riesgos son suavizados al comienzo.**
- **El cambio es más controlable.**
- **Mayores niveles de reutilización.**
- **El equipo del proyecto puede aprender a lo largo del desarrollo.**

- **Mejor calidad global.**
- **Estructura estática del proyecto**

Un proceso describe **QUIÉN** está haciendo **QUÉ**, **CÓMO** y **CUÁNDO**. El RUP se representa usando cuatro elementos de modelamiento principales:

- **Trabajadores, el "quién"**
- **Actividades, el "cómo"**
- **Objetos, el "qué"**
- **Workflow, el "cuándo"**

3.4.2.2 Trabajadores

Un trabajador define el comportamiento y responsabilidades de un individuo, o un grupo de individuos trabajando juntos como equipo.

Se podría ver a un trabajador como un "sombrero" que un individuo puede usar en el proyecto. Un individuo podría eventualmente usar muchos sombreros diferentes. Esta es una distinción importante, ya que es más natural ver a un trabajador como un individuo o un equipo, pero en el RUP el trabajador es un rol que define cómo el individuo debería llevar a cabo el trabajo.

3.4.2.3 Actividad

Una actividad de un trabajador específico es una unidad de trabajo que un individuo en ese rol puede realizar. La actividad tiene un propósito claro, usualmente expresada en términos de crear o actualizar algún objeto, tales como un modelo, una clase, o una planificación. Cada actividad se asigna a un trabajador específico. La granularidad de una actividad es generalmente de un par de horas a un par de días, usualmente involucra a un trabajador, y afecta a uno o a un número pequeño de objetos. Una actividad debería usarse como un elemento de planificación y progreso; si es muy pequeño se descuidará, y si es muy grande, el progreso tendría que ser expresado como una parte de la actividad.

Ejemplos de actividades:

- **Planificar una iteración, actividad para el trabajador: Administrador de proyecto.**
- **Encontrar casos de uso y actores, actividad para el trabajador: Analista de sistema.**
- **Revisar el diseño, actividad para el trabajador: Revisor de diseño.**
- **Ejecutar pruebas de rendimiento, actividad para el trabajador: Probador de rendimiento.**

3.4.2.4 Objetos

Un objeto es una pieza de información que es producida, modificada o usada por un proceso. Los objetos son los productos tangibles del proyecto, las cosas que produce o usa el proyecto mientras se trabaja hacia el producto final. Los objetos son usados como entradas por los trabajadores para realizar una actividad, y son el resultado o salida de tales actividades. En términos de diseño orientado a objeto, así como las actividades son operaciones sobre un objeto activo (el trabajador), los objetos son los parámetros de estas actividades.

Los objetos pueden tomar varias formas:

- **Un modelo, tal como un modelo de casos de uso o un modelo de diseño.**
- **Un elemento dentro de un modelo, tal como una clase, un caso de uso, o un subsistema.**
- **Un documento, tal como un caso de negocios o documento de arquitectura de software**
- **Código Fuente**
- **Ejecutables**

3.4.2.5 Workflow

Una mera enumeración de todos los trabajadores, actividades y objetos, no basta para constituir un proceso. Se necesita una forma para describir secuencias significativas de actividades que produzcan algún resultado, y que muestren la interacción entre trabajadores.

Un workflow es una secuencia de actividades que producen un resultado con un valor observable. En términos de UML, un workflow puede ser expresado como un diagrama de secuencia, un diagrama de colaboración, o un diagrama de actividad.

Nótese que no siempre es posible o práctico representar todas las dependencias entre actividades. A menudo, dos actividades están más entrelazadas de lo que se muestra, especialmente cuando involucran al mismo trabajador o al mismo individuo. Las personas no son máquinas, por lo que los workflows no pueden ser interpretados literalmente como programas para personas, para ser seguidos exactamente o mecánicamente. En la próxima sección se discutirá el tipo de workflow más esencial en el proceso, llamado Workflow central (core workflow).

3.4.2.6 Workflow Central

Existen nueve tipos de workflows centrales de proceso en RUP, que representan una partición de todos los trabajadores y actividades en agrupaciones lógicas.

Los workflows centrales del proceso se dividen en seis workflows centrales de ingeniería:

- **Workflow de modelamiento de negocio**
- **Workflow de requerimientos**
- **Workflow de análisis y diseño**
- **Workflow de implementación**
- **Workflow de prueba**
- **Workflow de despliegue**

Y tres workflows centrales de apoyo:

- **Workflow de administración de proyectos**
- **Workflow de administración de la configuración y cambio**
- **Workflow de entorno**

Aunque los nombres de los seis workflows centrales de ingeniería pueden evocar a las fases secuenciales del proceso tradicional de cascadas, se debería mantener en mente que las

fases de un proceso iterativo son diferentes y que estos workflows se revisan una y otra vez a lo largo del ciclo de vida. El workflow real completo de un proyecto interpola estos nueve workflows centrales, y los repite con diversas intensidades y énfasis en cada iteración.

Figura 3.10: Workflows en proceso de desarrollo de software RUP, extraído de Rational Software Inc.

3.4.2.6.1 Modelamiento de negocios

Uno de los principales problemas en los esfuerzos de la ingeniería de negocios, es que las comunidades de ingeniería de software y de ingeniería de negocios no se comunican debidamente. Esto conduce a que la salida de la ingeniería de negocios no se use adecuadamente como entrada al esfuerzo de desarrollo de software, y viceversa. Rational Unified Process soluciona esto brindando un lenguaje común y un proceso a ambas comunidades, así también muestra cómo crear y mantener el rastro a los modelos de negocio y de ingeniería.

En modelamiento de negocios, se documentan los procesos de negocio haciendo uso de los llamados casos de usos de negocio. Esto asegura un entendimiento común entre todos los involucrados en el proyecto y de lo que el proceso de negocio necesita para ser admitido en la organización.

Los casos de uso de negocios son analizados para entender cómo el negocio debería apoyar los procesos de negocios.

Muchos proyectos pueden elegir no realizar un modelo de negocios.

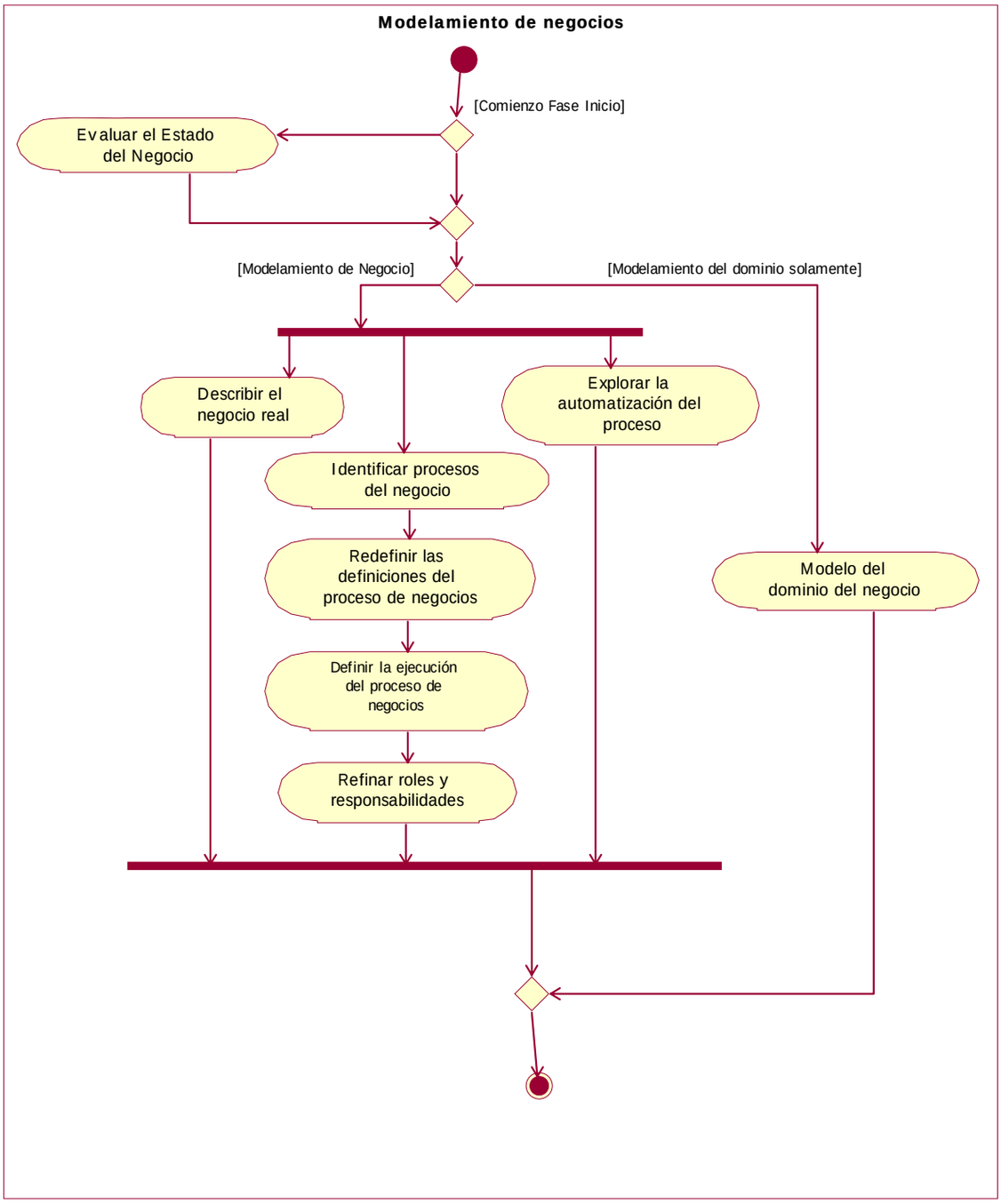


Figura 3.11: Diagrama de Actividad de workflow de Modelamiento de negocios.

Modelamiento de negocios

Tabla 3.2: Tareas, actividades y resultados clave workflow de modelamiento de negocio.

TAREAS	RESULTADO CLAVE
Evaluar el estado del negocio Capturar un vocabulario común de negocio Mantener políticas del negocio Evaluar el objetivo de la organización Establecer y ajustar metas Desarrollar directivas de modelamiento de negocio	Visión de negocio Evaluación de los objetivos de la organización Glosario de negocio Políticas de negocio
Describir el negocio actual Evaluar objetivos de la organización Encontrar los Actores y Casos de uso de negocio Establecer y ajustar metas. Encontrar las entidades y trabajadores del negocio	Evaluación de los objetivos de la organización Modelo de casos de uso de negocio Casos de uso de negocio Especificaciones adicionales de negocio Visión de negocio Modelo de objetos del negocio Realización de casos de uso de negocios
Identificar procesos de negocio Mantener políticas de negocio Establecer y ajustar reglas Definir arquitectura de negocio Capturar un vocabulario común de negocio Encontrar Actores y casos de uso	Políticas de negocio Visión de negocio Documento de arquitectura de negocio Glosario de negocio Modelo de casos de uso de negocio Casos de uso de negocio Especificaciones adicionales de negocio
Redefinir las definiciones de procesos de negocio Detallar un caso de uso de negocio Revisar modelo de casos de uso de negocio Estructurar el modelo de casos de uso de negocio	Modelo de casos de uso de negocio Casos de uso de negocios Especificaciones adicionales de negocio Revisar registro
Diseñar la ejecución del proceso de negocios Capturar un vocabulario de negocio común Encontrar entidades y trabajadores de negocio Mantener las políticas de negocio Definir la arquitectura de negocio	Glosario de negocio Modelo de objetos de negocio Realización de casos de uso de negocio Política de negocio Documento de arquitectura de negocio
Refinar roles y responsabilidades Detallar una entidad de negocio Detallar un trabajador de negocio Revisar modelo de objeto de negocio	Trabajador de negocio Entidad de negocio Unidad de organización Revisar registro Visión de negocio Modelo de casos de uso Modelo de análisis Especificaciones adicionales
Desarrollar un modelo del dominio Mantener políticas de negocio Capturar un vocabulario de negocio común Detallar una entidad de negocio Encontrar trabajadores y entidades de negocio Revisar el modelo de objetos de negocio	 Políticas de negocio Glosario de negocio Modelo de objeto de negocio Entidad de negocio Revisar registro

3.4.2.6.2 Requerimientos

El objetivo del workflow de requerimientos es describir qué es lo que el sistema debería hacer y permitir a los desarrolladores y clientes estar de acuerdo en la descripción. Para lograr esto, se extraen, organizan y documentan las funcionalidades y restricciones requeridas; se les sigue el rastro a los cambios y decisiones. Se crea un documento con la visión de negocio y se extraen las necesidades de los inversionistas. Los actores son identificados representando a los usuarios y a cualquier otro sistema que puede interactuar con el sistema que se está desarrollando. Los casos de uso se identifican representando el comportamiento del sistema. Como los casos de uso se desarrollan de acuerdo a las necesidades del actor, el sistema parece ser más relevante a los usuarios.

Cada caso de uso es descrito en detalle. La descripción de casos de uso muestra cómo el sistema interactúa paso a paso con los actores y qué es lo que hace el sistema. Los requerimientos no funcionales son descritos en las especificaciones suplementarias.

La función de los casos de uso es de lazo unificador a lo largo del ciclo de desarrollo. El mismo modelo de casos de uso es usado durante la captura de requerimientos, análisis y diseño, y prueba.

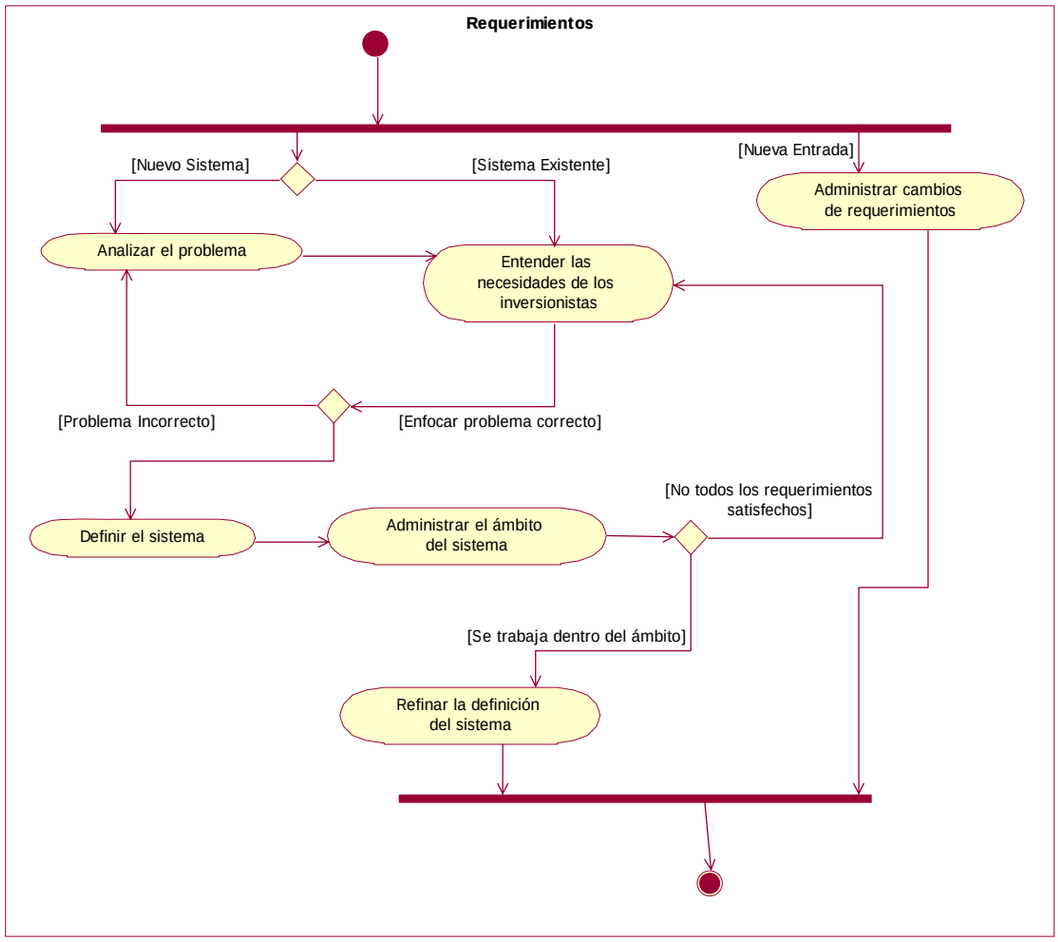


Figura 3.12: Diagrama de Actividad de workflow de Requerimientos.

Tabla 3.3: Tareas, actividades y resultados clave workflow de requerimientos.

TAREAS	RESULTADO CLAVE
Analizar el problema Capturar un vocabulario común Desarrollar visión Encontrar actores y casos de uso Desarrollar plan de administración de requerimientos	Glosario Visión Modelo de casos de uso Plan de administración de requerimientos Atributos de requerimientos
Entender las necesidades de los inversionistas Capturar un vocabulario común Desarrollar visión Producir como resultado los requerimientos de los inversionistas Encontrar Actores y casos de uso Administrar dependencias Revisar solicitud de cambio	Glosario Visión Solicitudes de inversionistas Modelo de casos de uso Especificaciones adicionales Atributos de requerimientos
Definir el sistema Desarrollar visión Capturar un vocabulario común Encontrar Actores y casos de uso Administrar dependencias	Glosario Modelo de casos de uso Especificaciones adicionales Atributos de requerimientos
Administrar el ámbito del sistema Desarrollar visión Administrar dependencias Priorizar casos de uso Revisar solicitud de cambio	Visión Atributos de requerimientos
Refinar la definición del sistema Detallar un caso de uso Detallar los requerimientos de software Modelar la interfaz de usuario Prototipar la interfaz de usuario	Especificaciones adicionales Caso de uso Especificación de requerimientos de software Historial de casos de uso Prototipo de interfaz de usuario Atributos de requerimientos
Administrar cambios de requerimientos Administración de dependencias Revisar solicitud de cambio Revisar requerimientos Estructurar el modelo de casos de uso Revisar registro	Modelo de casos de uso Atributos de requerimientos

3.4.2.6.3 Análisis y diseño

El objetivo del workflow de análisis y diseño es mostrar cómo se realizará el sistema en la fase de implementación. Se desea construir un sistema que: Realice –en un entorno de implementación específico– las tareas y funciones especificadas en la descripción de casos de uso, satisfaga todos sus requerimientos y que esté estructurado para ser robusto, es decir, posea una facilidad al cambio si sus requerimientos llegaran a cambiar en algún momento.

El workflow de análisis y diseño resulta en un modelo de diseño y opcionalmente en un modelo de análisis. El modelo de diseño sirve como una abstracción del código fuente; esto es, como si el modelo de diseño actuara como un plano de cómo el código fuente está escrito y estructurado. El modelo de diseño consiste en clases de diseño que se estructuran en paquetes de diseño y subsistemas de diseño con interfaces bien definidas, representando lo que se convertirá en componentes en la implementación. También contiene descripciones de cómo los objetos de estas clases de diseño colaboran para realizar los casos de uso.

Las actividades de diseño se centran alrededor de la noción de arquitectura. La producción y validación de esta arquitectura son el principal objetivo de las primeras iteraciones de diseño. La arquitectura es representada por un número de vistas arquitectónicas. Estas vistas capturan las principales decisiones de diseño estructural. En esencia, las vistas arquitectónicas son abstracciones o simplificaciones del diseño completo, en la cual se hacen más visibles las características importantes, dejando a un lado los detalles. La arquitectura es un vehículo importante no sólo para el desarrollo de un buen modelo de diseño, sino también para incrementar la calidad de cualquier modelo construido durante el desarrollo del sistema.

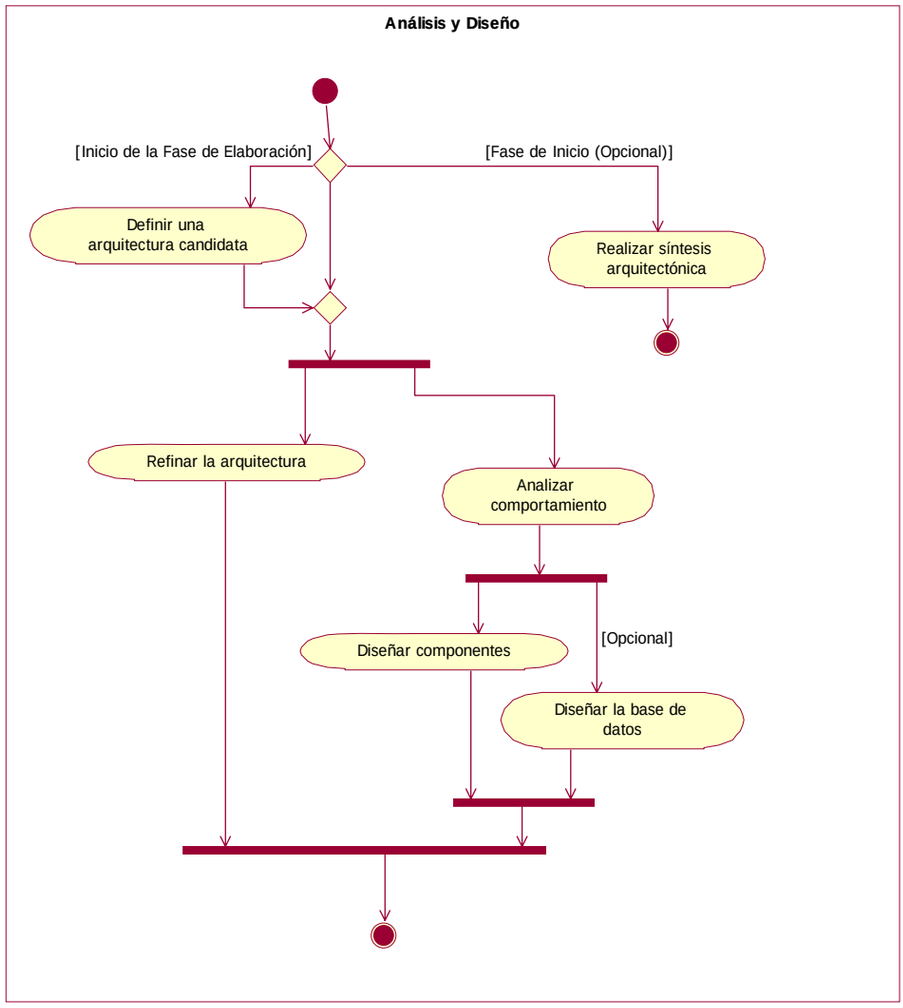


Figura 3.13: Diagrama de Actividad de workflow Análisis y Diseño.

Tabla 3.4: Tareas, actividades y resultados clave workflow de análisis y diseño.

TAREAS	RESULTADO CLAVE
Definir una arquitectura candidata Análisis arquitectónico Análisis de casos de uso	Documento de arquitectura de software Diseño del modelo Realización de casos de uso
Realizar síntesis arquitectónica Análisis arquitectónico Construir prueba de concepto arquitectónico Evaluar la viabilidad de la prueba de concepto arquitectónico	Diseño del modelo Documento de arquitectura de software Realización de caso de uso Lista de riesgo Visión
Analizar comportamiento Análisis de casos de uso Identificar elementos de diseño Revisar el diseño	Diseñar modelo Realizaciones de casos de uso
Diseñar componentes Diseño de casos de uso Diseñar clase Diseñar clases y paquetes de prueba Diseño de subsistema Diseño de cápsula Revisar el diseño Implementar el componente	Diseñar paquete Probar clase Probar especificación de interfaz Revisar registro Cambiar solicitud Cápsula Protocolo Diseñar clase Interfaz Diseñar subsistema
Diseñar la base de datos Diseño de clase Diseño de base de datos Revisar el diseño Implementar componente	Diseñar clase Modelo de datos Componente
Refinar la arquitectura Priorizar los casos de uso Describir la arquitectura de ejecución Describir distribución Identificar mecanismos de diseño Identificar elementos de diseño Incorporar elementos de diseño existentes Revisar la arquitectura Estructurar el modelo de implementación	Lista de riesgo Directivas de diseño Especificaciones adicionales Documento de arquitectura de software Cambiar solicitud Revisar registro

3.4.2.6.4 Implementación

Los objetivos de la implementación son:

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

El sistema se materializa a través de componentes. EL RUP describe cómo reutilizar componentes existentes, o implementar nuevos componentes con responsabilidades bien definidas, haciendo que el sistema sea más fácil de mantener, e incrementando las posibilidades de reutilización.

Los componentes son estructurados dentro de subsistemas de implementación. Estos subsistemas toman la forma de un directorio, con información de estructura y administración adicional. Por ejemplo, un subsistema puede ser creado como un directorio o una carpeta en un sistema de archivos, o packages si se usa JAVA.

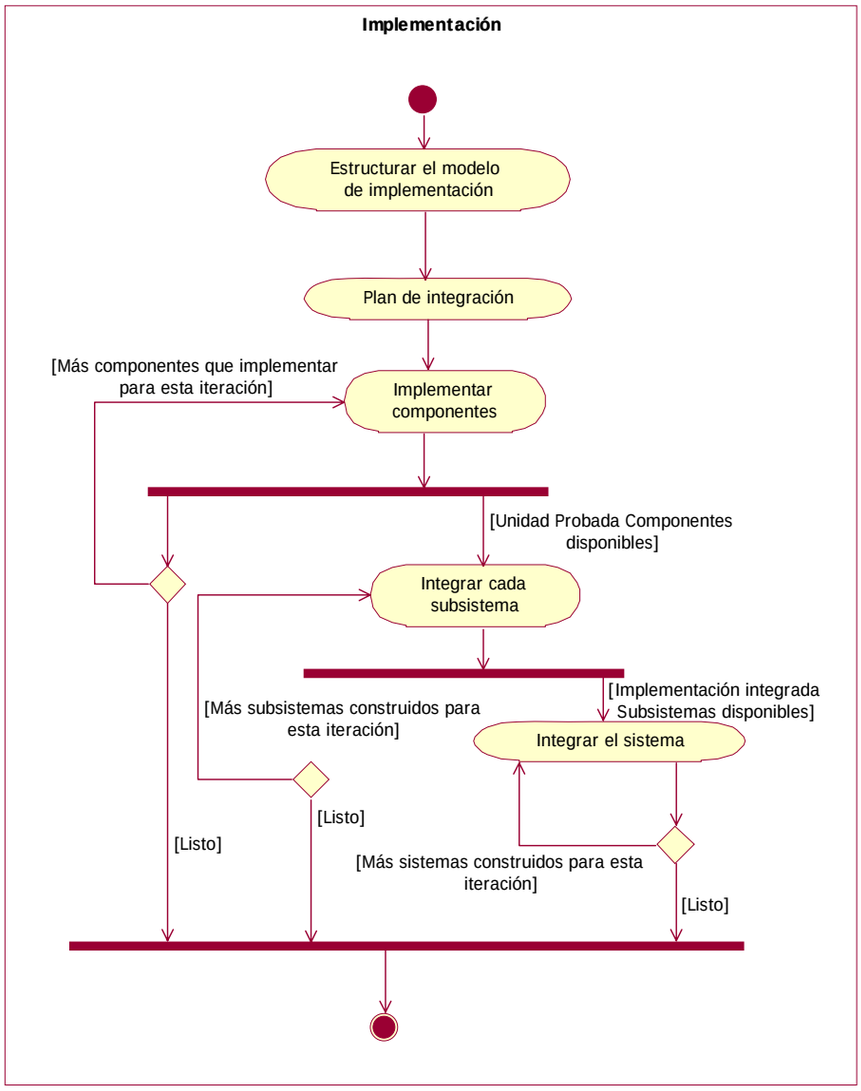


Figura 3.14: Diagrama de Actividad de workflow de Implementación.

Tabla 3.5: Tareas, actividades y resultados clave workflow de implementación.

TAREAS	RESULTADO CLAVE
Estructurar el modelo de implementación Estructurar el modelo de implementación	Subsistema de implementación Documento de arquitectura de software Modelo de implementación
Plan de integración Planificar la integración del sistema	Plan de construcción de integración
Implementar componentes Planificar la integración de subsistemas Reparar un defecto Implementar componentes Revisar código Realizar pruebas de la unidad Perform Unit Tests	Componentes Plan de construcción de integración Probar componente Revisar registro
Integrar cada subsistema Integrar subsistemas	Implementación de subsistema Construir
Integrar el sistema Integrar el sistema	Construir

3.4.2.6.5 Prueba

Los objetivos de la prueba son:

- **Verificar la interacción entre objetos.**
- **Verificar la debida integración de todos los componentes del software.**
- **Verificar que todos los requerimientos han sido correctamente implementados.**
- **Identificar y asegurar que los defectos se han solucionado antes del despliegue del software.**

El RUP propone un método iterativo, lo cual significa que se realizan pruebas a través de todo el proyecto. Esto permite encontrar los defectos tan pronto como sea posible, lo cual reduce los costos de reparación de defectos en forma radical. Las pruebas se realizan a través de tres dimensiones de calidad: confiabilidad, funcionalidad, desempeño de aplicaciones y de sistemas. Para cada una de estas dimensiones de calidad, el proceso describe cómo se realiza el ciclo de vida de prueba de la planificación, diseño, implementación, ejecución y evaluación.

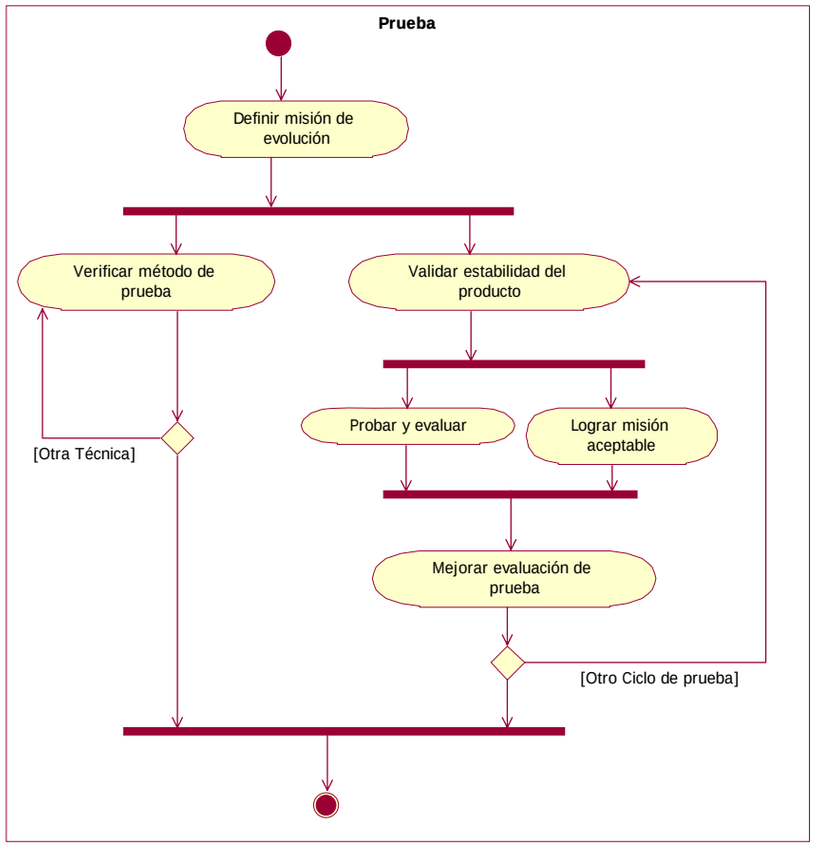


Figura 3.15: Diagrama de Actividad de workflow de Prueba.

Tabla 3.6: Tareas, actividades y resultados clave workflow de prueba.

TAREAS	RESULTADO CLAVE
Definir misión de evolución Identificar motivadores de prueba Identificar objetivos de prueba Identificar Ideas de prueba Definir evaluación y necesidades de rastreabilidad Definir aproximación de prueba Concordar con la misión	Plan de prueba Prueba de arquitectura de automatización Prueba de directivas
Verificar método de prueba Definir configuraciones del entorno de prueba Identificar mecanismos de prueba Definir elementos de prueba Definir detalles de prueba Implementar prueba Implementar comitiva de prueba Obtener compromiso de prueba	Prueba de arquitectura de automatización Prueba de la configuración del entorno Prueba de la especificación de interfaz Prueba de directivas Prueba de script
Validar estabilidad del producto Definir detalles de prueba Ejecutar pruebas Implementar prueba Analizar falla de prueba Determinar los resultados de la prueba Evaluar y asegurar calidad	Resumen de evolución de prueba Resultados de prueba
Probar y evaluar Identificar ideas de prueba Definir detalles de prueba Definir elementos de prueba Implementar prueba Implementar comitiva de prueba Estructurar la implementación de la prueba Ejecutar conjunto de pruebas Analizar falla de prueba Determinar resultados de prueba Evaluar y mejorar los esfuerzos de prueba	Prueba de datos Prueba del resumen de evolución Cambiar solicitudes Resultados de prueba Script de prueba Test Script Comitiva de prueba Probar lista de ideas Precedentes Carga de trabajo de modelo
Lograr misión aceptable Identificar ideas de prueba Implementar prueba Implementar comitiva de prueba Analizar fallas de prueba Determinar resultados de prueba Evaluar y mejorar esfuerzos de prueba Evaluar y asegurar la calidad	Lista de temas Cambio de solicitudes Prueba del resumen de evaluación Resultados de prueba
Mejorar evaluación de prueba Desarrollar directivas de prueba Definir elementos de prueba Estructurar la implementación de prueba Identificar ideas de prueba Definir detalles de prueba Definir evaluación y rastreabilidad de las necesidades Implementar conjunto de prueba Implementar prueba	Directivas de prueba Prueba de Script Prueba del conjunto Prueba de datos Prueba de la configuración del entorno

3.4.2.6.6 Despliegue

El propósito del workflow de despliegue es producir en forma exitosa una versión, y entregar el software a sus usuarios finales. Cubre un gran rango de actividades incluyendo:

Tabla 3.7: Actividades del workflow de despliegue.

Producir versiones externas para el software.
Empaquetar el software.
Distribuir el software.
Instalar el software.
Proveer ayuda y asistencia a los usuarios.

- **En muchos casos, también incluye actividades tales como:**
  - **Planificación y conducción de pruebas betas ("beta test")**
  - **Migración de software o información existente**
  - **Aceptación formal**

Aunque las actividades de despliegue se centran más alrededor de la fase de transición, muchas de las actividades necesitan ser incluidas en fases tempranas para prepararse para el despliegue al final de la fase de construcción.

Los workflows de despliegue y entorno contienen menos detalles que el resto de los workflows del Rational Unified Process.

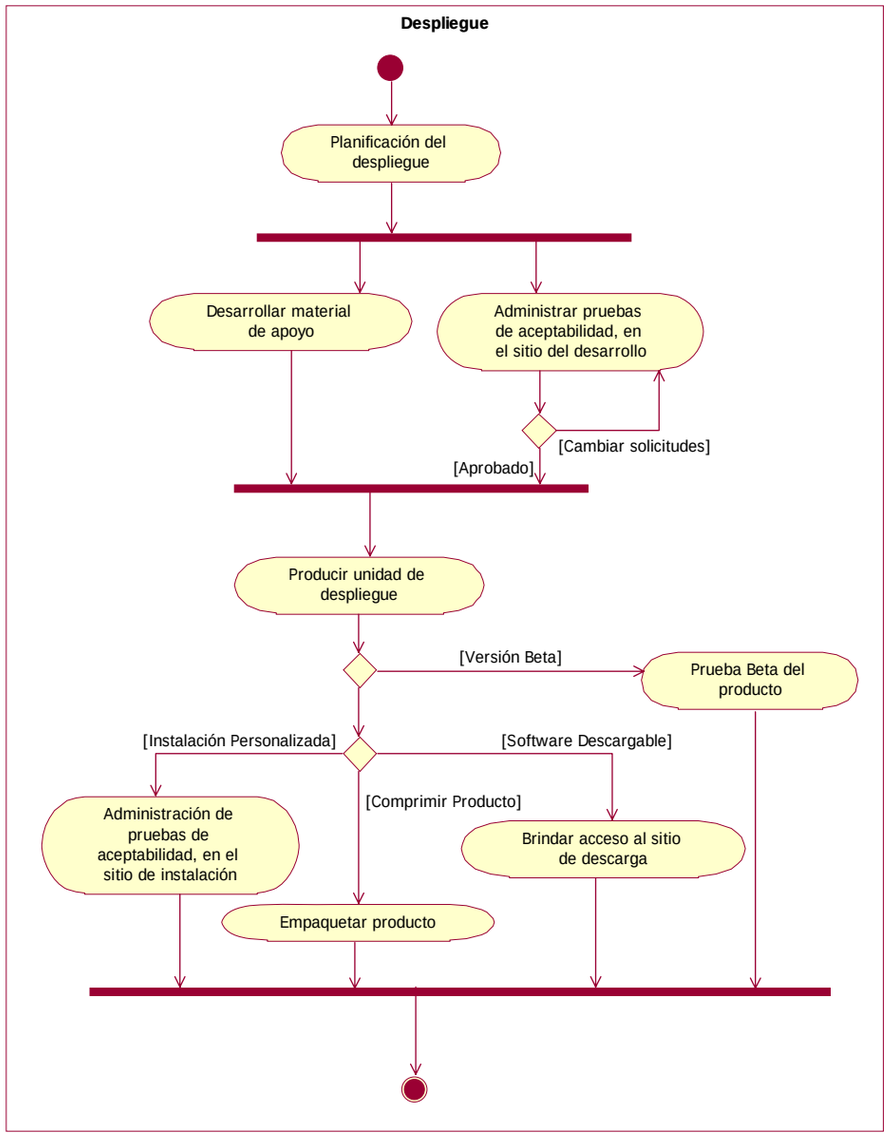


Figura 3.16: Diagrama de Actividad de workflow de Despliegue.

Tabla 3.8: Tareas, actividades y resultados clave workflow de despliegue.

TAREAS	RESULTADO CLAVE
Planificación del despliegue Desarrollar plan de despliegue Definir presupuesto de materiales	Plan de despliegue Presupuesto de materiales
Desarrollar material de apoyo Desarrollo de material de entrenamiento Desarrollo de material de apoyo	Material de entrenamiento Material de apoyo al usuario final
Administrar pruebas de aceptabilidad, en el sitio del desarrollo Administrar pruebas de aceptabilidad	Resultados de prueba Cambio de solicitudes Infraestructura de desarrollo
Producir unidad de despliegue Escribir notas de versión (Release Notes) Desarrollar elementos de instalación	Notas de versión (Release Notes) Elementos de instalación Unidad de despliegue
Prueba Beta del producto Administrar Prueba Beta	Cambio de solicitudes
Brindar acceso al sitio de descarga Brindar acceso al sitio de descarga	Unidad de despliegue
Empaquetar producto Comenzar fabricación Verificar productos fabricados Crear trabajo de arte del producto	Producto Trabajo de arte del producto
Administración de pruebas de aceptabilidad, en el sitio de instalación Administración de prueba de aceptabilidad	Prueba de resultados Cambiar solicitudes Infraestructura de desarrollo

3.4.2.6.7 Administración de configuración y control del cambio

Este workflow describe como controlar los numerosos objetos producidos por mucha gente que trabaja en un proyecto común. El control ayuda a evitar confusiones costosas, y asegura que los objetos resultantes no estén en conflicto debido a alguno de los siguientes problemas:

- **Actualizaciones simultáneas**- Cuando dos o más trabajadores trabajan sobre un mismo objeto, el último que realiza cambios destruye el trabajo del anterior.
- **Notificaciones limitadas**- Cuando un problema es resuelto en un objeto compartido por diversos desarrolladores, y algunos de ellos no son notificados del cambio.

- **Múltiples Versiones**- Los programas más grandes son desarrollados en versiones evolutivas. Una versión podría estar en uso por clientes, mientras que otra puede estar en pruebas y la tercera estar aún en desarrollo. Si se encuentran problemas en cualquiera de estas versiones, los arreglos deben propagarse en todas ellas.

La confusión puede guiar a costosos arreglos y tener que volver a trabajar en cosas ya hechas, a menos que los cambios sean cuidadosamente controlados y monitoreados.

Este workflow provee de directivas para administrar múltiples variantes de sistemas de software que evolucionan, y seguir la pista a las versiones usadas en ciertas construcciones de software, realizando construcciones de programas individuales o versiones completas de acuerdo a las especificaciones definidas por el usuario y reforzando políticas específicas de desarrollo. Se describe cómo se puede administrar el desarrollo en paralelo, desarrollos realizados en múltiples sitios y cómo automatizar el proceso de construcción. Esto es especialmente importante en un proceso iterativo donde probablemente se quiera construir a diario, algo que podría ser imposible sin una fuerte automatización. También se describe cómo se puede mantener un seguimiento de las revisiones sobre el por qué, cuándo y por quién fue modificado un objeto.

También cubre los cambios en la administración de requerimientos, es decir cómo reportar defectos y cómo administrarlos a través del ciclo de vida, cómo usar la información contenida en los defectos para seguir el progreso y las tendencias del proyecto.

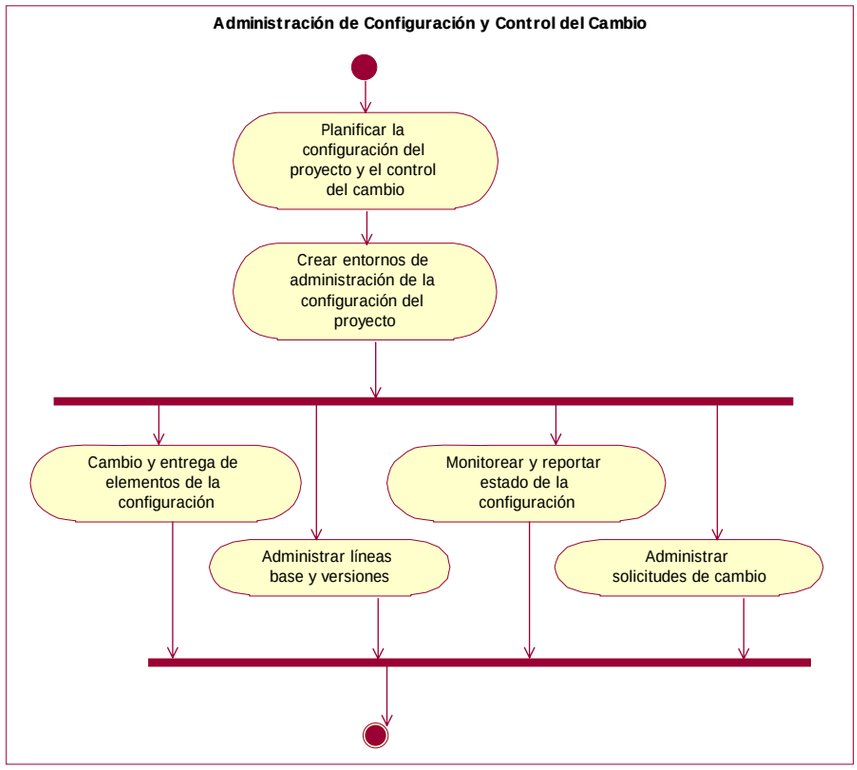


Figura 3.17: Diagrama de Actividad de workflow Administración de configuración y control del cambio

Tabla 3.9: Tareas, actividades y resultados clave workflow de Administración de la configuración y control del cambio.

TAREAS	RESULTADO CLAVE
Planificar la configuración del proyecto y el control del cambio Establecer proceso de control de cambio Establecer políticas de administración de la configuración Escribir plan de administración de la configuración	Plan de administración de la configuración
Crear entornos de administración de la configuración del proyecto Crear espacios de trabajo de integración Establecer entorno de administración de la configuración	Resultados clave Repositorio del proyecto Espacios de trabajo (integración)
Cambio y entrega de elementos de la configuración Crear líneas base Promover líneas base Hacer cambios Crear espacios de trabajo de desarrollo Entregar cambios Actualizar espacio de trabajo	Resultados clave Orden de trabajo (completado) Espacio de trabajo (integración) Espacio de trabajo (Desarrollo)
Administrar líneas base y versiones Crear líneas base (principios) Promover líneas base Crear unidad de despliegue	Resultados clave Repositorio del proyecto Unidad de despliegue
Monitorear y reportar estado de la configuración Realizar auditorias de configuración Reporte en estado de la configuración	Descubrimientos de la auditoría de la configuración Medidas del proyecto
Administrar solicitudes de cambio Confirmar solicitud duplicada o rechazada Revisar solicitud de cambio Someter a consideración solicitud de cambio Actualizar solicitud de cambio Verificar cambios en la construcción	Solicitud de cambio

3.4.2.6.8 Administración de proyectos

La administración de proyectos software es el arte de balancear objetivos que compiten entre sí, administrar riesgo, y superar restricciones para entregar en forma exitosa, un producto que cumpla las necesidades de ambos clientes (los que pagan la cuenta y los usuarios).

El hecho de que muy pocos proyectos son indiscutiblemente exitosos es un antecedente suficiente para demostrar lo difícil de esta tarea.

Este workflow se centra principalmente en los aspectos específicos de un proceso de desarrollo iterativo. La meta en esta sección es hacer la tarea más fácil proveyendo:

- **Una estructura para manejar proyectos intensos de software**
- **Directivas prácticas para la planificación, staffing, ejecución, y monitorear proyectos.**
- **Una estructura para manejar el riesgo.**

Esto no es una receta para el éxito, pero presenta un método para administrar el proyecto, el cual mejorará marcadamente las posibilidades de entregar un software exitoso.

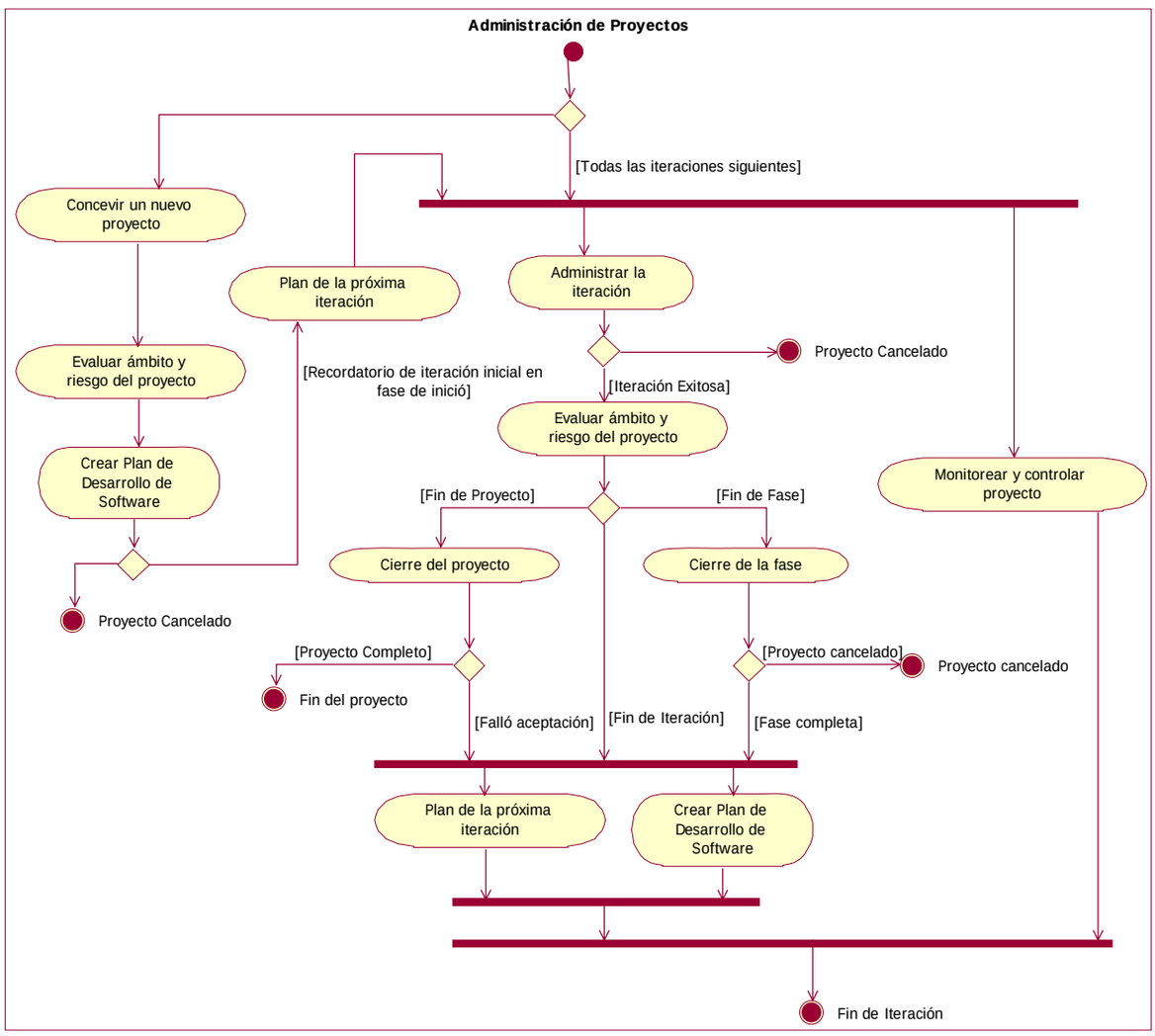


Figura 3.18: Diagrama de Actividad de workflow de Administración de proyectos.

Tabla 3.10: Tareas, actividades y resultados clave workflow de administración de proyectos.

TAREAS	RESULTADO CLAVE
Concebir un nuevo proyecto Identificar y evaluar riesgos Desarrollar caso de negocios Revisión de aprobación de proyecto Iniciar proyecto	Revisar registro Lista de riesgos Caso de negocio Plan de iteración Plan de desarrollo de software (SDP: Software Development Plan)
Evaluar ámbito y riesgo del proyecto Identificar y evaluar riesgos Desarrollar caso de negocio	Lista de riesgo Caso de negocio
Crear Plan de Desarrollo de Software Desarrollar plan de aseguramiento de la calidad Desarrollar plan de medición Desarrollar plan de aceptabilidad del producto Desarrollar plan de resolución de problemas Desarrollar plan de administración de riesgos Definir organización y staff del proyecto Definir procesos de monitoreo y control Planificar fases e iteraciones Compilar plan de desarrollo de software Revisión de la planificación del proyecto	Desarrollo de plan de aseguramiento de la calidad Revisión de registro Plan de medición Mediciones del proyecto Plan de aceptabilidad del producto Plan de resolución de problemas Plan de administración de riesgos Plan de Desarrollo de Software (SDP)
Plan de la próxima iteración Desarrollar plan de iteración Desarrollar caso de negocio Revisión de plan de iteración	Caso de negocio Revisión de registro
Administrar la iteración Adquirir Staff Iniciar iteración Revisión del criterio de evaluación de la iteración Evaluar la iteración Revisión de aceptabilidad de la iteración	Orden de trabajo Solicitud de cambio Evaluación de la iteración Plan de iteración Revisión de registro
Evaluar ámbito y riesgo del proyecto Identificar y evaluar riesgos Desarrollar caso de negocios	Lista de riesgos Caso de negocios
Cierre del proyecto Preparar cierre del proyecto Revisión de aceptabilidad de proyecto	Plan de desarrollo de software (SDP) Evaluación de estado Revisión de registro
Cierre de la fase Preparar cierre de fase Revisión del hito de ciclo de vida	Evaluación de la iteración Plan de desarrollo de software (SDP) Evaluación de estado Revisión de registro Lista de elementos
Monitorear y controlar proyecto Asignar y planificar horas de trabajo Monitorear estado del proyecto	Plan de desarrollo de software (SDP) Mediciones del proyecto

3.4.2.6.9 Entorno

El propósito del workflow de entorno es proveer a la organización desarrolladora de software del entorno de desarrollo de software –tanto procesos como herramientas– que son necesarias para apoyar el desarrollo en equipo. Se centra en las actividades para configurar el proceso en el contexto de un proyecto. También se centra en las actividades para desarrollar las directivas necesarias para apoyar un proyecto. Se provee de un procedimiento paso a paso de cómo implementar un proceso en una organización.

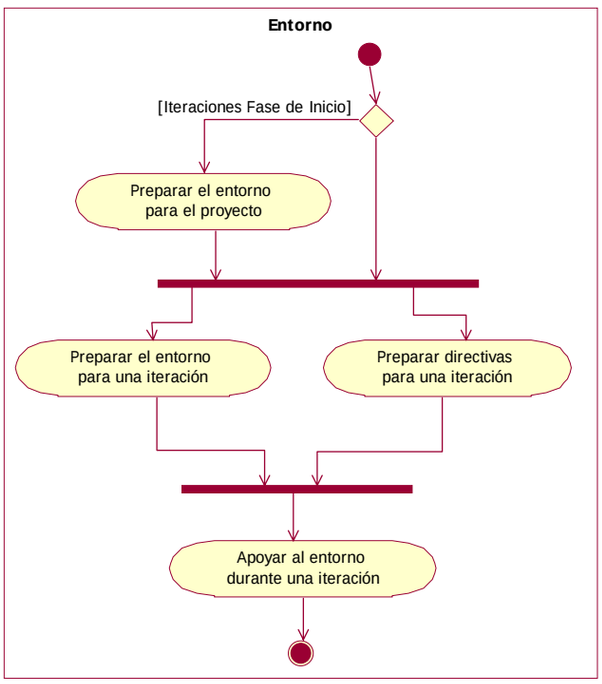


Figura 3.19: Diagrama de Actividad de workflow de Entorno.

Tabla 3.11: Tareas, actividades y resultados clave workflow de administración de entorno.

TAREAS	RESULTADO CLAVE
Preparar el entorno para el proyecto Evaluar la organización actual Desarrollar plantillas específicas para el proyecto Desarrollar caso de desarrollo Seleccionar y adquirir herramientas	Evaluación de desarrollo organizacional Plantillas especificas del proyecto Caso de desarrollo Herramientas
Preparar el entorno para una iteración Desarrollar plantillas específicas del proyecto Desarrollar caso de desarrollo Lanzar caso de desarrollo Configurar herramientas Verificar instalación y configuración de herramientas	Plantillas específicas del proyecto Caso de desarrollo Herramientas
Preparar directivas para una iteración Desarrollar directivas de modelamiento de negocios Desarrollar directivas de diseño Desarrollar estilo de manual Desarrollar directivas de programación Desarrollar directivas de prueba Desarrollar directivas de herramienta Desarrollar directivas de modelamiento de casos de uso Desarrollar directivas de interfaz de usuario	Directivas de modelamiento de negocios Directivas de diseño Estilo de manual Directivas de programación Directivas de prueba Directivas de herramienta Directivas de modelamiento de casos de uso Directivas de interfaz de usuario
Apoyar al entorno durante una iteración Apoyar el entorno Apoyar el desarrollo	Infraestructura de desarrollo

3.5 Conclusión

En el presente capítulo se ha explicado en detalle el proceso de desarrollo de software creado por Rational.

Una de las características importantes de RUP, y que no es fácil de entender a primera vista, consiste en el funcionamiento de su ciclo de vida. Se mostró que cada evolución estaba formada por cuatro fases: inicio, elaboración, construcción y transición. Cada una de éstas es realizada por los nueve workflows de RUP, con intensidades variables dependiendo de la fase en que se encuentre. Además, dichos workflows, recorren las fases en forma paralela.

Capítulo 4 : Análisis de la herramienta Visual Studio .NET

4.1 Visual Studio .NET

Visual Studio .NET es un completo conjunto de herramientas de desarrollo para construir aplicaciones Web en ASP, Servicios Web XML, aplicaciones para computadores personales y aplicaciones para dispositivos móviles. Todos los lenguajes apoyados por la plataforma .NET Framework, tales como: Visual Basic .NET, Visual C++ .NET, Visual C# .NET y Visual J# .NET hacen uso del mismo entorno de desarrollo integrado (IDE [13]), el cual permite compartir herramientas y facilitar la creación de soluciones informáticas que requieran de una mezcla de lenguajes.

El entorno general de Visual Studio .NET está compuesto de barras de botones, menús y ventanas flotantes. En su parte central, muestra la llamada **Página de inicio**, la cual se puede personalizar dependiendo de las necesidades de cada usuario. Por defecto, muestra una lista de los últimos proyectos en los que se ha trabajado, dando la opción al usuario de abrirlos o crear un proyecto nuevo. En esta lista de proyectos recientes, aparecerán los últimos proyectos con los que se haya trabajado, sin importar si estos son aplicaciones de ventanas, servicios Web, o componentes.

Visual Studio .NET cuenta con múltiples editores que se pueden usar en los proyectos sin importar el lenguaje de programación empleado. Se tienen editores de código con realzado sintáctico y la tecnología IntelliSence que permite desplegar ventanas flotantes que muestran listas de parámetros, métodos y propiedades; ayudando a los desarrolladores a autocompletar las líneas de código. Esta técnica también es válida cuando se trabaja en el desarrollo de páginas ASP+ o archivos XML.

[13] **IDE:** Entorno de desarrollo integrado, del inglés Integrated Development Environment

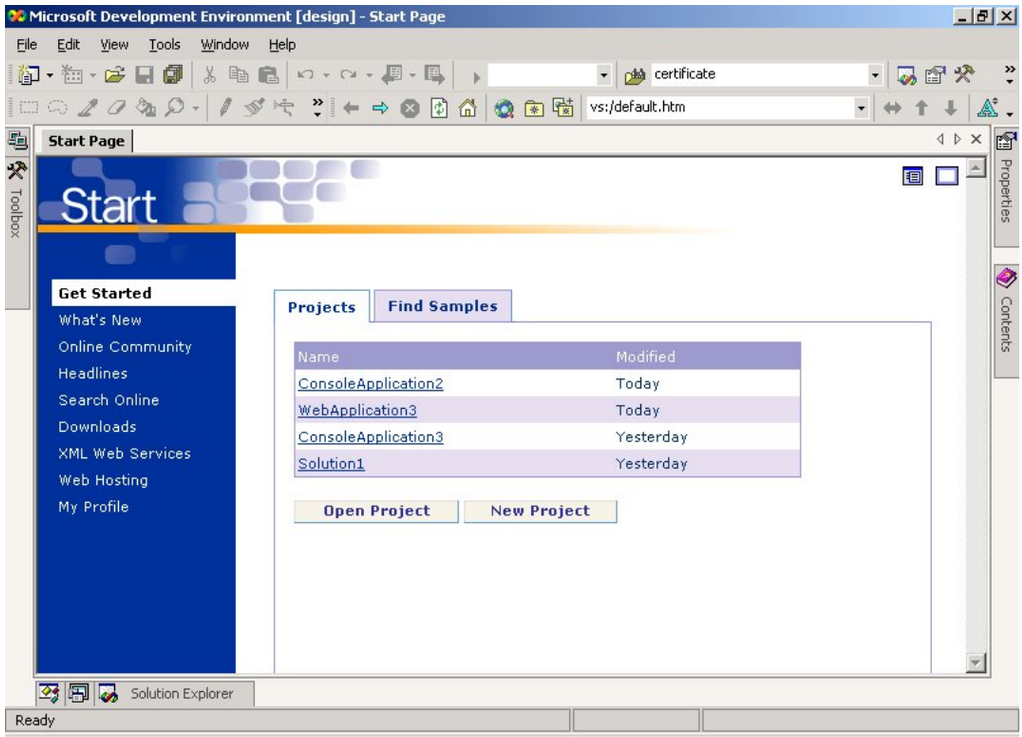


Figura 4.1: Entorno de desarrollo Visual Studio .Net y su página de inicio.

Algunos editores tales como los de HTML y XML permiten contar con vistas alternativas. En el caso de HTML, por ejemplo, se dispone de una vista de código y una vista de diseño de la página que se está desarrollando. Para el caso de los archivos XML, se presentan mediante las vistas de código o como si fueran una tabla de datos.

Además, los editores de Visual Studio .NET cuentan con múltiples diseñadores que se utilizan al desarrollar interfaces gráficas de usuario, ya sea para aplicaciones de ventanas o aplicaciones Web. Estos diseñadores cuentan con elementos tales como la caja de herramientas (ToolBox) y componentes, o la ventana de propiedades y eventos, las que aparecen a ambos lados de la ventana central.

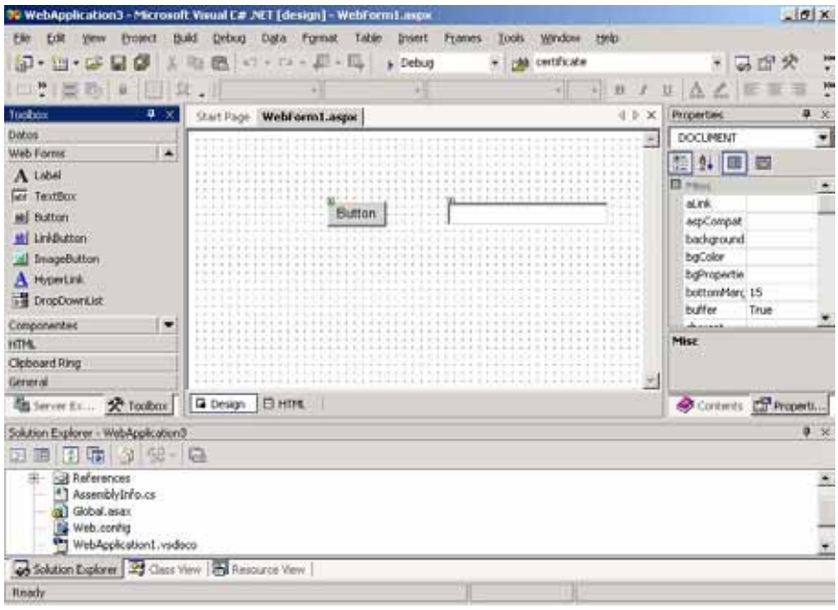


Figura 4.2: Entorno de desarrollo Visual Studio .Net.

En la parte inferior de la pantalla se encuentran las ventanas del Explorador de Soluciones (Solution Explorer) y Vista de clase (Class view). Más adelante se verá que al integrar Rational XDE Professional a Visual Studio .NET, tanto la caja de herramientas como las ventanas de la parte inferior se verán completadas con elementos adicionales. Es así como en la caja de herramientas se dispondrá de todos los elementos del modelo que participen en un tipo de diagrama UML determinado. En la parte inferior se agregarán las ventanas del Explorador de modelos (Model Explorer) y la ventana de documentación del modelo (Model documentation).

Las diversas ventanas del entorno Visual Studio .NET se pueden mantener fijas, siempre visibles, auto-ocultables o bien flotantes. Esto facilita el trabajo al permitir mantener siempre despejada el área del desarrollador.

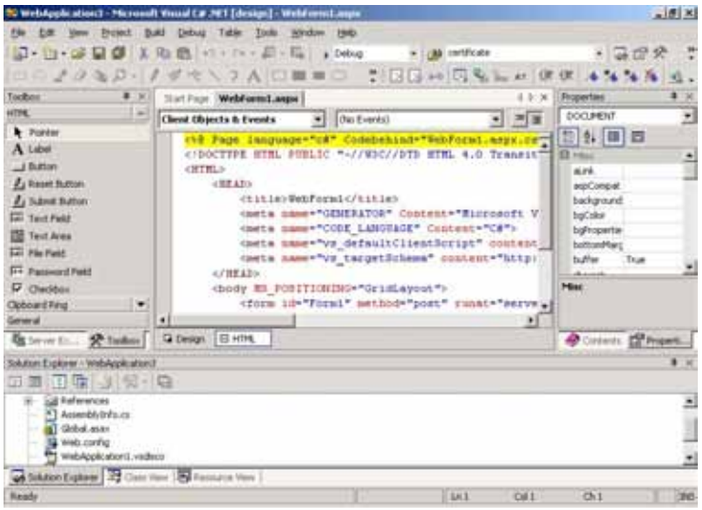


Figura 4.3: Entorno de desarrollo Visual Studio .Net, ventana de código.

4.1.1 Creación de proyectos

Para editar código fuente o diseñar las interfaces de usuario, primero es necesario iniciar un nuevo proyecto y agregar los elementos que se necesiten. La creación de nuevos proyectos, formularios, páginas ASP+, etc., se efectúa en forma muy similar a como se realiza la creación de un nuevo proyecto en Visual Basic 6.0. La ventana de nuevo proyecto está dividida en dos partes, la parte izquierda muestra una lista de carpetas que contienen todos los posibles proyectos y asistentes clasificados por lenguajes u objetivos, mientras que en la parte derecha aparecen los elementos existentes en la carpeta seleccionada.

Figura 4.4: Ventana de creación de un nuevo proyecto.

En la figura se muestran carpetas con proyectos Visual Basic, Visual C#, Visual C++, etc. También se cuenta con carpetas que contienen proyectos para el empaquetamiento y distribución de las aplicaciones.

Por ejemplo, al seleccionar la carpeta **Visual C# Projects**, en la parte derecha se muestran los siguientes elementos:

Tabla 4.1: Tipos de aplicaciones para un proyecto C#.

Templates	Descripción
Windows Application	Aplicación de ventanas tradicional o interfaz para una aplicación Web distribuida.
Class Library	Clase o componente reutilizable que puede ser compartida con otros proyectos. Este tipo de proyecto se considera sin ventana y no contiene una clase Windows Form.
Windows Control Library	Control personalizado para ser usado en Windows Forms.
ASP.NET Web Application	Aplicación Web ASP .NET programable.
ASP.NET Web Service	Servicio Web XML realizado con ASP.NET, funcionalidad que puede ser publicada y llamada por aplicaciones externas.
Web Control Library	Control personalizado que puede ser usado en páginas Web Forms.
Console Application	Aplicación de línea de comandos.
Windows Service	Aplicaciones de larga ejecución que no tienen una interfaz gráfica de usuario. Aplicaciones de Servicios Windows (a veces llamadas “Servicios NT”), pueden monitorear eventos tales como el rendimiento del sistema.
Empty Project	Un proyecto vacío. El molde crea la estructura de archivos necesaria para almacenar la información de la aplicación; cualquier referencia, archivo y componente debe ser agregado en forma manual.
Empty Web Project	Para usuarios avanzados que desean partir programando en un entorno vacío y agregar sus propias funcionalidades basadas en servidor.
New Project In Existing Folder	Proyecto en blanco dentro de una carpeta de aplicación existente, para usar archivos de un proyecto pre-existente. Este molde puede ser usado tanto por proyectos locales como por proyecto Web.

Después de iniciar el proyecto, el cual contará con un formulario de ventana, formulario Web o el elemento que corresponda dependiendo de su naturaleza, se pueden añadir nuevos elementos a éste seleccionando en la barra de menú **File | Add New Item**. En la ventana de agregar nuevo ítem, que tiene una estructura similar a la ventana para crear un nuevo proyecto, en la parte izquierda se muestran carpetas que listan las categorías de los nuevos elementos a agregar. Mientras que en la parte derecha se despliegan los elementos pertenecientes a la categoría seleccionada.

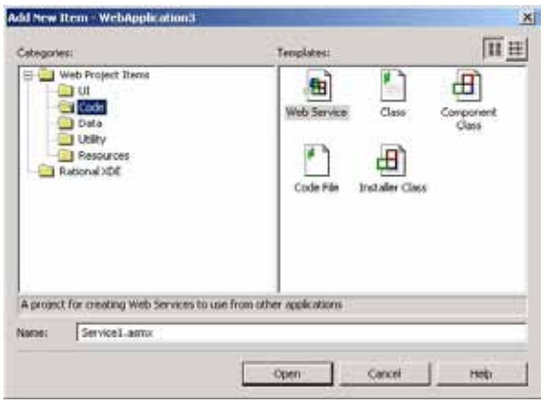


Figura 4.5: Ventana para agregar un nuevo item al proyecto.

Los elementos que forman parte de un proyecto, pueden ser gestionados a través del Explorador de soluciones (Solution Explorer). En él, es posible abrir, configurar y acceder a las propiedades de cada archivo de código, formulario, componente, etc.

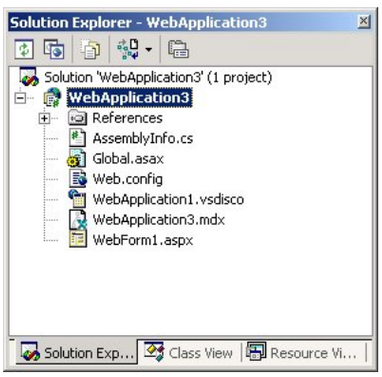


Figura 4.6: Explorador de soluciones (Solution Explorer)

También es posible distinguir la etiqueta **Class View**, donde se despliega un árbol de clases y métodos que participan en la aplicación.

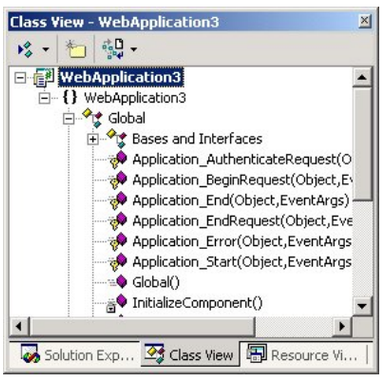


Figura 4.7: Vista de clase (Class View)

Visual Studio .NET cuenta con otros elementos que son comunes para los distintos compiladores que posee. El más importante de estos elementos es el depurador integrado, que puede utilizarse para depurar un proyecto formado por módulos escritos en distintos lenguajes. Cuenta con ventanas para ver y editar puntos de parada de ejecución (Break Points), inspeccionar variables y evaluar expresiones. Así, como una ventana donde podemos ejecutar comandos sobre Visual Studio, como por ejemplo, introducir una sentencia para modificar el valor de una variable durante la ejecución.

Es posible acceder a cualquier dirección Web sin necesidad de salir de Visual Studio .NET. **Dicha característica se puede explotar integrando a la herramienta una Intranet de desarrollo o producción que tendría como objetivo incorporar el proceso de desarrollo,** basado en el proceso RUP descrito en el Capítulo 3, **a la metodología de desarrollo de aplicaciones bajo la plataforma .NET. Para el caso de equipos de desarrollo geográficamente dispersos se puede implementar una Extranet de producción.**

Otro elemento común es la ventana de ayuda dinámica (Dynamic Help), despliega ayuda en el momento y contexto que se requiera. Permite ir explorando los métodos y propiedades de cada clase.

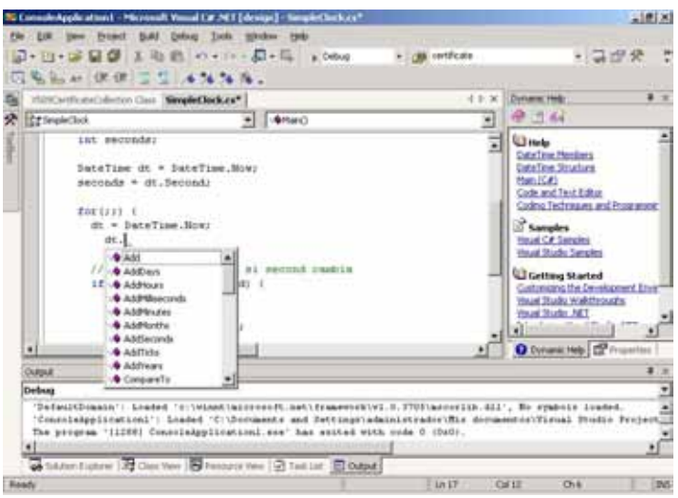


Figura 4.8: Ventana de ayuda dinámica (Dynamic help).

Una vez que se ha concluido un proyecto, se pueden crear distintas soluciones de distribución e instalación, dependiendo si la aplicación es de tipo Web o de ventanas, y según se requiera crear un proyecto para instalación.

Para iniciar un proyecto de distribución e instalación, se debe seleccionar la carpeta: **Setup and Deployment Projects**. Se puede elegir un proyecto vacío, en el cual se seleccionan, en forma manual los archivos a incluir, los archivos a copiar y los accesos directos a crear, o bien, se puede usar un asistente que se encargue de todas estas tareas.

Figura 4.9: Ventana para iniciar un nuevo proyecto de distribución.

Al igual que Java, al momento de instalar la aplicación empaquetada en los archivos de distribución, en el equipo de destino, éste debe contar con la plataforma .NET previamente instalada. De otra forma la aplicación no podrá ser ejecutada. Microsoft autoriza la redistribución de su plataforma de ejecución .NET, la que se puede anexar a los paquetes de instalación de la aplicación. El archivo que la contiene se llama DotNetfx.exe, el cual se instala automáticamente al descomprimir el archivo DOTNETREDIST.EXE.

4.1.2 C#

Microsoft C# (pronunciado C Sharp) es un nuevo lenguaje de programación diseñado para construir una amplia gama de aplicaciones empresariales que se ejecutan en .NET Framework.

C# es una evolución de los lenguajes de programación C y C++. Es un lenguaje simple, moderno, seguro y orientado a objeto. El código C# es compilado como **managed code**, lo cual significa que se beneficia de los servicios de CLR (common language runtime). Estos servicios incluyen interoperabilidad entre lenguajes, recolección de basura, seguridad mejorada y un control de versiones mejorado.

C# se presenta como Visual C# en Visual Studio .NET. El apoyo para Visual C# incluye moldes de proyectos (model templates), diseñadores, páginas de propiedades, code wizards, un object model, y otras características del entorno de desarrollo. La librería para la programación en Visual C# es la plataforma .NET Framework, ver **Anexo 2**.

- C# provee acceso a los estilos de API comunes: Las APIs .NET Framework, COM, Automation, y C-style. También apoya el modo no seguro, donde se pueden usar punteros para manipular la memoria que no está bajo el control del recolector de basura.

4.1.3 Web Forms

Las Web Forms son una tecnología ASP.NET que se usa para crear páginas Web programables. Las Web Forms se transforman ellas mismas a código HTML y script compatible con los navegadores de Internet, por lo que cualquier Browser sobre cualquier plataforma puede ver dichas páginas. Para crear las Web Forms, se diseñan páginas Web haciendo uso de la técnica **Drag&Drop**, es decir, arrastrando y soltando controles desde la caja de herramientas a la ventana de diseño, luego se agrega código, de una forma similar a como se efectúa la creación de aplicaciones en Visual Basic 6.

4.1.4 Windows Forms

Las Windows Forms (o formularios de ventanas), corresponden a la nueva plataforma para el desarrollo de aplicaciones basadas en ventana, que a su vez se basa en la plataforma .NET Framework. Esta infraestructura provee de un amplio conjunto de clases limpio y orientado a objeto que permite desarrollar dichas aplicaciones.

4.1.5 XML Web Services

Los servicios Web XML son aplicaciones que pueden recibir solicitudes e información haciendo uso de XML sobre HTTP. Los servicios Web XML no están sujetos a una tecnología de componentes en particular, o a una convención de llamada o invocación a objetos, por ende pueden ser llamados por cualquier lenguaje, modelo de componentes, o sistema operativo. En

Visual Studio .NET, se pueden crear e incluir servicios Web XML haciendo uso de Visual Basic, Visual C#, Jscript, las Managed Extensions para C++, o ATL server.

XML Support

El **Extensible Markup Language (XML)** provee de un método para describir información estructurada. XML es un subconjunto de **SGML**[14], el cual está optimizado para entregas sobre la Web. El World Wide Web Consortium (W3C) define el estándar XML por lo que la información estructurada será uniforme e independiente de aplicaciones. Visual Studio .NET apoya completamente a XML, proveyendo del diseñador de XML (XML Designer) para hacer más simple la edición de XML, y la creación de esquemas XML (XML schemas).

4.2 Aplicaciones Windows

Al desarrollar aplicaciones de ventanas para un sistema operativo determinado, se presuponía utilizar un conjunto de funciones de bajo nivel conocidas como API (Application Programming Interface), que en el caso de Windows, se llaman Win32 y están compuestas por funciones tales como RegisterClass(), CreateWindow(), SendMessage(), etc., mediante las cuales se registran clases, se crean ventanas y envían mensajes. La plataforma .NET Framework nos brinda una capa de abstracción que nos permite no tener que acceder a la API. Contando con un conjunto de clases de más alto nivel. Por lo que para crear una aplicación, no se tendrán que usar funciones aisladas, ni entregar **handles** [15] como parámetros, para operar sobre ventanas, ni enviar mensajes para conseguir acciones, ni establecer funciones **callbacks** para recibir notificaciones.

Los formularios de ventanas, o WinForms, están construidos en lenguaje C#, y definen clases que cuentan con propiedades, métodos y eventos. En la plataforma .NET una ventana es un objeto, y por lo tanto, se pueden usar sus propiedades para cambiar sus características, usar sus métodos para realizar diversas operaciones y responder a los eventos que genera.

Así como cualquier otra aplicación escrita en C#, una aplicación que use formularios de ventanas, tendrá que contar con un punto de entrada que es el método estático **Main()**. Una

[14] **SGML:** Standard Generalized Mark-up Language

aplicación de ventanas usa el espacio de nombres (namespace) **System.Windows.Forms** en el que se definen clases como **Application**, **Form** o **Button**.

La clase **Application**, definida en el espacio de nombres **System.Windows.Forms**, está compuesta de propiedades y métodos estáticos. Esto significa que podemos usarlos sin necesidad de crear un objeto de la clase, sólo basta con poner el nombre de la clase y a continuación el nombre de la propiedad o método a usar. Por ejemplo: **Application.Exit();** que cierra la aplicación.

Por último, hay que enfatizar el hecho de que para poder ejecutar una aplicación basada en Ventanas (o Winforms), es necesario tener instalada la plataforma común de ejecución en el equipo de destino. La CLR (Common Language Runtime) se a portado a diversos sistemas operativos incluyendo todas las versiones de Windows y algunas distribuciones de Linux. La comunidad Linux actualmente cuenta con proyectos en curso para mapear la plataforma .NET, destacando las versiones de CLR y los compiladores de C# que han ido evolucionando.

4.3 Aplicaciones Web

Las aplicaciones Web nacen de la necesidad de brindar servicios mediante aplicaciones que serán ejecutadas en máquinas que están más allá del alcance y control de los desarrolladores. Tal es el caso cuando no es posible solicitar a un cliente que instale en sus equipos una versión del CLR. En casos como éste, lo más recomendable es desarrollar una aplicación Web. Este tipo de aplicaciones se basa en el diseño de páginas Web o WebForms. Así como ocurre en el caso de las aplicaciones de ventana (o WinForms) es posible desarrollar aplicaciones Web en cualquiera de los lenguajes que son apoyados por la plataforma .NET. Además, se cuenta con la gran ventaja de poder separar lo que es el diseño de la página Web e interfaces, de lo que es la lógica de negocios. Todo esto es lo que engloba ASP+.

Las páginas ASP significaron una revolución en el diseño de páginas Web interactivas. Una página Web contiene el diseño visual, generalmente en HTML, y el código encargado de

---

[15] **Handles:** También conocidos como manejadores, son señales de control que notifican eventos y operaciones.

procesar la información, y generar dinámicamente el contenido, generalmente escrito en JScript o VBScript. La página ASP es procesada en el servidor (IIS, Internet Information Server) que ejecuta el código y combina la salida de éste con el código HTML de la página original. Así se genera un nuevo documento que es creado dinámicamente y es enviado al cliente. Las ventajas de ASP son muchas, facilitan la generación dinámica de contenido, por ejemplo, a partir de información almacenada en una base de datos, sin necesidad de escribir una CGI (Common Gateway Interface), sólo bastaría algo de código script para obtener un resultado rápidamente. Pero no todo es tan bueno, ni tan bonito. ASP también tiene debilidades. Una página ASP por lo general es una mezcla de HTML con contenido, diseños, estilos, código script en JScript, VBScript o ambos. Esto resulta en **diferentes lenguajes y sintaxis mezcladas en un mismo documento**, lo que constituye algo difícil de mantener. Otra de las debilidades, es que el código script que contiene una página ASP, es interpretado cada vez que se requiere ejecutarlo. Es posible ejecutar controles ActiveX desde una página ASP, pero antes es necesario instalarlos y registrarlos en el sistema. Si por ejemplo, se estuviera trabajando en el mantenimiento del contenido de un servidor Web remoto, el registro de dichos componentes presupone un serio problema. Todos estos problemas se terminan con el desarrollo en ASP+.

La independencia de lenguajes, permite poder desarrollar los scripts de una página ASP+ en cualquier lenguaje que sea apoyado por la plataforma .NET. Es por esto, que se puede trabajar en C# o Visual Basic, además de JScript y VBScript. Este último ya casi extinto debido a que Visual Basic es un súper conjunto de aquel lenguaje.

En general, la disposición de las herramientas del entorno de desarrollo integrado es la misma que para las aplicaciones basadas en ventanas, por lo que siempre se contará con una caja de herramientas (ToolBox), que en este caso contendrá controles para el diseño de la interfaz Web. Además, la ventana de propiedades despliega las propiedades del control específico seleccionado, así como los eventos posibles a los que está sujeto dicho control. Por otro lado, al seleccionar más de un elemento, en la ventana de propiedades se desplegarán sólo los eventos o propiedades comunes para aquellos elementos.

Figura 4.10: Ventana de propiedades.

La siguiente figura muestra algunos de los tipos de herramientas presentes en la caja de herramientas (ToolBox).

Figura 4.11: Caja de herramientas (ToolBox) .

Si no se contara con Visual Studio .NET, se podría configurar el sistema en forma manual, creando un nuevo directorio virtual que contendrá la nueva aplicación Web.

Con respecto al rendimiento de un código script que es insertado en una página ASP, éste no es el mejor, debido a que es analizado e interpretado cada vez que un cliente solicita el documento, por lo que el tiempo que lleva generar una respuesta final no es óptimo. Muy por el contrario, en ASP+, este código es compilado la primera vez que se solicita. Luego, es almacenado temporalmente en un espacio cache, por lo que cuando un nuevo cliente solicita el mismo código, éste es ejecutado directamente. Así se eliminan los tiempos de análisis, interpretación y compilación en las solicitudes posteriores.

El código compilado es un archivo con extensión DLL, y queda en una carpeta llamada **bin**, dentro del directorio de la aplicación Web.

En ASP, incluir el código HTML de diseño y el código script de lógica de negocio, en un mismo archivo, no es lo más recomendable, debido a que se obtiene un documento final con una mezcla heterogénea de lenguajes. La mejor estrategia es almacenar en forma separada la lógica de negocio, por ejemplo en forma de componente. Así las modificaciones de diseño no afectarán al código ejecutable y viceversa.

En ASP, la forma de lograr esto es a través de componentes ActiveX que luego se insertan en el código HTML. Pero esto no es una tarea fácil, ya que crear componentes ActiveX puede ser una tarea muy compleja dependiendo del lenguaje de implementación que se utilice.

Cuando al fin se ha creado el componente ActiveX, aparecen los otros problemas, ya que no basta sólo con copiar el componente en el servidor Web, sino que también es necesario registrarlo en el registro de Windows. Esta tarea se realiza a menudo ejecutando el programa **regsrv32** y poniendo como parámetro de entrada el nombre del archivo DLL que contiene al componente. Esta tarea debe hacerse en forma local, por lo que tratar de configurar el componente en un equipo remoto puede ser complicado.

El otro problema que surge es que el servidor Web bloquea las librerías mientras las mantiene en uso. Por lo que si se desea actualizar la componente ActiveX rescribiendo la DLL, será necesario **DETENER** el servidor Web para desbloquear la DLL, luego copiar la DLL actualizada para rescribir la antigua, y luego volver a poner en marcha el servidor Web. Este proceso puede parecer muy simple para usuarios que mantienen servidores Web locales o de bajo tráfico, sin embargo en un ambiente de operaciones, en que el tráfico es alto y a veces muy alto, este simple proceso podría dejar fuera a muchos usuarios del sistema, interrumpir transacciones, causar pérdida de información importante, etc.

Al utilizar la tecnología .NET, y específicamente ASP+, los problemas anteriormente mencionados desaparecen. Los componentes son desarrollados en C# o Visual Basic (o cualquier otro lenguaje apoyado por la plataforma .NET). Una componente en .NET se realiza definiendo una clase al igual que cualquier programa orientado a esta plataforma y no es necesario usar COM para crear un ActiveX. Por lo tanto al crear una clase, ya se está creando un

componente. Además estos componentes no necesitan ser registrados, basta con copiarlos en una carpeta del servidor.

En ASP+, el código se genera dinámicamente, y se almacena temporalmente en un archivo DLL.

La información que es enviada a los clientes es sólo código HTML, sin ningún control encapsulado. Esto último permite acceder a las aplicaciones Web desde cualquier navegador de Internet.

4.3.1 Formularios Web

La creación de formularios Web se realiza en forma análoga a la creación de los WinForms vistos anteriormente, haciendo uso de la técnica **Drag&Drop**. En una aplicación para Windows, todo el código se almacena en un archivo con extensión **cs** (en el caso de C#), en cambio, en una aplicación WebForm se utilizan dos archivos: uno con extensión **aspx**, que contiene el diseño de la interfaz gráfica de usuario, y otro para almacenar el código (**cs** o **vb**, según corresponda). En forma adicional, puede existir un archivo de configuración de la aplicación Web.

La conexión entre el código **aspx** y el código C# se puede establecer de diversas formas, pero las dos formas más usuales son: derivar la página ASP+ desde una clase definida en C#, o utilizar desde la página en ASP+ objetos escritos en C#.

4.3.2 Uso de componentes C# desde ASP+

Anteriormente se veían los problemas relacionados con el uso de componentes ActiveX desde páginas ASP. Además se mostraba la solución que plantea el nuevo modelo de ASP+.

La configuración por defecto de ASP+, hace que cuando se ingresa a una aplicación, los archivos presentes en la carpeta **bin** sean guardados en memoria. Esta característica se puede cambiar, modificando los archivos de configuración **config.web**.

4.3.3 Compilación dinámica

Una alternativa a la compilación directa del código es la **compilación dinámica**, que sirve si el sitio de la aplicación web está hospedado en un servidor de la empresa, por lo que se tiene control y otras personas ajenas a la empresa no tienen acceso a él. Esta compilación consiste en dejar que el motor de ASP+ tome el código fuente y lo compile cuando lo necesite. Esto se realiza insertando la siguiente línea al principio del documento ASP+: **<%@ Page Src="prueba.cs" %>**. Así se le indica a ASP+ que importe un archivo C# con código fuente.

Los **componentes del servidor** son una de las novedades más interesantes de ASP+. Al crear formularios HTML, se utilizan controles tales como: cajas de texto (textbox) y botones (buttons), con capacidades limitadas, pero que pueden usarse desde cualquier navegador de Internet, ya sea Internet Explorer, Netscape Navegator, o cualquier otro con capacidades básicas HTML.

En ASP+ se introducen los controles de servidor, que son un gran conjunto de componentes que se pueden usar en la elaboración de cualquier documento ASP+. Cuando un cliente solicita la página estos generan automáticamente código HTML. Al igual que los componentes WinForms vistos anteriormente, los controles Web tienen propiedades, métodos y eventos. El espacio de nombres **System.Web.UI.WebControls** contiene todas las clases de componentes que se pueden usar en una página ASP+.

El código en C# se ejecuta en el servidor, y el cliente sólo recibe código HTML.

4.3.4 Conexión entre diseño y lógica.

Al utilizar los controles de servidor, no es necesario dejar el código perteneciente a la lógica de la aplicación junto al código HTML en un mismo documento ASP+. Existe la posibilidad de separarlo en un módulo independiente que se compilará automáticamente, o en una librería precompilada. **Es necesario establecer una correspondencia bi-direccional entre los controles del servidor usados en la página ASP+ y los objetos que se creen en el código C#.** Cuando se trabaja con Visual Studio .NET, es esta herramienta la que se encarga de ir

construyendo las clases y sus métodos a medida que se van agregando componentes al documento ASP+.

Lo primero que se hace es declarar una clase que sea derivada de la clase **Page**. Esta clase, que está definida en el namespace **Systema.Web.UI;** es la clase base para todas las páginas ASP+. Lo que realmente se está haciendo es definir los elementos de la página en lenguaje C#.

4.3.5 Validación de datos

Los navegadores Web que se utilizan actualmente, se ejecutan sobre sistemas que tienen una cierta capacidad de procesamiento. Por lo que ciertas tareas que se realizan con ASP+ en el servidor, podrían realizarse en el cliente, lográndose con esto un mejor aprovechamiento del ancho de banda y de la capacidad de procesamiento del servidor.

Cuando se utiliza un formulario Web para solicitar datos al usuario, existe una tarea fundamental que debe considerarse, la validación de los datos de usuario. Esta validación, podría realizarse en el servidor, que al recibir los datos, los comprobaría y notificaría al usuario cualquier error. El problema de este esquema es que cada corrección del usuario debe viajar hasta el servidor, y cada notificación de éste debe volver de nuevo al cliente. Si los errores son varios, el proceso puede ser lento.

La alternativa es que este proceso de validación se realice por completo en el cliente, y que la información se envíe al servidor sólo cuando se encuentre correcta y completa. Para realizar esto, se podría escribir código C# que estuviera encapsulado en el documento ASP+, de esta forma el código se ejecutaría en el cliente y no en el servidor.

En ASP+, existe un conjunto de componentes de servidor que se encarga de generar automáticamente el código necesario para realizar la validación. Estos componentes no son visibles y por lo general se ponen al final del formulario, y se asocian con los controles de entrada de datos.

Existen cuatro tipos de validaciones distintas, las que se describen en la siguiente tabla:

Tabla 4.2: Tipos de validaciones.

RequiredFieldValidator	Se utiliza para asegurarse que un campo no quede en blanco al enviar el formulario.
RangeValidator	Permite controlar que una entrada se encuentre entre un rango de valores.
CompareValidator	Permite controlar, a través de un operador relacional, que una entrada sea mayor, menor, igual o distinta que alguna propiedad o algún otro valor.
RegularExpressionValidator	Se utiliza para verificar que la entrada de un dato se ajuste a cierta plantilla o expresión

Además de estas cuatro validaciones, se puede agregar una quinta, que es más flexible que las anteriores, ya que permite asociar un método a la comprobación de datos, su nombre es **CustomValidator**.

4.3.6 Pagelets

Cuando se diseña un sitio Web que tiene muchas páginas, por lo general estas tienen componentes que se repiten en distintos lugares y con bastante frecuencia. En vez de introducir el mismo código HTML cada vez que se necesite uno de estos elementos, es posible convertir con facilidad dichos fragmentos de código en componentes. El proceso es muy simple, y el resultado es que se contará con un componente más que se podrá usar en ASP+, y que ha sido definido por el desarrollador haciendo uso sólo de HTML, y si es necesario, algún otro componente del servidor Web.

4.4 Servicios Web

La mayoría de los clientes actuales que existen en Internet, ya sean usuarios físicos o aplicaciones de terceros, son consumidores de servicios. Uno de los principales objetivos de la plataforma .NET es lograr que la red actual, compuesta en su mayor parte de páginas Web y archivos, se convierta en una red de servicios.

Un servicio Web, es un servicio, que una determinada empresa pone a disposición de los usuarios, que puede estar disponible como servicio interno o accesible para toda la Internet.

Para crear un servicio Web, se puede usar cualquiera de los lenguajes apoyados por la plataforma .NET, además, los consumidores pueden encontrarse en cualquier lugar, utilizando cualquier hardware y sistema operativo. Un servicio Web puede ser ofrecido o consumido desde

cualquier parte, sólo se requiere contar con el protocolo adecuado, por lo que no es necesario contar con Windows, ni con la plataforma .NET, ni con el lenguaje C#.

Los consumidores pueden usar una simple solicitud HTTP, desde un cliente Web, o bien utilizar un protocolo RPC sobre XML conocido como SOAP (Simple Object Access Protocol).

4.4.1 Servicios y Aplicaciones

Una aplicación es un servicio o un conjunto de servicios instalados en un computador, Microsoft Word por ejemplo, es una aplicación formada por varios componentes o servicios distintos que ayudan al usuario a realizar tareas de edición de documentos, índices, tablas, etc.

Supóngase, que los distintos componentes que conforman la aplicación Microsoft Word, no están instalados en el computador personal del usuario, sino que están en un servidor Web, por ejemplo: el sitio de Microsoft. Estos componentes serían los servicios Web, el sitio de Microsoft sería quien ofrece dichos servicios, y por último, el usuario sería el consumidor de estos.

La aparición de los servicios Web va a influir en la forma en que hacen negocios muchas empresas. Con la expansión de las redes empresariales y su interconexión a través de Internet, serán muchos los que opten por arrendar u ofrecer sus productos (o servicios) en forma de servicios Web, en lugar de vender licencias de uso de esos productos. A muchas empresas consumidoras les puede interesar arrendar estos productos en forma de servicios Web, sabiendo que siempre contarán con las últimas versiones y no tendrán que actualizar el software de la empresa.

Para que dicho esquema de servicios tenga éxito, es necesario que las diversas empresas interesadas, lleguen a un acuerdo en cuanto a los estándares a ser utilizados para describir estos servicios, la comunicación con ellos, la búsqueda, etc.

4.4.2 Universalidad de un servicio

En la actualidad, cuando se publica un documento en un servidor Web, el autor, no se preocupa del software o del sistema operativo que usará el cliente para poder acceder a ese documento. Esto es así gracias a que HTML es un estándar universal, al igual que el protocolo

HTTP, el cual se utiliza para solicitar y transmitir dichos documentos. Entonces, se puede decir, que existe un servicio llamado **Web** que es universal debido ya que está disponible para todos sin limitaciones.

Existen empresas como Microsoft, IBM y muchas otras, que están definiendo protocolos en forma conjunta, desarrollando borradores de estándares que permitan describir un servicio, publicarlo para que los consumidores puedan encontrarlo, establecer una comunicación con él, etc. El objetivo final es lograr que los **WebServices** alcancen el mismo nivel de **universalidad** que se describe en el párrafo anterior.

Entonces, un servicio Web se puede crear usando cualquier lenguaje, sobre cualquier sistema operativo y arquitectura o plataforma de hardware. Para que dicho servicio esté disponible para los consumidores, se le debe describir haciendo uso de un lenguaje estándar, **WSDL (Web Service Description Language)**. La aplicación cliente puede implementarse en cualquier lenguaje y ejecutarse en cualquier sistema operativo y hardware, obteniendo la descripción WSDL de un servicio de directorio. Contando con esta información, se comunica con el servicio usando los protocolos HTTP o SOAP.

4.4.3 Protocolos y lenguajes

La meta perseguida por los sistemas distribuidos se ve reflejada fielmente en los servicios Web, para tratar de conseguir esta meta se han utilizado hasta ahora tecnologías como **CORBA**, **DCOM** o **Java RMI**. El problema se reduce a hacer llamadas a procedimientos remotos haciendo uso de un protocolo **RPC (Remote Procedure Call)**, que no dependa de un cierto sistema, como es el caso de DCOM, o lenguaje, como ocurre con RMI.

Hace algún tiempo se propuso un protocolo llamado SOAP, que permite realizar llamadas a RPC haciendo uso del lenguaje XML para describir la llamada y los parámetros, y haciendo uso del protocolo HTTP como transporte. El usar el protocolo HTTP para el transporte de las llamadas implica evitar varios obstáculos tales como los cortafuegos (Firewalls) que impiden la entrada a los servidores por puertos que no sean estándares. Al ser considerado estándar, HTTP es ideal para ser utilizado por SOAP.

Como SOAP describe sus llamadas y respuestas utilizando XML, es posible usarlo desde cualquier sistema operativo o lenguaje. Actualmente existen varias implementaciones que permiten usar SOAP desde Windows, Unix y Linux.

Utilizando SOAP, cualquier cliente puede consumir un servicio prestado por un proveedor. Pero es necesario que el cliente conozca con anterioridad la localización y naturaleza del servicio, sabiendo a qué métodos puede llamar, con qué parámetros, y qué tipo de respuesta obtendrá.

La descripción de un servicio se realiza a través de **WSDL**, el cual es un archivo XML en el que se identifica el servicio y se facilita el esquema para poder utilizarlo, generalmente entregando información sobre los distintos protocolos que es posible utilizar. ASP+ puede generar esta descripción automáticamente.

Anteriormente, se utilizó **SDL**, que fue la propuesta de Microsoft para describir servicios Web. Sin embargo, se ha desarrollado **WSDL** que por el apoyo que ha recibido de parte de la industria se perfila como el futuro estándar para describir servicios Web.

Una vez que se han resuelto los problemas de conectividad entre proveedor y consumidor, y de descripción de los servicios, queda aún un último obstáculo por resolver: **¿Cómo encontrará el consumidor al proveedor de los servicios que necesita?**. Contextualizando este problema a uno de la vida real, cuando se necesita contratar los servicios de algún profesional por lo general se recurre a una guía, en donde se encuentra la información de contacto. Al finalizar los acuerdos para la creación del lenguaje WSDL, las empresas participantes, anunciaron que estaban trabajando en la creación de un servicio llamado **UDDI (Universal Description, Discovery and Integration Registry),** el cual sería un servicio de directorio a nivel mundial en el que los proveedores de servicios podrán registrar sus productos o servicios en forma gratuita. A su vez, los consumidores harán uso de este registro UDDI para buscar lo que necesiten. En él podrán encontrar la descripción de los servicios, información sobre el proveedor, y todo lo necesario para contactarlo.

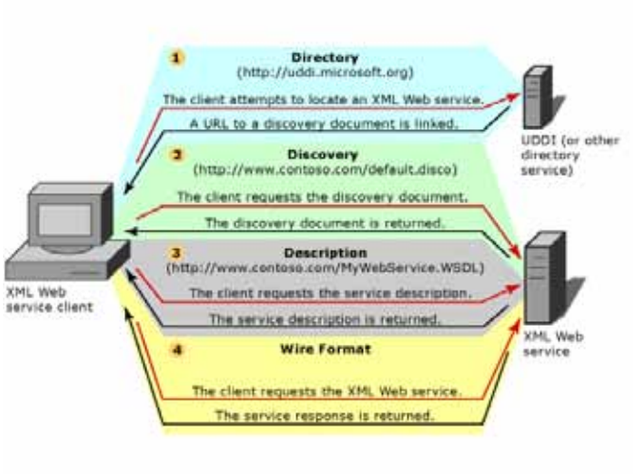


Figura 4.12: Vinculación de proveedores y consumidores de servicios Web, extraído de Microsoft.

El servicio de directorio UDDI de Microsoft se encuentra en la dirección: http://uddi.microsoft.com. En él es posible registrar y buscar servicios Web.

4.4.3.1 Obtención de un Proxy

Para consumir el servicio que se ha instalado en la máquina local, asumiendo que la aplicación consumidora estará en cualquier otro punto (máquina remota) y no precisamente en la máquina en la cual se ejecuta el servicio, se requiere de un **Proxy**. Un Proxy es un módulo de código que hace aparecer como local un servicio que se ejecuta remotamente. Por lo que su consumo resulta igual de simple como si se utilizara un componente cualquiera.

En C#, se utiliza la aplicación **wsdl.exe**, la cual se encarga de generar a partir de un archivo **.WSDL**, un archivo **.CS**, el cual será utilizado por el cliente al momento de consumir el servicio.

A través de este ejemplo se ha creado un componente con un método, que se puede utilizar desde cualquier cliente, gracias a los protocolos HTTP y SOAP, sin necesidad de usar DCOM, registrar controles, configurar, etc.

Por último, es importante reiterar que para que una aplicación .NET se pueda ejecutar en un computador específico, éste debe contar con el **entorno común de ejecución** instalado. Como se dijo anteriormente, éste se encuentra disponible en un paquete llamado **DOTNETREDIST.EXE**, el cual contiene en su interior un archivo llamado **DOTNETFX.EXE**, este

último instala el entorno común de ejecución dejando cualquier equipo en condiciones de ejecutar aplicaciones desarrolladas con esta tecnología. Por otro lado, si se ha instalado la plataforma .NET Framework SDK, automáticamente se instalarán las herramientas de desarrollo y el entorno común de ejecución. En el futuro, los sistemas operativos de Microsoft, contarán con este entorno común de ejecución incluido, por lo que no será necesario instalarlo en forma separada.

4.5 Desarrollo de componentes

La plataforma .NET, constituye un marco de desarrollo compuesto por una jerarquía de clases, entre las cuales existe un importante número de componentes y controles. Se acepta comúnmente, que un componente es un objeto que puede crearse visualmente y puede ser personalizado dentro de un contenedor mediante un diseñador, mientras que un control, además, cuenta con una interfaz de usuario. Dicho de otra forma, todo control es un componente, que cuenta con elementos de interfaz, tales como un botón, una lista, una etiqueta de texto, etc.

Visual Studio .NET cuenta con una gran cantidad de componentes. Sin embargo, dicho conjunto de componentes no cubre el dominio completo de las necesidades de todos los desarrolladores. Es por esta razón que han surgido empresas dedicadas a la construcción y venta de componentes específicos.

Actualmente Visual Studio .NET, permite seleccionar los componentes desde una caja de herramientas (toolbox). Estos componentes son arrastrados con el Mouse, usando la técnica Drag&Drop, antes mencionada, y puestos dentro de un contenedor, también se pueden editar sus propiedades, asociar código a los eventos, etc.

4.6 Acceso a bases de datos

En la actualidad, cualquiera que sea el desarrollo de aplicaciones, lo más probable es que ésta requiera de un amplio acceso a datos, a través de un Sistema Administrador de Bases de Datos Relacionales (RDBMS, Relational Data Base Manager System), como lo son SQL Server 7, SQL Server 2000, Oracle 8i, Oracle 9i, y DB2.

Las soluciones para facilitar el acceso a datos han sido muchas a través del tiempo, destacando las soluciones ligadas a ciertos lenguajes de programación, que por lo general

dependían de un fabricante, plataforma o sistema operativo. Sin embargo, han habido hitos importantes en el desarrollo de acceso a datos, tales como el lenguaje estándar SQL, o el acceso a través del estándar ODBC.

Originalmente Windows no contaba con herramientas de acceso a datos, lo que significó que muchos fabricantes incluyeran en sus productos dichas herramientas. Fue así, por ejemplo, que junto a las versiones de Visual Basic y Visual C++, se pudiera también contar con una herramienta que facilitaba el acceso a datos, **DAO (Data Access Objects)**, además otros lenguajes de programación tales como Delphi de la empresa Borland, incorporaba un mecanismo alternativo llamado **BDE (Borland Database Engine)**.

Tras la aparición de Windows 2000, los sistemas que facilitaban el acceso a datos, se desplazaron al sistema operativo. Siendo ahora éste el encargado de realizar dicha gestión. La base de este nuevo sistema eran los controladores **OLE DB**, que permitían el acceso a diversos sistemas RDBMS, y **ADO (ActiveX Data Objects)**, como interfaz de alto nivel, para comunicarse con esos controladores. Así todas las herramientas de desarrollo podían hacer uso de dichas interfaces, a través de **COM**, y ya no era necesario redistribuir el motor de acceso a datos siempre y cuando el sistema destino fuera Windows 2000.

La aparición de la plataforma .NET, trae consigo una evolución de los mecanismos de acceso a datos. La nueva versión de ADO, llamada ADO+, simplifica la construcción de aplicaciones en múltiples niveles, al facilitar el trabajo sobre conjuntos de datos sin necesidad de mantener una conexión continua con el sistema RDBMS.

La plataforma Microsoft .NET incluye una arquitectura nueva para el acceso a bases de datos denominada "ADO.NET". Esta arquitectura tiene muchas similitudes con el ADO tradicional, pero realmente fue desarrollada para facilitar el acceso a bases de datos que no cuentan con una conexión permanente.

Capítulo 5 : Análisis de la herramienta Rational XDE

5.1 Modelamiento con Rational XDE

Los modelos permiten organizar, entender, crear y describir sistemas complejos. Permiten representar gráficamente procesos del mundo real y al hacerlo, promueven un mejor entendimiento de los requerimientos, diseños más limpios y sistemas mejor mantenidos.

Rational XDE, o Rational e**X**tended **D**evelopment **E**nvironment, combina el diseño y el desarrollo en un entorno fuertemente integrado. Microsoft Visual Studio .NET y Rational XDE permiten trabajar en un solo entorno, evitando la necesidad de cambiar a herramientas que estén fuera de éste.

Hay muchas maneras en las que Rational XDE mejora la forma de trabajar. Por ejemplo:

- Permite moverse desde el análisis y el diseño hacia el código dentro del mismo entorno.
- Permite especificar la forma de sincronizar los modelos y el código, en forma manual o automática.
- Permite definir tanto plantillas de código como plantillas de modelos para ahorrar tiempo y ayudar a reforzar los estándares.
- Permite crear tanto modelos en UML como de formato libre y validar estos modelos para que cumplan con la especificación UML.
- Permite construir múltiples modelos dentro del mismo proyecto que pueden ser seguidos uno a uno.
- Permite reutilizar patrones definidos (patterns) para que los equipos de desarrollo puedan compartir código.

5.1.1 Trabajando con modelos, elementos del modelo y relaciones

Los modelos se usan para representar una abstracción de un sistema desde una perspectiva particular y a un cierto nivel de detalle. Los elementos del modelo y las relaciones, se agregan a los modelos para definir el detalle de estos.

5.1.2 Entendiendo los modelos y los diagramas

Los modelos pueden contener uno o más diagramas, los cuales representan ilustraciones de alguno o todos los elementos del modelo y relaciones.

5.1.2.1 Modelos

Un modelo representa los fundamentos de un sistema complejo sin los detalles no esenciales. Al sacar los detalles no esenciales, se puede entender y dar forma tanto al problema como a la solución.

Un modelo consiste de uno o más elementos del modelo. Hay muchos tipos de elementos del modelo que se pueden usar para representar diferentes abstracciones en un modelo. Por ejemplo, un modelo de casos de uso puede contener los siguientes elementos del modelo: dos diagramas, tres actores, una relación de asociación y un caso de uso.

El explorador de modelos (Model Explorer), despliega los modelos abiertos y los elementos del modelo que ellos contienen.

Figura 5.1: Explorador de modelos.

5.1.2.2 Diagramas

Los diagramas proveen de un medio de visualización y manipulación de los elementos en un modelo. Diferentes diagramas representan diferentes vistas del sistema que se está desarrollando.

- A los diagramas se pueden agregar conectores y formas [16] para representar las relaciones y los elementos del modelo respectivamente, que se encuentran en el modelo.

[16] **Formas:** También llamados elementos visuales, visual elements o view items.

Los diagramas pueden ilustrar múltiples vistas de un modelo, el mismo elemento del modelo puede estar representado en uno o más diagramas.

La ventana de diagramas despliega los diagramas abiertos para que se puedan ver y modificar las formas y los conectores que aparecen en ella.

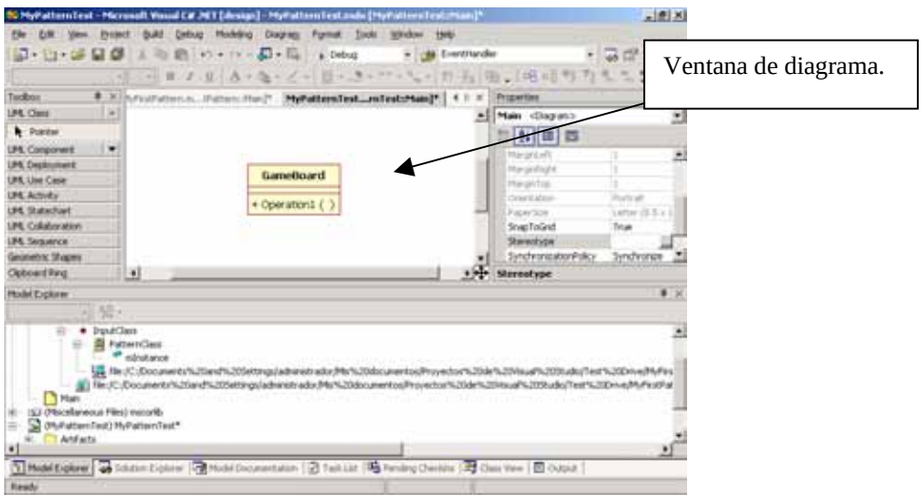


Figura 5.1: Ventana de diagramas en Rational XDE.

5.1.3 Entendiendo los elementos del modelo y las formas

Los elementos son los bloques de construcción de UML, y comprenden elementos del modelo y elementos visuales (o formas). Los elementos del modelo representan abstracciones del sistema que se está modelando, mientras que los elementos visuales o formas proveen de proyecciones textuales o gráficas que facilitan la manipulación de los elementos del modelo.

Los elementos se agrupan en paquetes que referencian a los elementos del modelo. Un modelo[17] es una abstracción de un sistema, representado por una jerarquía de paquetes.

Un elemento del modelo es una abstracción[18] de una característica estructural o una característica de comportamiento del sistema que se está modelando. Los elementos del modelo agregan contenido semántico a éste. Una forma representa a un elemento del modelo en el diagrama. Las formas no agregan semántica al modelo.

---

17 Modelo: Esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja, que se elabora para facilitar su comprensión y el estudio de su comportamiento. R.A.E.

18 Abstracción: Es el efecto de separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción. R.A.E.

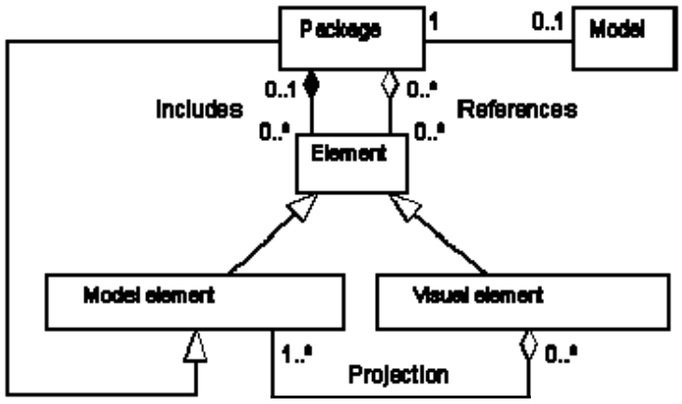


Figura 5.2: Las dos familias de elementos que conforman el contenido de los modelos.

5.1.3.1 Elementos del modelo

Los elementos del modelo de UML pueden agruparse en seis categorías:

- **Elementos del modelo estructurales:** Describen las partes estáticas del sistema que se está modelando. Por ejemplo, los **classifiers** [19] tales como los actores, clases, componentes, nodos y casos de uso.

- **Elementos del modelo de comportamiento:** Describen las partes dinámicas del sistema que se está modelando. Se encuentran por lo general en máquinas de estado y diagramas de interacción. Se pueden mencionar, como ejemplo, las actividades, decisiones, mensajes, objetos y estados.

- **Elementos del modelo organizacionales:** Agrupan a los elementos del modelo en conjuntos. Los paquetes son los elementos del modelo principales que se usan para organizar el modelo.

19 Classifier: Es un elemento del modelo que exhibe rasgos de comportamiento y estructura. Se llama classifier a toda entidad que puede tener intancia.

- **Elementos del modelo de anotación:** Proveen de comentarios y descripciones. Las notas y las restricciones son los elementos del modelo utilizados para realizar anotaciones.
- **Relaciones:** Definen conexiones semánticas entre elementos del modelo.
- **Diagramas:** Proveen de vistas de los elementos del modelo y relaciones en un modelo.

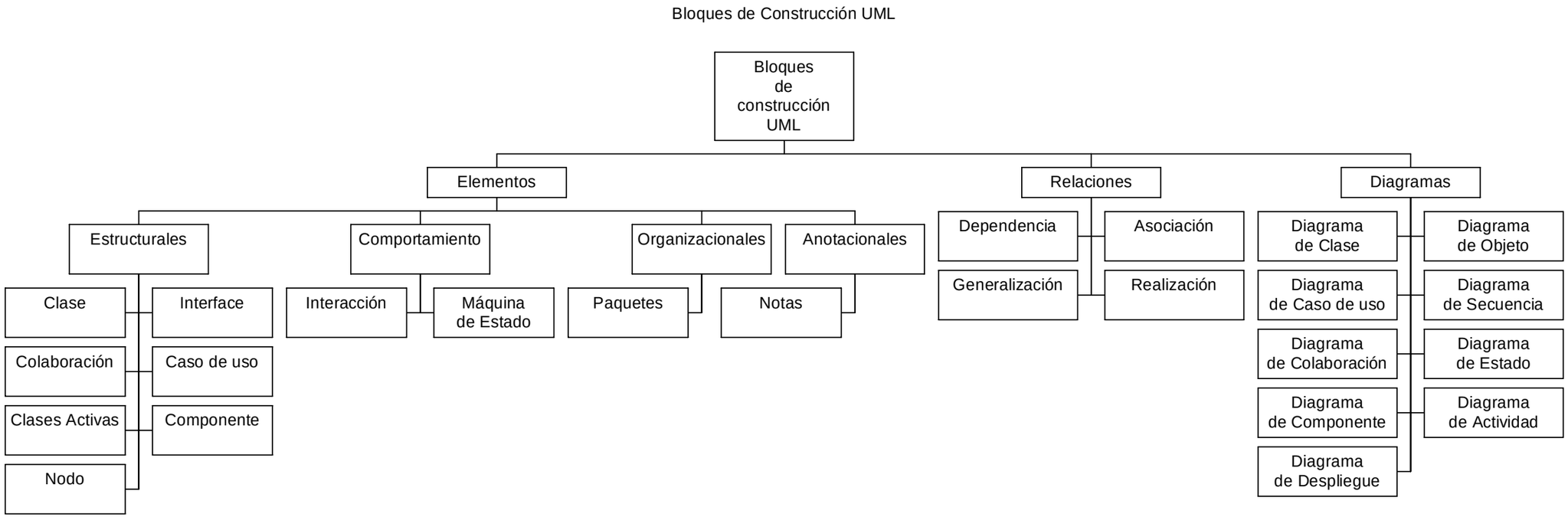


Figura 5.3: Bloques de construcción de UML.

Todos los elementos del modelo tienen propiedades. Algunos elementos del modelo de UML, como las clases, se encuentran principalmente definidas por características tales como su nombre, sus atributos y operaciones.

El explorador de modelos despliega los modelos abiertos y los elementos del modelo que contienen. Los elementos del modelo aparecen bajo el modelo o bajo los elementos del modelo a los cuales pertenecen.

5.1.3.2 Formas

Las formas son representaciones gráficas o textuales de sus elementos del modelo respectivos. Son puestas en un diagrama para ilustrar elementos del modelo relacionados conceptualmente. Las formas de UML **muestran** toda o parte de la semántica de sus elementos del modelo respectivos; *Sin embargo, como se dijo, las formas no agregan información semántica al modelo.*

Una forma sólo representa a un elemento del modelo, pero un elemento del modelo puede ser representado por múltiples instancias de una forma en uno o más diagramas.

Cada forma posee propiedades, las cuales gobiernan su apariencia y posición en la ventana de diagrama.

5.1.4 Entendiendo las relaciones y los conectores

Una relación es una conexión entre elementos del modelo. Las relaciones de UML agregan semántica a los modelos. Un conector es una línea en un diagrama que representa una relación. Los conectores muestran toda o parte de la semántica de sus relaciones subyacentes; Sin embargo, los conectores no agregan ninguna información semántica al modelo (los conectores son elementos visuales).

5.1.4.1 Relaciones

Rational XDE apoya las relaciones de UML que pueden ser usadas para definir la estructura entre elementos del modelo. Estas relaciones pueden agruparse en las siguientes categorías:

- **Asociaciones:** Indican que las instancias de un elemento del modelo están conectadas a las instancias de otro elemento del modelo.
- **Dependencias:** Indican que un cambio a un elemento del modelo puede afectar a otro elemento del modelo.
- **Generalizaciones:** Indican que un elemento del modelo es una especialización de otro elemento del modelo.
- **Realizaciones:** Indican que un elemento del modelo provee de una especificación que otro elemento del modelo implementa.
- **Transiciones:** Gatillan cambios de estado y/o un flujo entre actividades.

Se pueden lograr variaciones a estas relaciones estableciendo propiedades y usando palabras clave. Algunas relaciones poseen propiedades que pueden ser utilizadas para señalar un tipo específico de relación. Por ejemplo, se podría especificar un tipo de relación de asociación específica, tal como una **asociación de agregación** o **composición**, estableciendo la propiedad **kind** (tipo) de la asociación. Otras relaciones poseen palabras clave que se pueden utilizar para especificar una variación. Por ejemplo, la palabra clave **«bind»** se agrega a un conector de **dependencia** para indicar una **relación de ligadura (binding relationship)**.

En Rational XDE, el explorador de modelos despliega los modelos abiertos, sus elementos del modelo y sus relaciones.

5.1.4.2 Conectores

Los conectores son representaciones gráficas de relaciones respectivas. Se sitúan en los diagramas para unir a los elementos visuales, que representan a los elementos del modelo, que poseen una relación.

Un conector representa sólo una relación, y cada relación es representada generalmente por un conector. Sin embargo, una relación también puede ser representada por múltiples instancias de un conector, o ninguna, en uno o más diagramas.

Cada conector posee propiedades que establecen su apariencia en la ventana de diagrama.

5.2 Visión general de Rational XDE

Visión general de la interfaz de Rational XDE en el entorno de desarrollo integrado Microsoft Visual Studio.NET.

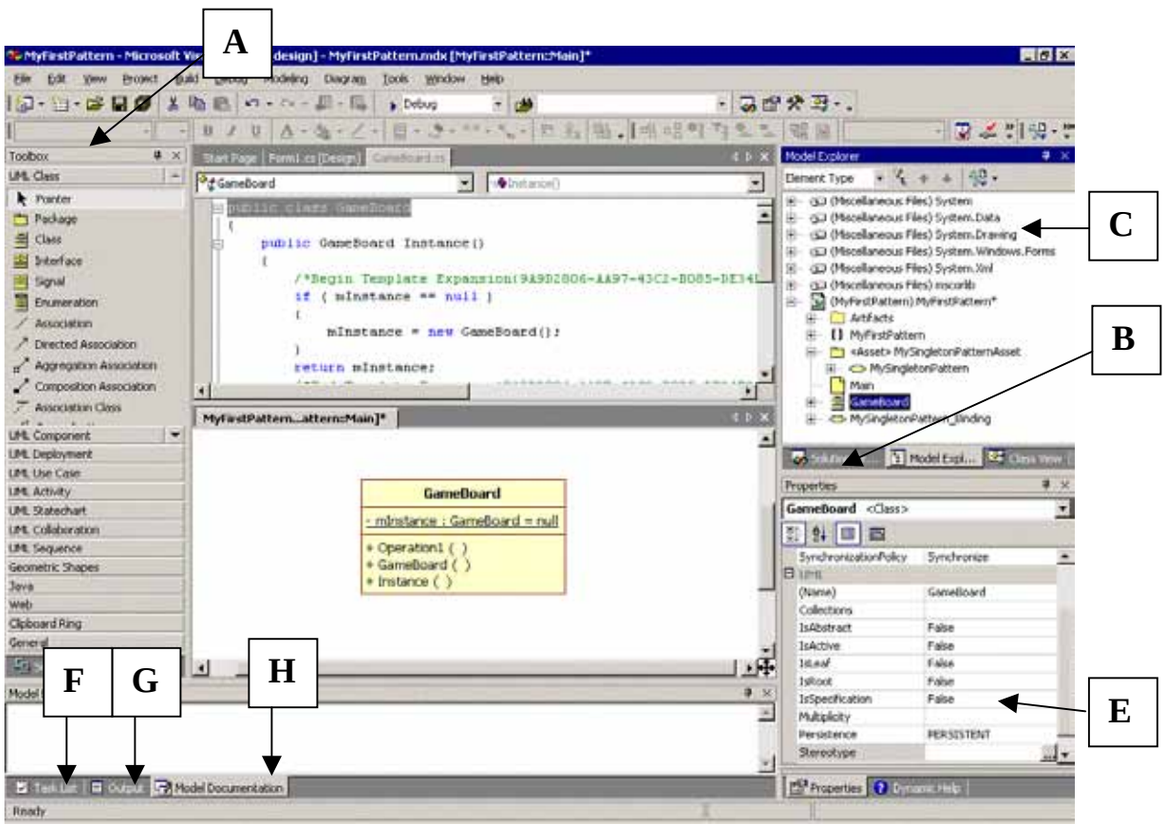


Figura 5.4: Vista general de Rational XDE. Se encuentra fuertemente integrado a Visual Studio .NET.

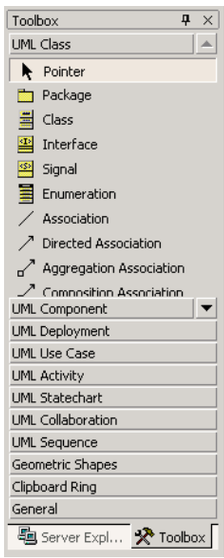


Figura 5.5: Toolbox de Rational XDE.

A. Toolbox – Caja de Herramientas

La caja de herramientas es una representación gráfica de todos los elementos de modelamiento disponibles que el usuario puede arrastrar sobre el diagrama. La caja de herramientas se agrupa en categorías de modelamiento tales como los tipos de diagrama UML y objetos generales que son usados en los diagramas de estilo libre. Las flechas Arriba/Abajo permiten navegar a lo largo de la lista de elementos de modelamiento. Los usuarios pueden agregar sus propios elementos, así como cambiar elementos de categorías.

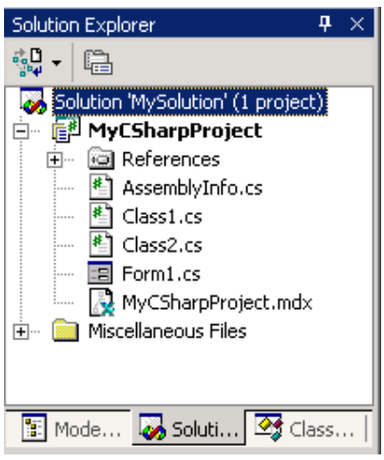


Figura 5.6: Solution Explorer de Rational XDE.

B. Solution Explorer – Explorador de soluciones

El explorador de soluciones provee de una visión global del proyecto o solución. Múltiples proyectos pueden ser vistos en cualquier momento. Un proyecto almacena una colección de archivos que pueden incluir referencias, modelos, código fuente, unidades de almacenamiento, archivos de texto y otros artículos relacionados con el proyecto.

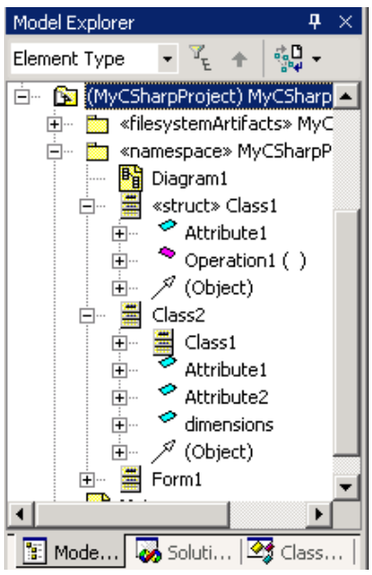


Figura 5.7: Model Explorer de Rational XDE.

C. Model Explorer - Explorador de modelos

El explorador de modelos despliega todos los elementos que pueden ser asociados con un proyecto, tales como recursos de patrones, diagramas, otro modelo, documentación externa, etc. Los usuarios pueden arrastrar los elementos del modelo desde el explorador de modelos a un diagrama.

Similar al explorador de soluciones, el explorador de modelos permite al usuario ver y modificar múltiples modelos en cualquier momento.

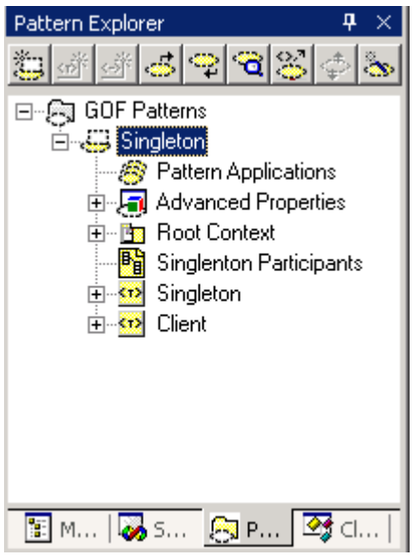


Figura 5.8: Pattern Explorer de Rational XDE.

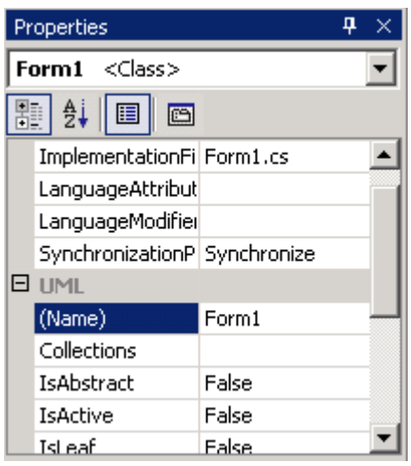


Figura 5.9: Ventana de propiedades.

D. Pattern Explorer - Explorador de Patrones

Disponible desde:

View | Other Windows | Pattern Explorer**.**

El explorador de patrones despliega todos los elementos que están siendo usados por un patrón específico.

Similar al explorador de soluciones, el explorador de patrones permite al usuario ver y modificar múltiples patrones en cualquier momento. Ver **Anexo 3**.

E. Properties View - Ventana de propiedades

La ventana de propiedades es sensible al contexto y desplegará todas las propiedades de un elemento determinado. Estas propiedades pueden ser modificadas desde dentro de la ventana de propiedades.

Cuando se crean patrones, se abre una ventana de propiedades específica para la creación de patrones.

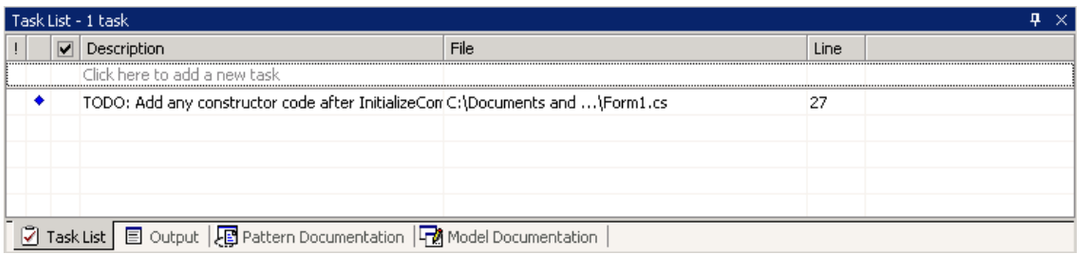


Figura 5.10: Ventana de tareas.

F. Task Window - Ventana de tareas

La ventana de tareas muestra la validación de modelos, errores de compilación y construcción. Los usuarios también pueden ingresar sus propias tareas. Las tareas que aparecen en la ventana son sensibles al contexto. Cuando se ve el código fuente la ventana desplegará errores y advertencias de compilación y de construcción. Cuando se vean diagramas de modelos, se desplegarán errores de validación del modelo y advertencias.

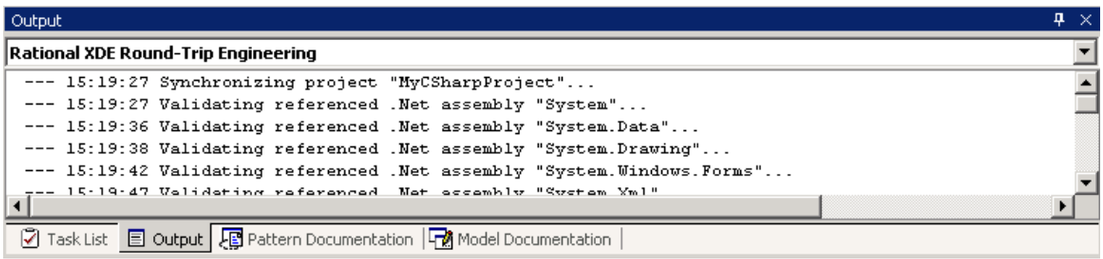


Figura 5.11: Ventana de salida.

G. Output Window - Ventana de salida

La ventana de salida es una ventana de registro que despliega el resultado de varias acciones de programa, tales como creación o modificación de un nuevo proyecto, archivo de modelo, elemento del modelo, etc.

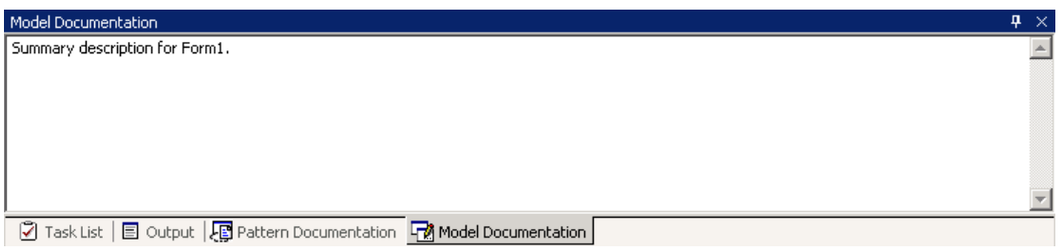


Figura 5.12: Ventana de documentación del modelo.

H. Model Documentation Window – Ventana de documentación del modelo

Despliega documentación al nivel del elemento del modelo.

5.3 Conclusiones

En el presente capítulo se ha retomado el tema del lenguaje de modelamiento unificado (UML), ahora aplicado a la herramienta de modelamiento visual Rational XDE professional v.2002, y se ha dado una visión general a su entorno de modelamiento.

El capítulo 6, revisará aspectos importantes de la Ingeniería de Software, enfocándose en el Modelo Constructivo de Costos (COCOMO II), para proyectos de desarrollo de software.

Capítulo 6 : Análisis del modelo de estimación de costos COCOMO II

6.1 Introducción a COCOMO II

COCOMO II (COnstructive COst MOdel) es un modelo paramétrico que se usa para estimar el esfuerzo y planificación de proyectos de desarrollo de software. El sistema COCOMO, es un software interactivo que brinda apoyo a la estimación de presupuesto y planificación. Gracias a la flexibilidad provista por este software, el administrador de proyectos software (o líder del equipo de desarrollo), puede desarrollar un modelo (o múltiples modelos) de proyectos para identificar potenciales problemas en los recursos, personal, presupuesto, y planificación mientras el software está siendo desarrollado.

6.1.1 Tipos de proyectos Software

Es posible distinguir y tipificar 5 categorías o tipos de proyectos software en los cuales se producirá el desarrollo de software en los próximos años. Estas categorías constituyen el modelo de mercado previsto para las futuras prácticas del software.

- **Programación de usuario final**

  Este tipo de programación está orientado a usuarios comunes de software, los cuales serán capaces de modificar o crear reportes en las aplicaciones que son pertinentes a su dominio de conocimiento en particular, y así producir resultados personalizados en forma rápida. Esto permitirá generar soluciones de procesamiento de información, rápidas, flexibles y manejables. Este mercado estará compuesto por los usuarios que generen, ellos mismos, sus aplicaciones de procesamiento de información a través de generadores de aplicaciones.

  Se prevé que esta categoría de proyectos software contará con un total estimado de 55 millones de personas en Estados Unidos.

  Este sector no necesita un modelo de estimación COCOMO II, ya que el desarrollo de este tipo de aplicaciones por lo general tiene una duración de días u horas. Por esta razón, bastará con una planificación simple, basada en actividades.

- **Composición de aplicaciones**

Trata con un espacio de soluciones sumamente diversificado, por lo que no es posible generar soluciones empaquetadas. La ventaja de este tipo de aplicaciones es que son muy fáciles de construir a partir de componentes interoperables. Algunos ejemplos son: desarrolladores para interfaces gráficas de usuario, bases de datos o gestores de objetos y componentes de dominio específico. El número estimado de personas dentro de este sector del mercado del software será de 0.7 millones de personas en Estados Unidos.

- **Generador de aplicaciones y ayuda a la composición**

Generará paquetes de utilidades para la programación de usuario. Software de apoyo a la construcción de aplicaciones, como hojas de cálculo, sistemas de consultas y sistemas de planificación de inventarios. Permiten a los usuarios definir la aplicación que procesará la información. Se incluyen sistemas para la ingeniería, manufactura y la planificación asistida por computadores. En esta área se contará con aproximadamente 0.6 millones de personas en Estados Unidos.

- **Integración de sistema**

El sector de integración de sistemas tratará con sistemas a gran escala, altamente encapsulados o sistemas sin precedentes. Gran parte de estos podrán construirse con software perteneciente al área de composición de aplicaciones. Sin embargo, aun existirá espacio para el desarrollo tradicional mediante ingeniería Up-Front (Refiérase al **Anexo 4**). Se estima que el número de personas vinculadas a esta categoría será de 0.7 millones de personas en Estados Unidos.

- **Infraestructura**

En este sector se encontrarán productos software tales como sistemas operativos, sistemas gestores de bases de datos, sistemas de gestión de interfaces de usuario y sistemas de redes. Este sector se proyectará hacia la creación de soluciones "middleware" para problemas genéricos como los sistemas distribuidos y el procesamiento transaccional. La cantidad de personas vinculadas con este sector, se estima en 0.75 millones en Estados Unidos.

El modelo COCOMO II para el sector de composición de aplicaciones, está basado en el método de los puntos objetos, el cual será descrito en la sección 6.3. Por otro lado, los sectores de generadores de aplicaciones, integración de sistemas, e infraestructura, están basados en tres modelos matemáticos que estiman el esfuerzo y el tiempo del proceso de desarrollo de software, el cual es un proceso en espiral. Más concretamente, apuntará a un proceso basado en RUP, el cual ha sido descrito en el Capítulo 3. Estas tres categorías de software, pueden ser desarrolladas a través de un proceso que es susceptible de ser representado a través de una mezcla de los tres modelos de estimación COCOMO II.

Dependiendo del nivel de detalle con que se cuente, estos modelos entregarán un resultado cada vez más exacto. En especial si dichos modelos han sido calibrados para ser utilizados dentro de una determinada organización.

Los modelos de estimación de costos, ordenados dependiendo de su grado de exactitud, desde el menos preciso al más preciso, son los siguientes:

- Modelo de composición de aplicaciones
- Modelo de Diseño Anticipado
- Modelo de Post-Arquitectura

La figura de la sección 8.4.2.1, muestra un diagrama UML de los futuros sectores del mercado del software.

6.2 Introducción a los Modelos de Estimación de costo de COCOMO II

Los modelos de estimación de costo de COCOMO II, se aplican a los siguientes tipos de proyectos software: **Generador de aplicaciones**, **Integración de Sistemas** e **Infraestructura**.

6.2.1 Modelo de Composición de aplicaciones

Se utiliza en las primeras fases o ciclos en espiral[20] del proceso de desarrollo de software, que involucran, por lo general, algún tipo de prototipado. Este modelo se dirige a un tipo de aplicaciones que es muy diverso como para poder desarrollarse rápidamente en una herramienta de dominio específico. Además, este tipo de aplicaciones, no se conoce lo suficiente como para

ser desarrollada mediante componentes interoperables. Ejemplos de estos sistemas son: bases de datos o gestores de objetos, procesos transaccionales o distribuidos, y componentes de dominio específico tales como paquetes de control financieros, médicos, etc. (Más información acerca del ciclo de vida en espiral puede ser encontrada en el **Anexo 5**).

6.2.2 Modelo de Diseño Anticipado

Se utiliza en las siguientes fases del proceso de desarrollo de software, en donde por lo general se incluye la exploración de arquitecturas alternativas o estratégicas de desarrollo incremental.

El modelo de diseño anticipado incluye la exploración de arquitecturas de software alternativas y conceptos de operación. Aún no es posible dar paso a la estimación de grano fino. Se utiliza cuando no se conoce mucho acerca del tamaño del producto que se va a desarrollar, la naturaleza de la plataforma objetivo, la naturaleza del personal involucrado en el proyecto o la especificación detallada del proceso de desarrollo que se va a usar.

COCOMO II utiliza puntos de función sin ajustar como métrica de medida y un conjunto de siete drivers de costo de grano grueso.

6.2.3 Modelo Post-Arquitectura

En los puntos anteriores se vio que las primeras fases del proceso de desarrollo de software se ajustaban mejor al modelo de composición de aplicaciones, las siguientes fases serían apoyadas, generalmente, por el modelo de diseño anticipado.

El modelo Post-Arquitectura incluye el actual desarrollo y mantenimiento de un producto software. Se utiliza cuando se ha desarrollado por completo la arquitectura del proyecto software y constituye el modelo más detallado.

Este modelo utiliza el número de líneas de código fuente (SLOC) y/o puntos de función para medir el tamaño del software (Size), y un conjunto de 17 drivers de costo, y 5 factores que determinan el exponente de escala del proyecto.

---

[20] **Ciclo de Vida en Espiral:** Desarrollo en espiral, Barry Boehm (1988).

6.2.4 Suposiciones y distribución de actividades y fases del modelo COCOMO II

El modelo COCOMO II, ha sido desarrollado para ser utilizado por proyectos que empleen ya sea el modelo de desarrollo en cascada o el modelo de desarrollo en espiral. Para que estos sean equivalentes, la implementación en cascada debe ser fuertemente conducida por riesgos, para evitar incurrir en rehacer grandes cantidades de trabajo que no están incluidas en la estimación basada en el modelo en espiral.

A la implementación del modelo en espiral utilizada por COCOMO II, también se le debe agregar una característica: un conjunto de hitos bien definidos, los cuales se han descrito en el capítulo tres y se ven representados en las figuras 3.1 y 6.3. Estos son: LCO (Life cycle Objetives, Objetivos del ciclo de vida), LCA (Life cycle Architecture, Arquitectura del ciclo de vida), IOC (Initial Operational Capability, Capacidad operacional inicial), y corresponden a los hitos de las fases del RUP. El modelo COCOMO II, estima el esfuerzo de desarrollo que se encuentra entre los hitos LCO e IOC. El hito correspondiente a la fase de transición, el cual libera una nueva versión del producto, no está considerado en los cálculos de estimación de esfuerzo de desarrollo del modelo COCOMO II.

6.3 Modelo de costo de Composición de aplicaciones

Como se ha planteado anteriormente, el modelo de costos de composición de aplicaciones, se utiliza en las primeras fases del proceso de desarrollo de software, que involucran, por lo general, algún tipo de prototipado. Además de esto, se basa en el cálculo de puntos objetos.

Los puntos objetos consisten en el conteo de pantallas, informes y módulos desarrollados en 3GL[21], cada uno ponderado por un factor de complejidad de tres niveles (Bajo, Medio y Alto). (Más información refiérase al **Anexo 6**).

Cálculo de puntos objeto

1. Realizar el recuento de objetos: Estimar el número de GUIs[22], informes y componentes que conforman la aplicación.

[21] **3GL:** Lenguaje de tercera generación, del inglés: 3th Generation Language. Incluye el desarrollo visual de GUIs.

[22] **GUIs:** Interfaz grafica de usuario, del inglés: Graphic user interface.

2. Clasificar cada instancia de objeto dentro de niveles de complejidad baja, media y alta dependiendo de los valores de las dimensiones de la característica. Ver tabla 6.1.

Tabla 6.1: Complejidad asociada a las instancias de objetos.

Para GUIs				Para Informes			
Nº de vistas que contiene	Nº y fuente de tablas de datos			Nº de secciones que contiene	Nº y fuente de tablas de datos		
	Total<4 (<2 srvr <3 clnt)	Total<8 (<2/3 srvr <3-5 clnt)	Total 8+ (>3 srvr >5 clnt)		Total<4 (<2 srvr <3 clnt)	Total<8 (<2/3 srvr <3-5 clnt)	Total 8+ (>3 srvr >5 clnt)
< 3	SIMPLE	SIMPLE	MEDIO	0 ó 1	SIMPLE	SIMPLE	MEDIO
3 – 7	SIMPLE	MEDIO	DIFICIL	2 ó 3	SIMPLE	MEDIO	DIFICIL
> 8	MEDIO	DIFICIL	DIFICIL	4 +	MEDIO	DIFICIL	DIFICIL

3. Pesar el número de cada celda usando la tabla 6.2. El peso refleja el esfuerzo relativo que se requiere para implementar una instancia de ese nivel de complejidad:

Tabla 6.2: Pesos asociados a los niveles de complejidad.

Tipo de Objeto	Complejidad-Peso		
	Simple	Medio	Difícil
GUI	1	2	3
Informe	2	5	8
Componente 3GL			10

4. Determinar Puntos Objetos: Suma todas las instancias de objeto pesadas para conseguir un número que será el recuento de puntos objetos.

$$Puntos\,Objetos = \sum_i GUIs_i \times PCA + \sum_i Informes_i \times PCA + \sum_i Componentes_i \times PCA \qquad Ec.6.1$$

$$donde\ PCA = Peso\ de\ la\ Complejidad\ Asociada\ (Ver\ Tabla\ 6.2)$$

Determinación del esfuerzo en el modelo de composición de aplicaciones

1. Estimar el porcentaje de reutilización que se espera lograr en este proyecto. Calcular los nuevos puntos objetos a desarrollar (NOP).

$$NOP = \frac{(Puntos\ Objeto) \times (100 - \%\,\mathrm{Re}\,utilización)}{100} \qquad Ec.6.2$$

2. Determinar un rango de productividad, a partir de la tabla 6.3. Se debe seleccionar la productividad (PROD) a través del criterio de media subjetiva.

$$PROD = \frac{NOP}{mes - \mathrm{hom}\,bre} \qquad Ec.6.3$$

Tabla 6.3: Ratio de productividad PROD.

Experiencia y capacidad de los desarrolladores	Muy bajo	Bajo	Nominal	Alto	Muy Alto
ICASE madurez y capacidad	Muy bajo	Bajo	Nominal	Alto	Muy Alto
PROD	4	7	13	25	50

3. Calcular el valor del esfuerzo [mes-hombre] estimado según la ecuación:

$$PM = \frac{NOP}{PROD} \qquad Ec.6.4$$

Una vez calculado este valor es posible determinar TDEV, que es la duración estimada del proyecto y el staff promedio requerido por éste.

6.4 Modelo de Diseño Anticipado y Post-Arquitectura

Los modelos de diseño anticipado y post-arquitectura se utilizan en el desarrollo de **generadores de aplicaciones**, **integración de sistemas**, o desarrollo de **infraestructura**. Son muy parecidos al momento de realizar una estimación. Esto debido a que su principal diferencia radica en la cantidad de detalle que se utiliza en cada uno de ellos. La ecuación básica, para obtener una estimación de esfuerzo con ambos modelos se muestra en la Ec.6.5:

Tanto el Diseño Anticipado como el modelo de Post-Arquitectura usan la misma fórmula para estimar la cantidad de esfuerzo y duración que tomará un determinado proyecto de software.

El subíndice NS significa **nominal-schedule** (planificación nominal, estándar), y no incluye al driver de costo **SCED** (**Required Development Schedule:** planificación de desarrollo requerida). La cantidad de esfuerzo en [mes-hombre] (Person-Month), PM_{NS}, se estima con la siguiente ecuación:

$$PM_{NS} = A \times (Size)^{E} \times \prod_{i=1}^{n} EM_i \qquad Ec.6.5$$

$$donde: \qquad E = B + 0{,}01 \times \sum_{j=1}^{5} SF_j$$

La duración promedio, $TDEV_{NS}$, estará dada por la siguiente ecuación, Ec.6.6:

$$TDEV_{NS} = C \times \left(PM_{NS}\right)^{F} \qquad Ec.6.6$$

$$donde: F = D + 0{,}2 \times 0{,}01 \times \sum_{j=1}^{5} SF_j$$

$$= D + 0{,}2 \times \left(E - B\right)$$

El valor de n, que es el número de multiplicadores de esfuerzo, EM_i, es 16 para el modelo de Post-Arquitectura y 6 para el modelo de Diseño Anticipado. SF_i representa a los factores de escala exponenciales. Los valores $A, B, EM_1, \ldots, EM_{16}, SF_1, \ldots, SF_6$ para el modelo de Post-Arquitectura de COCOMO II (Versión 2000), son obtenidos a través de calibración de los parámetros actuales y valores de esfuerzo para los 161 proyectos en la base de datos de COCOMO II. Los valores de C y D para la ecuación de planificación de tiempo se obtienen a través de calibración de los valores de planificación para los mismos 161 proyectos que se encuentran también en la base de datos de COCOMO II.

Los valores $A, B, C, D, SF_1, \ldots, SF_5$ para el modelo de Diseño anticipado son los mismos que los del modelo de Post-Arquitectura. Los valores de $EM_1, \ldots, EM_6$ para el modelo de Diseño anticipado se obtienen combinando los valores de los 16 multiplicadores de esfuerzo correspondientes al modelo de Post-Arquitectura.

Los efectos de la compresión o estiramiento de la planificación se cubren a través de un driver de costo adicional, llamado **Required Development Schedule (SCED)**. También se incluyen en la calibración de los 161 proyectos en COCOMO II.2000.

Los valores de $A, B, C \; y \; D$ en la calibración de COCOMO II.2000 son los siguientes:

$$A = 2{,}94 \qquad B = 0{,}91$$
$$C = 3{,}67 \qquad D = 0{,}28$$

Los detalles de la calibración se presentan en la sección 6.6. En donde se presentan ecuaciones para calibrar A, B, C y D usando los valores de la base de datos de la empresa. *Se recomienda por lo menos calibrar A y C para los entornos de desarrollo locales para incrementar la exactitud del modelo.*

6.4.1 Estimación del Tamaño (Size)

El cálculo de la estimación del tamaño es muy importante para un buen modelo de estimación. Sin embargo, el determinar el tamaño de un proyecto software puede ser bastante difícil. Por lo general, un proyecto software, se compone de código nuevo, código reutilizado de otros proyectos –con o sin modificaciones– y código automáticamente traducido.

El modelo COCOMO sólo utiliza la información del tamaño que afecte la estimación del esfuerzo, dentro de los cuales se encuentran: el código nuevo, el código copiado y el modificado.

Para el caso del código nuevo y el reutilizado, se utiliza un método que los hace ser equivalentes. La forma básica de representar el tamaño en COCOMO, es el conteo del número de nuevas líneas de código. El conteo del código que es copiado y luego modificado debe ser ajustado para crear una cuenta que es equivalente a las nuevas líneas de código. El ajuste toma en cuenta la cantidad de diseño, código y pruebas que fueron cambiados. También considera lo entendible del código y la familiaridad del programador con éste.

Para el código automáticamente traducido, existe una razón de productividad de traducción, la cual se usa por separado para determinar el esfuerzo desde la cantidad de código a ser traducido.

6.4.1.1 Conteo de líneas de código

Existen muchas fuentes para estimar las nuevas líneas de código. La mejor fuente, es la información histórica. Por ejemplo, podría existir información que convirtiera los puntos de función, componentes, o cualquier cosa disponible en forma anticipada durante la evolución del proyecto en un buen estimador de las líneas de código de dicho proyecto. Si se carece de información histórica, ésta puede ser reemplazada por la opinión de un experto para derivar dichas estimaciones probables, muy probables o poco probables acerca del tamaño del proyecto.

El tamaño del código se expresa en miles de líneas de código fuente (KSLOC). Una fuente de líneas de código, por lo general excluye al software de apoyo no entregado, tal como drivers de prueba. Sin embargo, si éste es desarrollado con el mismo cuidado que el software entregado, con sus propias revisiones, planes de prueba, documentación, etc., entonces, debería

ser contado **[20]**. *El objetivo es medir la cantidad de trabajo intelectual utilizado en el desarrollo del producto software.*

Definir una línea de código es difícil debido a diferencias conceptuales relacionadas con el conteo de sentencias ejecutables y declaraciones de datos en diferentes lenguajes. Las dificultades surgen cuando se intenta definir medidas consistentes a través de diferentes lenguajes de programación. En COCOMO II, se ha elegido como sentencia lógica fuente a una línea de código estándar. Para definir la medida de líneas de código se usa el checklist de definición del Software Engineering Institute (SEI). El SEI a desarrollado este checklist como parte de un sistema de checklists de definición, formularios de reporte, y formularios suplementarios para apoyar definiciones de mediciones **[21]**.

Un checklist de definición de SLOC se usa para apoyar el desarrollo del modelo COCOMO II. El **Anexo 7**, muestra la definición de líneas de código fuente para los lenguajes C, C++ y C#. Cada X en la columna “Incluir” identifica un tipo de sentencia particular, o atributo incluido en la definición, y viceversa para los que se encuentran en la columna “Excluir”.

6.4.1.2 Introducción a los puntos de función

El método de estimación de costos de puntos de función se basa en la cantidad de funcionalidad en un proyecto software y en un conjunto de factores del proyecto. Los puntos de función son estimadores útiles ya que se basan en información que está disponible en forma temprana en el ciclo de vida del proyecto.

Los puntos de función miden un proyecto software cuantificando la funcionalidad de procesamiento de información asociada con información externa principal o control de entrada, salida, o tipos de archivo. Se pueden identificar cinco tipos de funciones de usuario como se define en la tabla 6.4.

Tabla 6.4: Tipos de funciones de usuario.

Punto de Función	Abreviación	Descripción
External Input	**EI**	Cuenta cada información de usuario o tipo de control de entrada de usuario, que entre en los limites externos del sistema software que se está midiendo.
External Output	**EO**	Cuenta cada información de usuario o tipo de control de salida, que sale del limite externo del sistema que está siendo medido.
Internal Logical File	**ILF**	Cuenta cada grupo lógico principal de información de usuario o información de control en el sistema software como un tipo de archivo interno lógico. Incluye a cada archivo lógico que es generado, usado o mantenido por el sistema software.
External Interface Files	**EIF**	Archivos que han sido pasados o compartidos entre sistemas software deberían contarse como tipos de archivo de interfaz externa dentro de cada sistema.
External Inquiry	**EQ**	Cuenta cada combinación de entrada / salida única, donde una entrada causa y genera una salida inmediata, como un tipo de consulta externa.

Cada una de las instancias de estos tipos de funciones se clasifica por nivel de complejidad.

Los niveles de complejidad determinan un conjunto de pesos, los cuales se aplican al conteo de funciones específicas para determinar una cierta cantidad de puntos de función sin ajustar. Esta es la métrica de estimación de tamaño que utiliza COCOMO II. Sin embargo, el procedimiento habitual de cálculo de puntos de función no es utilizado por COCOMO II. Ya que éste, además, incluye el evaluar el grado de influencia[23] de **14 características de aplicaciones o características generales del sistema** en el proyecto software, determinando un valor para cada una de ellas a través de una escala que va desde 0.00 a 0.05, es decir (0% - 5%).

Las 14 características generales del sistema se suman y luego se agregan a un nivel base de 0.65 para producir un factor de ajuste en el rango 0.65 - 1.35. Mediante este factor de ajuste, es posible obtener el número de puntos de función ajustados (Puntos de función Albrecht).

Cada una de estas 14 características, tales como funciones distribuidas, desempeño y reutilización, tienen entonces un aporte máximo de contribución al esfuerzo estimado de un 5%.

Entonces, el hecho de tener, por ejemplo, un límite de 5% en el efecto debido a la reutilización es inconsistente con la experiencia COCOMO II, ya que en este modelo, se podría tener una reutilización mayor a un 5%. Debido a esto, se utilizan **los puntos de función sin ajustar** para poder estimar el tamaño (Size), luego se aplican los factores de reutilización, drivers

de costos y factores de escala a dicho tamaño, causando así el efecto de las 14 características de las aplicaciones en la determinación del esfuerzo del proyecto.

Otra justificación por la que en muchas organizaciones se han dejado de usar los Puntos de Función Ajustados, es debido a que si el software se desarrolla en entornos homogéneos, el valor del factor de ajuste tiende a ser uno. Por lo que no influirán en el resultado, y los puntos de función sin ajustar serán iguales a los **puntos de función ajustados**. El **Anexo 8** describe las características generales que dependen del entorno del sistema software, también, se describe el cálculo del valor del factor de ajuste para estimar los puntos de función ajustados.

6.4.1.3 Productividad

Según su definición económica estándar, productividad es: "Un Bien o Servicio por unidad de trabajo o costo". Hasta 1979, no existía una definición clara acerca de qué "Bien o Servicio" era la salida de un proyecto software. Esto cambió radicalmente gracias a A. J Albrecht[24], gracias a la publicación de su whitepaper titulado "Function Points".

Uno de los precursores de las investigaciones acerca de productividad fue Mr.Frederick Taylor (1856-1912), quien concluyó que los problemas relacionados con la productividad estaban directamente relacionados con la ignorancia en materias de administración. Años más tarde, entre 1927 y 1932, Elton Mayo condujo una serie de experimentos en la planta Western Electric Hawthorne Works de Chicago, destinados a medir la productividad en equipos de trabajos y cómo los cambios en el entorno influían sobre ellos, específicamente cambios en los niveles de luz y otras variables del entorno, tales como la temperatura, la humedad, etc. Hoy en día esta serie de experimentos se conocen gracias a los resultados obtenidos como el "Efecto Hawthorne".

El efecto Hawthorne reveló un resultado bastante inesperado. Se encontró que cuando las personas se sentían observadas o notadas (tomadas en cuenta), se producían incrementos en su productividad. También se encontró que el mejoramiento en la

---

[23] **Grado de influencia:** DI, del inglés, Degree of Influence, o TDI, Total Degree of Influence.
[24] **Albrecht:** Perteneciente al Equipo de IBM Research, publica por primera vez el concepto denominado puntos de función. Los Puntos de Función Albrecht han sido refinados desde 1979.

productividad se debía a factores sociales tales como la moral del equipo de trabajo, buenas relaciones interpersonales y administración efectiva.

Una definición de productividad, aplicable a la informática y al ámbito que circunda los proyectos software es: razón de salidas/entradas dentro de un período de tiempo con una debida consideración de calidad. Ver ecuación Ec.6.7.

$$\text{Pr}\,oductividad = \frac{Salidas}{Entradas}\ \ (dentro\,de\,un\ período\,de\,tiempo, con\,cierta\,calidad) \qquad Ec.6.7$$

La productividad puede mejorase de una de estas maneras:

- Incrementando las salidas para las mismas entradas
- Reduciendo las entradas y manteniendo sus salidas
- Incrementando las salidas y decrementando las entradas.

Llevado al contexto de los proyectos software, las salidas se ven representadas por los puntos de función. Mientras que la entrada es el esfuerzo requerido para desarrollar dicha funcionalidad.

Entonces, la productividad del software está dada por la siguiente ecuación, Ec.6.8:

$$\text{Pr}\,oductividad\ del\ Software = \frac{Puntos\ de\ función}{Entradas} \qquad Ec.6.8$$

Los puntos de función son las salidas del proceso de desarrollo software. Además, se consideran como las unidades del software o procesos elementales. Los puntos de función se mantendrán constantes sin importar quien desarrolle el software o que lenguaje se utilice para implementarlo.

Eventualmente, cada punto de función tendrá un costo asociado. Sin embargo, no todos los administradores de proyectos software son capaces de descomponer el software en estas unidades fundamentales y por lo general se termina asignando un costo único para todos los puntos de función. Lo que puede llevar a sobre dimensionar el costo de un proyecto software.

Cada punto de función tendrá un costo unitario distinto. Es por esto, que el costo unitario para los External Inputs es diferente al costo unitario para los External Outputs. Además, los External Inputs On-line (en línea), no tendrán el mismo costo unitario (Costo por puntos de función), que un External Input que se procese por lotes (Batch).

6.4.1.4 Costos del Software

Los costos del software se pueden clasificar como:

Costos fijos: Son los costos que permanecen constantes en la totalidad del proyecto sin considerar los cambios de actividad. Por lo tanto, los cambios en la actividad del proyecto no los afectan. Ejemplos de costos fijos son: el capital invertido en computadores personales para un proyecto software. Los sueldos de los desarrolladores pueden ser tratados como costos fijos por las empresas, ya que deben ser pagados regularmente sin importar los niveles de productividad de ésta.

Costos variables: Son los costos que varían, en la totalidad del proyecto software, y que tienen una relación directa con los cambios en el nivel de actividad[25] del proyecto. Es importante entender que las métricas del software se relacionan con los costos del proyecto software y no con los costos totales de la empresa u organización. Además, los costos fijos de un proyecto software son muy bajos, pero los costos variables son muy altos.

Gastos generales: Son los costos que no son atribuibles a un proyecto software en particular. Por ejemplo, los costos de mantener una LAN[26] en funcionamiento caen en la categoría de gastos generales. Los sueldos de ejecutivos también caen dentro de estos costos, ya que no es posible atribuirlos a un proyecto específico.

Costo promedio: Es el costo total dividido por el número de unidades producidas. El costo total de un proyecto software, dividido por el número de puntos de función resulta en el número de unidades monetarias (pesos, dólares, etc.) por punto de función. Además, si

[25] **Actividad:** Facultad de obrar; Prontitud en el obrar; Conjunto de operaciones o tareas propias de una persona o entidad (RAE). La actividad puede ser medida de diferentes formas, por ejemplo: número de horas.

se divide por ejemplo el número total de horas por el número de puntos de función se obtiene el número promedio de horas por punto de función.

Costo marginal: Es el cambio en el costo total atribuible a una unidad de cambio en la salida. Mientras el tamaño del proyecto aumenta, se aprecia un aumento en el costo marginal. Por lo tanto, el costo del software aumenta cuando el software crece. En el desarrollo de software no existen muchas economías de escala.

Cuando se estima el costo de proyectos de desarrollo software, se deben incluir los costos directos e indirectos para dicho proyecto software. Las actividades directas, serán aquellas que pueden ser rastreadas y asociadas a una actividad de desarrollo, de mejoramiento, o de mantención de software. Las actividades indirectas no pueden ser rastreadas hacia una tarea asociada con el proyecto software.

6.4.2 Agregando código nuevo, adaptado y reutilizado

Hasta este momento los cálculos del tamaño del producto (Size) se han realizado exclusivamente para el código nuevo.

El código que es tomado desde otra fuente y es usado en el producto en desarrollo también contribuye al tamaño efectivo de éste. El código preexistente, el cual es tratado como una caja negra[27] e insertado en el producto, recibe el nombre de código reutilizado. Y el código preexistente que es modificado para su uso en el producto recibe el nombre de código adaptado.

El tamaño efectivo del código reutilizado y el código adaptado debe ser ajustado a su equivalente en nuevo código. A este código ajustado, se le llama líneas de código fuente equivalente (ESLOC: Equivalent Source Lines of Code). El ajuste se basa en el esfuerzo adicional que toma modificar el código para incluirlo en el producto.

El modelo de tamaño trata la reutilización como puntos de función y líneas de código fuente de la misma forma, ya sea en el modelo de Diseño Anticipado o en el modelo de Post-Arquitectura.

---

[26] **LAN:** Red de área local, del inglés Local Area Network.

6.4.2.1 Efectos de la reutilización no lineal

A través del análisis de costos de reutilización, en más de 3000 módulos reutilizados por la NASA Software Engineering Laboratory, se ha determinado que la función de costos de reutilización, que relaciona la cantidad de modificaciones de código reutilizado con el costo resultante por reutilizar ese código, es no lineal, en dos formas significativas.

El esfuerzo requerido para reutilizar código no comienza en cero. Existe por lo general un costo de un 5% por evaluar, seleccionar y asimilar la componente reutilizable.

Esto significa que pequeñas modificaciones en el código fuente generan costos desproporcionadamente grandes. Esto se debe a dos factores:

- El costo de entender el software a modificar.
- El costo relativo de verificar módulos de interfaces.

Según G. Parikh y N. Zvegintzov, IEEE Computer Society, el 47% del esfuerzo en la mantención del software, involucra el entender el software a ser modificado.

Es así, como apenas se va desde la reutilización sin modificación (black-box), a una reutilización con modificación (white-box) es posible encontrar esta penalización de entender el software.

El tamaño de las penalizaciones del entendimiento de software y de la verificación de interfaces de módulo, puede ser reducido mediante una buena estructuración del software.

La estructuración del software puede reducir el número de interfaces que requieran una verificación, y el software que está bien estructurado, explicado y relacionado a su misión, será más fácil de entender.

---

[27] **Caja Negra:** En Teoría de sistemas el término caja negra se utiliza para caracterizar a un sistema del cual se conocen sus variables de entrada y sus variables de salida, pero se desconoce el proceso interno que ha llevado a dichos resultados.

6.4.2.2 Un modelo de reutilización

El tratamiento dado por COCOMO II a la reutilización de software usa un modelo de estimación no lineal. Esto involucra estimar la cantidad de software a ser adaptado y tres factores del grado de modificación:

- Porcentaje de diseño modificado (DM).
- Porcentaje de código modificado (CM).
- Porcentaje de esfuerzo de integración requerido para integrar el software reutilizado o adaptado (IM).

Estos tres factores (DM, CM e IM), utilizan el mismo modelo lineal que es usado en COCOMO 81, pero COCOMO II agrega algunos incrementos no lineales. En la ecuación Ec.6.11 muestra como calcular las líneas de código fuente equivalentes.

$$Equivalent\ KSLOC = Adapted\ KSLOC \times \left(1 - \frac{AT}{100}\right) \times AAM \qquad Ec.6.11$$

donde:

$$AAM = \begin{cases} \dfrac{AA + AAF \cdot \left[1 + \left(0.02 \cdot SU \cdot UNFM\right)\right]}{100} & , AAF \leq 50 \\ \dfrac{AA + AAF + SU \cdot UNFM}{100} & , AAFF > 50 \end{cases}$$

$$AAF = (0.4) \cdot DM + (0.3) \cdot CM + (0.3) \cdot IM$$

SU : *Software Unders* tan *ding increment*

AAM : Adaptation Adjustment Modifier

SU, el grado de entendibilidad del software, se obtiene a través de la tabla 6.5. Se determina tomando el promedio subjetivo de las tres categorías en dicha tabla.

Tabla 6.5: Grado de entendibilidad del Software (SU: Software Understanding).

	Very Low	Low	Nominal	High	Very High
Estructura	Muy baja Cohesión, Alto acoplamiento, código Spaghetti.	Moderadamente baja cohesión, Alto acoplamiento.	Razonablemente bien estructurado; algunas áreas débiles.	Alta Cohesión, Bajo Acoplamiento.	Fuerte modularidad, ocultamiento de información en estructuras de dato y control
Claridad de la aplicación	No hay una coincidencia entre el programa y la vista de la aplicación	Alguna correlación entre el programa y la aplicación	Correlación moderada entre el programa y la aplicación	Buena correlación entre el programa y la aplicación	Clara coincidencia entre el programa y la vista de la aplicación
Auto descriptivo	Código confuso, documentación faltante, poco clara u obsoleta	Algo de comentarios de código y encabezados, algo de documentación útil	Nivel moderado de comentarios de código, encabezados y documentación	Código bien comentado y encabezados, documentación útil y algunas áreas débiles	Código auto-descriptivo, documentación actualizada, bien organizada y con un diseño fundamentado.
incremento SU a ESLOC	**50**	**40**	**30**	**20**	**10**

El otro incremento no lineal de reutilización trata con el grado de estimación y asimilación necesario para determinar cuando un módulo de software reutilizado es apropiado para la aplicación y puede integrarse a la descripción global del producto.

La tabla 6.6, provee de una escala de rangos y valores para los incrementos de estimación y asimilación.

Tabla 6.6: Grado de evaluación y asimilación del Software (AA).

Incremento AA	Nivel de esfuerzo AA
0	Ninguno
2	Búsqueda básica de módulo y documentación
4	Algo de prueba y evaluación de módulo (T&E), documentación.
6	Prueba y evaluación considerable, documentación.
8	Prueba y evaluación extensiva, documentación
AA: Assessment & Assimilation (Evaluación y asimilación), es un porcentaje	

La cantidad de esfuerzo requerido para modificar software existente es una función no sólo de la cantidad de modificación (AAF: Adaptation Adjustment Factor) y el entendimiento del software existente (SU). También influye el grado de familiaridad del programador con el software

(UNFM). El factor UNFM se aplica multiplicativamente al incremento del esfuerzo. Si el programador trabaja con el software todos los días, entonces, UNFM tiende a cero, por lo que no agregará nada al incremento SU.

Si el programador, nunca ha visto el software antes, el multiplicador tendrá un valor 1,0 (UNFM tiende a 1,0), y se incrementará por completo el esfuerzo referente a la entendibilidad del software (se penaliza por completo). Ver tabla 6.7.

Tabla 6.7: Grado de familiaridad del programador con el software.

Incremento UNFM	Nivel de familiaridad
0,0	Completamente Familiar
0,2	Mayormente familiar
0,4	Algo familiar
0,6	Considerablemente familiar
0,8	Mayormente no familiar
1,0	Completamente no familiar
UNFM: Familiaridad del programador con el software (Unfamiliarity), es un porcentaje	

La ecuación que estima el tamaño equivalente de nuevas líneas de código (Equivalent KSLOC), Ec.6.11, está basada en el tamaño del producto que está siendo adaptado (Adapted KSLOC), y un modificador que cuenta el esfuerzo que involucra el hacer caber el código adaptado dentro de un producto existente.

AAM, utiliza los factores anteriormente explicados:

SU: Software understanding
UNFM: Programmer Unfamiliarity
AA: Assessment & Assimilation

Junto a un factor llamado AAF (Adaptation Adjustment Factor), el cual contiene los valores de DM, CM e IM.

- **DM:** Es el porcentaje de diseño del software el cual es modificado para adaptar el software a los nuevos objetivos y entorno. Es necesariamente una cantidad subjetiva.
- **CM:** Es el porcentaje de código modificado. Es el porcentaje del código del software el cual es modificado para adaptar el software a los nuevos objetivos y entorno.

- **IM:** Porcentaje de integración requerida para el software adaptado. Es el porcentaje de esfuerzo requerido para integrar el software adaptado dentro del producto final y probar el producto resultante.

Si no existe DM o CM, el componente se está usando sin modificaciones, por lo que no se necesita usar el factor de incremento SU. Cuando el código es modificado, si se aplica SU (Software Understanding).

6.4.2.3 Directivas para cuantificar el software adaptado (modificado)

Se proveerá de directivas para estimar los factores del software adaptado, para diferentes categorías de software utilizando COCOMO II. El código adaptado, es código preexistente al cual se le realizan algunos cambios, mientras que el código reutilizado no tiene cambios en el código fuente preexistente, es decir se utiliza tal como está. COTS es el software Off-The-Shelf (de repisa), el cual se trata generalmente como software reutilizable que no involucra modificaciones. Una diferencia es que puede haber algo de código para pegar o adosar este software, este código asociado debe ser contabilizado.

Como no existe código fuente modificado en la reutilización y en los COTS, DM=0, CM=0, y SU y UNFM no se aplican. AA (Assessment & Assimilation) e IM (Integration Required for Adapted Software) pueden tener valores distintos de cero en este caso. La reutilización no significa que la integración y la prueba sean gratis. Sin embargo, en el enfoque de la reutilización, con líneas de producción con buena arquitectura, la integración y prueba es mínima.

Para el caso del software adaptado, CM > 0, generalmente DM > 0, y todos los otros factores de reutilización normalmente tienen valores distintos de cero. Se espera que IM sea moderado para el software adaptado, pero puede ser más alto que un 100% para la adaptación en aplicaciones más complejas. La tabla 6.8, muestra los rangos válidos de los factores de reutilización con notas adicionales para las diferentes categorías.

Tabla 6.8: Directivas y restricciones de los parámetros del software adaptado.

	Parámetros de reutilización					
Categoría de código	**DM**	**CM**	**IM**	**AA**	**SU**	**UNFM**
Nuevo, software completamente original			No aplicable			
Adaptado, cambios al software preexistente	0% - 100% normalmente > 0%	$0^{+}\% - 100\%$, usualmente > DM y debe ser > 0%	$0\% - 100^{+}\%$ IM usualmente moderado y puede ser >100%	0% - 8%	0% - 50%	0 - 1
Reutilizado, software existente no modificado	0%	0%	0 % - 100%, raramente 0%, pero puede ser muy pequeño	0% - 8%	No Aplicable	
COTS, Off-The-Shelf Software, por lo general requiere código para pegar o adherir el COTS	0%	0%	0% - 100%	0% - 8%	No Aplicable	

6.4.2.4 Evolución y volatilidad de los requerimientos (REVL)

COCOMO II utiliza un factor llamado REVL, para ajustar el tamaño efectivo del producto, debido a la evolución de los requerimientos y la volatilidad causada por factores tales como la evolución de la misión (objetivos) o interfaces de usuario, actualización de tecnologías, o volatilidad de COTS (off-the-Shelf Software). Es el porcentaje de código desechado debido a la evolución de los requerimientos. Por ejemplo, un proyecto que entrega 100.000 instrucciones (líneas de código fuente) y que descarga el equivalente a unas 20.000 instrucciones adicionales, tiene un REVL con valor 20. Entonces, REVL, se utiliza para ajustar el tamaño efectivo del proyecto a 120.000 instrucciones para una estimación en COCOMO II.

El uso de REVL para calcular el tamaño efectivo se da en la ecuación Ec.6.12

$$Size = \left(1 + \frac{REVL}{100}\right) \times Size_D \qquad Ec.6.12$$

$Size_D$: Reutilización Equivalente de Software Entregado

6.4.2.5 Código automáticamente traducido

El modelo de reutilización de COCOMO II, requiere refinamientos adicionales para estimar los costos de la reingeniería y conversión de software.

La diferencia principal entre reingeniería y conversión es la eficiencia de herramientas automatizadas para reestructurar el software. Esto puede conducir a valores muy altos para el porcentaje de código modificado (CM) en el modelo de reutilización de COCOMO II, pero con valores muy pequeños correspondientes al esfuerzo.

El enfoque de estimación de reingeniería y conversión de COCOMO II, involucra estimar un factor adicional, AT, que es el porcentaje de código al que se le realiza reingeniería mediante traducción automática.

El esfuerzo debido a la traducción automática se muestra en la ecuación Ec.6.13:

$$PM_{AUTO} = \frac{ASLOC \times \left(\frac{AT}{100}\right)}{ATPROD} \qquad Ec.6.13$$

$ASLOC$: *Adapted SLOC* (*Líneas de Código Adaptado*)

AT : *Traducción Automática*

$ATPROD$: Pr*oductividad para la traducción automática*

ATPROD, es el factor de traducción automática, el cual resultó ser $2400 \cdot \left[\frac{SLOC}{mes - \text{hom}\, bre}\right]$ en el caso de estudio Nist **[22]**. Según Keith Meiklejohn, Software Project Engineer en Aviónica Táctica de General Dynamics - Advanced Information Systems, se podría usar la mitad de este valor para producir una predicción conservadora, es decir $1200 \cdot \left[\frac{SLOC}{mes - \text{hom}\, bre}\right]$.

En la tabla 6.9, se presenta la variación en porcentaje de reingeniería automatizada, determinada por el proyecto Nist.

Tabla 6.9: Variación en porcentaje de reingeniería automatizada.

Objetivo de la Reingeniería	AT (% de traducción automática)
Procesamiento Batch	96%
Batch con SORT	90%
Batch con DBMS	88%
Batch, SORT, DBMS	82%
Interactivo	50%

El valor de PM_{AUTO} no se utiliza en el cálculo de EKSLOC (Equivalent KSLOC), y tampoco debe influir en PM_{NS} cuando se esté determinando TDEV.

6.4.2.6 Tamaño de la mantención del software

La mantención del software incluye el agregar y modificar nuevas capacidades, excluyendo la reconstrucción del producto por sobre el 50% y el desarrollo de interfaces del sistema que se puedan dimensionar (como GUIs) por sobre el 20%.

El cálculo del tamaño de mantención ($Size_M$) se muestra en la siguiente ecuación, Ec. 6.14

$$\left(Size_M\right) = \left[(Base\ Code\ Size) \times MCF\right] \times MAF \qquad Ec.6.14$$

MAF (Maintenance Adjustment Factor): Es el factor de ajuste de mantención.

MCF (Maintenance Change Factor): Es el porcentaje de cambio que afecta al tamaño del código base (Base Code Size).

En el MCF se pueden usar períodos de mantención superiores al año, conceptualmente MCF representa una razón, ver ecuación Ec. 6.15.

$$MCF = \frac{Tamaño\ Agregado + Tamaño\ Modificado}{Tamaño\ Código\ Base} \qquad Ec.6.15$$

Una versión más simple se puede utilizar cuando se conoce la cantidad de código agregado o modificado del código base existente, ver ecuación Ec.6.16. El código borrado no se considera.

$$\left(Size_M\right) = \left(Tamaño\ Agregado + Tamaño\ Modificado\right) \times MAF \qquad Ec.6.16$$

El tamaño se puede referir a miles de líneas de código fuente, puntos de función, o puntos objetos. Es mejor estimar MCF en términos de la fracción de la aplicación total que se ha modificado, en lugar de las fracciones de entrada (input), salida (output), pantallas, reportes, etc.

La experiencia ha demostrado que contar los items que son modificados puede conducir a sobrestimaciones significativas, debido a que cambios relativamente pequeños, pueden afectar a un gran número de items.

El tamaño de mantención inicial (descrito anteriormente), se ajusta a través de un factor de ajuste de mantenimiento (MAF: Maintenance Adjustement Factor). COCOMO II, utiliza los factores SU y UNFM de su modelo de reutilización para modelar los efectos de un software pobremente estructurado o inentendible en el valor del esfuerzo de mantenimiento. La ecuación Ec.6.17 muestra el cálculo de MAF.

$$MAF = 1 + \left(\frac{SU}{100} \times UNFM \right) \qquad Ec.6.17$$

El uso de $\left(Size_M \right)$ para determinar el esfuerzo de mantención se discute más adelante.

6.4.3 Estimación de esfuerzo

El esfuerzo se encuentra expresado en $\left[mes - \text{hom}\, bre \right]$, y es la cantidad de tiempo que una persona gasta trabajando en un proyecto de desarrollo de software en un mes.

COCOMO II, trata el número $\left[hora - \text{hom}\, bre \right]$ por $\left[mes - \text{hom}\, bre \right]$, $\frac{PH}{PM}$, como un factor ajustable con un valor nominal de 152 horas por [mes-hombre]. Este número excluye el tiempo dedicado a feriados, vacaciones y fines de semana.

La unidad de medida $\left[mes - \text{hom}\, bre \right]$ es muy distinta al tiempo que tomará completar el proyecto. Este último se conoce como TDEV (tiempo de desarrollo, del inglés Time to Develop). Por ejemplo, se puede estimar que un proyecto requiera 50 [mes-hombre] de esfuerzo y un tiempo de 11 meses para completar. Si se utiliza un valor distinto de $\frac{PH}{PM}$, por ejemplo 160 en lugar de 152, COCOMO II ajustará el esfuerzo estimado, en este caso, reduciéndolo cerca de un 5%. El esfuerzo reducido se traducirá en una estimación de tiempo de desarrollo más pequeña.

El modelo de estimación de esfuerzo de COCOMO II se ha introducido en la ecuación Ec. 6.5, y se resume en la Ec. 6.18. La forma de esta ecuación se utiliza para los modelos de costos

de Diseño Anticipado y Post-Arquitectura para estimar el esfuerzo entre los hitos LCO[28] y IOC[29] para los modelos de ciclo de vida MBASE/RUP. Para el caso de un modelo en cascada se usan los puntos SRR[30] y SAR[31]. Las entradas son el tamaño del desarrollo de software (Size), una constante A, un exponente E, y un número de valores llamados multiplicadores de esfuerzo (EM: Effort Multiplier). El número de multiplicadores de esfuerzo depende del modelo de estimación que se use, de Diseño Anticipado o Post-Arquitectura.

$$PM = A \times (Size)^E \times \prod_{i=1}^{n} EM_i \qquad Ec.6.18$$

$donde$:

$$A = 2{,}94$$

El exponente E se explica en detalle en la sección 6.4.3.1, los multiplicadores de esfuerzo se tratan en la sección 6.4.3.2. La constante A refleja un promedio de productividad global, que aproxima a una constante de productividad en $\left[\frac{mes - \text{hom}\, bre}{KSLOC}\right]$ para el caso en que E=1.0. La productividad cambia cuando incrementa E debido a los efectos no lineales en Size.

El tamaño (Size) está en KSLOC. Este se deriva de estimar el tamaño de los módulos de software que constituirán la aplicación. Por lo general, es estimado a partir de los puntos de función sin ajustar (UFPs), luego convertidos a SLOC, y luego divididos por 1000 para transformarlos a KSLOC.

Los drivers de costo son utilizados para capturar las características del desarrollo de software que afectan el esfuerzo para completar un proyecto.

Todos los drivers de costo en COCOMO II tienen un rango de niveles **cualitativos** que expresa el impacto del driver sobre el esfuerzo de desarrollo. Este rango puede variar entre Extra Bajo (Extra Low) y Extra Alto (Extra High). Para cada driver de costo existe un valor asociado correspondiente a un multiplicador de esfuerzo (EM). Así, con este esquema, es posible

[28] **LCO:** Life Cycle Objetives (Objetivos del Ciclo de Vida).
[29] **IOC:** Initial Operational Capability (Capacidad Operacional inicial).
[30] **SRR:** Software Requirements Review (Revisión de Requerimientos de Software).
[31] **SAR:** System Acceptance Review (Revisión de Aceptabilidad del Software).

transformar un rango cualitativo de un driver de costo en uno cuantitativo para ser usado en el modelo. El valor de EM asignado al rango nominal de un driver de costo es 1.00.

El rango de los drivers de costo se basa en un fuerte razonamiento lógico que explica una fuente significativa de esfuerzo en el proyecto en forma independiente.

La principal diferencia entre el modelo de Diseño Anticipado y el modelo de Post-Arquitectura, es el número de drivers de costo y las áreas de influencia que ellos explican.

Para el modelo de Diseño Anticipado, existen 7 drivers de costo, y para el modelo de Post-Arquitectura, existen 17.

La entrada (o parámetro de entrada) más significativo en el modelo COCOMO II es Size. Este es tratado como un driver de costo especial ya que tiene un factor exponencial E.

6.4.3.1 Factores de Escala

El exponente E, es una acumulación de 5 factores de escala (SF: Scale Factors).

El factor de escala (o exponencial) E, explica el ahorro o gasto relativo de escala encontrado en proyectos software de distintos tamaños **[23]**.

Ahorros o gastos de escala del Software

Los modelos de estimación de costos del software por lo general tienen un factor exponencial para describir los ahorros o gastos relativos de escala que se encuentran en los proyectos software de diferentes tamaños. El exponente, E, en la ecuación Ec.6.19 se usa para capturar estos efectos.

Si **E < 1.0**, el proyecto presenta economías de escala. Si el tamaño del proyecto se duplica, el esfuerzo del proyecto es menor que el doble. La productividad del proyecto aumenta mientras aumenta su tamaño. Algunas economías de escala del proyecto se pueden lograr a través de herramientas específicas del proyecto, pero en general son difíciles de lograr. Para proyectos pequeños, costos fijos de puesta en marcha, tales como ajustes de herramientas y compaginación de estándares y reportes administrativos son a menudo una fuente de economía de escala.

Si **E=1.0**, los ahorros y gastos están en equilibrio. Este modelo lineal, se usa por lo general para la estimación de costos de pequeños proyectos software. COCOMO II lo usa para el modelo de composición de aplicaciones.

Si **E>1.0**, el proyecto exhibe un gasto de escala. Esto es debido, por lo general, a dos factores: crecimiento del costo operativo de las comunicaciones interpersonales y crecimiento del costo operativo de la integración de grandes sistemas. Los proyectos grandes tendrán más personal, por lo que también tendrán una mayor cantidad de caminos de comunicaciones interpersonales. Integrar un producto pequeño como parte de un producto grande requiere no sólo el esfuerzo de desarrollar el producto pequeño, si no también el esfuerzo adicional para diseñar, mantener, integrar y probar sus interfaces con las otras partes del producto. Ver tablas 6.10 y 6.11.

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j \qquad Ec.6.19$$

$$donde:$$

$$B = 0{,}91 \qquad (Para\ COCOMO\ II.2000)$$

Tabla 6.10: Factores de escala para el modelo de COCOMO II de Diseño Anticipado.

Factor de Escala (SFj)	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
PREC	Completamente sin precedentes	Prácticamente sin precedentes	Casi sin precedentes	Algo familiar	Muy familiar	Completamente familiar
FLEX	Riguroso	Relajación ocasional	Algo de Relajación	Conformidad general	Algo de conformidad	Objetivos generales
RESL	Poco (20%)	Algo (40%)	A menudo (60%)	Generalmente (75%)	En su mayor parte (90%)	Por completo (100%)
TEAM	Interacciones muy difíciles	Algunas interacciones difíciles	Interacciones básicamente cooperativas	Bastante cooperativo	Altamente cooperativo	Interacciones completas
PMAT	SW-CMM NIVEL 1 BAJO	SW-CMM NIVEL 1 ALTO	SW-CMM NIVEL 2	SW-CMM NIVEL 3	SW-CMM NIVEL 4	SW-CMM NIVEL 5
Significado de los Drivers de Costo						
PREC: Precedencia.						
FLEX: Flexibilidad de desarrollo.						
RESL: Resolución de Arquitectura/Riesgo						
TEAM: Cohesión de Equipo						
PMAT: Madurez del proceso						

Tabla 6.11: Valores de los Factores de escala para el modelo de COCOMO II de Diseño Anticipado pertenecientes a la versión USC-COCOMOII.1999.0.

Factor de Escala (SFj)	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Por ejemplo: Si se tuvieran factores de escala con valores bajos, entonces un proyecto de 100 KSLOC tendrá un esfuerzo relativo de:

$$SF = \sum SF_j = 4{,}96 + 4{,}05 + 5{,}65 + 4{,}38 + 6{,}24 = 25{,}28$$
$$E = 0{,}91 + 0{,}01 \times SF = 0{,}91 + 0{,}01 \times 25{,}28 = 1{,}1628$$
$$Esfuerzo\ \mathrm{Re}lativo = (Size)^E = (100)^{1{,}1628} = 211{,}64\,[mes - \mathrm{hom}bre]$$

6.4.3.1.1 Precedencia (PREC)

Si un producto es similar a algunos proyectos desarrollados previamente, entonces la precedencia incrementa. Ver tabla 6.12.

Tabla 6.12: Niveles de precedencia.

Característica	Muy Bajo	Nominal / alto	Extra Alto
Entendimiento organizacional de los objetivos del producto	General	Considerable	Minucioso
Experiencia trabajando con sistemas software relacionados	Moderado	Considerable	Extensivo
Desarrollo concurrente de nuevo hardware asociado y procedimientos operacionales	Extensivo	Moderado	Algo
Requiere de algoritmos y arquitecturas de procesamiento de información innovadoras	Considerable	Algo	Mínimo

6.4.3.1.2 Flexibilidad de desarrollo (FLEX)

Ver tabla 6.13.

Tabla 6.13: Niveles de flexibilidad de desarrollo.

Característica	Muy Bajo	Nominal / alto	Extra Alto
Necesita conformidad del software con requerimientos preestablecidos	Completo	Considerable	Básico
Necesita conformidad del software con especificaciones de interfaces externas	Completo	Considerable	Básico
Combinación de inflexibilidades sobre las cuales se premia el término anticipado	Alto	Medium	Low

6.4.3.1.3 Resolución Arquitectura / Riesgo (RESL)

Ver tabla 6.14.

Tabla 6.14: Niveles de Resolución de arquitectura y riesgo (RESL).

Característica	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
El plan de administración de riesgos ha identificado todos los items de riesgo críticos, establece el hito para resolver los PDR[32] o LCA[33].	Nada	Poco	Algo	Generalmente	Mayormente	Completamente
La planificación, el presupuesto, y los hitos internos a través de PDR o LCA son compatibles con el plan de administración de riesgos	Nada	Poco	Algo	Generalmente	Mayormente	Completamente
Porcentaje de tiempo de desarrollo dedicado a establecer una arquitectura, dados los objetivos generales del producto	5	10	17	25	33	40
Porcentaje requerido de arquitectos de software expertos disponibles en el proyecto	20	40	60	80	100	120
Herramientas de apoyo disponibles para resolver items de riesgo, para desarrollar y verificar especificaciones de arquitectura	Nada	Poco	Algo	Bueno	Fuerte	Completo
Nivel de incertidumbre en drivers clave de arquitectura: misión, interfaz de usuario, COTS, hardware, tecnología, desempeño	Extremo	Significativo	Considerable	Algo	Poco	Muy Poco
Número y criticalidad de items de riesgo	>10 Crítico	5 - 10 Crítico	2 - 4 Crítico	1 Crítico	>5 No Crítico	<5 No Crítico

[32] **PDR:** Revisión del diseño del producto, del inglés Product Design Review
[33] **LCA:** Arquitectura del ciclo de vida, del inglés Life Cycle Architecture.

6.4.3.1.4 Cohesión de equipo (TEAM)

El factor de escala cohesión de equipo justifica las fuentes de turbulencia y entropía en el proyecto debido a dificultades en la sincronización de los sostenedores del proyecto: usuarios, clientes, desarrolladores, mantenedores y otros. Estas diferencias pueden surgir debido a diferencias de objetivos o culturales; dificultades al recopilar objetivos y la falta de experiencia y familiaridad de los sostenedores en operar como un equipo. La tabla 6.15 provee una definición detallada de los rangos globales de TEAM. El nivel que se seleccionará corresponde al promedio subjetivo de los pesos de las características listadas.

Tabla 6.15: Componentes del nivel de cohesión de equipo (TEAM).

Característica	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Consistencia de los objetivos y culturas de los sostenedores del proyecto	Poco	Algo	Básico	Considerable	Fuerte	Completo
Habilidad y voluntad de los sostenedores para acomodar los objetivos de los otros sostenedores del proyecto	Poco	Algo	Básico	Considerable	Fuerte	Completo
Experiencia de los sostenedores de operar como un equipo	Nada	Poco	Poco	Básico	Considerable	Extensivo
Construcción de equipos de sostenedores para lograr los compromisos y la visión compartida de proyecto	Nada	Poco	Poco	Básico	Considerable	Extensivo

6.4.3.1.5 Madurez del proceso (PMAT)

El procedimiento para determinar PMAT (Process Maturity) gira en torno al SW-CMM desarrollado por el SEI (Software Engineering Institute, Carnegie Mellon University, Pittsburgh).

Existen dos formas de establecer el nivel de madurez del proceso software, en este caso **EFT-UP**.

La primera, captura el resultado de la evolución de la empresa de desarrollo de software basado en CMM, ver tabla 6.16.

Tabla 6.16: Niveles de PMAT para el nivel estimado de madurez del proceso (EPML).

Nivel PMAT	Nivel de Madurez	EPML
Muy Bajo	CMM Nivel 1 Bajo	0
Bajo	CMM Nivel 1 Alto	1
Nominal	CMM Nivel 2	2
Alto	CMM Nivel 3	3
Muy Alto	CMM Nivel 4	4
Extra Alto	CMM Nivel 5	5

6.4.3.1.5.1 Cuestionario de las áreas clave del proceso

La segunda forma de determinar PMAT, gira en torno a una evaluación de las 18 áreas claves del proceso de desarrollo de software (KPAs: Key process Areas).

El procedimiento para determinar PMAT es decidir el porcentaje de conformidad (o cumplimiento) para cada una de las áreas claves del proceso.

Si la empresa ha experimentado una evaluación CMM reciente, entonces se usa el porcentaje de conformidad para el conjunto global de KPAs. Si no se ha llevado a cabo una evaluación, se utiliza el nivel de conformidad con las metas (u objetivos) de las KPAs, de modo de establecer cual es el nivel de conformidad. Esto se realiza a través de una tabla Likert que se presenta en el **Anexo 9**.

Con este cuestionario se calcula un nivel de madurez del proceso equivalente **(EPML: Equivalent Process Maturity Level)** según la ecuación Ec. 6.20.

$$EPML = 5 \times \left(\sum_{i=1}^{n} \frac{KPA\%_i}{100} \right) \times \frac{1}{n} \qquad Ec.6.20$$

La sumatoria implica que se debe considerar cada KPA, por lo que el máximo valor que alcanza n será igual a 18. Sin embargo, no siempre serán 18 KPAs las que influirán en el cálculo de EPML, ya que las KPAs evaluadas como: "No se aplica" o "No se sabe", no se cuentan. Por esta razón, en algunas oportunidades n será menor que 18.

El porcentaje, KPA%, viene dado en la tabla del **Anexo 9**, y constituye un peso asociado a cada nivel de evaluación, ver tabla 6.17.

Tabla 6.17: Niveles de las áreas claves del proceso (KPAs).

Almost Always	Frecuently	About Half	Occasionally	Rarelly if Ever	Does Not Apply	Don't Know
Casi siempre	Frecuentemente	Por la mitad	Ocasionalmente	Rara vez si es que...	No se Aplica	No se sabe
100%	75%	50%	25%	1%	-	-

Un valor EPML de 0 corresponde a un nivel de PMAT Muy Bajo (Very Low) en la tabla 6.17. Las futuras revisiones de PMAT probablemente requerirán del análisis del modelo de madurez de capacidad de integración (Capability Maturity Model Integration) del SEI.

6.4.3.2 Multiplicadores de esfuerzo

6.4.3.2.1 Drivers de costo del modelo Post-Arquitectura

Como se ha dicho éste es el modelo más detallado. Pretende ser usado cuando la arquitectura del ciclo de vida del producto software se haya realizado por completo. Este modelo se utiliza en el desarrollo y mantenimiento de productos software en las siguientes áreas de aplicaciones: Generadores de aplicaciones, Integración de sistemas, o infraestructura.

Los 17 multiplicadores de esfuerzo del modelo Post-Arquitectura (EM: Effort Multipliers), se utilizan en COCOMO II para ajustar el esfuerzo nominal, [mes-hombre], para caracterizar al producto software que está bajo desarrollo. Entonces a cada driver de costo se le asocia un multiplicador de esfuerzo, el cual puede tomar un determinado rango de valores. El nivel nominal siempre tiene un multiplicador de esfuerzo con valor 1.00, el cual no modifica el esfuerzo estimado. Cualquier otro valor que no sea nominal, producirá cambios en el esfuerzo estimado.

Por ejemplo, un rango Alto (High) de confiabilidad requerida por el software (RELY) agregará un 10% al esfuerzo estimado, como lo determina la calibración de COCOMO II.2000. Un nivel muy alto (Very High) agregará un 26% al esfuerzo estimado.

Es posible asignar rangos de niveles intermedios a los multiplicadores de esfuerzo correspondientes para el proyecto. Por ejemplo, COCOMO II, apoya los niveles de incremento de los drivers de costo en incrementos de un cuarto (1/4), es decir: Low+ 0.25; Nominal+ 0.50; High+ 0.75; etc.

Cuando la evaluación de un driver de costo se encuentre a mitad de camino entre los incrementos de un cuarto (1/4), siempre se redondeará en dirección al valor nominal. Por ejemplo, si el nivel de un driver de costo se encuentra entre **Low+0.5** y **Low+0.75**, entonces se debe seleccionar Low+0.75; o si por ejemplo, un nivel se encuentra entre High+0.25 y High+0.50, entonces se debe seleccionar High+0.25, redondeando lo más próximo a Nominal, ver tabla 6.18.

Tabla 6.18: Representación de redondeo al más próximo a nominal.

Very Low	**Low**		**Nominal**	**High**		Very High	Extra High
Muy Bajo	**Bajo**		**Nominal**	**Alto**		Muy Alto	Extra Alto
	+0.5	**+0.75**		**+0.25**	+0.5		

Normalmente se usa interpolación lineal para determinar valores multiplicadores intermedios, pero la interpolación no lineal es más exacta para el extremo superior de los drivers de costo (TIME y STOR) y el extremo inferior (SCED).

El modelo COCOMO II se puede utilizar para estimar el esfuerzo y el tiempo que tomará un proyecto completo o un proyecto que consiste en múltiples módulos.

El tamaño y los rangos de los drivers de costo pueden ser diferentes para cada módulo, con excepción del tiempo de desarrollo requerido (SCED) y los factores de escala.

A la pitatoria de los multiplicadores de esfuerzo se le conoce como el **factor de ajuste de esfuerzo** (EAF: Effort Adjustment Factor).

El factor de ajuste de esfuerzo está compuesto por cuatro categorías de factores en las que se incluyen los drivers de costo pertinentes:

- **Los factores del producto:** RELY, DATA, **CPLX**, RUSE, DOCU.
- **Los factores de la plataforma:** TIME, STOR, PVOL.
- **Los factores del personal:** ACAP, APEX, PCAP, PLEX, LTEX, PCON.
- **Los factores del proyecto:** TOOL, SITE, SCED.

6.4.3.2.1.1 Factores del producto

Los factores del producto causan variaciones en el esfuerzo requerido para desarrollar software debido a las características intrínsecas del producto software que se está desarrollando.

Un producto que es complejo, tiene requerimientos altos de confiabilidad, o trabaja con una base de datos de prueba de gran tamaño, requerirá de un mayor esfuerzo para ser

completado. Existen 5 factores del producto, donde la complejidad **(CPLX)** tiene la mayor influencia sobre el esfuerzo estimado.

6.4.3.2.1.1.1 Confiabilidad requerida del software (RELY)

Pretende medir el grado de confiabilidad en la operación del software en un período de tiempo. Tomando en cuenta el efecto de una falla en el software. Por ejemplo, si el efecto de una falla en el software es sólo un ligero inconveniente, el valor de RELY será Muy Bajo (Very Low). Por el contrario, si el efecto de la falla del software arriesga vidas humanas, entonces el valor de RELY es Muy Alto (Very High). La tabla 6.19 provee el rango de valores que puede tomar RELY.

Tabla 6.19: Rango de valores para el driver de costo RELY.

RELY Descripción:	Ligero Inconveniente	Bajo, Perdidas fácilmente recobrables	Moderado, Perdidas fácilmente recobrables	Alto, Perdidas Financieras	Riesgo a vidas humanas	
Rangos	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**0.82**	**0.92**	**1.00**	**1.10**	**1.26**	**-**

6.4.3.2.1.1.2 Tamaño de la base de datos (DATA)

Este driver de costo intenta capturar el efecto que tiene sobre el desarrollo del producto los requerimientos de grandes cantidades de datos de prueba. El rango de valores está determinado por el cálculo de D/P ($\frac{\text{DataBase Bytes}}{Sloc\ del\ \text{Pr}\ ograma}$), la razón de bytes en la base de datos de prueba a la cantidad de SLOC en el programa. La importancia por la que se considera este driver, es debido al esfuerzo requerido para generar los datos que serán usados para probar el programa. Cuando no se cuente con una base de datos de prueba, será necesario realizar la estimación del tamaño de dicha base de datos. Refiérase al **Anexo 10**.

Entonces se puede decir que DATA captura el esfuerzo necesario para ensamblar y mantener la información requerida para completar la prueba del programa a través de IOC(Initial Operational Capability). Ver tabla 6.20.

Tabla 6.20: Driver de costo DATA.

DATA* Descripción:		D / P < 10	10 ≤ D / P < 100	10 ≤ D / P < 1000	D / P ≥ 1000	
Rangos	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**N/a**	**0.90**	**1.00**	**1.10**	**1.26**	**-**

DATA, se considera Bajo (Low) si D/P es menor que 10 y es Muy Alto (Very High) si es mayor que 1000. P se mide en líneas de código fuente equivalentes (SLOC), la cual puede involucrar puntos de función o conversiones de reutilización.

6.4.3.2.1.1.3 Complejidad del producto (CPLX)

La complejidad se divide en 5 áreas: operaciones de control, operaciones computacionales, operaciones dependientes del dispositivo, operaciones de administración de información, y operaciones de administración de interfaces de usuario. Utilizando la tabla 6.21, se deben seleccionar las áreas o combinaciones de éstas que caractericen al producto o el componente del producto que se está valorando. El valor de complejidad será, entonces, el promedio subjetivo de los pesos de las áreas seleccionadas. La tabla 6.22 muestra los valores para los multiplicadores de esfuerzo del driver de costo CPLX de COCOMO II.2000.

Tabla 6.21: Nivel de componentes de complejidad (CPLX).

	Operaciones de control	Operaciones computacionales	Operaciones dependientes del dispositivo	Operaciones de administración de información	Operaciones de administración de interfaces de usuario
Muy Bajo	Código rectilíneo con unos pocos operadores de programación estructurada no anidados: DO, CASE, IF - THEN - ELSE. Composición simple de módulos a través de llamadas a procedimientos o scripts simples.	Evaluación de expresiones simples. Ejemplo: $A = B + C(D - E)$	Lectura simple, sentencias de escritura con formatos simples.	Arreglos simples en la memoria principal. COTS simples, consultas y actualizaciones a bases de datos.	Formularios simples de entrada, generadores de reporte.
Bajo	Anidamiento directo de operadores de programación estructurada. En su mayoría predicados simples.	Evaluación de expresiones de nivel moderado. Ejemplo: $D = \sqrt{B^2 - 4AC}$	No se necesita el conocimiento de un procesador en particular o características de un dispositivo de E/S. La E/S es realizada a un nivel GET/PUT.	Subconjunto de un sólo archivo si cambios de estructura de datos, sin edición, sin archivos intermedios. COTS - DB moderadamente complejos, consultas y actualizaciones.	Uso de constructores simples de interfaces gráficas de usuario (GUIs)
Nominal	En su mayoría anidamiento simple. Algo de control entre módulos. Tablas de decisión. Llamadas de retorno o paso de mensajes simples, incluyendo el procesamiento distribuido apoyado por middleware.	Uso de rutinas de matemática y estadística estándar. Operaciones básicas de matrices / vectores.	El procesamiento de E/S incluye la selección de dispositivo, verificación de estado y procesamiento de errores.	Entra de múltiples archivos y un sólo archivo de salida. Cambios estructurales simples, edición simple. COTS - DB complejos, consultas y actualizaciones.	Uso simple de elementos de interfaz de usuario
Alto	Operadores de programación estructurada altamente anidados con muchos predicado compuestos. Control de colas y stack. Procesamiento homogéneamente distribuido. Control en tiempo real suave de un sólo procesador	Análisis numérico básico: interpolación multivariada, ecuaciones diferenciales comunes. Redondeo y truncado básico	Las operaciones de E/S se realizan a un nivel físico (traducción de la dirección de almacenamiento física, buscar, leer, etc.). Superposición de E/S optimizada.	Triggers simples activados por contenidos en el flujo de datos. Compleja reestructuración de información.	Desarrollo de elementos de interfaz de usuario y extensión. E/S simple de voz, multimedia.
Muy Alto	Código recursivo y de entradas múltiples. Manejo de interrupciones de prioridad fija. Sincronización de tareas, llamadas de retorno complejas. Procesamiento distribuido heterogéneo. Control en tiempo real difícil de un sólo procesador.	Análisis numérico difícil, pero estructurado: próximo a las ecuaciones de la matriz singular, ecuaciones diferenciales parciales. Paralelización simple.	Rutinas para el enmascaramiento, servicio y diagnostico de interrupciones. Manejo de línea de comunicación. Sistemas encapsulados de desempeño intensivo.	Coordinación de bases de datos distribuidas. Triggers complejos. Optimización de búsqueda.	Gráficos dinámicos moderadamente complejos en 2D/3D, multimedia.

Tabla 6.22: Driver de costo CPLX.

CPLX						
Rangos	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**0.73**	**0.87**	**1.00**	**1.17**	**1.34**	**1.74**

6.4.3.2.1.1.4 Desarrollado para reutilización (RUSE)

Este driver de costo se considera en el cálculo del esfuerzo adicional necesario para construir componentes que se planean reutilizar, ya sea en el proyecto actual o en proyectos futuros. Este esfuerzo se concentra en crear diseños de software más genéricos, documentación más elaborada, y una prueba más extensa para asegurar que los componentes se encuentran listos para ser usados en otras aplicaciones. "Entre Proyectos" se podría aplicar la reutilización entre módulos del proyecto de una aplicación. "Entre Programas" se podría aplicar la reutilización entre múltiples proyectos de aplicación para una sola organización. "Entre líneas de producto" podría aplicarse, si la reutilización se extiende a través de múltiples organizaciones. "Entre múltiples líneas de productos" se podría aplicar la reutilización en las líneas de productos con espacios de soluciones que apuntan a otro objetivo.

Desarrollar para reutilizar implica restricciones en los drivers de costo RELY y DOCU. El nivel de RELY debería estar, como mucho, un nivel más abajo que el nivel de RUSE. El valor del driver de costo DOCU debería estar por lo menos en Nominal para valores Altos y Nominal del driver de costo RUSE, y por lo menos Alto para valores de RUSE que sean Muy altos y Extra Altos. Ver tabla 6.23.

Tabla 6.23: Niveles del driver de costo RUSE.

RUSE Descripción:		Ninguno	Entre Proyectos	Entre Programas	Entre líneas de producto	Entre múltiples líneas de productos
Rangos	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**-**	**0.95**	**1.00**	**1.07**	**1.15**	**1.24**

6.4.3.2.1.1.5 Documentación acorde a las necesidades del ciclo de vida (DOCU)

Algunos modelos de costo del software incluyen un driver de costo para indicar el nivel requerido de documentación. En COCOMO II, la escala de rangos para el driver de costo **DOCU** es evaluada en términos de la aplicabilidad de la documentación del proyecto a sus requerimientos de ciclo de vida. La escala va desde Muy Bajo (requieren ser cubiertos Muchos ciclos de vida) a Muy Alto (Muy excesivo para las necesidades del ciclo de vida). Ver tabla 6.24.

Tabla 6.24: Niveles del driver de costo DOCU.

DOCU Descripción:	Muchos ciclos de vida necesitan ser descubiertos	Algunos ciclos de vida necesitan ser descubiertos	Tamaño correcto para las necesidades del ciclo de vida	Excesivo para las necesidades del ciclo de vida	Muy excesivo para las necesidades del ciclo de vida	
Rangos	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**0.81**	**0.91**	**1.00**	**1.11**	**1.23**	**-**

Este driver de costo puede ser influenciado por el driver de costo RUSE.

6.4.3.2.1.2 Factores de la plataforma

La plataforma se refiere al conjunto de infraestructura de hardware y software de la máquina destino.

6.4.3.2.1.2.1 Restricción de tiempo de ejecución (TIME)

Esta es una medida de las restricciones de tiempo de ejecución impuestas al sistema software. El valor es expresado en términos del porcentaje esperado de tiempo de ejecución disponible que será utilizado por el sistema o subsistema consumiendo el recurso tiempo de ejecución. Los rangos de valores van desde el Nominal, cuando se utiliza menos del 50% del tiempo de ejecución, a Extra Alto, cuando el 95% del tiempo de ejecución es consumido.

Tabla 6.25: Niveles del driver de costo TIME.

TIME Descripción:			Uso ≤ 50% del tiempo de ejecución disponible	70% de uso del tiempo de ejecución disponible	85% de uso del tiempo de ejecución disponible	95% de uso del tiempo de ejecución disponible
Rangos	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Multiplicador de esfuerzo	-	-	1.00	1.11	1.29	1.63

6.4.3.2.1.2.2 Restricciones de almacenamiento principal (STOR)

Este valor representa el grado de restricción impuesta al almacenamiento principal de un sistema o subsistema software. Dado el considerable incremento en el tiempo disponible de procesador y almacenamiento principal uno podría cuestionarse si estas restricciones son aun relevantes. Sin embargo, muchas aplicaciones aun continúan expandiéndose y consumiendo cualquier recurso que esté disponible, en particular los productos COTS (Off-the-Shelf Software), por lo que este driver de costo aun es relevante. La tabla 6.26 presenta los valores que puede tomar este driver de costo.

Tabla 6.26: Niveles del driver de costo STOR.

STOR Descripción:			Uso ≤ 50% del espacio disponible	70% de uso del espacio disponible	85% de uso del espacio disponible	95% de uso del espacio disponible
Rangos	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Multiplicador de esfuerzo	-	-	1.00	1.05	1.17	1.46

6.4.3.2.1.2.3 Volatilidad de la plataforma (PVOL)

La palabra "plataforma" se entiende como el complejo de hardware y software (Sistema Operativo, Base de datos, etc.) que el producto software visita para realizar sus tareas. Si el software que se está desarrollando, por ejemplo, es un sistema operativo, entonces la plataforma es el hardware del computador. Si el producto a ser desarrollado es un sistema administrador de bases de datos, entonces la plataforma es el hardware y el sistema operativo. Si, por ejemplo, se está desarrollando un navegador de redes (por ejemplo, un Browser) la plataforma es la red, el hardware, el sistema operativo y los repositorios de información distribuidos. La plataforma incluye a cualquier compilador o ensamblador que apoye el desarrollo del sistema software. El rango de valores que puede tomar fluctúa entre Bajo (Low), cuando hay un cambio importante

cada 12 meses, a Muy Alto (Very High), cuando hay un cambio importante cada 2 semanas. Ver tabla 6.27.

Tabla 6.27: Niveles del driver de costo PVOL.

PVOL Descripción:		Cambios importantes cada 12 Meses; Cambios menores cada 1 mes	Cambios importantes cada 6 meses; Cambios menores cada 2 semanas.	Cambios principales cada 2 meses; Cambios menores cada 1 semana	Cambios principales cada 2 semanas; Cambios menores cada 2 días	
Rangos	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**-**	**0.87**	**1.00**	**1.15**	**1.30**	**-**

6.4.3.2.1.3 Factores del personal

El factor más importante después del tamaño del proyecto (Size) es el factor que involucra al personal, ya que tiene una fuerte influencia cuando se determina la cantidad de esfuerzo requerido en el desarrollo de productos software. Los factores del personal sirven para evaluar las capacidades del equipo de desarrollo, **no son para evaluar las capacidades individuales**. Lo más probable es que los valores de estos factores cambien durante la evolución del ciclo de vida de un proyecto software, reflejando así la experiencia adquirida o las rotaciones de personal desde y hacia el proyecto.

6.4.3.2.1.3.1 Capacidad del Analista (ACAP)

Se consideran analistas a las personas que trabajan sobre los requerimientos, diseños de alto nivel y diseño detallado.

Los atributos principales que deberían ser evaluados son las habilidades de análisis y diseño, eficiencia y rigurosidad, junto con las habilidades de comunicar y cooperar. Esta evaluación no debe considerar el nivel de experiencia del analista, ya que ésta se mide a través de los drivers de costo APEX, LTEX y PLEX. Los valores posibles se despliegan en la tabla 6.28.

Tabla 6.28: Niveles del driver de costo ACAP.

ACAP Descripción:	Percentil 15º	Percentil 35º	Percentil 55º	Percentil 75º	Percentil 90º	
Rangos	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**1.42**	**1.19**	**1.00**	**0.85**	**0.71**	**-**

6.4.3.2.1.3.2 Capacidad del programador (PCAP)

Las tendencias actuales continúan enfatizando la importancia de contar con un analista altamente capacitado. Sin embargo, el incremento en el uso de paquetes COTS (off-the-Shelf), y el consiguiente aumento en forma significativa de la productividad, asociado con las habilidades del programador para manejar estos paquetes, muestran la gran importancia que tienen las capacidades del programador.

La evaluación debería basarse en las capacidades de los programadores vistos como un equipo, en lugar de evaluar las capacidades individuales. Los factores principales que deberían ser considerados son la competencia, eficiencia y rigurosidad, junto con las habilidades de comunicar y cooperar. La experiencia del programador no debería ser considerada aquí, ya que se incluye en los drivers de costo APEX, LTEX, y PLEX. La tabla 6.29 muestra los valores posibles para el driver de costo PCAP.

Tabla 6.29: Niveles del driver de costo ACAP.

PCAP Descripción:	Percentil 15º	Percentil 35º	Percentil 55º	Percentil 75º	Percentil 90º	
Rangos	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Multiplicador de esfuerzo	1.34	1.15	1.00	0.88	0.76	-

6.4.3.2.1.3.3 Continuidad del personal (PCON)

La escala de valoración para el driver de costo PCON se encuentra en términos del rendimiento o productividad anual del personal del proyecto: desde un 3% para una continuidad Muy Alta, a un 48% que es una continuidad Muy Baja. La tabla 6.30 muestra los valores que puede tomar el driver de costo PCON.

Tabla 6.30: : Niveles del driver de costo PCON.

PCON Descripción:	48% al año	24% al año	12% al año	6% al año	3% al año	
Rangos	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Multiplicador de esfuerzo	1.29	1.12	1.00	0.90	0.81	-

6.4.3.2.1.3.4 Experiencia en la aplicación (APEX)

El valor de este driver de costo depende del nivel de experiencia en aplicaciones que tenga el equipo que está desarrollando el sistema o subsistema software. El rango de valores posibles para este driver de costo, se define en términos del nivel de experiencia equivalente del

equipo de trabajo con este tipo de aplicaciones. Un valor Muy Bajo es para un nivel de experiencia en la aplicación menor a dos meses, mientras que un nivel Muy Alto es para un valor de experiencia de 6 años o más. La tabla 6.31 muestra los valores que toma el driver de costo APEX.

Tabla 6.31: Niveles del driver de costo APEX.

APEX Descripción:	≤ 2 meses	6 meses	1 año	3 años	6 años	
Rangos	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Multiplicador de esfuerzo	1.22	1.10	1.00	0.88	0.81	-

6.4.3.2.1.3.5 Experiencia en la plataforma (PLEX)

El modelo de Post-Arquitectura amplía la influencia en la productividad de la experiencia en la plataforma, PLEX, reconociendo la importancia del uso de plataformas más poderosas, incluyendo interfaces de usuario más gráficas, bases de datos, redes, y capacidades de middleware distribuido. La tabla 6.32 muestra los valores que puede tomar este driver de costo.

Tabla 6.32: Niveles del driver de costo PLEX.

PLEX Descripción:	≤ 2 meses	6 meses	1 año	3 años	6 años	
Rangos	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Multiplicador de esfuerzo	1.19	1.09	1.00	0.91	0.85	-

6.4.3.2.1.3.6 Experiencia en lenguaje y herramienta (LTEX)

Esta es una medida del nivel de experiencia en lenguajes de programación y herramientas de software que posee el equipo que está desarrollando el sistema o subsistema del proyecto. El desarrollo de software incluye el uso de herramientas que realizan representaciones de requerimientos y diseño, administración de la configuración, extracción de documentos, administrador de librerías, verificadores de consistencia, control y planificación, etc. Además de la experiencia en el lenguaje de programación del proyecto, la experiencia en el conjunto de herramientas de soporte también afecta el esfuerzo de desarrollo. Se da un valor Bajo a la experiencia inferior a 2 meses. Mientras que a una experiencia de 6 años o más se le da un valor Muy Alto. La tabla 6.33 muestra los valores que puede tomar LTEX.

Tabla 6.33: Niveles del driver de costo LTEX.

LTEX Descripción:	≤ 2 meses	6 meses	1 año	3 años	6 años	
Rangos	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**1.20**	**1.09**	**1.00**	**0.91**	**0.84**	**-**

6.4.3.2.1.4 Factores del proyecto

Los factores del proyecto influyen sobre el esfuerzo estimado a través de factores tales como: el uso de herramientas modernas de software, el emplazamiento del equipo de desarrollo, y la compresión de la planificación del proyecto.

6.4.3.2.1.4.1 Uso de herramientas de software (TOOL)

Las herramientas de software han mejorado significativamente desde la década del 70. El rango de herramientas va desde simples editores de código evaluados con un valor Muy Bajo, a herramientas de administración del ciclo de vida integradas con un valor Muy Alto. La tabla 6.34 muestra los valores que puede tomar este driver de costo.

Tabla 6.34: Niveles del driver de costo TOOL.

TOOL Descripción:	Editar, Codificar, Debug	Simple, Frontend, Backend CASE, Poca integración	Herramientas de ciclo de vida básicas, moderadamente integradas	Herramientas de ciclo de vida maduras, robustas, moderadamente integradas	Herramientas de ciclo de vida proactivas, maduras y robustas. Bien integradas con proceso, Métodos y reutilización	
Rangos	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**1.17**	**1.09**	**1.00**	**0.90**	**0.78**	**-**

6.4.3.2.1.4.2 Desarrollo en emplazamientos múltiples (SITE)

Determinar el valor de este driver de costo involucra la evaluación de dos factores los cuales serán promediados en base al juicio del evaluador: lugar de emplazamiento (desde completamente disponibles a distribución internacional) y apoyo a las comunicaciones (desde correo tradicional y algo de acceso a teléfono a multimedia completamente interactiva).

Por ejemplo, si un equipo está completamente disponible, no necesita una comunicación por multimedia interactiva para lograr un valor Extra Alto. Sólo bastaría con una comunicación por correo electrónico con un acceso de banda angosta. La tabla 6.35 muestra la distribución de valores posibles para ambos factores, los cuales deben ser promediados basándose en un juicio de valor.

Tabla 6.35: Niveles del driver de costo SITE.

SITE Descripción de la disposición:	Internacional	Inter-ciudad e Inter-empresa	Inter-ciudad o Inter-empresa	La misma ciudad o comuna	Mismo complejo o edificio	Totalmente disponible
SITE Descripción de la comunicación:	Algo de teléfono y correo	Teléfono individual, fax	Correo electrónico de banda angosta	Comunicación electrónica de banda ancha	Comunicación electrónica de banda ancha, video conferencia ocasional	Multimedia Interactiva
Rangos	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Multiplicador de esfuerzo	1.22	1.09	1.00	0.93	0.86	0.80

6.4.3.2.1.4.3 Tiempo requerido de desarrollo (SCED)

Este valor mide las restricciones de planificación impuestas sobre el equipo de desarrollo que lleva a cabo el proyecto. SCED está definido en términos del porcentaje de tiempo en que se extiende o acelera un proyecto con respecto su planificación Nominal, para un esfuerzo dado.

La planificación acelerada tiende a producir más esfuerzo en las fases más tempranas para eliminar el riesgo y refinar la arquitectura, mientras que un mayor esfuerzo en las últimas fases para lograr más pruebas y documentación en paralelo. En la tabla 6.36, la compresión de la planificación SCED en un 75% se considera Muy Baja. Un alargamiento de un 160% se considera Muy Alto. El alargamiento no incrementa o decrementa el esfuerzo.

SCED es el único driver de costo que se usa para describir los efectos de la compresión o expansión del tiempo para el proyecto en su totalidad. Los factores de escala también se usan para describir el proyecto como un todo. Todos los otros drivers de costo se utilizan para describir cada módulo en un proyecto de múltiples módulos. La tabla 6.36 muestra los valores posibles de SCED.

Tabla 6.36: Niveles del driver de costo SCED.

SCED Descripción:	75% del valor nominal	85% del valor nominal	100% del valor nominal	130% del valor nominal	160% del valor nominal	
Rangos	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Multiplicador de esfuerzo	1.43	1.14	1.00	1.00	1.00	-

6.4.3.2.2 Drivers de costo del modelo de Diseño Anticipado.

El modelo de diseño anticipado usa KSLOC o puntos de función sin ajustar para su tamaño (Size).

Los UFPs (unadjusted function points) se convierten a su equivalente en SLOC, y luego se les divide por 1000 para ser transformados a KSLOC. La aplicación de los factores de escala exponenciales son los mismos tanto para el modelo de Diseño Anticipado como para el modelo de Post-Arquitectura. Sin embargo, en el modelo de diseño anticipado se utiliza un conjunto reducido de drivers de costo.

Los drivers de costo del modelo de Diseño Anticipado se obtienen combinando los drivers del modelo Post-Arquitectura.

Al igual que el modelo Post-Arquitectura, cuando la evaluación de un driver quede entre dos niveles, el resultado siempre se aproximará al nivel Nominal. Por ejemplo si un driver de costo se encuentra a mitad de camino entre Muy Bajo (Very Low) y Bajo (Low), se seleccionará Low.

La ecuación de esfuerzo es la Ec. 6.21, pero el número de multiplicadores de esfuerzo se redujo a 7 ($n = 7$) :

$$PM = A \times (Size)^E \times \prod_{i=1}^{n} EM_i \qquad Ec.6.21$$

$donde$:

$$A = 2{,}94$$

En la tabla 6.37 se entrega un resumen de los multiplicadores de esfuerzo.

Tabla 6.37: Multiplicadores de esfuerzo para el modelo de Diseño Anticipado y sus multiplicadores de esfuerzo combinados equivalentes en modelo de Post-Arquitectura.

Drivers de costo del modelo de Diseño Anticipado	Drivers de costo combinados del modelo Post-Arquitectura
PERS	ACAP, PCAP, PCON
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STORE, PVOL
PREX	APEX, PLEX, LTEX
FCIL	TOOL, SITE
SCED	SCED

6.4.3.2.2.1 Enfoque global

Este enfoque se usa para hacer un mapeo completo del conjunto de drivers de costos del modelo de Post-Arquitectura y sus escalas de valoración y llevarlos al modelo de Diseño Anticipado. Involucra el uso y combinación de equivalencias numéricas de los niveles de

valoración. Específicamente, un valor Muy Bajo en un driver de costo del modelo de Post-Arquitectura le corresponde un valor 1; un Bajo un 2; Nominal es 3; Alto es 4; Muy Alto es 5; y Extra Alto es 6. Para los drivers de costo del modelo de Diseño Anticipado, se suman los valores de los drivers de costo que contribuyen desde el modelo de Post-Arquitectura. Y los resultados totales, ponen en un modelo de Diseño Anticipado expandido, con una escala de valoración que va desde Extra Bajo a Extra Alto. La escala de valoración de este modelo, siempre tiene un valor Nominal total igual a la suma de los valores Nominales de los elementos del modelo de Post-Arquitectura que contribuyeron.

6.4.3.2.2.1.1 Capacidad del personal (PERS)

El driver de costo PERS del modelo de Diseño Anticipado combina los siguientes drivers de costo del modelo de Post-Arquitectura: capacidad del analista (ACAP), capacidad del programador (PCAP) y continuidad del personal (PCON). Cada uno de ellos con un valor de escala que va desde Muy bajo con un valor igual a 1, a Muy Alto con un valor igual a 5. La suma de sus valores numéricos produce valores entre 3 y 15. Estos valores se ponen sobre la escala junto a los valores del driver PERS correspondientes como se muestra en la tabla 6.38. Los multiplicadores de esfuerzo asociados se derivan a partir de los multiplicadores de esfuerzo de los drivers ACAP, PCAP y PCON, promediando el producto de cada combinación de multiplicadores de esfuerzo asociados con el valor de diseño anticipado dado.

Los multiplicadores de esfuerzo para PERS, así como para cualquier otro driver de costo del modelo de Diseño Anticipado, se derivan a partir de los drivers de costo del modelo de Post-Arquitectura promediando sus valores correspondientes a los multiplicadores de esfuerzo (en este caso ACAP, PCAP, PCON).

Tabla 6.38: Niveles del driver de costo PERS.

PERS							
Suma de ACAP, PCAP, PCON	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
Percentil Combinado de ACAP y PCAP	20%	35%	45%	55%	65%	75%	85%
Productividad Anual del personal (PCON)	45%	30%	20%	12%	9%	6%	4%
Rangos	**Extra Bajo**	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**2.12**	**1.62**	**1.26**	**1.00**	**0.83**	**0.63**	**0.5**

6.4.3.2.2.1.2 Complejidad y confiabilidad del producto (RCPX)

Ver tabla 6.39.

Tabla 6.39: Niveles del driver de costo RCPX.

RCPX							
Suma de RELY, DATA, CPLX, DOCU	5, 6	7, 8	9 - 11	12	13 - 15	16 - 18	19 - 21
Complejidad del Producto	Muy Poca	Poca	Algo	Básica	Fuerte	Muy Fuerte	Compleja
Tamaño de la Base de Datos	Muy simple	Simple	Algo	Moderado	Complejo	Muy Complejo	Extremadamente Complejo
Rangos	**Extra Bajo**	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**0.49**	**0.60**	**0.83**	**1.00**	**1.33**	**1.91**	**2.72**

6.4.3.2.2.1.3 Desarrollado para reutilización (RUSE)

Este driver de costo es exactamente igual al planteado en el modelo de Post-Arquitectura.

Ver tabla 6.40.

Tabla 6.40: Niveles del driver de costo PDIF.

PDIF					
Suma de TIME, STOR, PVOL	8	9	10 - 12	13 - 15	16, 17
Restricciones de tiempo y almacenamiento	≤ 50 %	≤ 50 %	65%	80%	90%
Volatilidad de la plataforma	Muy Estable	Estable	Algo Volátil	Volátil	Altamente Volátil
Rangos	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**0.87**	**1.00**	**1.29**	**1.81**	**2.61**

6.4.3.2.2.1.4 Experiencia del personal (PREX)

Ver tabla 6.41.

Tabla 6.41: Niveles del driver de costo PREX.

PREX							
Suma de APEX, PLEX, LTEX	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
Experiencia en aplicaciones, plataformas, lenguajes y herramientas	≤ 3 meses	5 meses	9 meses	1 año	2 años	4 años	6 años
Rangos	**Extra Bajo**	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**1.59**	**1.33**	**1.22**	**1.00**	**0.87**	**0.74**	**0.62**

6.4.3.2.2.1.5 Instalaciones, dependencias (FCIL)

Ver tabla 6.42.

Tabla 6.42: Niveles del driver de costo FCIL.

FCIL							
Suma de TOOL y SITE	2	3	4, 5	6	7, 8	9, 10	11
Apoyo a TOOL	Mínimo	Algo	Conjunto de herramientas case simples	Herramientas de ciclo de vida básicas	Bueno, moderadamente integrado	Fuerte, moderadamente integrado	Fuerte, bien integrado
Condiciones de emplazamientos múltiples	Apoyo débil al desarrollo en emplaza-mientos múltiples	Algo de apoyo al desarrollo complejo en emplaza-mientos múltiples	Algo de apoyo al desarrollo moderadamente complejo en emplaza-mientos múltiples	Apoyo básico al desarrollo moderadamente complejo en emplaza-mientos múltiples	Fuerte apoyo al desarrollo moderadamente complejo en emplaza-mientos múltiples	Fuerte apoyo al desarrollo simple en emplaza-mientos múltiples	Apoyo muy fuerte al desarrollo en múltiples emplaza-mientos simple u ordenado
Rangos	**Extra Bajo**	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**1.43**	**1.30**	**1.10**	**1.0**	**0.87**	**0.73**	**0.62**

6.4.3.2.2.1.6 Tiempo requerido de desarrollo (SCED)

Este driver de costo es exactamente igual al planteado en el modelo de Post-Arquitectura.

6.4.4 Estimación de esfuerzo de múltiples módulos

A menudo los sistemas de software están compuestos de múltiples subsistemas o componentes. Se puede usar COCOMO II para estimar el esfuerzo y tiempo para múltiples componentes. **La técnica descrita es para un nivel de subcomponente. Para ver múltiples niveles de subcomponentes referirse a [20].**

El modelo COCOMO II no utiliza la suma de las estimaciones para cada componente, ya que esto ignoraría el esfuerzo debido a la integración de los componentes.

El método de cálculo de COCOMO II para múltiples módulos para n número de módulos tiene los siguientes pasos:

PASO 1:

Sume el tamaño (Size), para todos los componentes, $Size_i$, para obtener un tamaño agregado, ver ecuación Ec.6.22.

$$Size_{Aggregate} = \sum_{i=1}^{n} Size_i \qquad Ec.6.22$$

PASO 2:

Aplique los drivers que se encuentran a nivel de proyecto, los factores de escala y el driver de costo SCED al tamaño agregado ($Size_{Aggregate}$), para obtener el esfuerzo básico global para el proyecto total, PM_{Basic}, ver ecuación Ec.6.23. Los factores de escala se discutieron anteriormente en la sección 6.4.3.1 y SCED en la sección 6.4.3.2.1.4.3.

$$PM_{BASIC} = A \times \left(Size_{Aggregate}\right)^{E} \times SCED \qquad Ec.6.23$$

PASO 3:

Determinar el esfuerzo básico de cada componente, $PM_{BASIC,i}$, prorrateando el esfuerzo básico global de cada componente, basándose en su contribución al tamaño agregado ($Size_{Aggregate}$). Ver ecuación Ec.6.24.

$$PM_{BASIC,i} = PM_{BASIC} \times \left(\frac{Size_i}{Size_{Aggregate}} \right) \qquad Ec.6.24$$

PASO 4:

Aplicar los drivers de costo a nivel de componente (excluyendo SCED) al esfuerzo básico de cada componente. Ver ecuación Ec.6.25.

$$PM_i = PM_{BASIC,i} \times \sum_{j=1}^{16} EM_j \qquad Ec.6.25$$

PASO 5:

Sume cada esfuerzo de componente para obtener el esfuerzo agregado ($PM_{Aggregate}$), para el proyecto total. Ver ecuación Ec.6.26.

$$PM_{Aggregate} = \sum_{i=1}^{n} PM_i \qquad Ec.6.26$$

PASO 6:

El tiempo se estima repitiendo los pasos del 2 al 5 sin el driver de costo SCED usado en el paso 2. Luego utilizando este esfuerzo agregado modificado, $PM'_{Aggregate}$, el tiempo se deriva utilizando la siguiente ecuación Ec. 6.27.

$$TDEV = \left[C \times \left(PM_{NS}\right)^{D+0.2(E-B)}\right] \times \frac{SCED\%}{100} \qquad Ec.6.27$$

6.4.5 Estimación del tiempo

La versión inicial de COCOMO II, provee de una capacidad de estimación de tiempo similar a las capacidades de estimación disponibles en COCOMO 81 y Ada COCOMO. La ecuación base para el cálculo de tiempo para COCOMO II en sus modelos de Diseño Anticipado y Post-Arquitectura está dada por la ecuación Ec.6.28:

$$TDEV = \left[C \times \left(PM_{NS}\right)^{(D+0.2(E-B))}\right] \times \frac{SCED\%}{100} \qquad Ec.6.28$$

$$donde: C = 3{,}67$$
$$D = 0{,}28$$
$$B = 0{,}9$$

En la ecuación anterior, Ec. 6.28, C es el coeficiente de TDEV que debe ser calibrado. PM_{NS} es el multiplicador de esfuerzo estimado (PM), excluyendo el modificador de esfuerzo SCED como se indica en la ecuación Ec. 6.29.

$$PM_{NS} = A \times Size^{E} \times \prod_{i=1}^{n} EM_i \qquad Ec.6.29$$

$$donde: E = B + 0{,}01 \times \sum_{j=1}^{5} SF_j$$

D es uno de los exponentes en la ecuación de TDEV y también puede ser calibrado. E es el esfuerzo de escala exponencial que es derivado de la suma de los factores de escala del proyecto y B corresponde al factor de escala, calibrado, que se discute en la sección 6.6.2.

SCED% es el porcentaje de compresión/expansión en la escala de rangos del multiplicador de esfuerzo SCED.

El tiempo de desarrollo, TDEV, es el tiempo calendario en meses entre los hitos **LCO** (Life Cycle Objetives) y el **IOC** (Initial Operational Capability), para modelos de ciclos de vida MBASE/RUP[34] o SRR (Software Requirements Review) y SAR (Software Acceptance Review) para el modelo en cascada. Para el modelo en cascada, esto va desde la determinación de la línea base de un producto hasta la completitud de una actividad de aceptabilidad que certifica que el producto satisface sus requerimientos.

Como se dijo anteriormente, para los modelos de ciclos de vida MBASE/RUP, COCOMO II abarca el intervalo de tiempo entre los hitos LCO e IOC, ver figura.

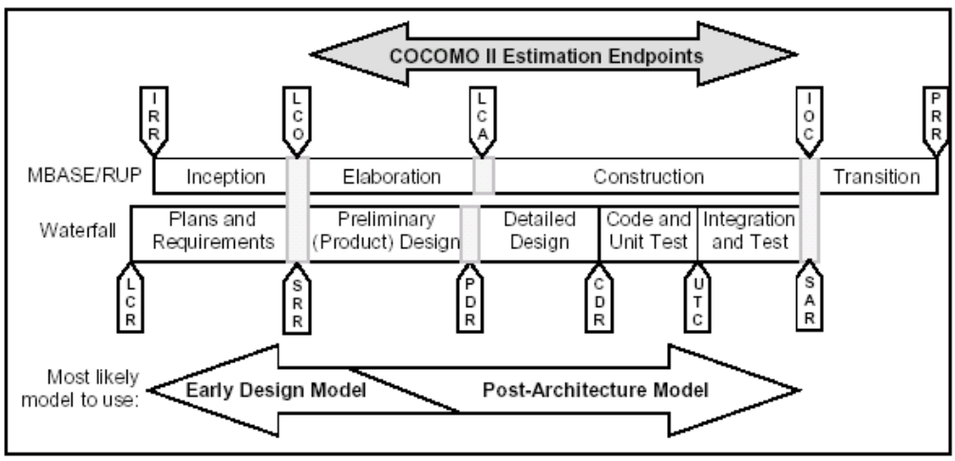


Figura 6.1: Intervalo de validez del modelo COCOMO. Extraído del manual de COCOMO II.

Junto con las nuevas versiones de COCOMO II, se dispondrá de un modelo de estimación en un intervalo de tiempo cada vez más extenso.

[34] **MBASE/RUP:** Model Based Architecture Software Engineering / Rational Unified Process

6.4.6 Mantenimiento del software

El mantenimiento del software se define como el proceso de modificar el software existente sin cambiar sus funcionalidades principales. El supuesto hecho por el modelo COCOMO II, es que el costo de mantención del software tiene, por lo general, los mismos drivers de costo que los utilizados en el desarrollo de software.

La mantención incluye el rediseño y codificación de una pequeña porción del producto original, el rediseño y desarrollo de interfaces, y modificaciones menores a la estructura del producto.

La mantención puede ser clasificada como una actualización, o como una reparación. La reparación del producto se puede dividir a su vez en una reparación correctiva, cuando hay fallas en el procesamiento, rendimiento, o implementación; o en una reparación adaptativa, cuando hay cambios en el procesamiento o entorno de datos; o mantenimiento de perfeccionamiento, cuando se desea mejorar el rendimiento o la mantenibilidad. El tamaño de mantenimiento, $Size_M$, ha sido descrito en la sección 6.4.2.6. La ecuación que lo representa se describe a continuación. Ver ecuación Ec.6.30.

$$(Size)_M = [(BaseCodeSize) \times MCF] \times MAF \qquad Ec.6.30$$

Existen ciertas consideraciones especiales para usar COCOMO II, en el mantenimiento de software.

Consideración 1:

El driver de costo SCED (Requiered Development Schedule), no se usa en la estimación de esfuerzos para mantenimiento. Esto debido a que el ciclo de mantenimiento tiene, por lo general, un tiempo fijo.

Consideración 2:

El driver de costo RUSE (Required Reusability), no se utiliza en la estimación de esfuerzo para mantenimiento. Esto debido al tiempo extra requerido para mantener la reusabilidad de un

componente, ya que éste es difícilmente equiparado por el esfuerzo de mantención, que ha sido reducido debido al diseño meticuloso, documentación y prueba de componentes.

Consideración 3:

El driver de costo RELY (Required Software Reliability) tiene un conjunto distinto de multiplicadores de esfuerzo para el mantenimiento. Para el caso del mantenimiento, el driver de costo RELY, depende de la confiabilidad requerida bajo la cual fue desarrollado el producto. Si el producto fue desarrollado con baja confiabilidad, el reparar una falla oculta en él, requerirá de un mayor esfuerzo. Si por el contrario, el producto fue desarrollado con una confiabilidad Muy Alta (Very High), entonces el esfuerzo requerido para mantener ese nivel de confiabilidad, será por sobre el valor Nominal. La tabla 6.43 muestra los valores posibles para el driver de costo RELY en mantenimiento.

Tabla 6.43: Rango de valores para el driver de costo RELY para mantenimiento.

RELY Descripción:	Ligero Inconveniente	Bajo, Perdidas fácilmente recobrables	Moderado, Perdidas fácilmente recobrables	Alto, Perdidas Financieras	Riesgo a vidas humanas	
Rangos	**Muy Bajo**	**Bajo**	**Nominal**	**Alto**	**Muy Alto**	**Extra Alto**
Multiplicador de esfuerzo	**1.23**	**1.10**	**1.00**	**0.99**	**1.07**	**-**

Consideración 4:

El valor en el exponente, E, se aplica al número de KSLOC que se haya cambiado, esto incluye a los KSLOC agregados y modificados, pero no incluye a los borrados, todo esto en lugar de aplicarlo a los KSLOC del sistema completo.

El tamaño de mantenimiento efectivo, $(Size)_M$, se ajusta a través del factor de ajuste de mantenimiento (MAF: Maintenance Adjustment Factor), para así considerar los efectos del sistema completo. La ecuación de estimación de esfuerzo de mantenimiento es la misma que utiliza el modelo de desarrollo de Post-Arquitectura de COCOMO II, sin incluir los drivers de costo SCED o RUSE. El esfuerzo de mantenimiento queda expresado a través de la siguiente ecuación, Ec.6.31.

$$PM_M = A \times (Size_M)^E \times \prod_{i=1}^{15} EM_i \qquad Ec.6.31$$

El enfoque de COCOMO II difiere del de COCOMO 81, en cuanto a la estimación de esfuerzos de mantenimiento, permitiéndole al Ingeniero de Software utilizar la duración de la actividad de mantenimiento que desee, TM.

Por otro lado, el nivel promedio de Staff requerido, SFPM, puede ser obtenido a través de la siguiente ecuación Ec.6.32:

$$FSPM = \frac{PM_M}{TM} \qquad Ec.6.32$$

Se ha explicado en forma detallada el modelo COCOMO II. Sin embargo, aún resta por explicar el proceso de calibración del modelo. Por lo general, este proceso se realizará mediante una herramienta de apoyo a la calibración llamada Calico. El proceso de calibración del modelo COCOMO II, busca determinar los valores correctos para las constantes A, B, C y D. Estos valores se pueden encontrar mediante regresiones múltiples usando información de proyectos anteriores.

6.5 Distribución de fases y actividades

En la tabla 6.44, se muestra el resultado de los valores para la distribución de fases y actividades para el modelo COCOMO II MBASE/RUP. Las primeras dos líneas, muestran el porcentaje de tiempo de desarrollo que tomará cada una de las fases para el modelo Rational y COCOMO II. La única diferencia entre ellos, es que el modelo Rational considera que la suma de porcentajes debe ser 100% para el conjunto de las cuatro fases, mientras que el modelo COCOMO II considera la suma de porcentajes igual a 100% sólo en las fases de Elaboración y Construcción. Esto es así debido a que el ámbito y la duración de las fases de Inicio y Transición son mucho más variables, y además existe menos información disponible para calibrar los modelos de estimación para estas fases. Esto significa, que la distribución del modelo COCOMO

II para las fases de elaboración y construcción es considerablemente más exacto y robusto que el modelo de distribución de fases de Rational.

La tercera y cuarta línea, muestran la distribución de esfuerzo para cada una de las fases. La suma de los porcentajes para las cuatro fases es de 125% para el tiempo de desarrollo y 118% para el esfuerzo.

Tabla 6.44: Distribución de fases y actividades para el modelo COCOMO II MBASE/RUP.

	Fase	Inicio	Elaboración	Construcción	Transición
1	Tiempo Rational	10	30	50	10
2	Tiempo COCOMO2	12.5	37.5	62.5	12.5
3	Esfuerzo Rational	5	20	65	10
4	Esfuerzo COCOMO2	6	24	76	12
5	Administración	14	12	10	14
6	Entorno	10	8	5	5
7	Requerimientos	38	18	8	4
8	Diseño	19	36	16	4
9	Implementación	8	13	34	19
10	Evaluación	8	10	24	24
11	Despliegue	3	3	3	30

Desde la línea cinco a la línea once, es posible observar la distribución de esfuerzo por actividad para cada una de las fases del RUP.

Así entonces es posible determinar el porcentaje de consumo de esfuerzo que tendrá una determinada actividad durante el ciclo de vida del proyecto software.

Por ejemplo: la actividad de administración se estima consumirá el 14% del esfuerzo durante la fase de Inicio, el 12% durante la Elaboración, 10% durante la Construcción y 14% durante la Transición.

Entonces para las cuatro fases, la actividad de administración consumirá:

$$(14\%)\cdot(6\%)+(12\%)\cdot(24\%)+(10\%)\cdot(76\%)+(14\%)\cdot(12\%)=13\%$$

Además, estos valores se pueden utilizar para determinar el nivel de personal para cada una de las fases.

Por ejemplo, si se supone un proyecto con las siguientes características:

$$KSLOC = 3.57 \qquad EAF = 5.05 \qquad SSF = 18.19$$

$$PM_{real} = 58.87\left[mes - \text{hom}\, bre\right]$$

$$TDEV = 5.58\left[meses\right]$$

$$STAFF = \frac{PM_{real}}{TDEV} = \frac{58.87\left[mes - \text{hom}\, bre\right]}{5.58\left[meses\right]} \approx 10.55\left[personas\right]$$

El cálculo del esfuerzo, la duración y el nivel promedio de staff para cada una de las fases se realizaría como sigue, ver tabla 6.45:

Tabla 6.45: Cálculo del esfuerzo, duración y nivel promedio de staff para cada fase.

	Inicio	Elaboración	Construcción	Transición
PM	(0.06)(58.87)=3.5322	**(0.24)(58.87)=14.1288**	**(0.76)(58.87)=44.7412**	(0.12)(58.87)=7.0644
TDEV	(0.125)(5.58)=0.6975	**(0.375)(5.58)=2.0925**	**(0.625)(5.58)=3.4875**	(0.125)(5.58)=0.6975
STAFF	5.064	**6.752**	**12.829**	10.128

Es importante tener en cuenta que estos valores dependen de los factores de escala y los drivers de costo del proyecto.

Además, es posible utilizar los valores de la tabla 6.44, para bosquejar una estimación de lo que estarán haciendo estas 10.55 personas (11 personas) durante cada una de las fases. Esto se determina multiplicando el consumo de esfuerzo por actividad en cada fase por el número estimado de personal, ver tabla 6.46.

Tabla 6.46: Personal por actividad en cada fase.

	Fase	Inicio	Elaboración	Construcción	Transición
5	Administración	(0.14)(10.55)	(0.12) (10.55)	(0.10) (10.55)	(0.14) (10.55)
6	Entorno	(0.10) (10.55)	(0.08) (10.55)	(0.05) (10.55)	(0.05) (10.55)
7	Requerimientos	(0.38) (10.55)	(0.18) (10.55)	(0.08) (10.55)	(0.04) (10.55)
8	Diseño	(0.19) (10.55)	(0.36) (10.55)	(0.16) (10.55)	(0.04) (10.55)
9	Implementación	(0.08) (10.55)	(0.13) (10.55)	(0.34) (10.55)	(0.19) (10.55)
10	Evaluación	(0.08) (10.55)	(0.10) (10.55)	(0.24) (10.55)	(0.24) (10.55)
11	Despliegue	(0.03) (10.55)	(0.03) (10.55)	(0.03) (10.55)	(0.30) (10.55)

Lo anterior se resume en la tabla 6.47.

Tabla 6.47: Nivel de Staff promedio por actividad en cada fase del ciclo de desarrollo.

	Fase	Inicio	Elaboración	Construcción	Transición
5	Administración	1.477	1.266	1.055	1.477
6	Entorno	1.055	0.844	0.5275	0.5275
7	Requerimientos	4.009	1.899	0.844	0.422
8	Diseño	2.0045	3.798	1.688	0.422
9	Implementación	0.844	1.3715	3.587	2.0045
10	Evaluación	0.844	1.055	2.532	2.532
11	Despliegue	0.3165	0.3165	0.3165	3.165

A la tabla de distribución de fases y actividades también se le pueden multiplicar los costos promedios de desarrollo, considerándose entonces un buen punto de partida para elaborar el presupuesto de un proyecto software.

Es importante determinar el costo de una actividad por mes. En la versión COCOMO II.1999, este valor se determinaba a través de la columna llamada **Labor Rate Column**. Esta variable debe contener la cantidad de dinero que se paga a un desarrollador que trabaja en un determinado módulo o componente cada mes, es decir $\left(\frac{\$}{mes}\right)$.

El valor del **Labor Rate** para un determinado módulo o componente se puede multiplicar para el esfuerzo estimado que conlleva realizar dicho módulo. Además, el modelo COCOMO II supone que el esfuerzo en [mes-hombre] contempla 152 horas de trabajo al mes, por lo que se debe ajustar este valor a 160 horas de trabajo al mes válidas en Chile.

A este esfuerzo se le llamará **Esfuerzo Normalizado** y se obtendrá a través de una regla de tres simple. Ver ecuación Ec.6.35.

$$PM_{Normalizado} = \frac{(Horas\ de\ Trabajo\ al\ mes)\cdot(PM)}{152} \qquad Ec.6.35$$

Es importante entender que estos números sólo bosquejan un buen punto de partida, por lo que se aproximan a los valores reales que se utilizan en la administración del proyecto. Además, hay que considerar que todo proyecto tiene factores especiales que deben ser considerados para ajustar dicho bosquejo.

Entonces, el esfuerzo normalizado estará dado por: $PM_{Normalizado} = \frac{160 \cdot 58{,}87}{152} = 61{,}9684$.

Por lo que si el costo mensual del personal a cargo del desarrollo del software o módulo del software es X, entonces el costo del desarrollo de éste estará dado por la ecuación Ec.6.36:

$$Costo = X \cdot PM_{Normalizado} \qquad Ec.6.36$$

El esfuerzo normalizado se puede obtener también para cada fase, por lo que sería posible determinar el costo de desarrollo por fase de cada componente o módulo del proyecto software.

6.6 Calibración del Modelo COCOMO II

Se discutirá la forma de calibrar las diversas variables de entorno del modelo COCOMO II.

Los estudios de la información usada para la calibración de todos los parámetros del modelo Post-Arquitectura COCOMO II, han demostrado que el modelo aumenta su exactitud significativamente cuando se ha calibrado para una empresa u organización determinada. Todo lo que se hizo, en los estudios de calibración del modelo, fue calibrar la constante A en la ecuación de esfuerzo.

Las principales razones para realizar la calibración son el hecho de considerar, o tomar en cuenta, las distribuciones de actividad y productividad del entorno local de desarrollo. En la figura 6.4, se muestra un gráfico para ocho proyectos ficticios[35]. El efecto logrado al calibrar la constante A, es aumentar o disminuir los valores retornados por la función de esfuerzo estimado (PM est.) en relación al esfuerzo real (PM real o PM en el gráfico).

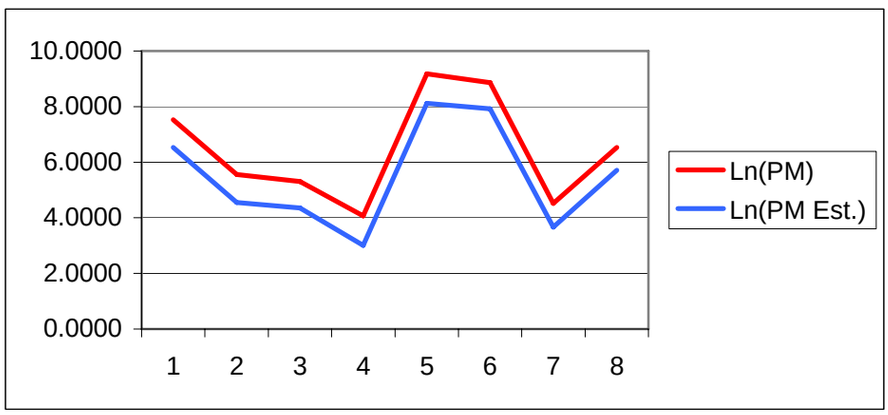


Figura 6.2: Gráfico que muestra la diferencia entre curvas de esfuerzo del modelo COCOMO II y la curva del esfuerzo estimado cuando A = 1.

6.6.1 Calibración de la constante A

La calibración al entorno local significa ajustar la constante en el modelo, en este caso A, ver ecuación Ec. 6.4. El caso de la calibración de las demás constantes en el modelo COCOMO II, se verá más adelante, ver secciones 6.6.2 y 6.6.3.

[35]**Proyectos ficticios:** Extraídos desde la planilla de cálculo para la calibración del modelo COCOMO II.1999.

Según Boehm, existe más de una forma de calibrar las constantes pertenecientes al modelo COCOMO II.

La primera técnica descrita para calcular la constante A, utiliza logaritmos naturales, y se recomiendan por lo menos el uso de 5 puntos de datos.

En este caso se utilizarán los 8 puntos de datos descritos por la tabla 6.48. En ella PM real se considera el esfuerzo realizado entre el término del análisis de requerimientos (correspondiente al fin de la fase de inicio) y el término de la integración y prueba del software (correspondiente al fin de la fase de construcción). Es decir, PM real, corresponde al esfuerzo real realizado entre los hitos de LCO (Life Cycle Objetives) e IOC (Inicial Operational Capability).

Tabla 6.48: Puntos de datos para 8 proyectos históricos realizados por la empresa u organización.

Proyecto	PM real	KSLOC	EAF	SSF	TDEV
1	1,854.55	134.47	1.89	29.28	19.69
2	258.51	132.00	0.49	16.72	8.77
3	201.00	44.03	1.06	22.48	8.62
4	58.87	3.57	5.05	18.19	5.58
5	9,661.02	380.80	3.05	26.77	32.80
6	7,021.28	980.00	0.92	25.21	28.76
7	91.67	11.19	2.45	23.50	6.77
8	689.66	61.56	2.38	26.48	13.56

Las actividades deben ser las mismas correspondientes a las descritas en el capítulo 3, sección 3.4.2.5.

Además del esfuerzo real realizado, también se requiere el tamaño del producto final (KSLOC), los factores de escala del proyecto (SSF: Sumatoria de los factores de escala), y los drivers de costo reflejados en EAF(Effort Adjustment Factor), que corresponde a la multiplicatoria o pitatoria de los multiplicadores de esfuerzo (EM).

A partir de esos valores, se debe generar una estimación del esfuerzo (PM est) usando la ecuación de esfuerzo Ec.6.4. , suponiendo que el valor de la constante A es igual a 1 (A=1). Se obtienen los logaritmos naturales del esfuerzo real (PM real) y el esfuerzo estimado (PM est), y luego se obtiene la diferencia de estos logaritmos (DIFF = LN(PM real) - LN(PM est)) . Se debe estimar el promedio de las diferencias (X), para los n = 8 proyectos, ver ecuación Ec.6.37.

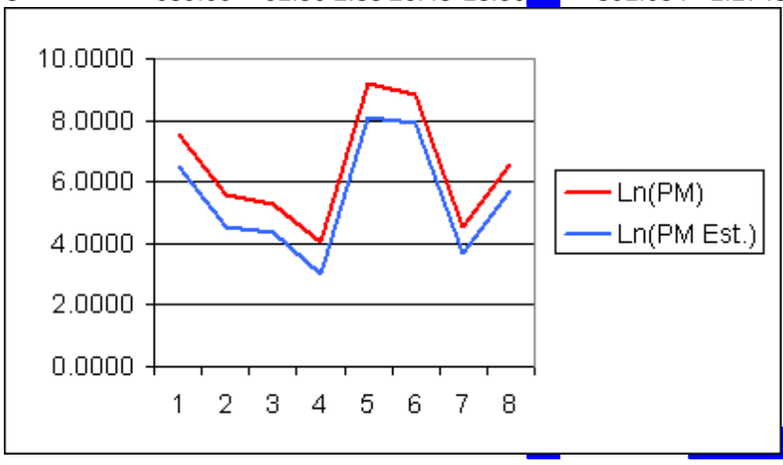
$$X = \frac{\sum_{i=1}^{n} DIF}{n} = \frac{7{,}703}{8} = 0{,}9628 \qquad Ec.6.37$$

Por último, para determinar la constante A, se extrae el antilogaritmo de X: $A = e^{X}$. Ver tabla 6.49.

En la tabla 6.49, Exp representa al exponente E = B + SSF/100, de la ecuación de esfuerzo, donde el valor de B se considera constante e igual a 0.91.

Tabla 6.49: Resumen de los cálculos necesarios para calibrar la constante A.

Proyecto	PM real	KSLOC	EAF	SSF	TDEV	PM Est.	Exp	Ln(PM)	Ln(PM Est.)	DIFF
1	1,854.55	134.47	1.89	29.28	19.69	686.709	1.2028	7.5254	6.5319	0.993
2	258.51	132.00	0.49	16.72	8.77	94.293	1.0772	5.5549	4.5464	1.009
3	201.00	44.03	1.06	22.48	8.62	77.738	1.1348	5.3033	4.3533	0.950
4	58.87	3.57	5.05	18.19	5.58	20.265	1.0919	4.0753	3.0089	1.066
5	9,661.02	380.80	3.05	26.77	32.80	3,338.771	1.1777	9.1759	8.1134	1.062
6	7,021.28	980.00	0.92	25.21	28.76	2,753.542	1.1621	8.8567	7.9206	0.936
7	91.67	11.19	2.45	23.50	6.77	38.895	1.145	4.5182	3.6609	0.857
8	689.66	61.56	2.38	26.48	13.56	301.054	1.1748	6.5362	5.7073	0.829
									Suma DIFF =	7.70319045
									X= (Suma DIFF)/n =	0.96289881
									A =	2.619278259



El ejemplo mostrado anteriormente sugiere que para la estimación de proyectos de desarrollo de software en el entorno local de la empresa u organización se utilice la constante A=2.619278259 en lugar de la constante sin calibrar del modelo COCOMO II (A=2.9).

6.6.2 Calibración de las constantes A y B

A partir de los mismos puntos de dato en la tabla 6.41, es posible determinar valores para las constantes A y B presentes en la ecuación de esfuerzo de desarrollo de proyectos software.

El procedimiento es netamente matemático a partir de la ecuación de esfuerzo, ver ecuaciones Ec.6.38:

$$PM = A \cdot Size^{B+0.01 \cdot SSF} \cdot \prod EM$$

$$PM = A \cdot Size^{B} \cdot Size^{0.01 \cdot SSF} \cdot EAF$$

$$\frac{PM}{Size^{0.01 \cdot SSF} \cdot EAF} = A \cdot Size^{B} \qquad \left| Log() \right.$$

$$Log\left(\frac{PM}{Size^{0.01 \cdot SSF} \cdot EAF} \right) = Log\left(A \cdot Size^{B} \right)$$

$$Log\left(\frac{PM}{Size^{0.01 \cdot SSF} \cdot EAF} \right) = Log(A) + B \cdot Log(Size) \qquad Ec.6.38$$

Con lo que se ha pasado desde una ecuación de forma exponencial a una de forma lineal.

Entonces, se tiene una expresión de la forma de la ecuación Ec.6.39:

$$Y_i = b_0 + b_1 \cdot X_i + e_i \qquad Ec.6.39,$$

donde se ha incorporado e_i que es el error presente en todo modelo de la realidad.

A partir de este punto es posible realizar un análisis de regresión simple que se deriva de lo siguiente:

$$e_i = Y_i - (b_0 + b_1 \cdot X_i)$$

$$\sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} [Y_i - (b_0 + b_1 \cdot X_i)]^2$$

$$\frac{\partial}{\partial b_0}\left(\sum_{i=1}^{n} e_i^2\right) = -2 \cdot \sum (Y_i - b_0 - b_1 \cdot X_i) = 0 \qquad (*)$$

$$\frac{\partial}{\partial b_1}\left(\sum_{i=1}^{n} e_i^2\right) = -2 \cdot \sum X_i \cdot (Y_i - b_0 - b_1 \cdot X_i) = 0 \qquad (**)$$

Simplificando (*) :

$$\sum_{i=1}^{n} Y_i = n \cdot b_0 + b_1 \cdot \sum_{i=1}^{n} X_i \qquad \left| \left(\frac{1}{n}\right)\right.$$

$$\frac{\sum_{i=1}^{n} Y_i}{n} = b_0 + b_1 \cdot \frac{\sum_{i=1}^{n} X_i}{n}$$

$$\overline{Y} = b_0 + b_1 \cdot \overline{X}$$

$$b_0 = \overline{Y} - b_1 \cdot \overline{X} \qquad Ec.6.40$$

Re *emplazando Ec.* 6.40 *en* (**) *y Simplificando* :

$$\sum_{i=1}^{n} X_i Y_i = b_0 \cdot \sum_{i=1}^{n} X_i + b_1 \cdot \sum_{i=1}^{n} X_i^2$$

$$\sum_{i=1}^{n} X_i Y_i = \left(\frac{\sum_{i=1}^{n} Y_i}{n} - b_1 \frac{\sum_{i=1}^{n} X_i}{n} \right) \cdot \sum_{i=1}^{n} X_i + b_1 \cdot \sum_{i=1}^{n} X_i^2$$

$$b_1 = \frac{\sum_{i=1}^{n} X_i Y_i - \frac{\left(\sum_{i=1}^{n} X_i\right)\left(\sum_{i=1}^{n} Y_i\right)}{n}}{\sum_{i=1}^{n} X_i^2 - \frac{\left(\sum_{i=1}^{n} X_i\right)^2}{n}} = \frac{\sum_{i=1}^{n} (X_i - \overline{X}) \cdot (Y_i - \overline{Y})}{\sum_{i=1}^{n} (X_i - \overline{X})^2} \qquad Ec.6.41$$

Siendo :

$$X = Log(Size) \quad \text{e} \quad Y = Log\left(\frac{PM}{EAF \cdot Size^{0.01 \cdot SSF}}\right)$$

Es posible reemplazar los valores en la tabla 6.49, para encontrar los valores de X e Y. Una vez calculados se puede determinar los valores de b_0 y b_1.

La tabla 6.51 muestra un ejemplo de cómo se realizaría el cálculo para los puntos de datos.

Tabla 6.50: Copia de la tabla 6.48.

Proyecto	PM real	KSLOC	EAF	SSF	TDEV
1	1,854.55	134.47	1.89	29.28	19.69
2	258.51	132.00	0.49	16.72	8.77
3	201.00	44.03	1.06	22.48	8.62
4	58.87	3.57	5.05	18.19	5.58
5	9,661.02	380.80	3.05	26.77	32.80
6	7,021.28	980.00	0.92	25.21	28.76
7	91.67	11.19	2.45	23.50	6.77
8	689.66	61.56	2.38	26.48	13.56

Tabla 6.51: Cálculo de los valores de X e Y, beta 0 y beta 1.

Proyecto	Y = Log(PM/EAF*KSLOC^0.01SSF)	X = Log(KSLOC)	XY	X2	Y2
1	2.368515224	2.128625405	5.042	4.531	5.610
2	2.367721306	2.120573931	5.021	4.497	5.606
3	1.908375488	1.643748685	3.137	2.702	3.642
4	0.966072309	0.552668216	0.534	0.305	0.933
5	2.809870571	2.58069694	7.251	6.660	7.895
6	3.128540371	2.991226076	9.358	8.947	9.788
7	1.326622565	1.048674815	1.391	1.100	1.760
8	1.988251809	1.789298611	3.558	3.202	3.953
Sumatoria =	16.86396964	14.85551268	35.292	31.944	39.187
		B= b1 = 0.9124			
		b0 = 0.4137			
		A = 2.5925			

Entonces la ecuación estimada de regresión, que corresponde al esfuerzo estimado está dado por la siguiente ecuación Ec.6.42: $\hat{Y}_i = 0{,}4137 + 0{,}9124 \cdot X_i$ $Ec.6.42$

El valor de la constante B se obtiene directamente como el término que acompaña a X_i en la ecuación de esfuerzo estimado, Ec.6.42. La constante A, sin embargo, se obtiene como el antilogaritmo en base 10 de b_0. Es decir: $A = 10^{b_0} = 10^{0.4137} = 2{,}5925$, aproximadamente.

Así entonces, es posible calcular el esfuerzo estimado, el residuo ($e = Y_i - \hat{Y}_i$) y el cuadrado del residuo (e^2), para cada proyecto. Una vez calculados estos, se puede calcular la varianza residual, ver ecuación Ec.6.43.

$$s^2 = \frac{\sum_{i=1}^{n} e_i^2}{n-2} = \frac{\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i \right)^2}{n-2} \qquad Ec.6.43$$

La constante apropiada, $n-2$, se justifica ya que al tener que estimar los valores de los parámetros b_0 y b_1 se pierden dos grados de libertad.

Mientras más pequeña sea esta varianza residual, la ecuación de estimación de esfuerzo obtenida a través de la regresión se ajustará más al modelo.

La tabla 6.52, muestra el cálculo del esfuerzo estimado, el residuo, el residuo al cuadrado para cada proyecto. Además, se calcula la varianza de la ecuación de estimación de esfuerzo (s2).

Tabla 6.52: Cálculo del esfuerzo estimado, el residuo, el residuo al cuadrado para cada proyecto y la varianza residual.

Proyecto	Y	X	Y est = b0 + b1X	Residuo = (Y-Y est)	Residuo^2
1	2.368515224	2.1286254	2.356	0.012657	0.000160
2	2.367721306	2.12057393	2.349	0.019210	0.000369
3	1.908375488	1.64374869	1.913	-0.005081	0.000026
4	0.966072309	0.55266822	0.918	0.048118	0.002315
5	2.809870571	2.58069694	2.768	0.041543	0.001726
6	3.128540371	2.99122608	3.143	-0.014354	0.000206
7	1.326622565	1.04867481	1.371	-0.043888	0.001926
8	1.988251809	1.78929861	2.046	-0.058004	0.003364
Sumatoria =	16.86396964	14.8555127	16.864	0.000200	0.010093
			s2 =	0.001682147	
			B = b1 =	0.9124	
			b0 =	0.4137	
			A =	2.5924	

6.6.3 Calibración de las constantes C y D

El desarrollo de las ecuaciones de calibración de las constantes C y D para la ecuación de planificación o tiempo de desarrollo (TDEV) del modelo COCOMO II, es similar al realizado para la calibración de las constantes A y B.

El primer paso es linealizar la ecuación de TDEV que tiene forma exponencial, ver ecuaciones Ec.6.44:

$$TDEV = C \cdot PM^{\,D+0{,}002\cdot SSF}$$

$$TDEV = C \cdot PM^{\,D} \cdot PM^{\,0{,}002\cdot SSF}$$

$$\frac{TDEV}{PM^{\,0{,}002\cdot SSF}} = C \cdot PM^{\,D} \qquad \big| Log()$$

$$Log\left(\frac{TDEV}{PM^{\,0{,}002\cdot SSF}}\right) = Log\left(C \cdot PM^{\,D}\right)$$

$$Log\left(\frac{TDEV}{PM^{\,0{,}002\cdot SSF}}\right) = Log(C) + D \cdot Log(PM) \qquad Ec.6.44$$

Entonces, los valores de X e Y son: $X = Log(PM)$ e $Y = Log\left(\frac{TDEV}{PM^{\,0{,}002\cdot SSF}}\right)$

Entonces, es posible calcular los parámetros b_0 y b_1 de la ecuación de regresión lineal. Donde: $b_0 = Log(C)$ y $b_1 = D$.

Además :

$$b_0 = \overline{Y} - b_1 \cdot \overline{X} \qquad Ec.6.45$$

$$b_1 = \frac{\sum_{i=1}^{n} X_i Y_i - \frac{\left(\sum_{i=1}^{n} X_i\right)\left(\sum_{i=1}^{n} Y_i\right)}{n}}{\sum_{i=1}^{n} X_i^{\,2} - \frac{\left(\sum_{i=1}^{n} X_i\right)^2}{n}} = \frac{\sum_{i=1}^{n}\left(X_i - \overline{X}\right)\cdot\left(Y_i - \overline{Y}\right)}{\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2} \qquad Ec.6.46$$

Los ocho puntos de datos permiten calcular los nuevos valores para X e Y que se muestran en la tabla 6.54. Además, esto posibilita el cálculo de b_0 y b_1.

Tabla 6.53: Puntos de datos, copia de la tabla 6.48.

Proyecto	PM real = Y	KSLOC	EAF	SSF	TDEV
1	1,854.55	134.47	1.89	29.28	19.69
2	258.51	132.00	0.49	16.72	8.77
3	201.00	44.03	1.06	22.48	8.62
4	58.87	3.57	5.05	18.19	5.58
5	9,661.02	380.80	3.05	26.77	32.80
6	7,021.28	980.00	0.92	25.21	28.76
7	91.67	11.19	2.45	23.50	6.77
8	689.66	61.56	2.38	26.48	13.56

Tabla 6.54: Cálculo de las constantes C y D.

Proyecto	Y = Log(TDEV/PM^(0,002*SSF))	X = Log(PM real)	XY	X2	Y2
1	1.102857667	3.268238547	3.604	10.681	1.216
2	0.862326351	2.412477348	2.080	5.820	0.744
3	0.831955571	2.303196057	1.916	5.305	0.692
4	0.682245454	1.769894036	1.208	3.133	0.465
5	1.302515713	3.985022981	5.191	15.880	1.697
6	1.264852572	3.846416293	4.865	14.795	1.600
7	0.738363989	1.962227231	1.449	3.850	0.545
8	0.981925578	2.838635038	2.787	8.058	0.964
Sumatoria =	7.767042895	22.38610753	23.100	67.522	7.923
		D = b1 =	0.2799		
		b0 =	0.1876		
		C =	1.5402		

Además, se muestra el cálculo de los residuos a partir de la ecuación de regresión: $\hat{Y}_i = 0{,}1876 + 0{,}2799 \cdot X_i$, en ella es posible identificar $b_1 = D = 0{,}2799$ y $b_0 = 0{,}1876$, de este último se obtiene el valor de C extrayendo su antilogaritmo en base 10, $C = 10^{0{,}1876} = 1{,}5403$.

Con los valores del tiempo estimado, obtenidos a través de la ecuación de regresión, es posible calcular el residuo y el residuo al cuadrado para cada proyecto y la varianza residual. Ver tabla 6.55.

Tabla 6.55: Cálculo del tiempo estimado, el residuo y el residuo al cuadrado para cada proyecto y la varianza residual (s2).

Proyecto	Y	X	Y est = b0 + b1X	Residuo = (Y-Y est)	Residuo^2
1	1.102857667	3.26823855	1.102	0.000478	0.000000
2	0.862326351	2.41247735	0.863	-0.000526	0.000000
3	0.831955571	2.30319606	0.832	-0.000309	0.000000
4	0.682245454	1.76989404	0.683	-0.000748	0.000001
5	1.302515713	3.98502298	1.303	-0.000492	0.000000
6	1.264852572	3.84641629	1.264	0.000641	0.000000
7	0.738363989	1.96222723	0.737	0.001537	0.000002
8	0.981925578	2.83863504	0.982	-0.000208	0.000000
Sumatoria =	7.767042895	22.3861075	7.767	0.000371	0.000004
			s2 =	7.0283E-07	
			D = b1 =	0.2799	
			b0 =	0.1876	
			C =	1.5403	

Así se concluye la calibración de las cuatro constantes del modelo de estimación COCOMO II.

6.6.4 Calibración de la distribución de tiempos y esfuerzos

Además de calibrar las ecuaciones de esfuerzo y tiempo. También es importante calibrar la distribución de dichas estimaciones. Según las investigaciones en el área de Ingeniería de Software, es posible presentar una distribución estándar, para esfuerzos y tiempos, la cual tiende a aproximarse a la distribución real de la mayoría de los proyectos software, ver Tabla 6.56.

Tabla 6.56: Distribución de tiempo y esfuerzo por fase del proyecto de desarrollo software.

	Fase	Inicio	Elaboración	Construcción	Transición
1	Tiempo Rational	10	30	50	10
2	Tiempo COCOMO2	12.5	37.5	62.5	12.5
3	Esfuerzo Rational	5	20	65	10
4	Esfuerzo COCOMO2	6	24	76	12
5	Administración	14	12	10	14
6	Entorno	10	8	5	5
7	Requerimientos	38	18	8	4
8	Diseño	19	36	16	4
9	Implementación	8	13	34	19
10	Evaluación	8	10	24	24
11	Despliegue	3	3	3	30

No obstante, esta distribución de tiempos y esfuerzos puede diferir de la del entorno local, debido al uso de diversas metodologías y procesos de desarrollo software.

Para ajustar el proceso y su distribución de esfuerzos del entorno local a las estimaciones de un proyecto software, se puede recolectar la información de esfuerzos por fases y actividades del proceso software. Dicha tabla puede expandirse para cubrir todas las fases y actividades del ciclo de vida del entorno local. Además, aunque no todas las actividades locales estén incluidas en las estimaciones de esfuerzo de COCOMO II, estas aún servirán para la planificación y seguimiento del progreso del proyecto.

6.6.5 Calibración de los factores de escala y los multiplicadores de esfuerzo

Es posible también realizar una calibración de los datos encontrados en las tablas de valoración para los factores de escala y los multiplicadores de esfuerzo. Esto mediante una regresión múltiple de los parámetros a estimar, ver ecuación Ec.6.47. Sin embargo, la cantidad de información necesaria para realizar esta regresión no siempre está disponible, por lo que no resulta fácil llegar a niveles aceptables de precisión. Los puntos de datos necesarios para esto se elevan por sobre 80 puntos de datos. Por otro lado, los drivers de costo y factores de escala del modelo COCOMO II han sido calibrados haciendo uso de una gran base de datos de proyectos, 83 puntos de datos y 161 puntos de datos dependiendo de la versión de COCOMO, por lo que se supone contar con valores aceptables para todos los drivers y factores de escala.

$$Log\left(\frac{PM}{Size^{1.01}}\right) = b0 + b_1 \cdot SF_1 \cdot Log(Size) + \ldots + b_5 \cdot SF_5 \cdot Log(Size) + \\ + b_6 \cdot Log(EM_1) + \ldots + b_{22} \cdot Log(EM_{22}) \qquad Ec.6.47$$

Para mayor información acerca de la calibración de los factores de escala y drivers de costo, se recomienda la lectura del WhitePaper de Sunita Devnani-Chulani, Bradford Clark y Barry Boehm, titulado: "Calibrating the COCOMO II Post-Architecture Model".

6.7 Conclusión

Los modelos de estimación de costos actuales nos ofrecen una herramienta poderosa en la obtención de resultados aplicables a la administración de proyectos informáticos. Esto gracias a la gran cantidad de información empírica recolectada a través del tiempo tras la realización de muchos proyectos software.

Del presente capítulo, se desprende la importancia del método de conteo de puntos de función ajustados y sin ajustar, la elaboración de un diseño preliminar del sistema a construir y el uso de herramientas de apoyo para el cálculo y administración de proyectos software.

Ya al final del capítulo, se desarrolla el tema de la calibración de las constantes del modelo COCOMO II para el entorno de desarrollo local. Y se introduce a la calibración de los factores de escala y drivers de costo.

Capítulo 7 : Prototipo Funcional R.E.M.

7.1 Introducción

El presente capítulo se divide en dos partes que giran en torno al desarrollo del prototipo funcional del sistema de recaudación electrónica a minoristas (R.E.M.), el cual fue solicitado por la empresa Trends S.A. durante el primer semestre del año 2002.

La primera parte describe la forma en que se desarrolló el prototipo desde la perspectiva del rol de analista-programador, sin poner énfasis en el diseño, ni en el proceso, ni en las estimaciones, ni en aspectos de administración de proyectos software, es decir, sin poner énfasis en la utilización de una metodología clara de desarrollo. Esta primera parte, que se llamará: **desarrollo artesanal del prototipo R.E.M.**, concluye con el paso de éste a operaciones.

La segunda parte, muestra una alternativa de desarrollo, que emplea la metodología de desarrollo sobre la plataforma .NET Framework, a la cual se le bautizó como **EFT-SDM (Metodología de desarrollo de Software de EFT Banca)**, cuyas bases han sido descritas a través de los capítulos previos y que será integrada en el capítulo 8. La figura de la sección 8.3, muestra un diagrama de componentes de UML que describe dicha metodología.

7.2 Prototipo de Recaudación Electrónica a Minoristas (R.E.M.)

7.2.1 Introducción al R.E.M.

El prototipo de recaudación electrónica a minoristas, surge de la inquietud de brindar un nuevo medio de pago a través del uso de tecnología WEB / WAP, a los diversos actores que forman parte de las cadenas de distribución de productos o servicios de las grandes empresas distribuidoras.

Las cadenas de distribución a las que apunta el sistema R.E.M., están formadas por los dueños de negocios minoristas, tales como: panaderías, mini-markets, quioscos, tiendas de abarrotes, etc., los transportistas que llevan el producto o servicio al cliente, y por las grandes empresas distribuidoras, tales como: Coca-Cola, SOPROLE, Pollos King, etc., que proveen de estos productos o servicios.

Por lo tanto, el sistema R.E.M., debe ser desarrollado teniendo en mente las necesidades de tres tipos de usuarios que forman parte del proceso de distribución de productos o servicios. El primero es el **proveedor**, que es un individuo perteneciente al área de administración de la empresa distribuidora, y que se encuentra ubicado físicamente en dicha empresa. El segundo, es el encargado de transportar el producto o servicio y llevarlo hacia el cliente, a esta persona se le llamará **transportista** o **recaudador**, ya que porta el dispositivo PCS con el cual se realizará la transacción a través de tecnología WAP. El tercero, es el **cliente** que paga por dicho producto o servicio.

La tecnología a utilizar, fue acotada por el sostenedor del proyecto, el cual pidió expresamente el uso de tecnología WAP aplicable a teléfonos celulares pertenecientes a la empresa ENTEL PCS. Además, se solicitó, realizar un sitio de apoyo a la gestión, que estuviese basado en tecnología WEB. La lógica del negocio, se concentraría en una base de datos sencilla, para el caso del prototipo, que estaría al alcance de los usuarios de ambos sitios.

El sistema de recaudación a minoristas debe presentar al usuario transportista y al usuario cliente (el minorista), una interfaz gráfica de usuario en un dispositivo PCS que permita la navegación vía WAP, además, debe permitir la identificación del usuario transportista mediante un código de usuario y una contraseña, los cuales lo habilitan para poder ingresar al área de navegación del sistema R.E.M., pudiendo así llevar un control de la ruta de distribución para cada día, visualizar los clientes a los cuales se ha despachado mercadería y ver a los clientes que aún tienen entregas pendientes. El sistema debe desplegar para cada orden de compra el rut del cliente, el número de factura, la cantidad a pagar por el cliente y la dirección de éste. Por el lado del minorista, debería permitir ver el detalle de la orden de compra (la glosa), y determinar la forma de pago. Estas podrían ser en efectivo, en cheque, por transferencia desde la cuenta corriente o crédito preaprobado. Además, el cliente contaría con un número secreto, con el cual autorizaría la transacción de compra y por ende la transferencia de fondos en el caso de las cuentas corrientes o créditos preaprobados. El sistema retornaría una aceptación o rechazo de la transacción, dependiendo de la validez de la contraseña del cliente y si el saldo de su cuenta

permitiera realizar el pago. De ser aceptada la transacción, se retornaría un número de transacción que sería la garantía del cliente.

Por el lado de la administración de la empresa proveedora, se contaría con el sitio de recaudación electrónica a minoristas el cual desplegaría cierta información importante de la distribución de los productos. Dicha información, incluiría, la lista de transportistas, las rutas asociadas a él, la lista de clientes por transportista, las órdenes de compra por cliente y el estado de cuenta del cliente, entre otros. La figura 7.1, muestra un esquema que caracteriza el modelo de negocios pedido por el sostenedor del proyecto.

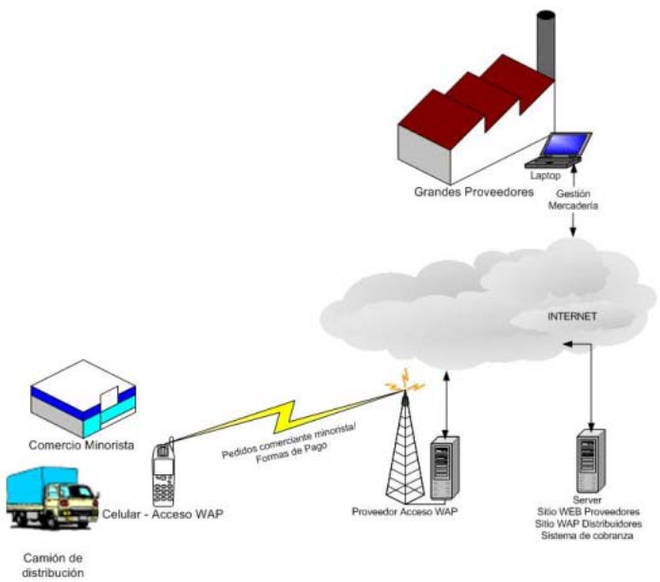


Figura 7.1: Modelo de negocios solicitado por el sostenedor del proyecto R.E.M.

Lo anterior, corresponde a los requerimientos básicos del sostenedor del proyecto, sin embargo, estos requerimientos fueron cambiando a través de su desarrollo, desde cambios fundamentales en el modelo de negocios hasta cambios en los requerimientos no funcionales, tales como los requerimientos estéticos de las interfaces gráficas de usuario.

Debido a la volatilidad en los requerimientos se construyeron dos versiones finales del prototipo R.E.M., la versión 1 y la versión 5, las que en realidad constituyen pequeñas variaciones al prototipo solicitado originalmente. Dichas versiones se explican con mayor detalle en la *sección 7.3.4.*

7.2.2 Estudio de factibilidad técnica

El estudio de factibilidad técnica fue desarrollado de una manera informal, y no concluyó, en ningún caso, en un documento en donde se expusieran las características técnicas necesarias para llevar a cabo el proyecto, más bien, se verificaron ciertas condiciones de disponibilidad para dejar el sistema operando. Estas condiciones tampoco fueron llevadas a un documento que diera cuenta de la factibilidad técnica del proyecto. Sin embargo, se realizaron anotaciones en borradores que ayudaban a mantener su curso. Esto debido a que los hitos exigían dar prioridad a mostrar avances efectivos, funcionales y demostrables por sobre el análisis y diseño. El método de desarrollo utilizado fue "Code and Fix" (codificar y arreglar).

Los aspectos que se consideraron importantes en la investigación de la factibilidad técnica del proyecto se exponen a continuación.

7.2.2.1 Tecnología WAP

Se investigó rápidamente la tecnología WAP, tratando de determinar cuáles eran los aspectos importantes que afectaban al proyecto. Se descubrió que existía un lenguaje llamado WML que es un subconjunto del lenguaje HTML. También se descubrió que los teléfonos PCS con acceso a WAP, contaban con un microbrowser que estaba grabado en su **Firmware**[36], el que interpretaba el código WML. Estos microbrowsers corresponden a lo que sería un navegador de internet (o browser) como los conocidos Internet Explorer o Netscape Navegator, pero para celulares. Además, dichos microbrowsers tendrían como cota superior un cierto número de versión del lenguaje WML (Wireless Markup Language), esto significa que el microbrowser será capaz de interpretar sólo el subconjunto de instrucciones que estén incluidas en dicha versión de WML, por lo que si se implementa un sitio WAP en una versión superior del lenguaje WML, el microbrowser no será capaz de interpretar todo el código. Es decir, sería necesario determinar la mínima versión del lenguaje WML que apoyan los equipos PCS seleccionados para el sistema de

[36] **Firmware:** Se refiere al software que ha sido almacenado en circuitos integrados, Read Only Memory.

recaudación a minoristas. El equipo celular seleccionado para las pruebas fue el **Motorola T193**, ver figura 7.2.

Figura 7.2: Motorola T193.

La topología de redes está formada por una red inalámbrica privada perteneciente en la mayoría de los casos a alguna empresa telefónica, en este caso ENTEL PCS. Además existe un punto de acceso a Internet a través de un servidor gateway WAP, el cual sirve de puente entre la red inalámbrica e Internet. La figura 7.3 ilustra la topología de redes.

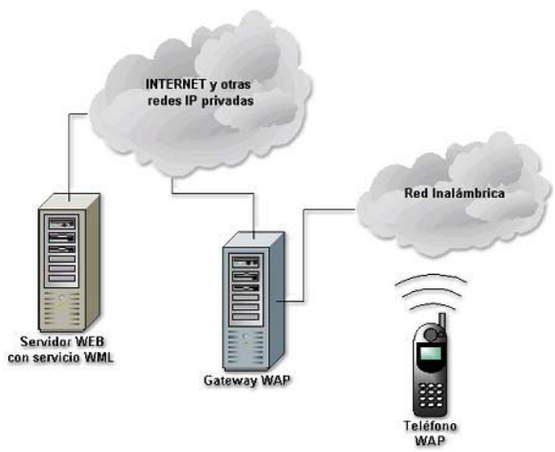


Figura 7.3: Ilustración de la topología de redes.

Como se dijo anteriormente, los teléfonos celulares con acceso a WAP, utilizan un lenguaje abierto desarrollado por el **WAP Forum**[37], el WML, que permite la presentación de texto e imágenes, entrada de información y formularios.

Este lenguaje, permite que el usuario envíe peticiones a través de las redes inalámbricas al gateway WAP, el cual convierte estas peticiones en HTTP, y las envía a través de internet.

[37] **WAP Forum:** Organización dedicada al desarrollo del protocolo WAP.

Cuando el servicio requerido responde a dicha petición, el gateway WAP retorna la información al teléfono celular con acceso a WAP.

El gateway WAP es el núcleo de la plataforma WAP, y actúa en forma análoga a un servidor proxy HTTP, permitiendo a los navegantes el acceso a diversos sitios en la WWW.

Algunos proveedores de contenido brindan dos formas de acceso al materiar disponible. Para los usuarios convencionales que navegan a través de un computador personal, el acceso es a través de un sitio WEB, por otro lado, para los usuarios que navegan a través de sus dispositivos celulares, el acceso es a través de un sitio WAP. Además, para ofrecer acceso a través de WAP, es necesario configurar el servidor WEB con apoyo a servicios WML, o WAP-enabled, ver figura 7.4.

Los pasos a seguir para configurar el servidor Internet Information Server para que brinde servicios WML (WAP-enabled) se resume en el **Anexo 11**.

Además de la traducción a HTML, los gateway WAP ofrecen diversos servicios, tales como: servicio de correo electrónico, servicio de fax, etc. Todos ellos dependen de la suite de servicios que ofrezca cada gateway. Actualmente existen varios gateway WAP disponibles en el mercado, dentro de los cuales se destaca el **Microsoft Mobile Information Server**, el cual se integra con la plataforma .NET, actúa como gateway, apoya protocolos de seguridad SSL, IPSec, VPN (Virtual Private Network). Además, apoya microbrowser WAP versión 1.1 y Phone.com, mobile WebForms, controls y ASP.NET.

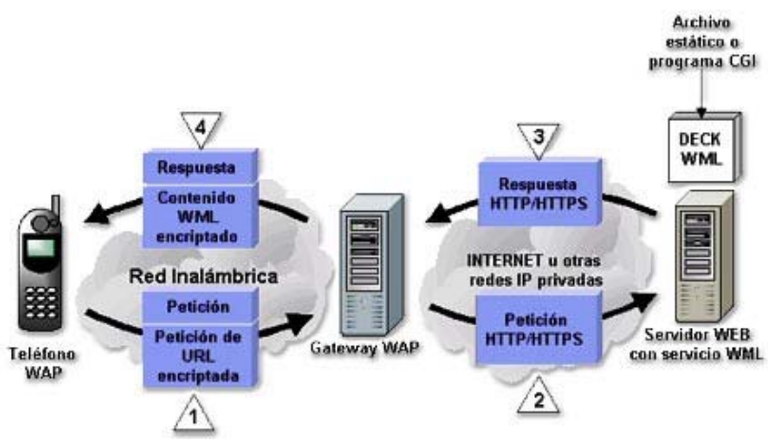


Figura 7.4: Peticiones y respuestas en la topología de redes.

El estudio de factibilidad técnica se realizó de manera informal, ya que las instrucciones del sostenedor del proyecto regían las tecnologías de desarrollo por el hecho de formar parte de sus requerimientos, por lo que sólo se confirmó la viabilidad de una manera informal.

7.2.2.2 Estudio de factibilidad económica

En el rol de analista-programador, no se realizó ningún estudio de factibilidad económica, limitándose a cumplir las órdenes de la administración del proyecto. Además, la administración del proyecto restringió la actividad de desarrollo a un máximo de 80 [horas-hombre], basado en una estimación por analogía de proyectos similares, y se supuso un precio de venta[38] de $1.5\left[\frac{UF}{hora-\text{hom}\,bre}\right]$ estimado en forma empírica.

7.3 Desarrollo artesanal del prototipo R.E.M. versión 5.

El desarrollo artesanal del prototipo R.E.M. fue modelado a través de varias entrevistas con el sostenedor del proyecto. Es así como se obtuvo una especificación de requerimientos vaga en base a dichas entrevistas, con una pobre documentación de requerimientos. En donde estos fueron anotados tan sólo en lenguaje natural y muchas veces fueron negociados durante la implementación del sistema.

Para modelar el sistema no se utilizó ningún tipo de diagrama especial, más bien se dibujaron interfaces gráficas de usuario, en donde se sugerían las funcionalidades de cada una de ellas. La figura 7.5, muestra la navegación del sitio WAP en forma esquemática.

---

[38] **Precio de venta:** Se ha estimado el precio de venta igual al precio de mercado de una forma subjetiva.

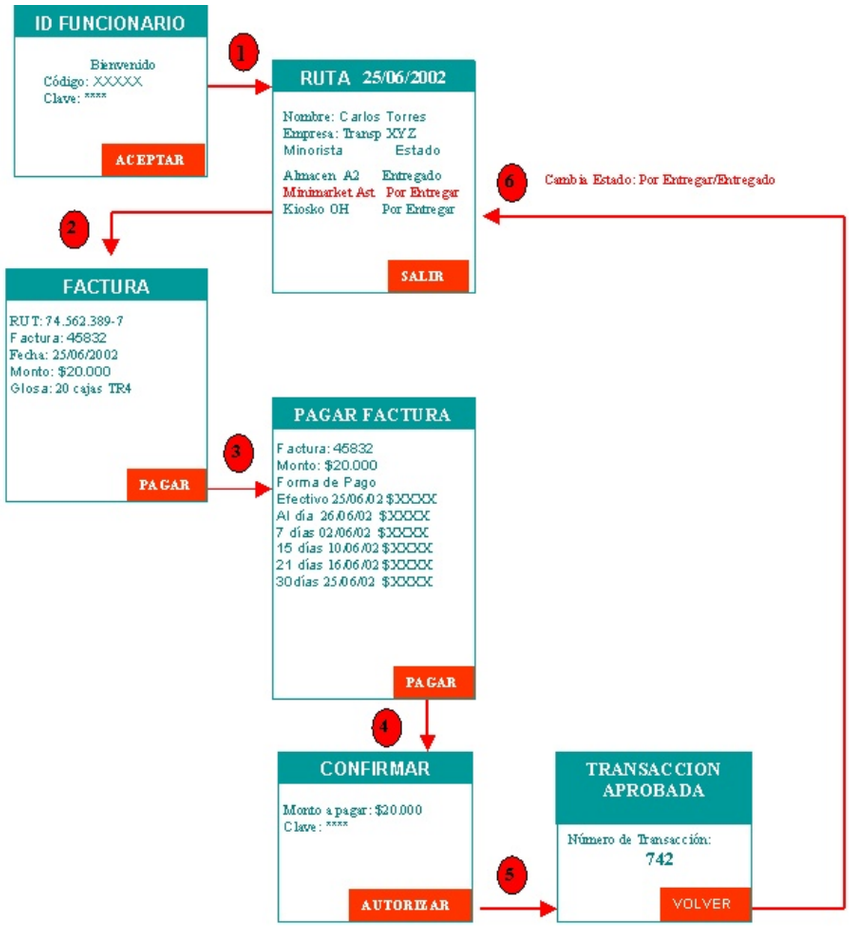


Figura 7.5: Navegación sitio WAP proyecto TRENDS.

La figura 7.5, fue extraída del informe de estado de avance del proyecto Trends con fecha 27 de Junio de 2002, elaborado por Sr. Gonzalo Patrone. En ella se observan los requerimientos del sostenedor con fecha 26 de junio de 2002. En el rectángulo rotulado como "PAGAR FACTURA", se ve como las alternativas de pago han sido reducidas drásticamente por el sostenedor, dejando sólo las alternativas de pago en efectivo, y pago con cheque al día, a 7, 15, 21 y 30 días. Lo anterior, obedece a un cambio en los requerimientos debido a la volatilidad de estos.

7.3.1 Factores de riesgo del proyecto

Una de las restricciones técnicas más importantes fue el hecho de no contar con un equipo celular para verificar cómo eran desplegadas las interfaces gráficas de usuario. Además, las modificaciones hechas a éstas debían ser ingresadas al sitio de pruebas en forma manual.

Esto significó que se debía acceder al sitio de pruebas en forma personal y realizar las modificaciones pertinentes desde ahí, el servidor de pruebas no estaba próximo a la estación de trabajo en que se desarrollaba el producto.

Además, el riesgo del proyecto se elevaba conforme aparecían nuevos requerimientos del sostenedor al límite de cumplirse las fechas acordadas de entrega. Los nuevos requerimientos en su mayoría correspondían a requerimientos no funcionales, en gran parte para mejorar la estética de las GUIs, dichas actividades de mejoramiento de las interfaces gráficas de usuario fueron frecuentes debido, como se señaló anteriormente, a que no se contaba con un dispositivo PCS que desplegara el diseño de las cards[39] del sitio WAP. Por otro lado, algunos requerimientos funcionales, comprometían el modelo de negocio. Por ejemplo, cuando se solicitó, que en lugar de desplegar las fechas de pago fijas (ver figura 7.5, recuadro PAGAR FACTURAS), se debía dar la opción al cliente de introducir fechas de pago y cuotas tantas veces como fuese necesario para cubrir el monto total de la deuda. Esto acarreó un cambio en el modelo de datos que consumió recursos por causa de la volatilidad de los requerimientos.

7.3.2 Plataforma de desarrollo utilizada

El modelo de desarrollo que se utilizó para implementar el prototipo R.E.M. siguió una mezcla de prototipado evolutivo con subfases exploratorias de **"Code & Fix"** de una manera informal. El prototipo del sitio WAP fue implementado y probado a través de un simulador de microbrowser desarrollado por Openwave. Se utilizó el kit de desarrollo de software **Openwave 5.1 SDK**, el cual permitió realizar pruebas al sitio WAP en forma local, por lo que no fue necesario en una primera instancia un equipo celular. El microbrowser contaba con las características expuestas en la tabla 7.1. Además, se utilizaron las siguientes herramientas que ayudaron en las distintas etapas de la implementación del sistema R.E.M.: **Macromedia DreamWeaver MX**, **Paint Shop Pro 6.0**, **Notepad**, **Internet Information Server con servicio WML**.

[39] **Cards:** En WML cada GUI se considera una carta (Card), y es posible tener un maso (Deck) con varias cards en un mismo archivo WML o ASP según se requiera.

Tabla 7.1: Características simulador Siemens S45,Openwave 5.1 SDK.

Características	
Browser	Openwave Mobile Browser 5.0.2 (GUI)
Markup Languages	**HDML 3.0, WML 1.3, WMLScript 1.2**
Image formats	WBMP, BMP (1-bit black/white)
Font & Charset	Siemens S45 Font, MS1252 Latin 1 (ANSI)
Display Size	101 x 80 pixels

La tabla 7.1 destaca las capacidades de Markup Language del simulador, las que resultaron compatibles con las del equipo elegido para las pruebas del prototipo Motorola T193.

7.3.3 Resumen de las actividades realizadas en el proyecto R.E.M. versión 5

Durante el desarrollo del prototipo R.E.M. versión 5, la administración del proyecto, llevó un control de las actividades. La tabla 7.2, muestra un cronograma de las actividades realizadas en el desarrollo artesanal del prototipo R.E.M., en ella se observan las fechas de inicio de las actividades, las actividades y subactividades, los recursos involucrados, y el consumo en tiempo de éstas.

Tabla 7.2: Cronograma de actividades realizadas en el desarrollo artesanal del prototipo R.E.M.

FECHA INICIO	ACTIVIDAD	RECURSOS	CONSUMO [Hr]
5/06/2002	Reunión inicial en oficinas de Trends S.A. Captura de requerimientos.	Jorge Correa Christian Hagedorn Gonzalo Patrone	**7.5**
6/06/2002	Primera Iteración (versión 1) - Requerimientos del sistema (WAP-WEB-BD) - Definición y diseño del proceso de pago. - Presentación al cliente del modelo preliminar. - Construcción de primera aproximación del sistema. - Construcción modelo Base de Datos. - Interacción con página ASP y WML. - Pruebas y ajustes al prototipo WAP. - Presentación del prototipo al cliente	Jorge Correa Gonzalo Patrone	**66.0**
17/06/2002	Segunda Iteración (versión 2) - Análisis de nuevos requerimientos. - Cambios al prototipo. - Cambios al modelo de datos. - Definición y diseño Sitio de Gestión WEB. - Construcción Sitio de Gestión WEB. - Instalación en servidor de QA[40] con acceso Internet. - Pruebas y ajustes al prototipo WAP y Sitio de Gestión WEB.	Jorge Correa Gonzalo Patrone	**26.0**
20/06/2002	Tercera Iteración (versión 3) - Análisis de nuevos requerimientos. - Cambios al prototipo. - Cambios al modelo de datos. - Definición y diseño Sitio de Gestión WEB. - Construcción Sitio de Gestión WEB. - Instalación en servidor de QA con acceso Internet. - Pruebas y ajustes al prototipo WAP y Sitio de Gestión WEB.	Jorge Correa Gonzalo Patrone	**15.0**
TOTAL DE HORAS DEL PROYECTO			**114.5**

La tabla 7.2 ha sido extraída del informe de estado de avance del proyecto Trends, 27 de Junio de 2002, elaborado por Sr. Gonzalo Patrone.

[40] **Servidor QA:** Servidor de pruebas, aseguramiento de la calidad, del inglés Quality Assurance.

7.3.4 Versiones finales del prototipo artesanal R.E.M.

De las múltiples versiones desarrolladas del prototipo R.E.M., cinco en total, se han rescatado dos versiones seleccionadas como finales por el sostenedor. Dichas versiones corresponden a las versiones 1 y 5 del prototipo R.E.M.

La versión 1 corresponde al prototipo que supone las fechas de pago fijas y la forma de pago ha sido reducida exclusivamente a pago con cheque. Las fechas son al día, a 7 días, a 15 días, a 21 días y a 31 días.

La versión 5 corresponde al prototipo que permite al usuario cliente del sistema, el minorista, establecer las fechas de pago y el monto de las cuotas hasta completar el valor total de los productos solicitados al proveedor.

7.3.4.1 Sitios WAP

La figura 7.6, muestra la dinámica de cards de la versión 1 del prototipo artesanal R.E.M., en ella se aprecia el flujo típico de eventos para realizar una transacción de pago.

En la card rotulada como "Bienvenido", el transportista debe ingresar un código de funcionario, que ha sido especificado como un código numérico de cinco dígitos, en donde los dos primeros dígitos corresponden al ID de empresa y los tres últimos corresponden al ID de funcionario. Ver ecuación Ec.7.1.

$$\underbrace{01}_{ID\ empresa}\ \underbrace{111}_{ID\ funcionario} \Rightarrow 01111 \qquad Ec.7.1$$

Por lo que el número 01111 identifica a un funcionario con ID de funcionario 111 que pertenece a la empresa con ID 01. La clave de cada funcionario ha sido especificada como un código numérico de cuatro dígitos. Si la identificación de funcionario es válida, entonces éste tendrá acceso a la ruta para el día, en donde se desplegará su nombre, la empresa a la cual pertenece y una lista de minoristas y el estado de la entrega (Est.). El estado de una entrega puede ser Entregado (Ent), No entregado (NEn) o Por entregar (PEn).

La tabla 7.3 resume las situaciones consideradas dentro de estos estados.

Tabla 7.3: Posibles estados de una orden.

Código	Estado	Descripción
PEn	Por Entregar	Es el estado por defecto en el que se encuentra una orden. Indica que la orden no ha sido entregada.
Ent	Entregado	Indica que una orden ya ha sido entregada.
NEn	No Entregado	Se reserva para dejar constancia de que el producto ha sido despachado y no se ha podido entregar por razones ajenas al proveedor. (Ej. No fue recibido)

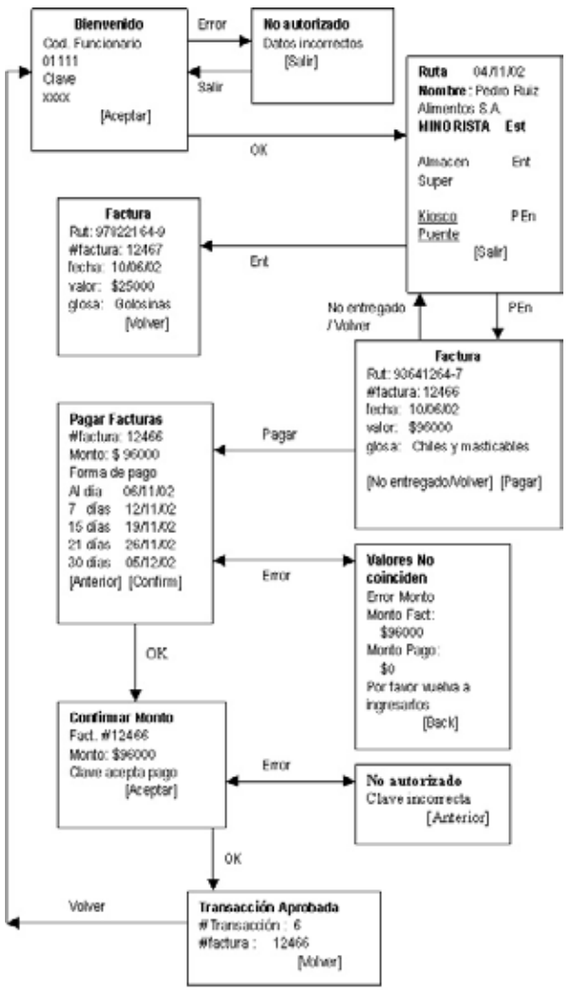


Figura 7.6: Dinámica de cards de la versión 1 del prototipo artesanal R.E.M.

La dinámica de cards de la versión 5 del prototipo artesanal R.E.M. se muestra en la figura 7.7. En ella se aprecia cómo es posible seleccionar la fecha de pago. Estas se deben ingresar en un formato numérico: **ddmmaa**. Esto significa, que los dos primeros dígitos

corresponden al día, los siguientes dos al mes y los últimos dos al año. Entonces, por ejemplo, la fecha de pago 1, 011202, correspondería al día 01 de diciembre (mes 12), del año 2002 (año 02).

Monto 1 corresponde a la cantidad de dinero que se desea pagar en la fecha de pago 1, que ha sido ingresada en la card anterior. La etiqueta "Por pagar" despliega la diferencia que falta por pactar, y "Monto" indica el total que se está pactando en cuotas. Una vez que el monto total de la deuda a sido pactado en cuotas, se despliega la última card rotulada Pagar facturas, la cual muestra una tabla resumen con las fechas y montos seleccionados por el minorista. Al final de ésta, se le pide al minorista ingresar la clave secreta que autoriza las transacciones que se muestran en dicha tabla.

En ambas versiones el cliente debe ingresar personalmente la clave secreta y ésta no debe ser revelada a nadie. La card en donde se solicita la contraseña, contiene información referente a la transacción que se realizará.

Por último, ambas versiones actualizan una base de datos sencilla la cual ha sido desarrollada en Microsoft Access, y a la cual se le ha instalado una ligera carga de prueba.

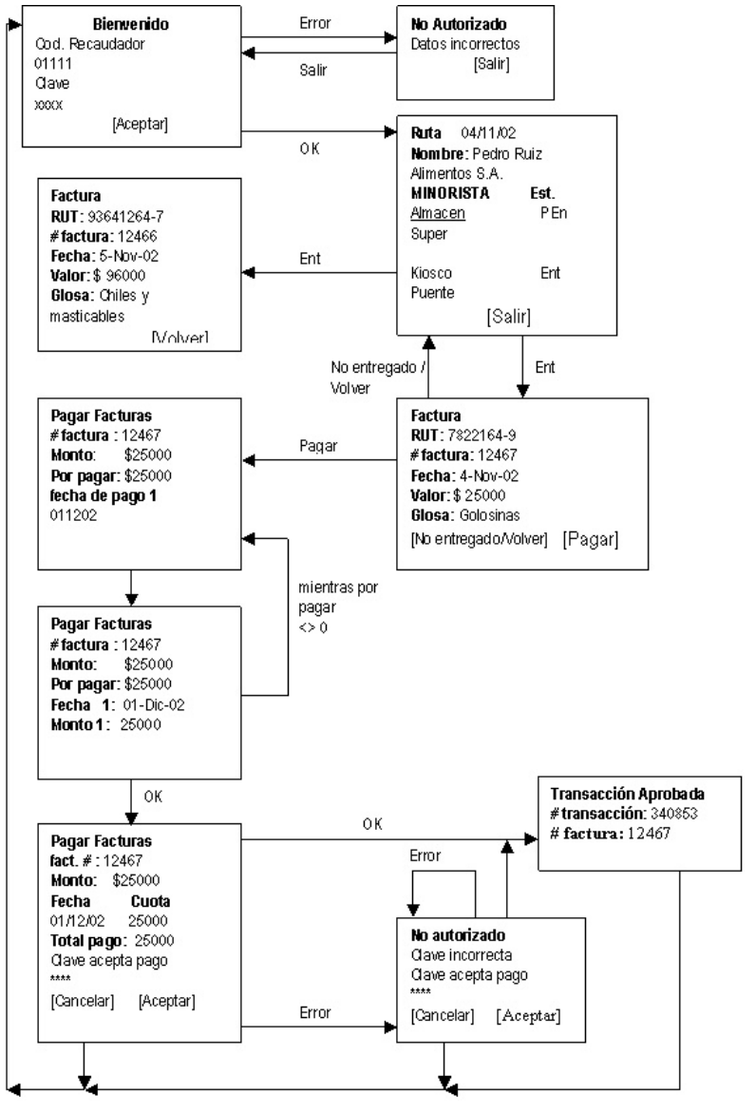


Figura 7.7: Dinámica de cards de la versión 5 del prototipo artesanal R.E.M.

La implementación de ambos prototipos se ha realizado en ASP, insertando Tags de WML para diferenciar las cards en un mismo deck.

7.3.4.1.1 Sitio WAP DEMO prototipo R.E.M. versión 1

La figura 7.8, muestra la navegación típica a través del sitio WAP del prototipo R.E.M. versión 1 en el simulador Siemens S45 provisto por la plataforma de desarrollo Openwave 5.1 SDK.

Figura 7.8: Secuencia de GUIs que muestran un escenario de navegación típico del sitio WAP DEMO prototipo R.E.M. versión 1.

Nótese la similitud de las GUIs en la figura 7.8 con algunas cards mostradas en la figura 7.6. La misma comparación es posible con las figuras 7.9 y con la dinámica de cards de la figura 7.7.

7.3.4.1.2 Sitio WAP DEMO prototipo R.E.M. versión 5

La 7.9, muestra la navegación típica a través del sitio WAP del prototipo R.E.M. 5 en el simulador Siemens S45 provisto por la plataforma de desarrollo Openwave 5.1 SDK.

Figura 7.9: Secuencia de GUIs que muestran un escenario de navegación típico del sitio WAP DEMO prototipo R.E.M. versión 5.

7.3.4.2 Sitios WEB de apoyo a la gestión

Para cada una de las versiones seleccionadas (versiones 1 y 5), se elaboró un sitio WEB de apoyo a la gestión con pequeñas modificaciones dependiendo de la versión.

7.3.4.2.1 Sitio WEB Demo prototipo R.E.M. versión 1

El sitio WEB de la versión 1 del prototipo R.E.M. se muestra en la figura 7.10.

Figura 7.10: Sitio WEB DEMO prototipo R.E.M. versión 1.

El logo que se encuentra en la esquina inferior izquierda (EFT Unplugged TRENDS S.A.), sirve para reinicializar la base de datos, para iniciar una nueva presentación de la Demo del prototipo R.E.M. Haciendo click con el mouse, sobre este logo, los valores de las transacciones se reinicializan a cero y las cuentas de clientes quedan con un saldo disponible de $200.000 cada una. Además, las rutas de cada uno de los transportistas aparecen con todas sus entregas pendientes (estado Por Entregar).

Es posible, entrar al sitio WEB de administración, ingresando el ID de la empresa y la contraseña correspondiente. Por ejemplo, Código: 01 y Clave: 1111.

Una vez dentro, se puede observar el nombre de la empresa, las rutas por funcionario y una serie de opciones. Así como un resumen de los pagos al día, a 7, 15, 21 y 31 días.

Figura 7.11: Sitio WEB Demo prototipo R.E.M. versión 1. Lista de rutas.

Es posible ver el número de ruta, el nombre del funcionario, y las cantidades de dinero en despachos que le falta por entregar, las que ya ha entregado y las que no a podido entregar. También, se observa el total en dinero de cada ruta y los totales para todas estas categorías. En la mitad inferior de la pantalla se observa un resumen de la forma de pago de cada una de las transacciones realizadas.

Existe un enlace sobre el nombre de cada uno de los funcionarios. Si se hace click con el mouse sobre algún nombre, se accederá a una página web en donde se despliegan los clientes que están incluidos en la ruta del funcionario, ver figura 7.12. En ella se detalla el nombre del funcionario, el número de ruta, la lista de clientes, las facturas asociadas a cada cliente, el estado del despacho y el monto del despacho.

En la mitad inferior de la interfaz gráfica de usuario, se puede observar un resumen igual al correspondiente de la figura 7.11, pero con la información pertinente a la ruta asociada al funcionario. En otras palabras, se pueden ver los montos que se han pactado, y la forma de pago seleccionada por el cliente, así como un total para la ruta. Por cada cliente de la lista, se tiene un enlace que despliega una página web en donde se detalla cierta información de la factura del cliente.

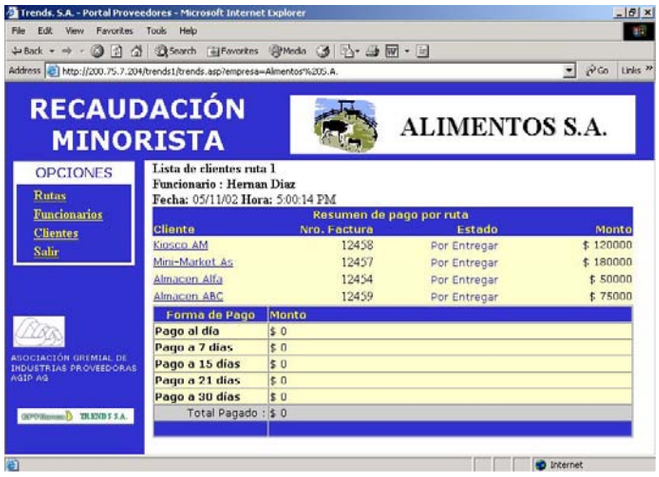


Figura 7.12: Sitio WEB Demo prototipo R.E.M. versión 1. Lista de clientes por ruta.

La figura 7.13, muestra el detalle de la información del cliente, en ella se observa, el nombre del cliente, Rut del cliente, dirección, fono y fecha. Además, se da cuenta de la forma de pago seleccionada por éste.

Figura 7.13: Sitio WEB Demo prototipo R.E.M. versión 1. Información cliente.

En el menú opciones, hasta el momento, se han revisado las páginas que guardan relación con el enlace "Rutas". Sin embargo, existen tres enlaces más que revisar y que vinculan a páginas que exhiben exactamente el mismo comportamiento tanto para la versión 1 de la demo del prototipo como para la versión 5, por lo que no serán vistos nuevamente en la **sección 7.3.4.2.2**. El primero de los enlaces a tratar es **"Funcionarios"**, que presenta una lista con la información de los funcionarios transportistas de la empresa. La información desplegada por esta página es el nombre de los funcionarios, el RUT y el número del teléfono celular que portan. Ver figura 7.14.

Figura 7.14: Sitio WEB Demo prototipo R.E.M. versión 1. Información funcionario transportista.

El segundo enlace es **"Clientes"**, el cual despliega la información referente a los clientes de la empresa con ID 01, en este caso Alimentos S.A.. La información que se entrega es nombre, RUT, dirección, teléfono, y saldo disponible (disponible). Ver figura 7.15.

Por último, el enlace **"Salir"**, termina la sesión de apoyo a la gestión para la empresa. Al presionar este enlace, el usuario se ve direccionado a la página de inicio del sitio, la cual se muestra en la figura 7.10.

Figura 7.15: Sitio WEB Demo prototipo R.E.M. versión 1. Información de clientes.

7.3.4.2.2 Sitio WEB Demo prototipo R.E.M. versión 5

El sitio WEB Demo del prototipo R.E.M. versión 5 se muestra en la figura 7.16.

Figura 7.16: Sitio WEB DEMO prototipo R.E.M. versión 5.

La página de inicio de la versión 5, sólo ha sufrido una modificación estética. Se ha incorporado un logotipo que se muestra en la figura 7.17.

Figura 7.17: Logotipo R.E.M.

El resto de las funcionalidades de la versión 1, como es la validación de sesión y reinicialización de los valores de la presentación se han mantenido intactos. Al ingresar al sitio de apoyo a la gestión, es posible observar la página web por defecto, ver figura 7.18, la cual mantiene cierta similitud con la de la versión 1. El menú de opciones se ha mantenido igual, con la salvedad de que el nombre del enlace **"Funcionarios"** a sido reemplazado por **"Recaudador"**, sin embargo, la funcionalidad de todos los enlaces sigue siendo la misma de la versión 1.

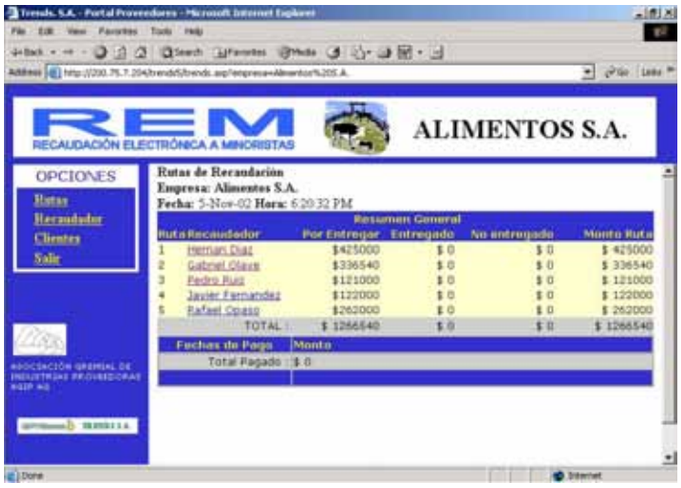


Figura 7.18: Sitio WEB Demo prototipo R.E.M. versión 5. Lista de rutas.

La mitad superior de la lista de rutas despliega la misma información que su par en la versión 1, mostrando el número de ruta, el nombre del recaudador y el monto asociado a cada estado de los despachos de una ruta. No obstante la parte inferior, muestra las fechas de pago y los montos pactados, así como el total para todas las rutas. Si se hace click en algún cliente, se despliega la lista de clientes por ruta que se muestra en la figura 7.19.

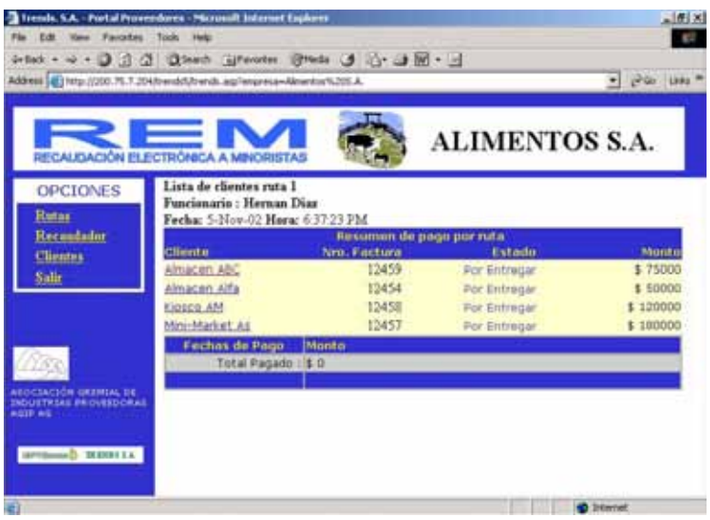


Figura 7.19: Sitio WEB Demo prototipo R.E.M. versión 5. Lista de clientes por ruta.

La información desplegada en la lista de clientes por ruta es igual a la de la versión 1. Sin embargo, en la mitad inferior de la interfaz gráfica de usuario, se observan las fechas y montos pactados para esa ruta, así como el total pagado. La figura 7.20, muestra el detalle por cliente, y sus fechas de pago y montos para la factura asociada.

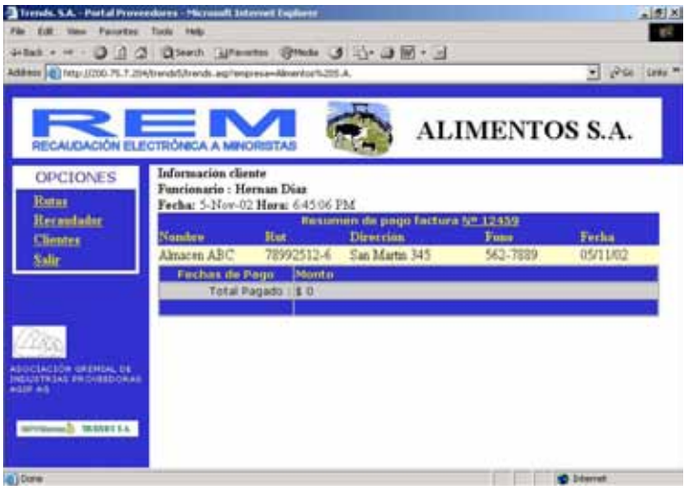


Figura 7.20: Sitio WEB Demo prototipo R.E.M. versión 5. Información cliente.

Como se dijo anteriormente, el resto de las funcionalidades se han mantenido exactamente iguales. Por otro lado, al no existir mayor documentación respecto al desarrollo de las versiones del prototipo artesanal R.E.M., la presente sección se ve concluída.

En la próxima sección, se presenta como ejemplo de la metodología **EFT-SDM**, el desarrollo de la versión 5 del prototipo artesanal R.E.M.. Con este ejemplo, es posible apreciar el ámbito y profundidad de cada una de las tareas, de cada uno de los workflows del proceso de desarrollo de **EFT-SDM**, el cual se describe en el Capítulo 8. En donde se incluyen todas las actividades de la administración de proyectos, arquitectura de software e implementación de los modelos.

Debido a que este ejemplo involucra muchas tareas, que por lo general, se estarán realizando en forma concurrente, se aplicarán diagramas de actividad para mostrar la secuencia de tareas por rol. Además, se asociará un **Swimlane** a cada rol para delimitar las actividades que le competen.

Las actividades referentes al diseño artístico de las páginas no han sido consideradas por no estar incluidas en el proceso de desarrollo de software.

7.4 Aplicación de Metodología de desarrollo de software EFT-SDM

A continuación se desarrollará la construcción del prototipo R.E.M. versión 5, haciendo uso de la metodología **EFT-SDM**. Nótese que las actividades presentes en el RUP han sido adaptadas para formar parte del **EFT-UP**.

7.4.1 Primera Iteración

7.4.1.1 Fase de inicio

Durante la fase de inicio es necesario que interactúen los siguientes workflows: Modelamiento de negocios, Requerimientos, Administración de Proyectos, Entorno y Análisis y Diseño. Sin embargo, no siempre es necesario realizar las actividades pertenecientes al workflow de modelamiento de negocios.

En la fase de inicio, es de vital importancia capturar los requerimientos del software a desarrollar. Esto se realiza a través de casos de uso y los diagramas de casos de uso.

7.4.1.1.1 Workflow de Requerimientos

El objetivo principal de este workflow es capturar los requerimientos de los usuarios del sistema. La figura 7.21, muestra un Diagrama de casos de uso del prototipo R.E.M. versión 5, y ofrece una primera aproximación de su arquitectura.

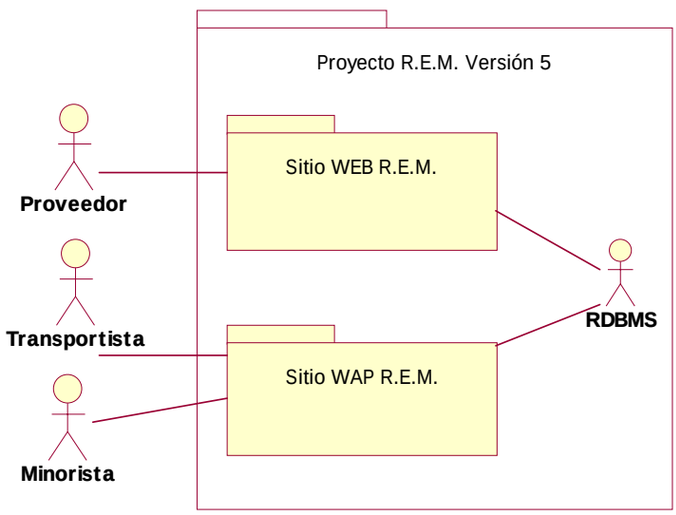


Figura 7.21: Diagrama de casos de uso inicial del prototipo R.E.M. versión 5.

En un nivel de abstracción alto, se pueden representar los actores y casos de uso del prototipo R.E.M. versión 5. Los casos de uso se encuentran agrupados en el interior de dos paquetes de UML que separan claramente dos subconjuntos en el Proyecto R.E.M.. El sitio WEB

R.E.M. y el sitio WAP R.E.M., los cuales interactúan con los actores: Proveedor, Transportista, Minorista y Sistema Administrador de Bases de Datos (RDBMS).

El diagrama de casos de uso inicial debe ir refinándose en la fase de inicio y elaboración entre iteraciones sucesivas del proyecto. Las figuras 7.22 y 7.23, muestran un nivel de abstracción más bajo, en el cual se observan los casos de uso en el interior de cada paquete, además de la interacción con los actores.

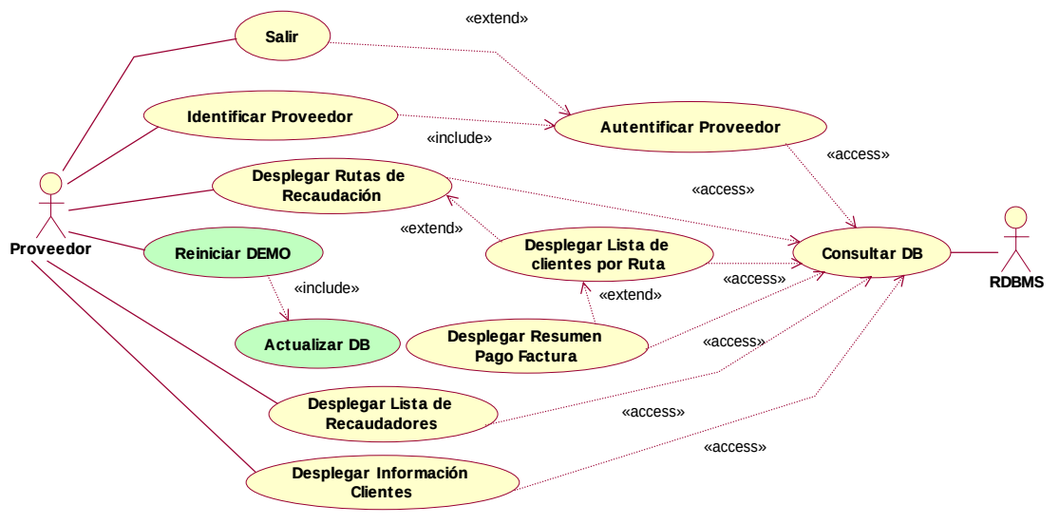


Figura 7.22: Diagrama de casos de uso prototipo R.E.M. v.5, paquete sitio WEB R.E.M.

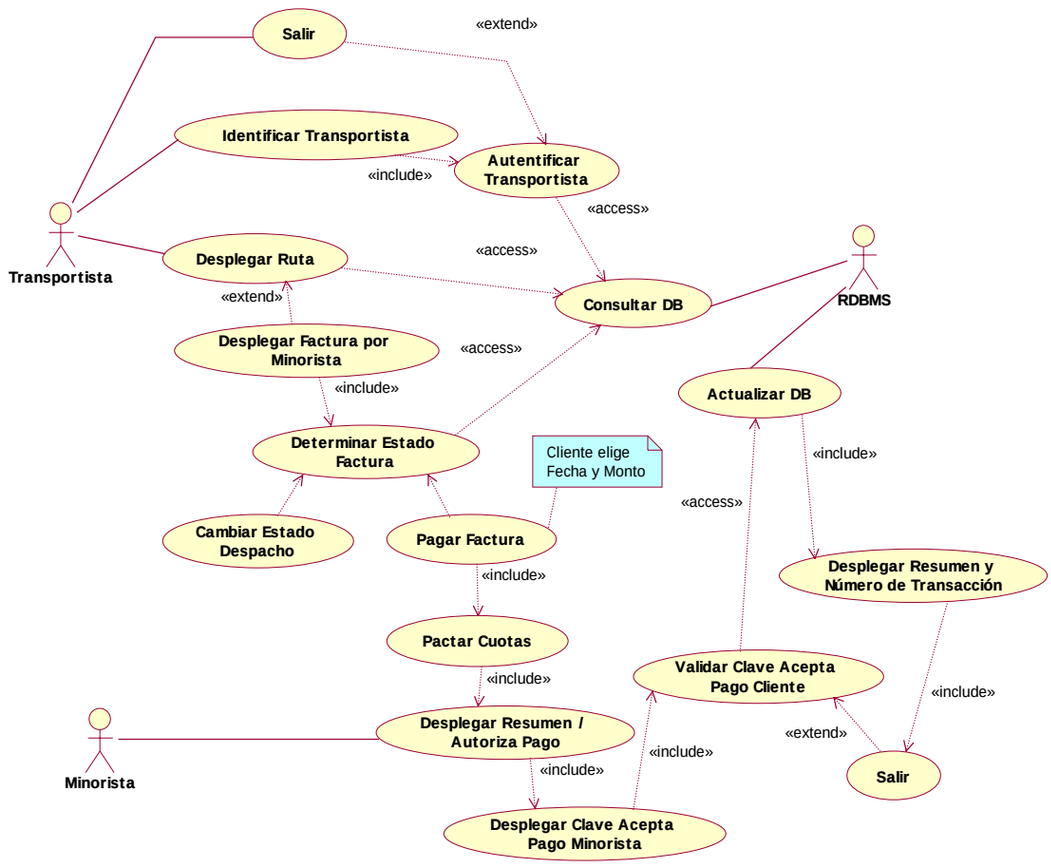


Figura 7.23: Diagrama de casos de uso prototipo R.E.M. v.5, paquete sitio WAP R.E.M.

Se comenzará trabajando sobre los casos de uso del diagrama del sitio WEB R.E.M., de la figura 7.22 y luego sobre los del sitio WAP R.E.M. de la figura 7.23. El objetivo es poder describir los **casos de uso de alto nivel** del sistema y posteriormente los **casos de uso expandidos**. *Los rótulos de tablas de las secciones 7.4.1.1.1.1, 7.4.1.1.1.2, 7.4.1.1.1.3 y 7.4.1.1.1.4, han sido omitidos en forma intencional para mejorar la legibilidad del documento.*

7.4.1.1.1.1 Casos de uso de alto nivel WEB R.E.M.

Caso de uso	Salir
Actores	Proveedor
Tipo	Secundario
Descripción	Permite que el proveedor salga del sitio WEB R.E.M.

Caso de uso	Identificar Proveedor
Actores	Proveedor
Tipo	Primario
Descripción	Permite que el proveedor se identifique mediante un código de funcionario y contraseña.

Caso de uso	Desplegar rutas de recaudación
Actores	Proveedor
Tipo	Primario
Descripción	Despliega las rutas y los transportistas asociados a ellas para una determinada empresa. Además, muestra los montos y las fechas pactadas para todas las rutas.

Caso de uso	Desplegar lista de clientes por ruta
Actores	Proveedor
Tipo	Primaria
Descripción	Despliega la lista de clientes para una ruta junto con los montos y fechas pactadas hasta el momento en dicha ruta.

Caso de uso	Desplegar resumen pago factura
Actores	Proveedor
Tipo	Primaria
Descripción	Despliega el número de factura, los montos y las fechas de pago para un determinado cliente.

Caso de uso	Desplegar lista de recaudadores
Actores	Proveedor
Tipo	Secundaria
Descripción	Despliega una lista con la información de los recaudadores: Nombre, Rut y Fono.

Caso de uso	Desplegar información clientes
Actores	Proveedor
Tipo	Secundario
Descripción	Despliega una lista con la información de los clientes: Nombre, Rut, Dirección, Fono, Disponibilidad.

Caso de uso	Autentificar proveedor
Actores	Proveedor, RDBMS
Tipo	Primario
Descripción	Permite consultar la base de datos para validar la identificación de un determinado proveedor.

Caso de uso	Consultar DB
Actores	RDBMS
Tipo	Primario
Descripción	Establece la ruta de acceso a la base de datos y administra consultas.

Caso de uso	**Reiniciar DEMO**
Actores	Proveedor
Tipo	**Opcional (Sólo disponible en versión DEMO del producto)**
Descripción	Reinicializa los valores de la Base de datos. Para comenzar una nueva demostración del producto con los valores iniciales del negocio.

Caso de uso	**Actualizar DB**
Actores	Proveedor, Incluida por caso de uso **Reiniciar DEMO**.
Tipo	**Opcional (Sólo disponible en versión DEMO del producto)**
Descripción	Actualiza campos de las tablas de la base de datos que están relacionados con las transacciones dispuestas en la carga de prueba del sistema.

7.4.1.1.1.2 Casos de uso expandidos WEB R.E.M.

No es necesario describir todos los casos de uso de alto nivel como casos de uso expandidos, sólo se describen los más importantes.

Caso de uso	**Salir**	
Actores	Proveedor, Extiende a Autentificación Proveedor	
Propósito	Permite finalizar sesión del proveedor y salir del sistema WEB R.E.M.	
Visión General	El proveedor presiona el enlace salir, o la ejecución del caso de uso Autentificación Proveedor determina que éste no debe ingresar al sistema. Salir presenta la página de inicio.	
Tipo	Secundario	
Referencias		
Curso Típico de Eventos		
Acción del Actor		**Respuesta del sistema**
1.- El proveedor presiona el enlace salir		2.- El sistema ejecuta el código para salir de la sesión de usuario.
		3.- Se redirecciona a la página de Login del sistema R.E.M.
Cursos Alternativos		
1.- La autentificación del proveedor determina que éste no debe ingresar al sitio.		

Caso de uso	Identificar proveedor
Actores	Proveedor
Propósito	Permite que el proveedor se identifique mediante un código de funcionario y clave. El propósito es capturar la información de usuario para su posterior validación.
Visión General	El proveedor ingresa su código de funcionario y clave.
Tipo	Primario
Referencias	Incluye el caso de uso Autentificación proveedor
Curso Típico de Eventos	
Acción del Actor	Respuesta del sistema
1.- El proveedor ingresa al sitio de recaudación electrónica de cuentas.	2.- Despliega GUI de identificación de usuario, con campos código de usuario y clave.
3.- Ingresa su código de funcionario, clave secreta y presiona el botón aceptar.	4.- El sistema captura la información ingresada por el usuario.
	5.- Envía la información a un módulo de validación, el cual realiza el caso de uso Autentificar Proveedor.
Cursos Alternativos	

Caso de uso	Desplegar rutas de recaudación
Actores	Proveedor. Extiende a Desplegar Lista de Clientes por Ruta.
Propósito	Desplegar las rutas de recaudación y sus transportistas responsables, así como los montos y fechas de pago pactadas hasta el momento. Además, debe entregar el estado y monto total para cada ruta y los montos entregados, por entregar y no entregados.
Visión General	El proveedor ingresa al sitio WEB R.E.M. y en la página principal se despliegan por defecto las rutas de recaudación. Alternativamente, se accede a esta página a través del enlace **Rutas**.
Tipo	Primario
Referencias	Accede a caso de uso Consultar DB.
Curso Típico de Eventos	
Acción del Actor	Respuesta del sistema
1.- El proveedor ingresa al sitio WEB R.E.M. a través del caso de uso Identificación Proveedor. Entrar(Empresa)	2.- El sistema ejecuta el caso de uso consultar DB, para encontrar las rutas del proveedor (Empresa)
	3.- El sistema despliega por defecto la lista de rutas para el proveedor.
Cursos Alternativos	
1.- El usuario presiona el enlace Rutas y carga la página inicial del sistema R.E.M.	

Caso de uso	Desplegar lista de clientes por ruta
Actores	Proveedor, Extiende a caso de uso Desplegar rutas de recaudación.
Propósito	Permite desplegar la lista de clientes minoristas que posee cada recaudador asociado a una ruta.
Visión General	El proveedor presiona el enlace sobre el nombre de un recaudador (o transportista) en la página principal que muestra la lista de rutas de recaudación, y se despliega la lista de clientes (minoristas) a los que se debe despachar.
Tipo	Primario
Referencias	Referencia a Desplegar Resumen pago Factura y accede a consultar DB
Curso Típico de Eventos	
Acción del Actor	Respuesta del sistema
1.- Cliente observa lista de rutas de recaudación y presiona enlace sobre el nombre de un recaudador	2.- El sistema ejecuta el caso de uso consultar DB, para encontrar los clientes minoristas que se encuentran en la ruta del recaudador
	3.- El sistema despliega la lista de clientes minoristas para la ruta del recaudador seleccionado.
Cursos Alternativos	

Caso de uso	Desplegar resumen pago factura
Actores	Proveedor, Extiende a caso de uso Desplegar lista de clientes por ruta.
Propósito	Desplegar la información de fechas y montos pactados para pagar la factura de un determinado minorista. Además, debe desplegar el nombre del minorista, su RUT, dirección y fono.
Visión General	El proveedor presiona el enlace sobre un determinado minorista el cual está asociado a una factura. Al hacer esto, se despliega la información de los montos y fechas pactadas para el pago de dicha factura.
Tipo	Primario
Referencias	Accede a Consultar DB.
Curso Típico de Eventos	
Acción del Actor	Respuesta del sistema
1.- El proveedor observa la lista de clientes por ruta, y presiona el enlace de un cliente minorista en dicha página.	2.- El sistema ejecuta el caso de uso consultar DB, para encontrar la información pertinente al cliente minorista asociado con la factura.
	3.- El sistema despliega la factura asociada con ese minorista.
Cursos Alternativos	

Caso de uso	Autentificar Proveedor
Actores	Proveedor, RDBMS, se incluye en caso de uso Identificación Proveedor.
Propósito	Permite consultar la base de datos para validar la identificación de un determinado proveedor.
Visión General	Caso de uso es gatillado por caso de uso Identificación Proveedor.
Tipo	Primario
Referencias	Accede a consultar DB.

Curso Típico de Eventos

Acción del Actor	Respuesta del sistema
1.- El proveedor ingresa código de recaudador y clave en caso de uso identificación proveedor. Luego presiona Aceptar	2.- El sistema envia la información al caso de uso Autentificación Proveedor
	3.- El sistema autoriza al proveedor a ingresar al sistema, éste es llevado a la página principal dentro del sistema.

Cursos Alternativos

3.- El usuario no es autorizado, por lo que es llevado fuera de los límites del sistema R.E.M.

Caso de uso	Reiniciar DEMO
Actores	Proveedor
Propósito	Reinicializar los valores de la Base de datos, para comenzar una nueva demostración del producto con los valores iniciales del negocio. El estado de las transacciones debe ser **Por Entregar**, el saldo disponible por cliente debe ser de **$200.000**, deben borrarse las cuotas pactadas.
Visión General	Caso de uso es gatillado por el agente que realiza la demostración (supuesto Proveedor).
Tipo	**Opcional (Sólo disponible en versión DEMO del producto)**
Referencias	Accede a **Actualizar DB**.

Curso Típico de Eventos

Acción del Actor	Respuesta del sistema
1.- El Proveedor presiona el enlace sobre el logotipo en la parte inferior izquierda de su interfaz gráfica de usuario de la página de acceso al sistema R.E.M.	2.- El sistema Gatilla Caso de uso **Actualizar DB**.
	3.- Se reinicializan los valores de la base de datos del Sistema R.E.M.

Cursos Alternativos

Caso de uso	Actualizar DB
Actores	Proveedor, Incluida por caso de uso **Reiniciar DEMO**.
Propósito	Actualiza campos de las tablas de la base de datos que están relacionados con las transacciones dispuestas en la carga de prueba del sistema.
Visión General	Caso de uso es gatillado por el caso de uso **Reiniciar DEMO**.
Tipo	**Opcional (Sólo disponible en versión DEMO del producto)**
Referencias	Accede a la base de datos.

Curso Típico de Eventos	
Acción del Actor	**Respuesta del sistema**
1.- El caso de uso Reiniciar DEMO pasa señal de control que autoriza a actualizar base de datos.	2.- Se invoca procedimiento almacenado para reiniciar tablas.
	3.- Caso de uso devuelve el flujo de ejecución a Reiniciar DEMO.

Cursos Alternativos
3.- Si se captura un error en la ejecución del procedimiento almacenado se debe desplegar un mensaje procesado del error.

El nivel de detalle de los diagramas de casos de uso debe refinarse dependiendo de la complejidad del sistema, el grado de entendimiento de éste y la iteración en que se encuentre.

7.4.1.1.1.3 Casos de uso de alto nivel WAP R.E.M.

Caso de uso	Salir
Actores	Transportista
Tipo	Secundario
Descripción	Permite que el transportista salga del sitio WAP R.E.M.

Caso de uso	Identificar Transportista
Actores	Proveedor
Tipo	Primario
Descripción	Permite que el transportista se identifique mediante un código de funcionario y contraseña.

Caso de uso	Desplegar ruta
Actores	Transportista
Tipo	Primario
Descripción	Despliega la ruta del día, mostrando los clientes y el estado de la entrega.

Caso de uso	Desplegar factura por minorista
Actores	Transportista
Tipo	Primaria
Descripción	Despliega el detalle de la factura para un determinado cliente.

Caso de uso	Determinar estado factura
Actores	Transportista
Tipo	Secundaria
Descripción	Determina el estado de despacho de una determinada factura, es decir, si está entregada, no entregada o por entregar, códigos: Ent, NEn, PEn respectivamente.

Caso de uso	Cambiar estado despacho
Actores	Transportista
Tipo	Secundaria
Descripción	Cuando una factura no se ha entregado, es posible cambiar el estado de la orden del estado por entregar al estado no entregado.

Caso de uso	Pagar Factura
Actores	Transportista
Tipo	Primario
Descripción	Permite comenzar a pactar las cuotas y las fechas de pago de la factura de un cliente en específico.

Caso de uso	Pactar cuotas
Actores	Minorista
Tipo	Primario
Descripción	Permite pactar las fechas y los montos hasta completar el valor total de la factura. El cliente elige la fecha y el monto.

Caso de uso	Desplegar Resumen / Autoriza Pago
Actores	Minorista
Tipo	Primario
Descripción	Presenta una tabla resumen con las fechas y los montos pactados, así como el total a pagar. Al final de la página despliega un cuadro de texto que permite al minorista ingresar su contraseña para aceptar el pago de las cuotas convenidas.

Caso de uso	Desplegar clave acepta pago minorista
Actores	Minorista
Tipo	Primario
Descripción	Despliega cuadro de texto que permite al usuario ingresar la clave que autoriza las transacciones.

Caso de uso	Validar clave acepta pago cliente
Actores	Minorista
Tipo	Primario
Descripción	Confirma la clave del minorista brindando acceso la base de datos.

Caso de uso	**Actualizar DB**
Actores	RDBMS y Minorista, invocada desde **validar clave acepta pago cliente**.
Tipo	Primario
Descripción	Ejecuta procedimiento almacenado para actualizar la información de la transacción.

Caso de uso	**Desplegar Resumen y Número de Transacción**
Actores	RDBMS y Minorista, es invocada desde Actualizar DB
Tipo	Primario
Descripción	Finaliza la transacción desplegando el número de transacción y el número de factura.

Caso de uso	**Consultar DB**
Actores	RDBMS, Minorista y Transportista, es invocada desde: Autentificar transportista, desplegar ruta y determinar estado factura.
Tipo	Primario
Descripción	Realiza consultas pertinentes a través de los procedimientos almacenados específicos para satisfacer a cada caso de uso que invoca.

Caso de uso	**Autentificar Transportista**
Actores	Transportista
Tipo	Primario
Descripción	Invoca consulta a base de datos a través del caso de uso Consultar DB. Determina si la información retornada por este caso de uso permite brindar acceso al transportista que se ha identificado.

7.4.1.1.1.4 Casos de uso expandidos WAP R.E.M.

Caso de uso	Salir
Actores	Transportista y Minorista, Extiende a Autentificar Transportista y a Validar Clave Acepta Pago Cliente.
Propósito	Permite finalizar sesión del Transportista o Minorista y salir del sistema WAP R.E.M.
Visión General	El transportista presiona el enlace salir o el minorista presiona el enlace volver al finalizar una transacción exitosa. Puede ser invocado desde el caso de uso Autentificar Transportista, que determina que éste no debe ingresar al sistema. También puede ser invocado por el caso de uso Validar Clave Acepta Pago Cliente. El caso de uso Salir, retorna a la página de inicio.
Tipo	Secundario
Referencias	
Curso Típico de Eventos	
Acción del Actor	**Respuesta del sistema**
1.- El proveedor presiona el enlace salir	2.- El sistema ejecuta el código para salir de la sesión de usuario.
	3.- Se redirecciona a la página de Login del sistema R.E.M.
Cursos Alternativos	
Curso Alternativo 1	
1.- La autentificación del proveedor determina que éste no debe ingresar al sitio.	
Curso Alternativo 2	
1.- El usuario Minorista presiona el enlace Volver al haber finalizado con éxito una transacción.	

Caso de uso	**Identificar Transportista**
Actores	Transportista
Propósito	Permite que el transportista se identifique mediante un código de recaudador y clave. El propósito es capturar la información de usuario para su posterior validación.
Visión General	El transportista ingresa su código de funcionario y clave.
Tipo	Primario
Referencias	Incluye al caso de uso Autentificación Transportista
Curso Típico de Eventos	
Acción del Actor	**Respuesta del sistema**
1.- El transportista ingresa al sitio WAP de recaudación electrónica de cuentas.	2.- Despliega GUI de identificación de usuario, con campos código recaudador y clave.
3.- Ingresa su código de recaudador, clave secreta y presiona el botón aceptar.	4.- El sistema captura la información ingresada por el usuario.
	5.- Envía la información a un módulo de validación, el cual realiza el caso de uso Autentificar Transportista.
Cursos Alternativos	

Caso de uso	**Desplegar ruta**
Actores	Transportista.
Propósito	Despliega la ruta del día, mostrando los clientes y el estado de la entrega. Cada minorista posee un enlace asociado a su nombre, con el cual es posible acceder al detalle de la factura.
Visión General	El proveedor ingresa al sitio WEB R.E.M. y en la página principal se despliegan por defecto las rutas de recaudación. Alternativamente, se accede a esta página a través del enlace **Rutas**.
Tipo	Primario
Referencias	Accede a caso de uso consultar DB.
Curso Típico de Eventos	
Acción del Actor	**Respuesta del sistema**
1.- El transportista ingresa al sitio WAP R.E.M. a través del caso de uso Identificar Transportista.	2.- El sistema ejecuta el caso de uso Desplegar Ruta.
	3.- El caso de uso Desplegar Ruta invoca al caso de uso Consultar DB, para encontrar la lista de minoristas que conforman la ruta del transportista para un día determinado.
	4.- Se despliega la lista de entregas a minorista en una tabla de dos columnas que describen el nombre del minorista y el estado del despacho. El nombre de cada minorista se encuentra asociado con un enlace que permite invocar al caso de uso Desplegar Factura Por Minorista. Los estados se ven representados por los siguientes códigos: **PEn**, **Ent**, **NEn**, que representan a los estados **Por Entregar**, **Entregado** y **No Entregado,** respectivamente.
5.- El usuario selecciona un enlace en la GUI.	
Cursos Alternativos	
1.- El usuario presiona el enlace Rutas y carga la página inicial del sistema R.E.M.	

Caso de uso	**Desplegar factura por minorista**
Actores	Transportista. Extiende a caso de uso Desplegar Ruta.
Propósito	Permite desplegar el detalle de la factura perteneciente a un determinado minorista.
Visión General	El transportista presiona el enlace asociado al nombre de un determinado minorista que se encuentra en su ruta. Se invoca al caso de uso Determinar Estado Factura.
Tipo	Primario
Referencias	Referencia a Determinar Estado Factura.
Curso Típico de Eventos	
Acción del Actor	**Respuesta del sistema**
1.- El transportista observa la lista de recaudación y presiona un enlace asociado con el nombre de un minorista.	2.- El sistema ejecuta el caso de uso Determinar Estado Factura.
	3.- Se despliega la información retornada por el caso de uso Determinar Estado Factura. En esta se incluye el RUT del minorista, el número de factura, la fecha, el valor y la glosa. Dependiendo del estado de la factura, es decir, si es PEn, NEn o Ent, se contará con enlaces para pagar. En el caso de que el estado sea **Ent**, volver a la página de inicio de sesión, o establecer como NEn mediante el enlace No Entregado, para dejar constancia que no se ha podido hacer el despacho al minoristas debido a que éste no se encontraba en el lugar convenido.
4.- El cliente puede seleccionar una de los enlaces posibles, según sea el estado de la factura: **No Entregado**, **Volver** o **Pagar**.	
Cursos Alternativos	

Caso de uso	**Determinar Estado Factura**
Actores	Transportista.
Propósito	Gatillar consulta a la base de datos a través del caso de uso Consultar DB, y retornar información acerca del estado de la factura al caso de uso Desplegar Factura por minorista.
Visión General	Presta servicios al caso de uso Desplegar Factura por Minorista.
Tipo	Primario
Referencias	Accede a Consultar DB.
Curso Típico de Eventos	
Acción del Actor	**Respuesta del sistema**
1.- El transportista observa la ruta del día y presiona el enlace asociado a un determinado minorista.	2.- El sistema ejecuta el caso de uso Desplegar Factura por Minorista, el cual invoca a su vez al caso de uso **Determinar Estado Factura**.
	3.- **Determinar Estado Factura** realiza una serie de consultas a través del caso de uso Consultar DB. Se obtiene la siguiente información correspondiente a la factura del minorista: Estado de la factura, RUT minorista, número factura, Fecha, Valor y Glosa.
	4.- **Determinar Estado Factura** retorna la información al caso de uso **Desplegar Factura por Minorista**, el cual le da un formato de salida apropiado y la despliega en la GUI del dispositivo.
5.- Dependiendo del estado de la factura, el transportista puede seleccionar uno de los siguientes enlaces: **No Entregado**, **Pagar**, o **Volver**.	
Cursos Alternativos	

Caso de uso	**Cambiar Estado Despacho**
Actores	Transportista.
Propósito	Cambiar el estado de una entrega de **Por Entregar** a **No Entregado**.
Visión General	Cuando una factura no se ha entregado, es posible cambiar el estado de la orden del estado **Por Entregar** al estado **No Entregado**. Esto a partir de un enlace provisto en la GUI.
Tipo	Secundario
Referencias	Accede a Actualizar DB.

Curso Típico de Eventos

Acción del Actor	Respuesta del sistema
1.- El transportista ingresa al sitio WAP R.E.M.. Luego accede al detalle de la factura de un determinado minorista a través del enlace asociado a éste.	2.- El sistema despliega la información del detalle de la factura del minorista.
	3.- Si no fue posible realizar la entrega al minorista, el transportista debe presionar el enlace No Entregado, se solicitará confirmar la identidad del transportista para ejecutar cambio de estado de la factura.
	4.- Se despliega un mensaje de notificación del cambio de estado de la factura.
	5.- Ejecuta el caso de uso Salir.

Cursos Alternativos

Caso de uso	**Pagar Factura**
Actores	Transportista
Propósito	Permite comenzar a pactar las cuotas y las fechas de pago de la factura de un cliente en específico.
Visión General	El transportista presiona el enlace Pagar desde el caso de uso **Desplegar factura por minorista** y comienza el proceso de pactar las cuotas y fechas de pago a través de la ejecución del caso de uso **Pactar Cuotas**.
Tipo	Primario
Referencias	Referencia a caso de uso Pactar Cuotas.

Curso Típico de Eventos

Acción del Actor	Respuesta del sistema
1.- El Transportista presiona el enlace Pagar desde la implementación del caso de uso Desplegar Factura por Minorista.	2.- El sistema gatilla caso de uso **Pactar Cuotas**, el cual inicia el proceso de pago.
3.- El transportista ingresa la fecha y el monto de los pagos según sea su disponibilidad. La fecha se debe ingresar como una secuencia de seis dígitos que representan el día, el mes y el año, cada uno consta de dos dígitos, por ejemplo: 2 de octubre de 2003, sería ingresado como 021003. El monto se ingresa mediante una secuencia de dígitos que representa la cifra a ser pagada. No se permite ningún tipo de separado como coma o punto.	4.- El sistema continua con el proceso de pago en el caso de uso Pactar Cuotas.
	5.- Al terminar el proceso de pago, se sale del sistema.

Cursos Alternativos

Caso de uso	**Pactar Cuotas**
Actores	Transportista.
Propósito	Establecer las fechas y montos de pago en que el minorista autoriza cancelar el precio total del producto que se ha despachado.
Visión General	Caso de uso es gatillado por el caso de uso **Pagar Factura**. Representa el proceso de pago en el que se realiza la selección de fechas y montos hasta completar el valor total del producto que se ha despachado.
Tipo	Primario
Referencias	Referencia Desplegar Resumen / Autoriza Pago.

Curso Típico de Eventos

Acción del Actor	Respuesta del sistema
1.- El Transportista presiona el enlace Pagar desde la implementación del caso de uso **Desplegar Factura por Minorista**.	2.-Se ejecuta la implementación del caso de uso **Pagar Factura**.
4.- El transportista ingresa la fecha de pago 1 ingresando el día, el mes y año con dos dígitos cada uno, por ejemplo: 2 de octubre de 2003, sería ingresado como 021003.	3.- El sistema gatilla caso de uso **Pactar Cuotas**, el cual despliega la primera interfaz para seleccionar la fecha del pago, ésta incluye lo siguiente: número factura, monto por pagar y un cuadro de texto para ingresar la fecha de pago 1.
	5.- El sistema despliega la segunda interfaz para seleccionar el monto del pago, que incluye lo siguiente: número factura, monto por pagar, fecha 1 y un cuadro de texto para ingresar el monto de pago 1.
	6.- Si lo que resta por pagar es igual a cero, se ejecuta el paso 7. De lo contrario, se recalcula la cantidad que queda por pagar para pactar una nueva fecha y un nuevo monto constituyendo la fecha y monto de pago 2,3,4, ..., n en los pasos 3 y 5.
	7.- Se ejecuta la implementación del caso de uso Desplegar Resumen / Autoriza Pago.

Cursos Alternativos

3..- Si se captura un error en la ejecución del procedimiento almacenado se debe desplegar un mensaje procesado del error.

Caso de uso	Desplegar Resumen / Autoriza Pago
Actores	Transportista, Minorista.
Propósito	Mostrar al minorista un resumen de las fechas y montos pactados para pagar el valor del producto que se ha despachado.
Visión General	Al finalizar el proceso de pago implementado por el caso de uso Pactar Cuotas, el transportista debe entregar el dispositivo PCS al minorista para que éste confirme y autorice la transacción ingresando su número secreto (clave acepta pago).
Tipo	Primario
Referencias	Referencia Desplegar Clave Acepta Pago Minorista
Curso Típico de Eventos	
Acción del Actor	Respuesta del sistema
1.- Al finalizar el proceso de pago implementado por caso de uso **Pactar Cuotas**.	2.- Se despliega una tabla resumen al Transportista, que debe entregar el dispositivo al minorista para que verifique las fechas y los montos pactados, así como el valor total a pagar. Se invoca al caso de uso **Desplegar Clave acepta Pago Minorista**.
Cursos Alternativos	

Caso de uso	Desplegar Clave acepta Pago Minorista
Actores	Minorista.
Propósito	Desplegar cuadro de texto en donde se debe ingresar la contraseña del minorista que autoriza la transacción.
Visión General	El minorista debe ingresar su clave acepta pago.
Tipo	Primario
Referencias	Referencia **Validar Clave Acepta Pago Cliente**.
Curso Típico de Eventos	
Acción del Actor	Respuesta del sistema
	1.- El sistema despliega al minorista una tabla resumen con las fechas y los montos pactados, así como el valor total a pagar.
	2.- Despliega un cuadro de texto para ingresar la clave acepta pago del cliente, la cual debe ser una secuencia de 4 dígitos.
3.- El cliente ingresa la secuencia de cuatro dígitos que constituye su clave acepta pago.	4.- Se invoca el caso de uso **Validar Clave Acepta Pago Cliente**.
Cursos Alternativos	

Caso de uso	**Validar Clave Acepta Pago Cliente**
Actores	Minorista.
Propósito	Invocar **Consultar DB** para validar la contraseña del minorista y aprobar o rechazar la transacción.
Visión General	Caso de uso se gatilla cuando el minorista ingresa su clave acepta pago y presiona Aceptar. Tras lo cual se realiza una consulta a la base de datos para determinar si la transacción es válida o no. Si es válida, se actualiza la base de datos registrando las fechas y los montos. También se reduce la capacidad crediticia del minorista en un monto igual al valor total que se ha pactado.
Tipo	Primario
Referencias	Referencia **Consultar DB** y **Actualizar DB**.
Curso Típico de Eventos	
Acción del Actor	**Respuesta del sistema**
1.- Minorista ingresa su clave acepta pago, luego presiona Aceptar.	2.- Se inicia el caso de uso **Validar Clave Acepta Pago Cliente**, con lo cual se realiza una consulta a la base de datos para determinar la validez de la transacción.
	3.- Si el usuario es quien dice ser y cuenta con el dinero disponible en su crédito preaprobado entonces se aprueba la transacción. De lo contrario, se rechaza la transacción.
3.- El cliente ingresa la secuencia de cuatro dígitos que constituye su clave acepta pago.	4.- Si se aprueba la transacción, se invoca al caso de uso **Desplegar Resumen y Número de Transacción**.
5.- El minorista presiona el enlace salir o volver según sea el caso.	5.- Se invoca al caso de uso Salir.
Cursos Alternativos	
4.- Si se Rechaza la transacción, se despliega un mensaje de notificación de rechazo de transacción y se habilita un enlace para Salir.	

Caso de uso	**Desplegar Resumen y Número de Transacción**
Actores	Minorista. Extiende a **Validar Clave Acepta Pago Cliente**.
Propósito	Despliega un resumen de la transacción que consta del estado de la transacción, el número de transacción y el número de factura.
Visión General	Idem.
Tipo	Primario
Referencias	
Curso Típico de Eventos	
Acción del Actor	**Respuesta del sistema**
Cursos Alternativos	
(No es necesaria una mayor descripción)	

Los casos de **uso Autentificar Transportista**, **Consultar DB** y **Actualizar DB**, son análogos a sus pares descritos en el modelo de casos de uso del sitio WEB R.E.M. (**Autentificar Proveedor**, **Consultar DB**, **Actualizar DB**), en la sección 7.4.1.1.1.2.

No es necesario describir todos los casos de uso de alto nivel como casos de uso expandidos, sólo se describen los más importantes. El nivel de detalle de los diagramas de casos de uso debe refinarse dependiendo de la complejidad del sistema.

El conjunto de casos de uso de alto nivel y expandidos, representan el vocabulario del sistema, el cual se puede refinar en cada iteración para resolver ambigüedades. Además, constituye una pieza fundamental en la administración de requerimientos. Por esta razón, no deben ser subestimados bajo ninguna circunstancia, ya que deben estar presentes en cualquier negociación de requerimientos con el cliente.

7.4.1.1.2 Workflow de análisis y diseño

El objetivo del workflow de análisis y diseño es especificar en primera instancia un **modelo de análisis**, que no cuenta con todos los detalles del sistema, pero si representa una buena abstracción del comportamiento del sistema desde una perspectiva global. El modelo de análisis se infiere a partir de los casos de uso que se encuentran en continuo refinamiento en cada iteración, evolucionando hasta llegar a un punto en que se pueda hacer un buen modelo de diseño del sistema software.

7.4.1.1.2.1 Modelamiento Web del sitio WEB R.E.M.

El modelo de análisis se verá representado por el modelamiento Web del sitio WEB R.E.M.. El tipo de diagrama usado es un diagrama de clases fuertemente estereotipado, el cual representa una extensión al lenguaje de modelamiento unificado y que es explicado en el **Anexo 12**. La figura 7.24 muestra el modelamiento Web del sitio WEB R.E.M. y su lógica de negocio inicial, es decir, su página de acceso al sistema. La figura 7.25 muestra la continuación del modelamiento Web de la figura 7.24, a partir del FrameSet **WEB-REM**, una vez que el proveedor a ingresado dentro del sistema.

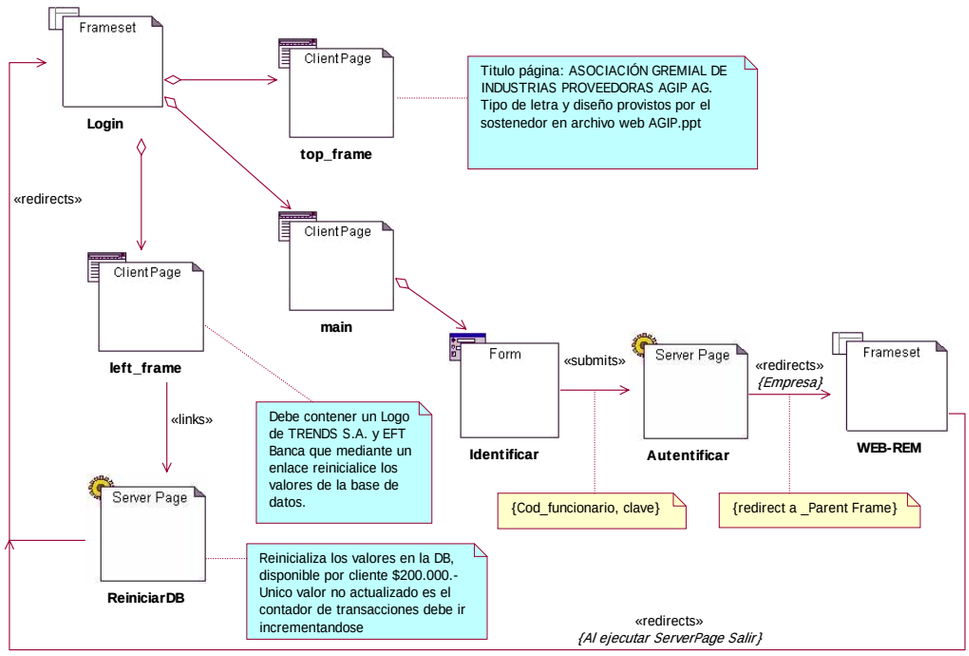


Figura 7.24: Modelamiento Web en UML de la lógica de negocio de la página de acceso al sistema WEB-REM. Corresponde al modelo de análisis del sistema WEB R.E.M.

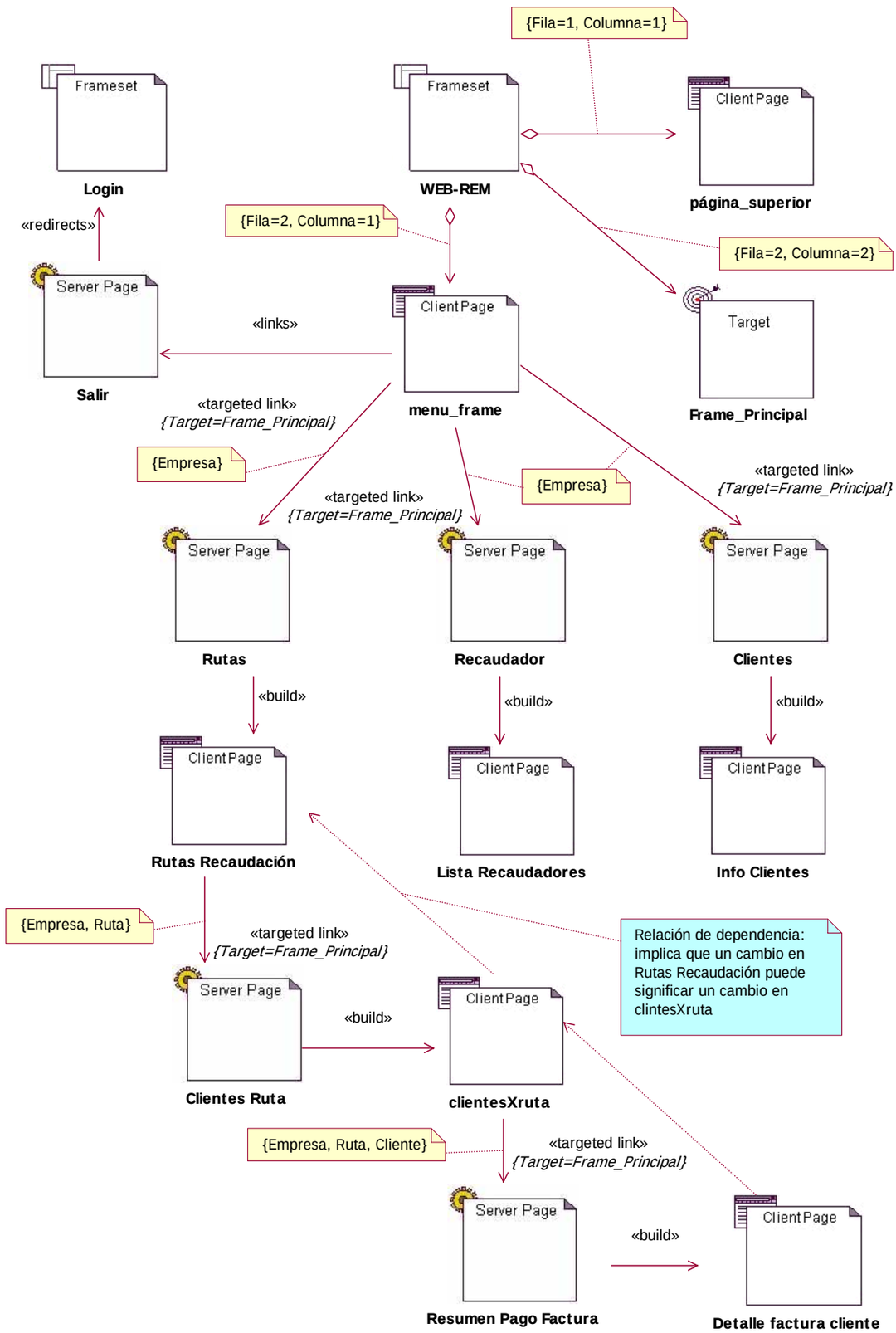


Figura 7.25: Modelamiento Web en UML de la lógica de negocio de la página de acceso al sistema WEB-REM. Corresponde al modelo de análisis del sistema WEB R.E.M.

En las figuras 7.24 y 7.25, nótese el paso de parámetros entre las llamadas sucesivas a páginas de cliente y a páginas de servidor. Estos modelos se asemejan al modelo que describiría el funcionamiento del sitio WEB-REM elaborado de una forma artesanal. Sin embargo, aparece como un buen modelo de análisis inicial para el diseño del nuevo sitio WEB-REM. El paso de mensajes muestra cómo no se están aprovechando las características de orientación a objeto del lenguaje C#.

Se considerará un nuevo punto de partida en el modelamiento del sitio WEB-REM. La figura 7.26 muestra la página inicial **default.aspx**, la cual es construida en parte por una página de servidor **WebForm1**. En alguna parte, en la lógica de procesamiento de la página de servidor **WebForm1** debe redireccionarse a la página de cliente **REM.aspx**, una vez validado el usuario.

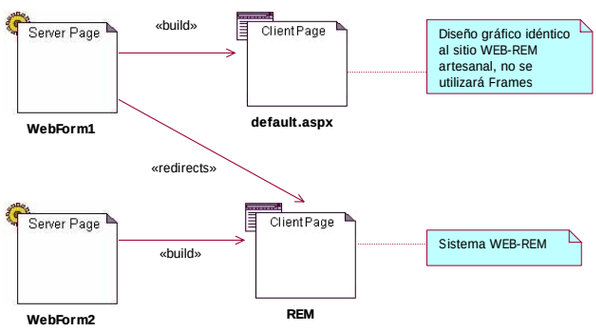


Figura 7.26: Modelamiento Web sitio prototipo REM

7.4.1.1.2.2 Diseño de la base de datos

La primera aproximación del modelo entidad relación que permitirá implementar la base de datos del sistema WEB-REM.

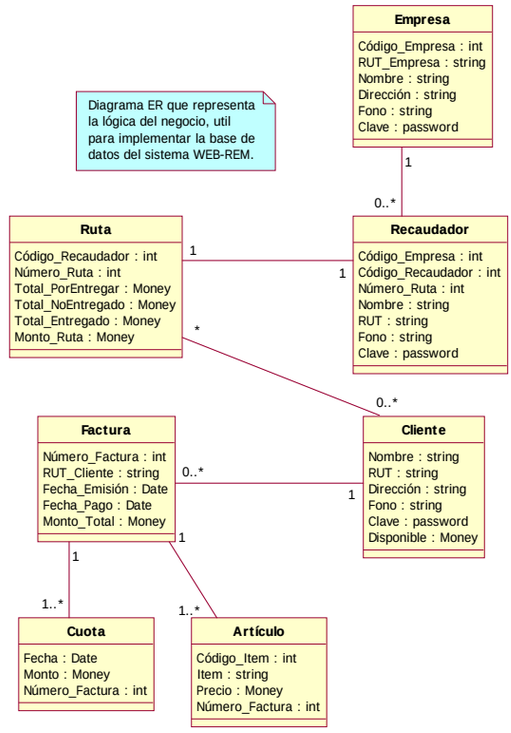


Figura 7.27: Modelo Entidad relación que representa las relaciones y estructuras de tablas en la base de datos del sistema WEB-REM.

La figura 7.27, representa un diagrama de clases con los objetos que intervienen en el prototipo WEB-REM.

7.4.1.1.3 Workflow de administración de proyectos

7.4.1.1.3.1 Estimación del proyecto software prototipo WEB R.E.M. versión 5

Se puede determinar a partir de la clasificación de aplicaciones descrita en el **Capítulo 6**, que la aplicación que se pretende construir pertenece al modelo de **composición de aplicaciones**. Es debido a esto, que la estimación de esfuerzo se obtendrá a partir del **modelo de costo de composición de aplicaciones**. Como se vio en el capítulo 6, los puntos objetos consisten en el conteo de pantallas, informes y módulos desarrollados en 3GL, cada uno ponderado por un factor de complejidad de tres niveles (Bajo, Medio y Alto). Estos fueron obtenidos observando los casos de uso y los diagramas de casos de uso. Además, se consideraron los modelos expuestos en las figuras 7.24 y 7.25 y el bosquejo del diseño de la base de datos de la figura 7.27. Sin embargo, dicho recuento, ha sido fuertemente guiado a partir de los casos, ver tabla 7.4.

- **Recuento de puntos objeto, generado por el workflow de arquitectura de software.**

Tabla 7.4: Recuento de puntos objeto basado en análisis y diseño preliminar.

WEB-REM				
Número de GUIs	**Cantidad**	**Complejidad**	**PCA**	**Puntos Objeto**
Página de inicio	**1**	MEDIO	2	2
Página REM	**1**	DIFICIL	3	3
SubTotal				**5**
Informes	**Cantidad**	**Complejidad**	**PCA**	
Rutas de recaudación	**1**	MEDIO	5	5
- Detalle cuotas	**1**	MEDIO	5	5
Lista de clientes por ruta	**1**	MEDIO	5	5
- Detalle cuotas	**1**	MEDIO	5	5
Resumen pago factura	**1**	MEDIO	5	5
- Detalle cuotas	**1**	MEDIO	5	5
Lista de recaudadores	**1**	MEDIO	5	5
Información clientes	**1**	MEDIO	5	5
SubTotal				**40**
Componentes	**Cantidad**	**Complejidad**	**PCA**	
cuadros de texto	**2**	FACIL	0	0
botón	**1**	FACIL	0	0
enlaces	**4**	FACIL	0	0
Imagenes	**5**	FACIL	0	0
Etiquetas (labels)	**8**	FACIL	0	0
Tablas	**8**	MEDIO	0	0
Archivos externos:				
- default.aspx	**1**	FACIL	0	0
- REM.aspx	**1**	FACIL	0	0
- default.aspx.cs	**1**	MEDIO	0	0
- REM.aspx.cs	**1**	MEDIO	0	0
- Autentificar.cs	**1**	MEDIO	0	0
SubTotal				**0**
Subtotal Puntos Objeto				**45**
			NOP =	**36**

Tabla 7.5: Recuento de puntos objeto basado en análisis y diseño preliminar del sitio WAP REM.

WAP-REM				
Número de GUIs	Cantidad	Complejidad	PCA	Puntos Objeto
Página de acceso	1	SIMPLE	1	1
Página principal	1	MEDIO	2	2
Ingresar Fechas	1	SIMPLE	1	1
Ingresar Monto	1	SIMPLE	1	1
Clave acepta pago	1	SIMPLE	1	1
Error	1	SIMPLE	1	1
SubTotal				7
Informes	Cantidad	Complejidad	PCA	
Rutas de recaudación	1	SIMPLE	2	2
Detalle factura	1	SIMPLE	2	2
Resumen transacción	1	SIMPLE	2	2
Transacción Aprobada	1	SIMPLE	2	2
SubTotal				8
Componentes 3GL	Cantidad	Complejidad	PCA	
Simples	1	MEDIO	0	0
SubTotal				0
Subtotal Puntos Objeto				15
			NOP =	12
			Total NOP=	48

Tabla 7.6: Resumen de estimaciones preliminares del desarrollo del prototipo R.E.M.

Estimaciones				
Suma SF	14.72		Total NOP=	48
% Reutilización	20			
Constantes			Trabaja	
A	2.5924		Días al mes	20
B	0.9124		Horas al mes	160
C	1.5402		Horas al día	9
D	0.2799			
F	0.30934			
Productividad	Valor	PM Norm.	TDEV	STAFF
Baja	13	3.887	2.344	1.66
Media	19	2.659	2.084	1.28
Alta	25	2.021	1.915	1.06

TDEV Estimado [días]	41.69
TDEV Estimado [horas]	375.2

NOP: Nuevos puntos objeto, dado por la siguiente ecuación:

$$NOP = \frac{(45)\times(100-20)}{100} = 36$$

$$PM_{Norm.} = \frac{160 \times PM}{152}$$

donde el valor de PM, está dado por :

$$PM = \frac{NOP}{PROD}$$

Los valores restantes son calculados a partir de las siguientes tablas que fueron extraídas del capítulo 6.

- **Productividad promedio (Tabla 6.3)**

  Donde la productividad a utilizar estará dada por el promedio aritmético de 13 y 25:

$$\text{Productividad} = PROD = \frac{13+25}{2} = 19$$

El valor de la **reutilización** se estima en un 20% del código presente en la versión artesanal del producto.

- **Cálculo de los Factores de Escala del proyecto**

  Se han seleccionado los niveles de acuerdo a las características del proyecto.

Tabla 7.7: Factores de escala del proyecto software.

Factor de Escala (SFj)	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
PREC	Completamente sin precedentes	Prácticamente sin precedentes	Casi sin precedentes	Algo familiar	Muy familiar	Completamente familiar
FLEX	Riguroso	Relajación ocasional	Algo de Relajación	Conformidad general	Algo de conformidad	Objetivos generales
RESL	Poco (20%)	Algo (40%)	A menudo (60%)	Generalmente (75%)	En su mayor parte (90%)	Por completo (100%)
TEAM	Interacciones muy difíciles	Algunas interacciones difíciles	Interacciones básicamente cooperativas	Bastante cooperativo	Altamente cooperativo	Interacciones completas
PMAT	SW-CMM NIVEL 1 BAJO	SW-CMM NIVEL 1 ALTO	SW-CMM NIVEL 2	SW-CMM NIVEL 3	SW-CMM NIVEL 4	SW-CMM NIVEL 5
Significado de los Drivers de Costo						
PREC: Precedencia.						
FLEX: Flexibilidad de desarrollo.						
RESL: Resolución de Arquitectura/Riesgo						
TEAM: Cohesión de Equipo						
PMAT: Madurez del proceso						

Tabla 7.8: Ponderaciones asociadas para el cálculo de los factores de escala del proyecto software.

PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

$$\sum SF = 4{,}96 + 1{,}01 + 1{,}41 + 1{,}1 + 6{,}24 = 14{,}72$$

$$TDEV_{NS} = 1{,}5402 \times (PM_{NS})^{F} = 1{,}5402 (PM_{NS})^{0{,}30934}$$
$$donde: F = 0{,}2799 + 0{,}2 \times 0{,}01 \times 14{,}72$$
$$= 0{,}30934$$

El valor total de nuevos puntos objetos será de 48, entonces:

- **Rango de estimaciones según niveles de productividad**

Tabla 7.9: Valores estimados según productividad.

Productividad	PM	PM Norm.	TDEV	STAFF
Baja	$\frac{48}{13} = 3{,}69$	3,887	$TDEV = 1{,}5402 \cdot (3{,}887)^{0{,}30934} = 2{,}344$	$\frac{3{,}887}{2{,}344} = 1{,}66$
Promedio	$\frac{48}{19} = 2{,}53$	2,659	$TDEV = 1{,}5402 \cdot (2{,}659)^{0{,}30934} = 2{,}084$	$\frac{2{,}659}{2{,}084} = 1{,}28$
Alta	$\frac{48}{25} = 1{,}92$	2,021	$TDEV = 1{,}5402 \cdot (2{,}021)^{0{,}30934} = 1{,}915$	$\frac{2{,}021}{1{,}915} = 1{,}06$

7.4.1.1.3.2 Planificación del proyecto software

La tabla de distribución que se usará es la siguiente:

Tabla 7.10: Distribución de tiempos y esfuerzos de los workflows para cada fase.

Fase	Inicio	Elaboración	Construcción	Transición
Tiempo COCOMO2	12.5	37.5	62.5	12.5
Esfuerzo COCOMO2	6	24	76	12
Administración	14	12	10	14
Entorno	10	8	5	5
Requerimientos	38	18	8	4
Diseño	19	36	16	4
Implementación	8	13	34	19
Evaluación	8	10	24	24
Despliegue	3	3	3	30

Según la **productividad promedio**, se espera la siguiente distribución de esfuerzo, tiempo y personal para cada fase.

Tabla 7.11: Cálculo de la distribución de tiempo de desarrollo, esfuerzo y personal para las fases del proyecto software.

Productividad Media	Inicio	Elaboración	Construcción	Transición
PM	0.1596	0.6382	2.0211	0.3191
TDEV en meses	0.2605	0.7816	1.3027	0.2605
STAFF	0.6124	0.8165	1.5514	1.2248
TDEV en días	5.2109	15.63	26.05	5.2109
		Total días :	41.69	

En la tabla 7.11, se observa que el mayor esfuerzo se concentra en las fases de elaboración y construcción, el staff requerido en la mayoría de las fases es de 1 persona, sin embargo, en la fase de construcción se recomienda contar con dos personas. La duración de la fase de elaboración y construcción es de 41,69 días. Se recomienda contar con 5,21 días para las fases de inicio y transición.

En cuanto a la distribución de esfuerzo, tiempo y personal, para cada workflow que participa en el desarrollo del proyecto, se tiene lo siguiente:

- **El esfuerzo estará distribuido de la siguiente forma:**

Tabla 7.12: Distribución de esfuerzo de los workflows del proceso para cada fase del proyecto.

Esfuerzo	Inicio	Elaboración	Construcción	Transición
Administración	0.02	0.08	0.20	0.04
Entorno	0.02	0.05	0.10	0.02
Requerimientos	0.06	0.11	0.16	0.01
Diseño	0.03	0.23	0.32	0.01
Implementación	0.01	0.08	0.69	0.06
Evaluación	0.01	0.06	0.49	0.08
Despliegue	0.00	0.02	0.06	0.10
		Total PM:	2.659	

Multiplicando los valores para cada uno de los workflows de la tabla 7.11 con el valor del tiempo COCOMO2 para la fase respectiva, por el valor del tiempo de desarrollo estimado para

una productividad media en la tabla 7.6, por el número de días trabajados al mes, se obtiene la siguiente información acerca del tiempo utilizado por los workflows en cada fase.

Tabla 7.13: Tiempo utilizado en cada fase en [días] y [horas].

TDEV	Inicio	Elaboración	Construcción	Transición
Administración	0.730	1.876	2.605	0.730
Entorno	0.521	1.251	1.303	0.261
Requerimientos	1.980	2.814	2.084	0.208
Diseño	0.990	5.628	4.169	0.208
Implementación	0.417	2.032	8.859	0.990
Evaluación	0.417	1.563	6.253	1.251
Despliegue	0.156	0.469	0.782	1.563
		Total TDEV:	41.6873	
		Total horas:	375	

En forma análoga, multiplicando la tabla 7.11, de distribución de tiempos y esfuerzo, por el nivel de personal estimado para una productividad media, se puede obtener una idea de lo que estará haciendo el staff asignado en cada fase y workflow del proyecto software.

Tabla 7.14: Personal utilizado por cada workflow en cada fase del proyecto.

Personal	Inicio	Elaboración	Construcción	Transición
Administración	0.18	0.15	0.13	0.18
Entorno	0.13	0.10	0.06	0.06
Requerimientos	0.48	0.23	0.10	0.05
Diseño	0.24	0.46	0.20	0.05
Implementación	0.10	0.17	0.43	0.24
Evaluación	0.10	0.13	0.31	0.31
Despliegue	0.04	0.04	0.04	0.38

Los valores no son muy significativos, ya que se estima que es necesaria una persona para las fases de elaboración y construcción en cada workflow, por lo que esta persona es la misma. Además se sugiere una persona para las actividades de la fase de inicio y una persona para las actividades de la fase de transición, además de una persona en la captura de requerimientos, análisis y diseño.

7.4.1.1.3.3 Evaluación de riesgos del proyecto

Los riesgos del proyectos serán divididos como sigue:

$$Riesgo\,Total = RPLAN + RPROD + RPERS + RPROC + RPLAT + RREUT$$

$$RPLAN = Riesgo\ de\ Planificación$$

$$RPROD = Riesgo\ del\ \mathrm{Pr}\ oducto$$

$$RPERS = Riesgo\ del\ Personal$$

$$RPROC = Riesgo\ del\ \mathrm{Pr}\ oceso$$

$$RPLAT = Riesgo\ de\ la\ plataforma$$

$$RREUT = Riesgo\ de\ \mathrm{Re}\ utilización$$

$$Riesgo\,Total\ de\ un\ módulo = \left(\frac{Riesgo\,Total}{373}\right)\cdot 100$$

Tabla 7.15: Ponderación de riesgos del proyecto listados por frente de riesgo y fase.

WEB-REM	Inicio	Elaboración	Construcción	Transición
RPLAN	0.2	0.15	0.1	0.05
RPROD	0.2	0.1	0.05	0.05
RPERS	0.15	0.15	0.15	0.15
RPROC	0.2	0.2	0.15	0.2
RPLAT	0.15	0.1	0.05	0
RREUT	0.05	0.05	0.05	0.05
SUMA	0.95	0.75	0.55	0.5
Riesgo total	0.25	0.20	0.15	0.13

La ponderación de los riesgos de cada uno de los seis frentes ha sido determinada según la experiencia adquirida en el desarrollo de proyectos software. **Se entiende como la probabilidad de que ocurra un incidente dentro de alguno de los seis ámbitos expuestos anteriormente. Esta probabilidad variará según la fase y la iteración en la que se encuentre dentro del proceso de desarrollo.**

Es posible realizar un análisis estadístico similar al realizado en el capítulo 6, para registrar la curva de riesgo asociada a cada ámbito. De modo de obtener una mejor aproximación a la realidad.

7.4.1.1.3.4 Roles del proyecto

Según las estimaciones obtenidas en las secciones anteriores, se contemplan tres roles que satifacen las actividades del proceso de desarrollo, ver tabla 7.16.

Tabla 7.16: Definición de roles del proyecto software.

ROL	Worflows involucrados
Administrador de Proyectos	Administración de Proyectos
Ingeniero de Software	Requerimientos Diseño Evaluación
Programador	Implementación Despliegue Entorno

Según esta definición de roles para el proyecto, se tienen los siguientes tiempos de desarrollo por fase:

Tabla 7.17: Tiempo en días por rol para cada fase del proyecto.

TDEV por Rol	Inicio	Elaboración	Construcción	Transición	Total
Admin. Proyecto	1	2	3	1	6
Ing. Software	3	10	13	2	28
Programador	1	4	11	3	19
Total	5	16	26	5	52

Si se planifican tres iteraciones, según el grado de entendimiento del problema, la distribución de tiempo de desarrollo para cada iteración estará dada por la siguiente tabla:

Tabla 7.18: Distribución de tiempo de desarrollo para las fases de cada iteración.

TDEV por Iteración [Horas]	Inicio	Elaboración	Construcción	Transición
Admin. Proyecto	2	6	8	2
Ing. Software	10	30	38	5
Programador	3	11	33	8
Total	16	47	78	16
Horas por Iteración	**125**			
Total Horas	**375**			
Horas Estimadas Iteración	**13.896**			

Con la cual, es posible realizar una carta Gantt para cada iteración. El **Anexo 13**, muestra la planificación de la primera iteración del proyecto REM versión 5 siguiendo la metodología EFT-SDM.

La distribución de personal por fase será la siguiente:

Tabla 7.19: Distribución del personal por fase.

Personal	Inicio	Elaboración	Construcción	Transición
Admin. Proyecto	0.2	0.2	0.1	0.2
Ing. Software	0.8	0.8	0.6	0.4
Programador	0.3	1.5	1.8	0.9
Total	**3**	**4**	**4**	**3**

En conjunto, la información recabada en la sección 7.4.1.1.3, aporta el conocimiento necesario para comenzar a realizar una planificación para cada iteración del proyecto software.

Además, es necesario realizar las estimaciones de costo pertinentes al proyecto, las cuales se exponen en la tabla 7.20, 7.21 y 7.22.

Tabla 7.20: Cálculo de las estimaciones de costo del proyecto REM versión 5 siguiendo metodología EFT-SDM, 1 programador.

Valor UF	16696.6	TDEV TOTAL	41.69				
Costo Mes Grupo	Roles	Cant.	Sueldo [UF/mes]	I	E	C	T
Gerente / Admin. Proyecto	Admin. Proyectos	1	80	1.40	3.60	5.00	1.40
Ing. Software	Requerimientos	1	50	4.06	12.00	15.00	2.00
	Diseño						
	Evaluación						
Programador	Implementación	1	45	1.18	4.05	11.81	3.04
	Despliegue						
	Entorno						
	Costo Mes Grupo [UF]		175	6.64	19.65	31.81	6.44

Tabla 7.21: Cálculo de las estimaciones de costo del proyecto REM versión 5 siguiendo metodología EFT-SDM, 2 programadores.

Valor UF	16696.6	TDEV TOTAL	41.69				
Costo Mes Grupo	Roles	Cant.	Sueldo [UF/mes]	I	E	C	T
Gerente / Admin. Proyecto	Admin. Proyectos	1	80	1.40	3.60	5.00	1.40
Ing. Software	Requerimientos	1	50	4.06	12.00	15.00	2.00
	Diseño						
	Evaluación						
Programador	Implementación	2	45	2.36	8.10	23.63	6.08
	Despliegue						
	Entorno						
	Costo Mes Grupo [UF]		175	7.83	23.70	43.63	9.48

Al verificar los costos del proyecto software en las fases de inicio y transición, cuando el costo de grupo está influenciado por la presencia de 1 programador, se obtiene un costo total de grupo de 6.64 [UF/mes] para la fase de inicio y 6.44 [UF/mes] para la fase de transición. Por otro lado, para las fases de elaboración y construcción, en las que se requieren 2 programadores, se obtiene un costo total de grupo de 23.7 [UF/mes] para la fase de elaboración y 43.63 [UF/mes] para la fase de construcción. La tabla 7.22, muestra los cálculos necesarios para determinar el costo total del proyecto.

Tabla 7.22: Resumen del costo total del proyecto.

Planificación	I	E	C	T
Costo Mes Grupo [UF]	6.64	23.70	43.63	6.44
Factor Riesgo WEB-REM	0.25	0.20	0.15	0.13
Costo Total Fase [UF]	8.34	28.47	50.06	7.30
Costo Total Proyecto [UF]	94.17			
Costo Total Proyecto [$]	$1,572,237			

7.4.1.1.4 Workflow de entorno

El workflow de entorno, se encarga de satisfacer las necesidades en cuanto a infraestructura de los equipos de trabajo y conectividad, además muchas veces cumple el rol de dar soporte al área de desarrollo. También le conciernen las condiciones de trabajo del área de desarrollo, velando porque éstas sean las adecuadas. Estas incluyen a los factores de higiene o ambientales, tales como: **niveles de ruido, temperatura, luz, etc.**

Se debe determinar el hardware y software, así como las necesidades de conectividad y configuraciones para cada rol correspondiente a un determinado workflow. La idea es no involucrar a los desarrolladores en la configuración de los equipos. Esta labor es llevada a cabo generalmente por un área de soporte. Se considera que los equipos están disponibles con la configuración mínima. Los desarrolladores deben verificar la condición de los equipos.

Se debe disponer de los siguientes equipos que constituyen los requerimientos mínimos para el desarrollo del proyecto WEB-REM. Estos se han organizado según los roles que participan en el proyecto.

Tabla 7.23: Requerimientos mínimos de infraestructura de hardware y software para el desarrollo del proyecto WEB-REM.

Admin. Proyecto		
	Hardware	CPU Pentium III - 700 Mhz / 128 MB RAM / 8GB HDD
	Software	Windows 2000 Professional
		Microsoft Office 2000
		Microsoft Project 2000
Ing. Software		
	Hardware	CPU Pentium III - 700 Mhz / 256 MB RAM / 10 GB HDD
	Software	Windows 2000 Server
		Microsoft Office 2000
		Microsoft Project 2000
		Microsoft Visual Studio .NET Architect Edition
		Rational XDE v.2002
Programador		
	Hardware	CPU Pentium III - 700 Mhz / 256 MB RAM / 10 GB HDD
	Software	Windows 2000 Server
		Microsoft Office 2000
		Microsoft Project 2000
		Microsoft Visual Studio .NET Architect Edition
		Rational XDE v.2002
Requerimientos Comunes		Acceso a Internet
		Acceso a LAN
		Correo electrónico

7.4.1.2 Fase de elaboración

7.4.1.2.1 Workflow de requerimientos

Suficiente con la fase anterior hasta el momento. Se debe iniciar un esfuerzo de prototipado en esta fase.

7.4.1.2.2 Workflow de análisis y diseño

Se realizan algunos bosquejos de modelos que pretenden servir de guía para la construcción del prototipo en esta iteración. La figura 7.28 muestra un diseño preliminar tentativo, que intenta modelar la página de acceso del sistema WEB-REM.

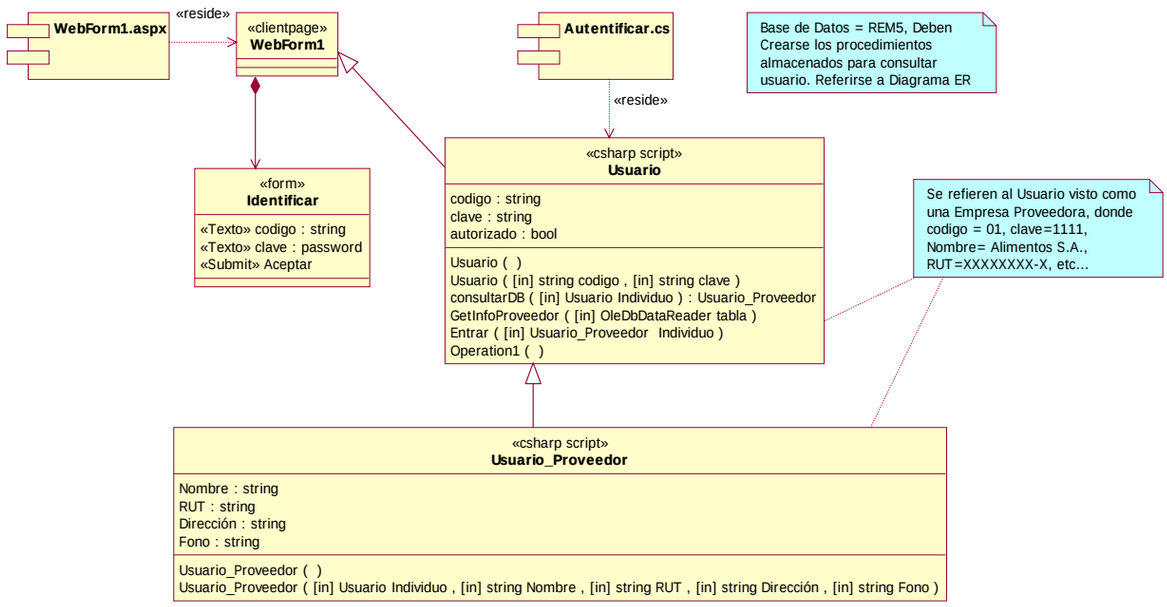


Figura 7.28: Modelo preliminar tentativo de los objetos tras la página de acceso.

La figura 7.29, muestra una posible extensión de la clase **Usuario_Proveedor** hacia un posible modelo de base de datos.

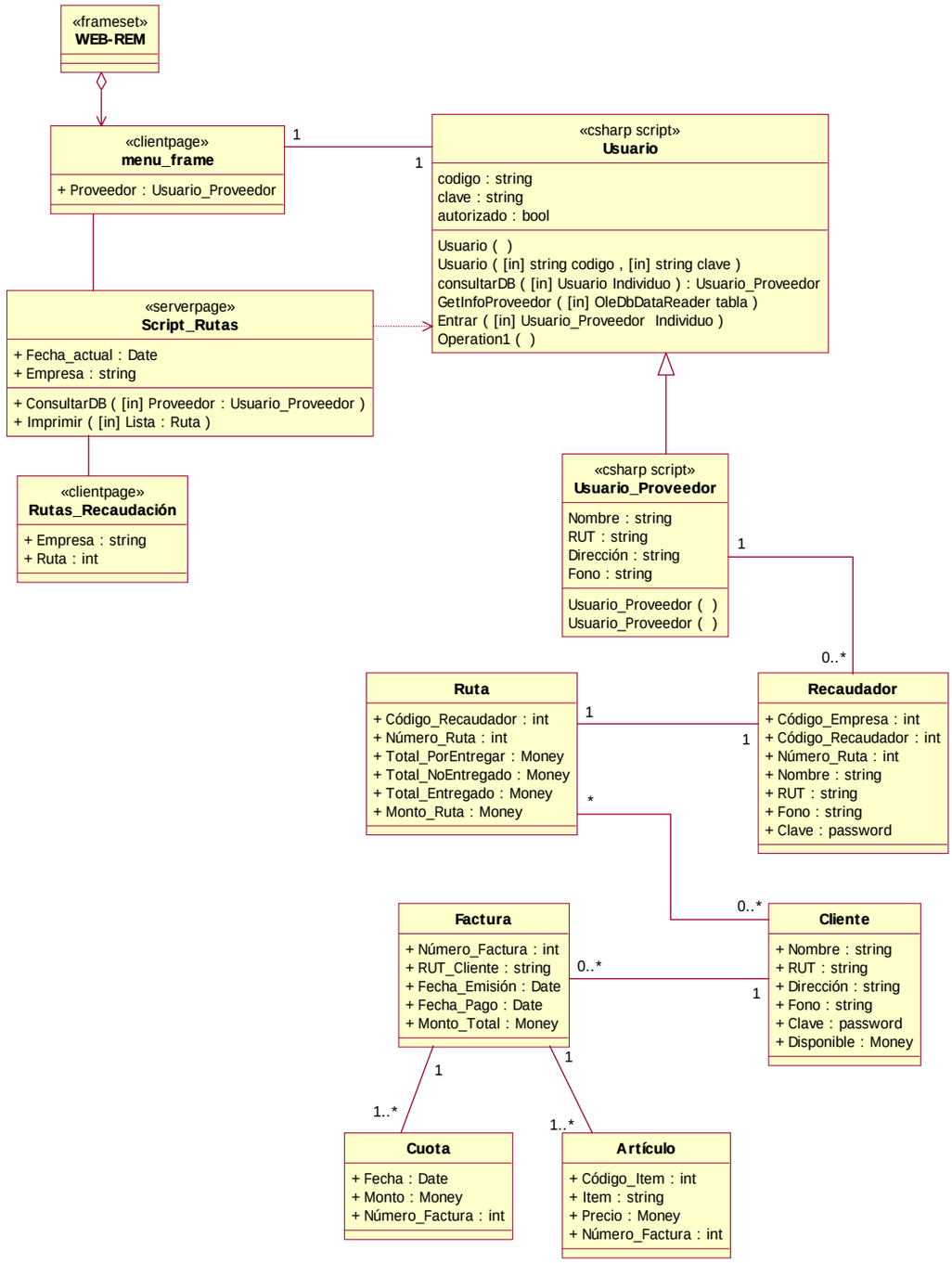


Figura 7.29: Extensión de la clase Usuario_Proveedor hacia un modelo tentativo de base de datos.

7.4.1.2.3 Workflow de implementación

Se comienzan a construir las GUIs y a codificar un primer prototipo funcional.

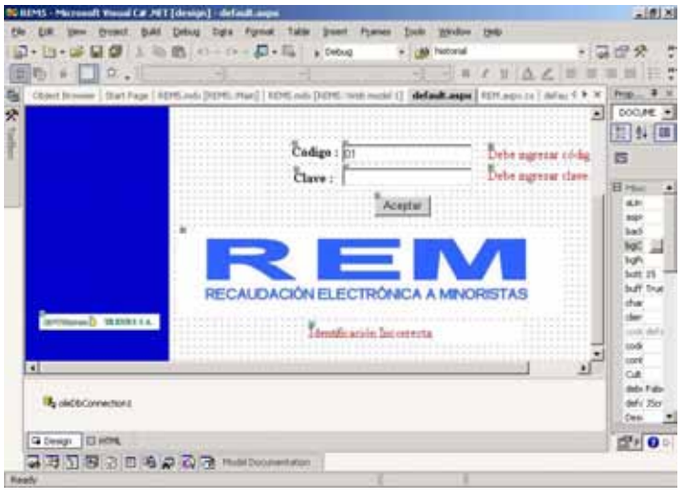


Figura 7.30: Construcción de la interfaz gráfica de usuario de la página de acceso a la aplicación WEB-REM versión 5 .

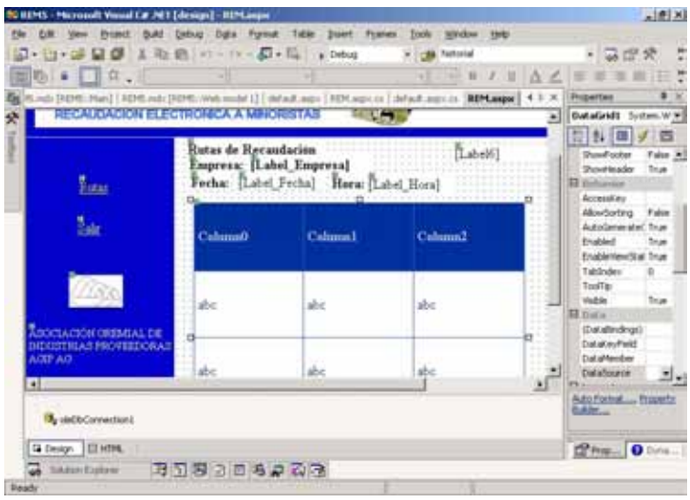


Figura 7.31: Construcción de la interfaz gráfica de la página principal de la aplicación WEB-REM versión 5 .

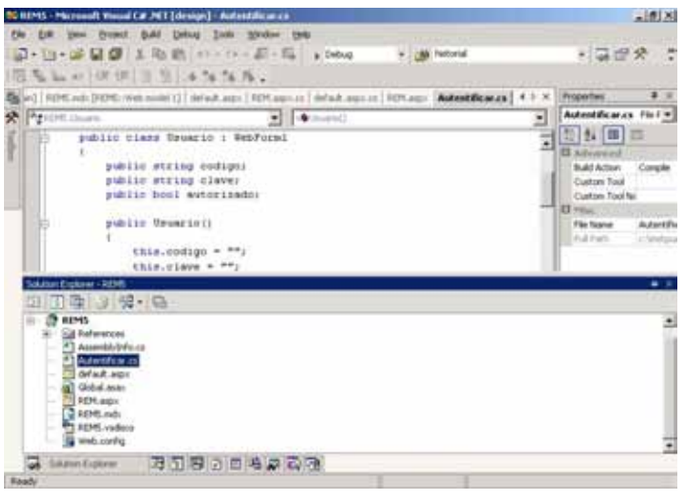


Figura 7.32: Implementación del prototipo funcional. Implementación de la clase Usuario.

Al sincronizar el modelo con el código fuente que se ha escrito se observa el siguiente modelo del sistema REM generado por **Rational XDE**, **ver figura 7.33**.

Una vez corregido el modelo, éste pasa de las manos del personal encargado de la implementación a las manos del personal encargado de los requerimientos y el diseño. Para el proyecto en curso, el nuevo modelo, ajustado al código pasa del programador al Ingeniero de Software.

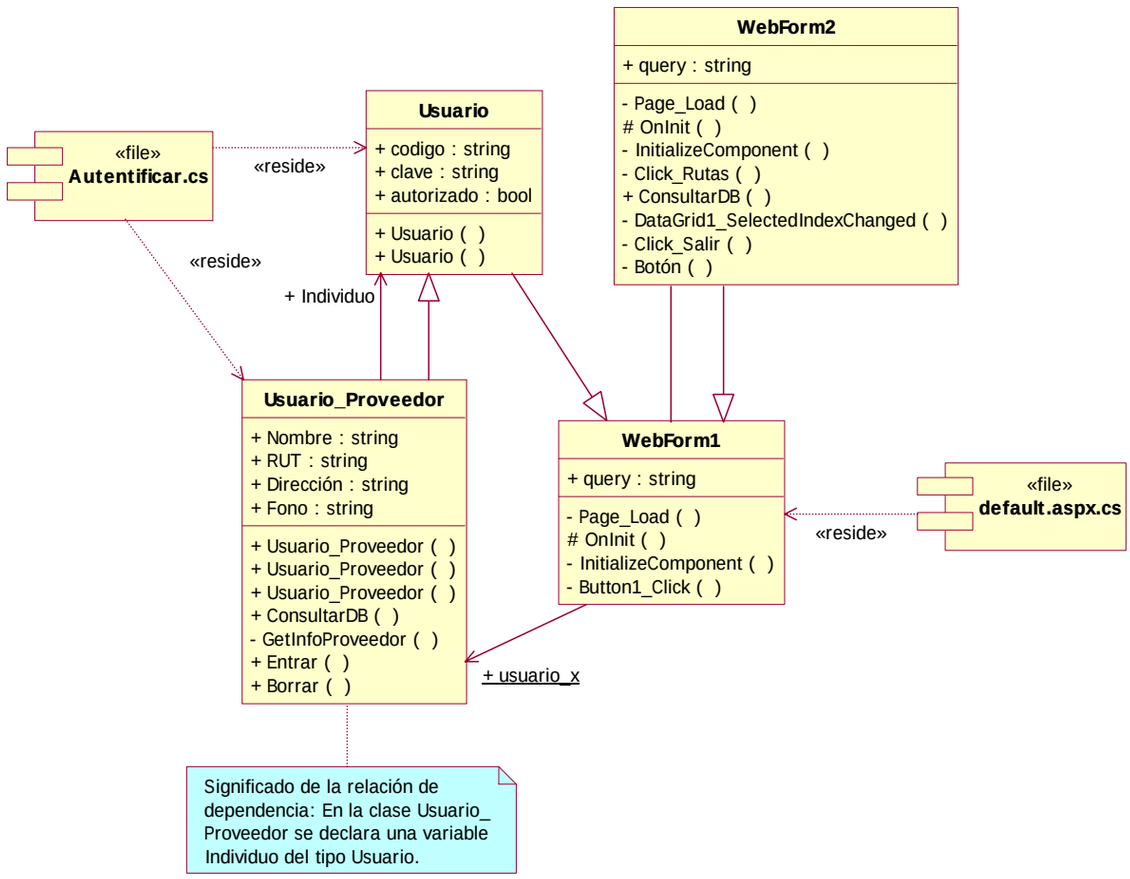


Figura 7.33: Muestra la misma vista de la figura 7.34, pero las clases se muestran sin su firma, o "signature", que describe el paso de parámetros.

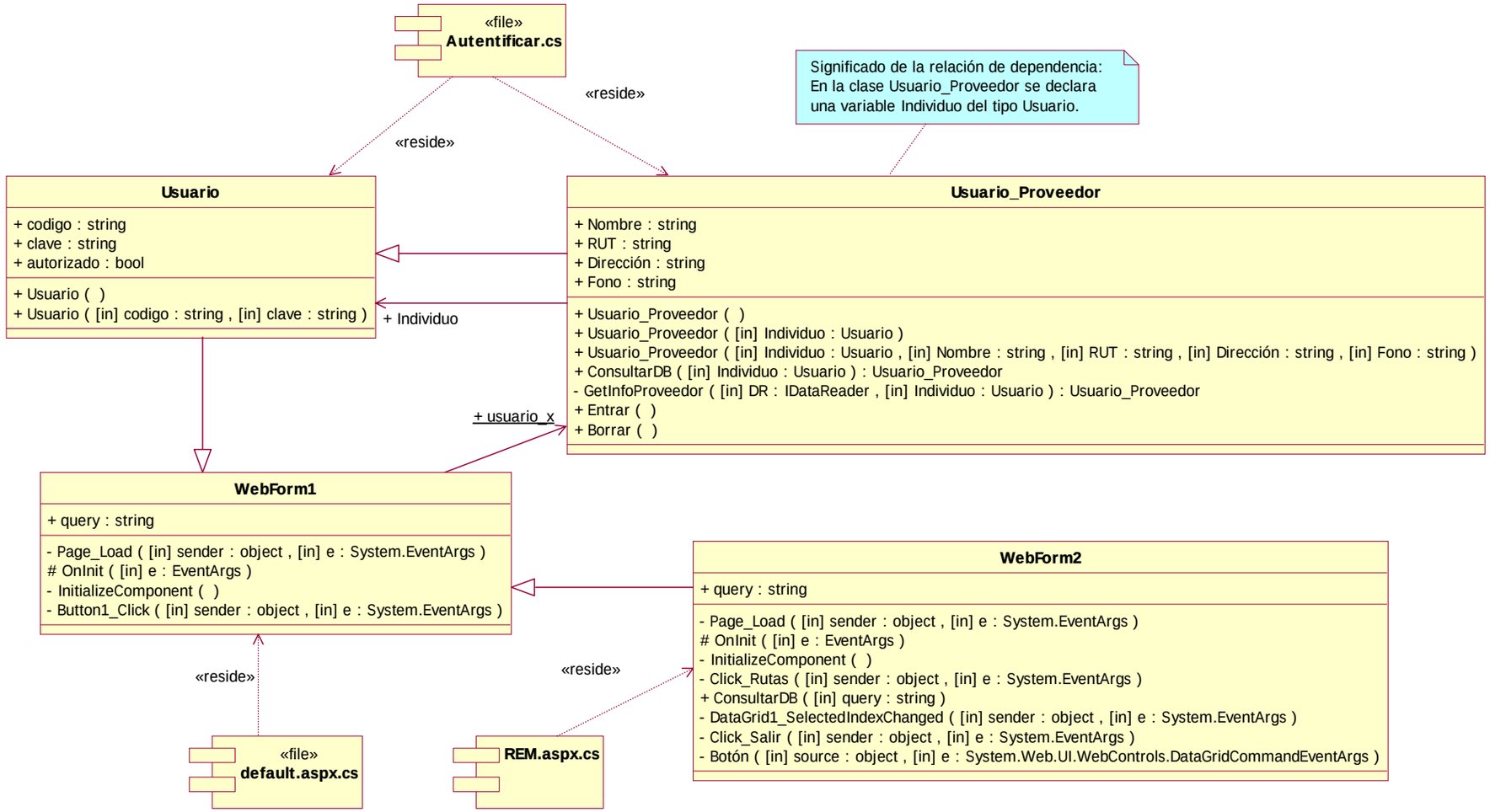


Figura 7.34: Modelo del prototipo WEB-REM construido mediante sincronización del modelo y el código a través de Rational XDE.

En WebForm1, se declara un objeto estático del tipo **Usuario_Proveedor**, mediante la siguiente línea de código presente en **default.aspx.cs**:

```
public static Usuario_Proveedor usuario_x;
```

Esto significa, que sólo existirá una instancia de dicho objeto durante la ejecución de la aplicación WEB, y esta instancia será accesible desde cualquier clase que pertenezca al espacio de nombres REM5. En otras palabras, será tratada como una variable global, por cualquier clase perteneciente al Namespace REM5. En el diagrama, esto se expresa mediante: +usuario_x , en la asociación dirigida desde WebForm1 a **Usuario_Proveedor**.

Los esfuerzos de prototipado continuan, así como la sincronización del código y el modelo, en la evolución del prototipo.

7.4.1.2.4 Workflow de administración de proyectos

El workflow de administración de proyectos en la fase de elaboración efectúa un control de las actividades planificadas en la fase de inicio y que se encuentran descritas en la carta gantt del proyecto, ver **Anexo 13**. Sin embargo, existe un ajuste que debe ser realizado por el administrador del proyecto entre una iteración y otra. Se comienza contrastando la planificación del proyecto con la realidad, para determinar el error de ésta. Lo importante es recabar suficiente información para poder hacer las correcciones pertinentes a la carta Gantt y reconstruirla en la fase de inicio para la planificación de la siguiente iteración. Según se tiene un mayor conocimiento del sistema, basado en el diseño refinado de éste, se deben refinar las estimaciones de costo del proyecto software.

7.4.1.3 Fase de construcción

7.4.1.3.1 Workflow de análisis y diseño

Se modela el sitio WAP-REM. Este es un modelo tentativo que pretende separar el diseño artístico de la parte funcional del sistema.

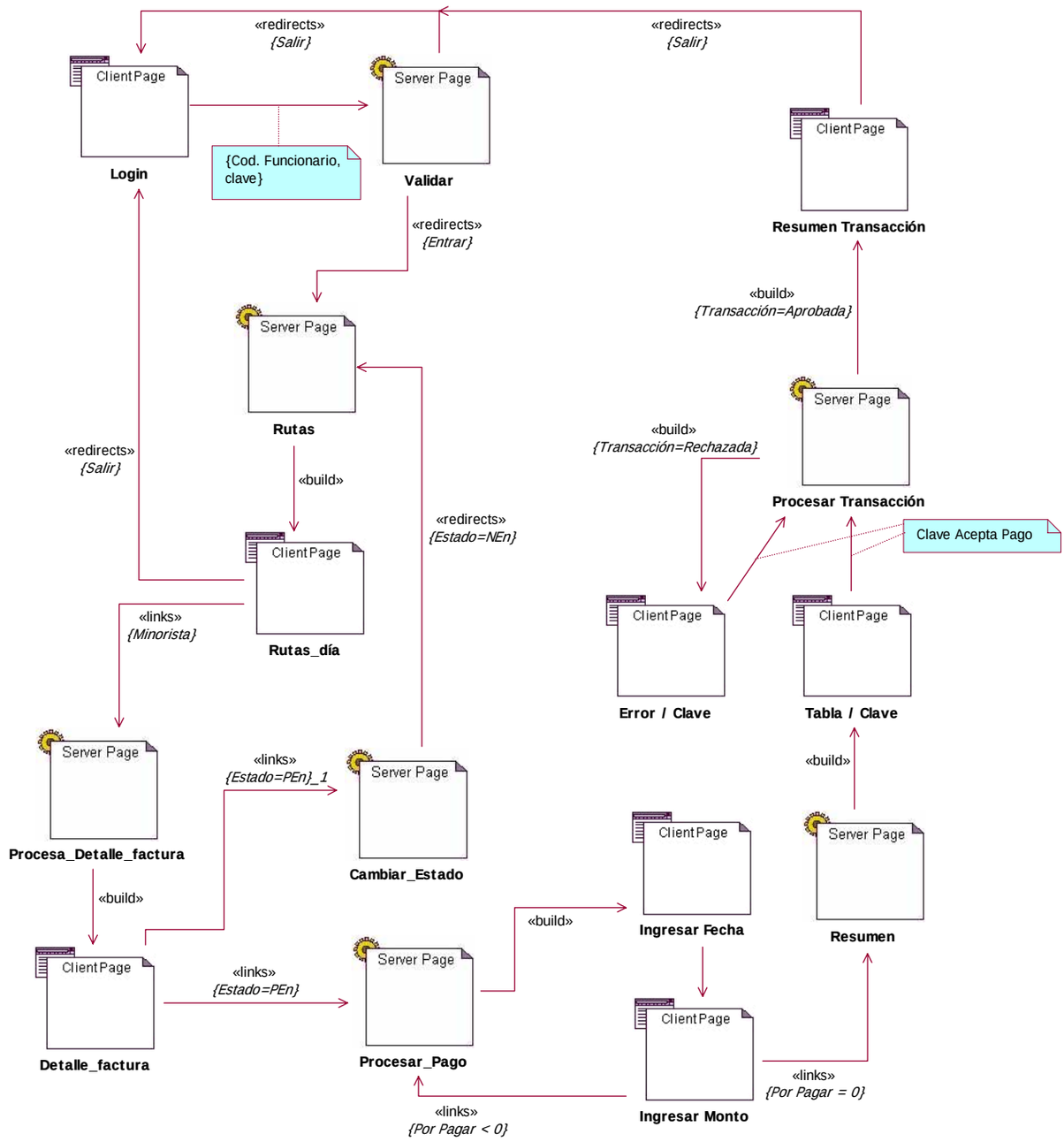


Figura 7.35: Modelamiento Web tentativo del prototipo WAP-REM.

Por otro lado, la sincronización de los esfuerzos de prototipado muestran los cambios en el modelo gatillados por el desarrollo, ver figura 7.36.

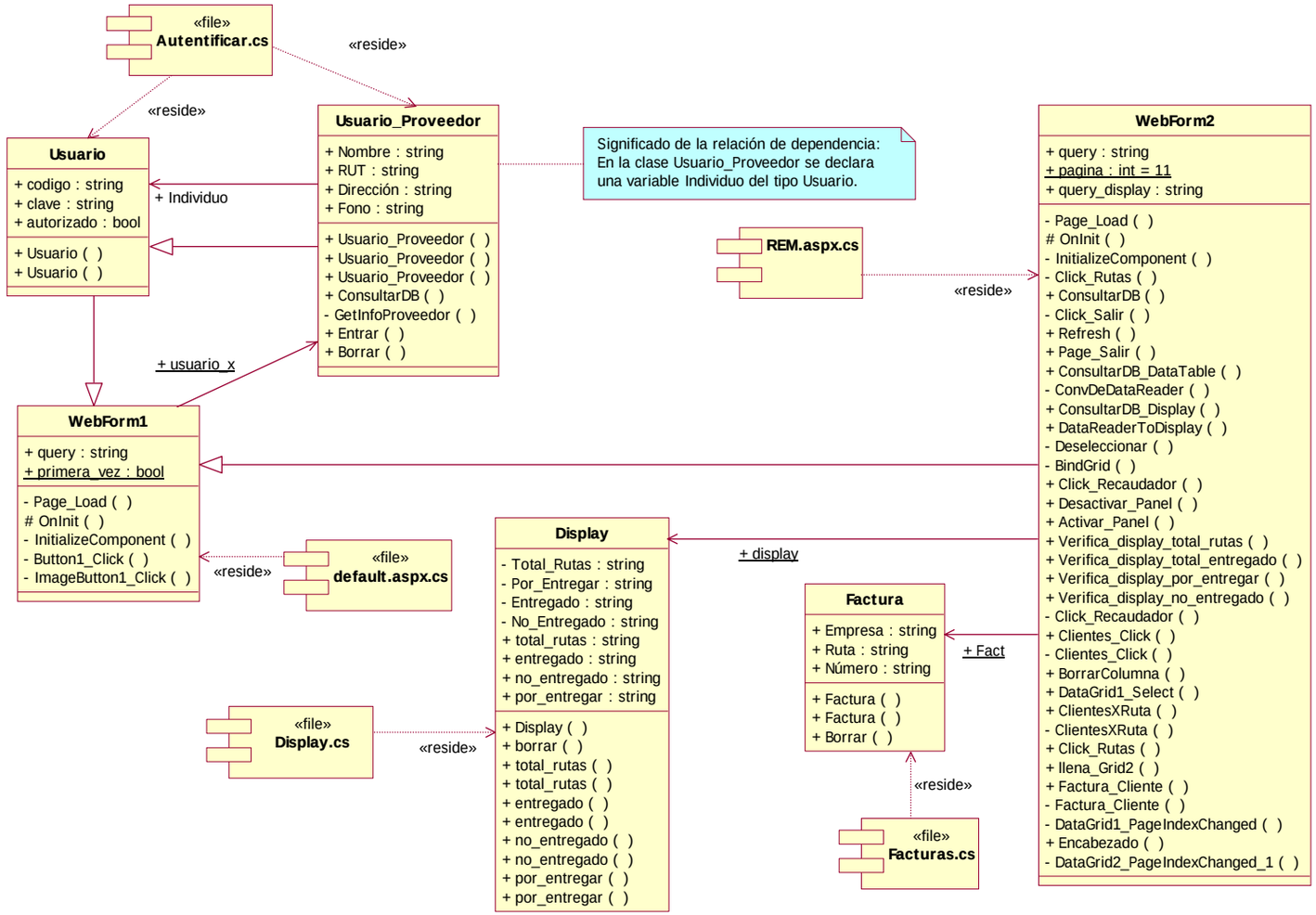
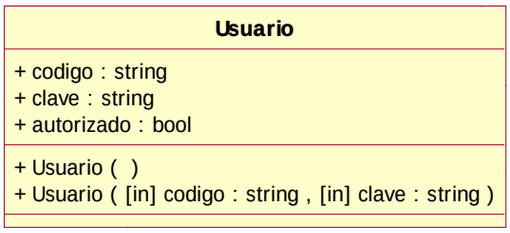


Figura 7.36: Cambios al modelo del sistema WEB-REM gatillados por el prototipado evolutivo.

7.4.1.3.1.1 Diccionario de datos

El presente diccionario de datos muestra las clases del modelo utilizadas por el prototipo WEB-REM versión 5 siguiendo la metodología EFT-SDM, junto con un detalle de los métodos de cada una de ellas. *Los rótulos de figuras y tablas de la presente sección 7.4.1.3.1.1, han sido omitidos en forma intencional para mejorar la legibilidad del documento.*



Nombre:	**Usuario**
Descripción:	**Constructor de la clase Usuario. Inicializa los valores de los miembros de la clase.**
Entrada:	

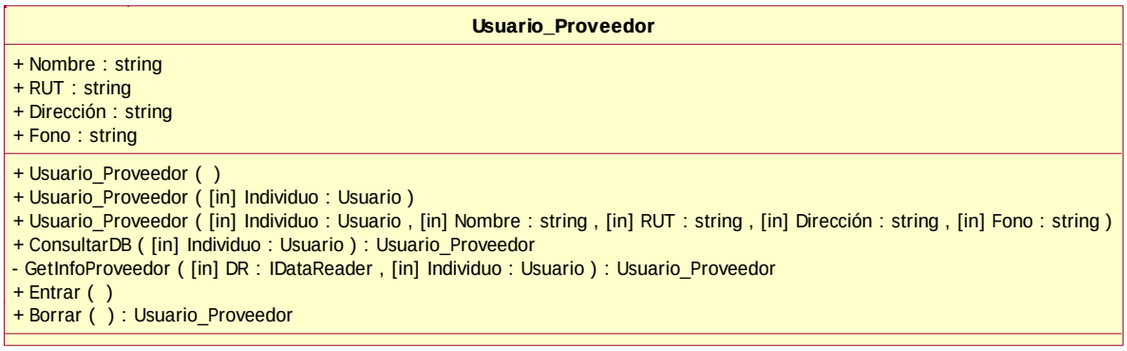
Salida:	`this.codigo = "";`
	`this.clave = "";`
	`this.autorizado = false;`

Dependencias Funcionales:	

Nombre:	**Usuario**
Descripción:	**Sobrecarga del constructor Usuario, permite modificar los valores de los miembros codigo y atributo**
Entrada:	`(string codigo, string clave)`

Salida:	Los valores de los miembros del objeto son modificados a los valores de los parámetros de entrada.
	`this.codigo = codigo;`
	`this.clave = clave;`

Dependencias Funcionales:	



Nombre:	**Usuario_Proveedor**
Descripción:	**Constructor de la clase Usuario_Proveedor**
Entrada:	

Salida:	`Usuario X = new Usuario();`
	`this.Individuo=X;`
	`this.Nombre= "";`
	`this.RUT= "";`
	`this.Dirección = "";`
	`this.Fono = "";`

Dependencias Funcionales:	

Nombre:	**Usuario_Proveedor**
Descripción:	**Permite modificar el dato miembro Individuo de la clase Usuario.**
Entrada:	(Individuo: Usuario)
Salida:	`this.Individuo = Individuo;`
Dependencias Funcionales:	

Nombre:	**Usuario_Proveedor**
Descripción:	**Permite modificar los valores de los datos miembros de un objeto Usuario_Proveedor con los valores de los parámetros de entrada**
Entrada:	`(Usuario Individuo, string Nombre, string RUT, string Dirección, string Fono)`

Salida:	`this.Individuo=Individuo;`
	`this.Nombre= Nombre;`
	`this.RUT= RUT;`
	`this.Dirección = Dirección;`
	`this.Fono = Fono;`

Dependencias Funcionales:	

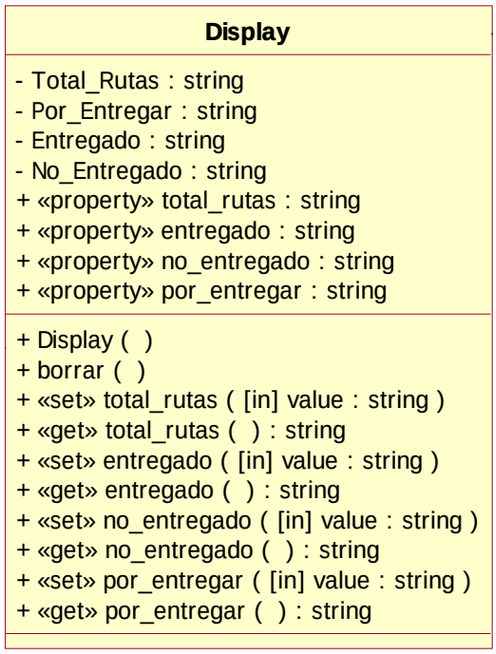
Nombre:	ConsultarDB
Descripción:	Realiza consulta a la base de datos a través del procedimiento almacenado sp_VerEmpresas, para validar el código y contraseña del usuario en la página de acceso (Login).
Entrada:	(Usuario Individuo)
Salida:	Usuario_Proveedor
Dependencias Funcionales:	oleDbConnection1, sp_VerEmpresas, GetInfoProveedor, Button1_Click.

Nombre:	GetInfoProveedor
Descripción:	Inicializa los valores de un objeto Usuario_Proveedor a partir de la información provista en los parámetros de entrada.
Entrada:	(IDataReader DR, Usuario Individuo)
Salida:	Usuario_Proveedor
Dependencias Funcionales:	
Nombre:	Entrar
Descripción:	Redirecciona a la página principal del sistema WEB-REM.
Entrada:	
Salida:	WebForm2.aspx
Dependencias Funcionales:	

Nombre:	Borrar

Descripción:	Borra los valores de un objeto Usuario_Proveedor:
	this.Individuo.codigo = "";
	this.Individuo.clave = "";
	this.Individuo.autorizado=false;
	this.autorizado=false;
	this.Nombre="";
	this.RUT="";
	this.Dirección="";
	this.Fono="";

Entrada:	
Salida:	Usuario_Proveedor
Dependencias Funcionales:	



Nombre:	**Display**
Descripción:	**Constructor de la clase Display. Inicializa los valores por defecto de un objeto Display.**
Entrada:	

Salida:	`Total_Rutas= "";`
	`Por_Entregar = "";`
	`Entregado = "";`
	`No_Entregado = "";`

Dependencias Funcionales:	

Nombre:	**Borrar**
Descripción:	**Borra los valores de los datos miembros de un objeto Display**
Entrada:	

Salida:	`Total_Rutas= "";`
	`Por_Entregar = "";`
	`Entregado = "";`
	`No_Entregado = "";`

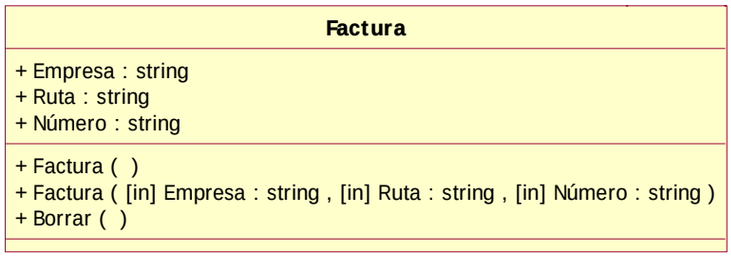
Dependencias Funcionales:	

Nombre:	**total_rutas**
Descripción:	Propiedad de un objeto Display que permite establecer y rescatar el valor del dato miembro privado **Total_Rutas**. Este mantiene la suma de dinero del total de todas las rutas desplegadas.
Entrada:	value: string
Salida:	**Total_Rutas:** string
Dependencias Funcionales:	

Nombre:	**Entregado**
Descripción:	Propiedad de un objeto Display que permite establecer y rescatar el valor del dato miembro privado **`Entregado`**. Este mantiene la suma de dinero de los despachos entregados.
Entrada:	value: string
Salida:	**`Entregado`**`: string`
Dependencias Funcionales:	

Nombre:	**no_entregado**
Descripción:	Propiedad de un objeto Display que permite establecer y rescatar el valor del dato miembro privado **`No_Entregado`**. Este mantiene la suma de dinero de los despachos no entregados.
Entrada:	**value:** string
Salida:	**`No_Entregado`**`: string`
Dependencias Funcionales:	

Nombre:	**por_entregar**
Descripción:	Propiedad de un objeto Display que permite establecer y rescatar el valor del dato miembro privado **`Por_Entregar`**. Este mantiene la suma de dinero de los despachos por entregar.
Entrada:	value: string
Salida:	**`Por_Entregar`**`: string`
Dependencias Funcionales:	



Nombre:	**Factura**
Descripción:	Constructor de la clase Factura. Inicializa los valores por defecto de un objeto Factura.
Entrada:	

Salida:	`this.Empresa="";`
	`this.Ruta="";`
	`this.Número="";`

Dependencias Funcionales:	

Nombre:	**Factura**
Descripción:	Sobrecarga del constructor de la clase Factura. Permite modificar los valores de los datos miembros de un objeto Factura con los valores de los parámetros de entrada.
Entrada:	`(string Empresa, string Ruta, string Número)`

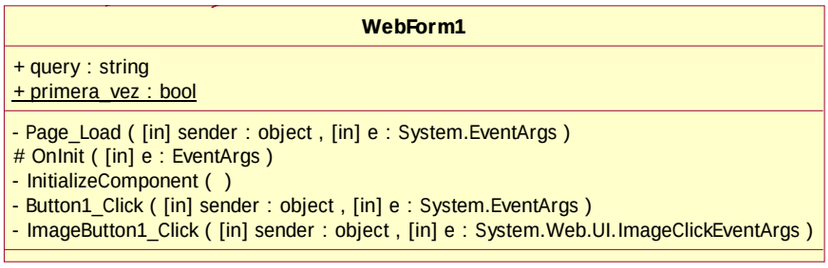
Salida:	`this.Empresa=Empresa;`
	`this.Ruta=Ruta;`
	`this.Número=Número;`

Dependencias Funcionales:	

Nombre:	**Borrar**
Descripción:	Permite borrar los valores de los datos miembros de un objeto Factura.
Entrada:	

Salida:	`this.Empresa="";`
	`this.Ruta="";`
	`this.Número="";`

Dependencias Funcionales:	



Nombre:	**Page_Load**
Descripción:	Método que determina las cosas que deben hacerse cada vez que se cargue la aplicación Web (WebForm1). Reservado para uso futuro. Declarado por Visual Studio .Net en forma automática.
Entrada:	`(object sender, System.EventArgs e)`
Salida:	`Void`
Dependencias Funcionales:	

Nombre:	**OnInit**
Descripción:	Método que determina la inicialización de componentes de la aplicación Web (WebForm1). Es modificado en forma automática por Visual Studio .NET, al usar el entorno de desarrollo gráfico. (Web Form Designer Generated Code)
Entrada:	
Salida:	
Dependencias Funcionales:	**InitializeComponent**

Nombre:	**InitializeComponent**
Descripción:	Inicializa los administradores de eventos de la aplicación WebForm1. Es modificado en forma automática por Visual Studio .NET, al usar el entorno de desarrollo gráfico. (Web Form Designer Generated Code)
Entrada:	
Salida:	
Dependencias Funcionales:	**oleDbConnection1, Button1, ImageButton1.**

Nombre:	**Button1_Click**
Descripción:	Acciones que deben ser realizadas cuando se captura el evento Button1 presionado. Verificar validadores de campos, llamar a ConsultarDB
Entrada:	`(object sender, System.EventArgs e)`
Salida:	**Se inicializa el objeto Usuario_Proveedor** **Salida 1:** - Si no ha sido ingresada la información en forma correcta, o esta no es válida, se debe volver a la página de inicio desplegando un mensaje del error. Se borra objeto Usuario_Proveedor. **Salida 2:** - Si la información es correcta y válida, se debe redireccionar a WebForm2.
Dependencias Funcionales:	**ConsultarDB(Usuario)**

Nombre:	**ImageButton1_Click**
Descripción:	Permite reinicializar los valores de la base de datos de la **DEMO** del prototipo REM5.
Entrada:	
Salida:	Valores de tablas reinicializados.
Dependencias Funcionales:	**oleDbConnection1, sp_Reiniciar**

WebForm2

+ query : string
+ pagina : int = 11
+ query_display : string

- Page_Load ([in] sender : object , [in] e : System.EventArgs)
OnInit ([in] e : EventArgs)
- InitializeComponent ( )
- Click_Rutas ([in] sender : object , [in] e : System.EventArgs)
+ ConsultarDB ([in] query : string)
- Click_Salir ([in] sender : object , [in] e : System.EventArgs)
+ Refresh ( )
+ Page_Salir ( )
+ ConsultarDB_DataTable ([in] query2 : string) : DataTable
- ConvDeDataReader ([in] DR : IDataReader) : DataTable
+ ConsultarDB_Display ([in] query2 : string)
+ DataReaderToDisplay ([in] DR : IDataReader)
- Deseleccionar ( )
- BindGrid ( )
+ Click_Recaudador ( )
+ Desactivar_Panel ( )
+ Activar_Panel ( )
+ Verifica_display_total_rutas ( )
+ Verifica_display_total_entregado ( )
+ Verifica_display_por_entregar ( )
+ Verifica_display_no_entregado ( )
- Click_Recaudador ([in] sender : object , [in] e : System.EventArgs)
+ Clientes_Click ( )
- Clientes_Click ([in] sender : object , [in] e : System.EventArgs)
+ BorrarColumna ([in] col : int)
+ DataGrid1_Select ([in] sender : Object , [in] e : EventArgs)
+ ClientesXRuta ( )
- ClientesXRuta ([in] sender : object , [in] e : System.EventArgs)
+ Click_Rutas ( )
+ llena_Grid2 ([in] query : string)
+ Factura_Cliente ( )
- Factura_Cliente ([in] sender : object , [in] e : System.EventArgs)
- DataGrid1_PageIndexChanged ([in] source : object , [in] e : System.Web.UI.WebControls.DataGridPageChangedEventArgs)
+ Encabezado ( )
- DataGrid2_PageIndexChanged_1 ([in] source : object , [in] e : System.Web.UI.WebControls.DataGridPageChangedEventArgs)

Nombre:	**Page_Load**

Descripción:	**Método que determina las cosas que deben hacerse cada vez que se cargue la aplicación Web (WebForm2).**
	Declarado por Visual Studio .Net en forma automática.
	
	Ordena no tener Buffer y no guardar páginas en cache de browser:
	
	`Page.Response.Buffer = false;`
	`Page.Response.CacheControl= "no-cache";`
	
	Realiza un Refresh de la página.

Entrada:	
Salida:	
Dependencias Funcionales:	**Refresh()**

Nombre:	**OnInit**
Descripción:	**Método que determina la inicialización de componentes de la aplicación Web (WebForm2). Es modificado en forma automática por Visual Studio .NET, al usar el entorno de desarrollo gráfico.** **(Web Form Designer Generated Code)**
Entrada:	
Salida:	
Dependencias Funcionales:	**InitializeComponent**

Nombre:	**InitializeComponent**
Descripción:	Inicializa los administradores de eventos de la aplicación WebForm1. Es modificado en forma automática por Visual Studio .NET, al usar el entorno de desarrollo gráfico. (Web Form Designer Generated Code)
Entrada:	
Salida:	
Dependencias Funcionales:	**oleDbConnection1, LinkButton1, LinkButton2, LinkButton3, LinkButton4, DataGrid1, DataGrid2.**

Nombre:	**Click_Rutas**
Descripción:	Acciones realizadas al presionar el enlace Rutas, asociado a LinkButton1. 1. `Activar_Panel` 2. `DataGrid1 y DataGrid1, CurrentPageIndex = 0` 3. `Click_Rutas()`
Entrada:	`(object sender, System.EventArgs e)`
Salida:	void
Dependencias Funcionales:	**Click_Rutas(), Activar_Panel()**

Nombre:	**ConsultarDB**
Descripción:	Realiza una consulta a la base de datos, especificada por el parámetro de entrada query. El resultado, es transferido al DataGrid1.
Entrada:	`(string query)`
Salida:	`void, llena DataGrid1`
Dependencias Funcionales:	**oleDbConnection1, DataGrid1**

Nombre:	Click_Salir
Descripción:	Actividades realizadas cuando de captura el evento que indica que se ha presionado el enlace LinkButton2. Llama a método **Page_Salir()**
Entrada:	`(object sender, System.EventArgs e)`
Salida:	void
Dependencias Funcionales:	**Page_Salir()**

Nombre:	Refresh
Descripción:	Verifica si el objeto Usuario_Proveedor está autorizado para estar en el formulario web WebForm2, establece las características de DataGrid1 y DataGrid2, despliega la lista de rutas de la empresa mediante el método **Click_Rutas()**. Si el objeto Usuario_Proveedor no está autorizado, ejecuta el metodo Page_Salir(). Además siempre ejecuta el método Encabezado().
Entrada:	
Salida:	void
Dependencias Funcionales:	**Click_Rutas(), Page_Salir(), Encabezado(), DataGrid1 y DataGrid2.**

Nombre:	Page_Salir
Descripción:	Debe borrar objeto Usuario_Proveedor, hacer expirar la página inmediatamente: Page.Response.ExpiresAbsolute=System.DateTime.Now; y redireccionar a default.aspx.
Entrada:	
Salida:	
Dependencias Funcionales:	

Nombre:	ConsultarDB_DataTable
Descripción:	Método que realiza consulta especificada en parámetro de entrada query2.
Entrada:	`(string query2)`
Salida:	Retorna un DataTable con el resultado de la consulta.
Dependencias Funcionales:	**oleDbConnection1, ConvDeDataReader**

Nombre:	ConvDeDataReader
Descripción:	Recorre un objeto IDataReader pasado como parámetro y retorna un objeto del tipo DataTable que contiene exactamente la misma información.
Entrada:	`(IDataReader DR)`
Salida:	`DataTable`
Dependencias Funcionales:	

Nombre:	**ConsultarDB_Display**
Descripción:	Inicializa los valores de los datos miembros de un objeto Display a través de una llamada al método `DataReaderToDisplay`. El objeto del tipo Display debe ser declarado estático y su nombre debe ser display.
Entrada:	`(string query2)`
Salida:	`void`
Dependencias Funcionales:	**DataReaderToDisplay**

Nombre:	**DataReaderToDisplay**
Descripción:	Obtiene valores para un objeto **Display** llamado **display**, a cuyos datos miembros se le asignan valores obtenidos desde el parámetro de entrada.
Entrada:	`(IDataReader DR)`
Salida:	`void`
Dependencias Funcionales:	**Debe declararse un objeto estático del tipo Display llamado display: static Display display.**

Nombre:	**Deseleccionar**
Descripción:	Permite deseleccionar la fila que ha escogido el usuario en el DataGrid1. `DataGrid1.SelectedIndex = -1;`
Entrada:	
Salida:	void
Dependencias Funcionales:	
Nombre:	**BindGrid**
Descripción:	Actualiza tamaños de página de DataGrid1 dependiendo del estado de la aplicación.Realiza DataBinding de DataGrid1 y dataGrid2; Dependiendo del estado de la aplicación se ejecuta método `Activar_Panel()`.
Entrada:	
Salida:	`void`
Dependencias Funcionales:	**Activar_Panel(), DataGrid1, DataGrid2**

Nombre:	Click_Recaudador
Descripción:	Actividades que deben ser realizadas por la aplicación cuando se presione el LinkButton3 correspondiente al recaudador. Debe consultarse la base de datos para determinar la información referente a los recaudadores. Se utiliza el procedimiento almacenado **`sp_Recaudadores`**.
Entrada:	
Salida:	`void`
Dependencias Funcionales:	**Deseleccionar(),sp_Recaudadores, ConsultarDB_DataTable(query), BorrarColumna(0), BindGrid(),Encabezado().**

Nombre:	Desactivar_Panel

Descripción:	Dependiendo del estado de la aplicación, se desactivan los siguientes paneles cuando el estado sea Página 21 y Página 31:
	`Panel1.Visible=false;`
	`Panel4.Visible=false;`
	`Panel5.Visible=false;`
	`Panel6.Visible=false;`
	`Panel7.Visible=false;`
	`Panel_Detalle.Visible=false;`

Entrada:	
Salida:	`void`
Dependencias Funcionales:	

Nombre:	Activar_Panel

Descripción:	Según corresponda, activa los siguientes paneles:
	`Panel7.Visible=true;`
	`Panel1.Visible=true;`
	`Panel4.Visible=true;`
	`Panel5.Visible=true;`
	`Panel6.Visible=true;`
	`Panel_Detalle.Visible = true;`
	Además ejecuta los métodos mostrados en dependencias funcionales.

Entrada:	
Salida:	`void`
Dependencias Funcionales:	**Verifica_display_total_rutas(), Verifica_display_total_entregado(), Verifica_display_por_entregar(), Verifica_display_no_entregado().**

Nombre:	Verifica_display_total_rutas
Descripción:	Si el atributo del objeto Display, **`display.total_rutas`**, no está vacío, debe desplegarlo, de lo contrario, lo debe ocultar.
Entrada:	
Salida:	`void`
Dependencias Funcionales:	**display.total_rutas** **Panel1, Label9, Label8.**

Nombre:	Verifica_display_total_entregado
Descripción:	Si el atributo del objeto Display, **`display.entregado`**, no está vacío, debe desplegarlo, de lo contrario, lo debe ocultar.
Entrada:	
Salida:	`void`
Dependencias Funcionales:	**display.entregado, Panel4, Label12, Label10.**

Nombre:	Verifica_display_por_entregar
Descripción:	Si el atributo del objeto Display, **`display.por_entregar`**, no está vacío, debe desplegarlo, de lo contrario, lo debe ocultar.
Entrada:	
Salida:	`void`
Dependencias Funcionales:	**display.por_entregar, Panel5, Label13, Label14.**

Nombre:	Verifica_display_no_entregado
Descripción:	Si el atributo del objeto Display, **`display.no_entregado`**, no está vacío, debe desplegarlo, de lo contrario, lo debe ocultar.
Entrada:	
Salida:	`void`
Dependencias Funcionales:	**display.no_entregado, Panel6, Label15, Label16.**

Nombre:	Click_Recaudador
Descripción:	Ejecuta el método borrar de la instancia display: **`display.borrar();`** Invoca a los métodos mostrados en el item dependencias funcionales.
Entrada:	`(object sender, System.EventArgs e)`
Salida:	`void`
Dependencias Funcionales:	**Click_Recaudador(),Desactivar_Panel().**

Nombre:	**Clientes_Click**
Descripción:	Actividades que debe realizar la aplicación cuando se presiona el enlace LinkButton4. Se efectúa una consulta a la base de datos para obtener la información de los clientes a través del procedimiento almacenado **sp_Clientes**.
Entrada:	
Salida:	`void`
Dependencias Funcionales:	**Deseleccionar(), sp_Clientes, ConsultarDB_DataTable(query), BorrarColumna(0), BindGrid(), Encabezado()**
Nombre:	**Clientes_Click**
Descripción:	Borra el objeto display, e invoca los métodos mostrados en el item dependencias funcionales.
Entrada:	`(object sender, System.EventArgs e)`
Salida:	`void`
Dependencias Funcionales:	**Clientes_Click(), Desactivar_Panel();**

Nombre:	**BorrarColumna**
Descripción:	Borra la primera columna del DataGrid1: `DataGrid1.Columns.RemoveAt(col);`
Entrada:	`(int col)`
Salida:	`void`
Dependencias Funcionales:	**DataGrid1**

Nombre:	**DataGrid1_Select**
Descripción:	El objetivo es determinar si se ha pulsado algún enlace en el interior del componente DataGrid1, y dependiendo del estado de la aplicación, si es página 11 o página 12, se ejecutan los métodos **`ClientesXRuta(sender,e)`** o **`Factura_Cliente(sender,e`**). Además, cuando la secuencia de ejecución es devuelta, se debe establecer el nuevo valor de la variable pagina.
Entrada:	`(Object sender, EventArgs e)`
Salida:	`void`
Dependencias Funcionales:	**pagina, ClientesXRuta(sender,e), Fact.Ruta, Fact.Número, Factura_Cliente(sender,e), Encabezado(), Deseleccionar()**

Nombre:	**ClientesXRuta**
Descripción:	Permite desplegar la Información de Clientes por Ruta.
Entrada:	
Salida:	`void`
Dependencias Funcionales:	**sp_clienteXruta, ConsultarDB_DataTable(query), sp_CuotasRutas, llena_Grid2(query), ConsultarDB_Display, BindGrid().**

Nombre:	ClientesXRuta
Descripción:	Invoca al método: `ClientesXRuta();`
Entrada:	`(object sender, System.EventArgs e)`
Salida:	`void`
Dependencias Funcionales:	ClientesXRuta()

Nombre:	Click_Rutas
Descripción:	`Permite desplegar la Información de las Rutas.`
Entrada:	
Salida:	`void`
Dependencias Funcionales:	Deseleccionar(),sp_rutasDeRecaudación, ConsultarDB_DataTable(query), sp_CuotasEmpresa, llena_Grid2(query), sp_GetMontoTotalRutasXEmpresa, ConsultarDB_Display(query_display), BindGrid(), Encabezado().

Nombre:	llena_Grid2
Descripción:	Permite llenar el DataGrid2, con la información consultada a través del parámetro de entrada query.
Entrada:	`(string query)`
Salida:	`void`
Dependencias Funcionales:	ConsultarDB_DataTable(query)

Nombre:	Factura_Cliente
Descripción:	Permite desplegar la `Información de una determinada Factura.`
Entrada:	
Salida:	`void`
Dependencias Funcionales:	sp_FacturaCliente, ConsultarDB_DataTable(query), sp_CuotasFactura, llena_Grid2(query), sp_GetTotalFactura, ConsultarDB_Display(query_display), BorrarColumna(0), BindGrid().

Nombre:	Factura_Cliente
Descripción:	Invoca al método: `Factura_Cliente();`
Entrada:	`(object sender, System.EventArgs e)`
Salida:	`void`
Dependencias Funcionales:	Factura_Cliente()

Nombre:	DataGrid1_PageIndexChanged
Descripción:	Al detectarse el cambio de índice de página del DataGrid1, se debe llamar a uno de los tres métodos descritos en el item dependencias funcionales, siendo el estado de la página; 11, 21 y 31 respectivamente.
Entrada:	(object source, System.Web.UI.WebControls.DataGridPageChangedEvent Args e)
Salida:	Void
Dependencias Funcionales:	pagina, Click_Rutas(),Click_Recaudador(), Clientes_Click().

Nombre:	Encabezado
Descripción:	Despliega en el panel superior el nombre de la empresa, la fecha y la hora.
Entrada:	
Salida:	Void
Dependencias Funcionales:	Usuario_Proveedor.usuario_x.Nombre, System.DateTime.Today.ToShortDateString(), System.DateTime.Now.ToShortTimeString().

Nombre:	DataGrid2_PageIndexChanged_1
Descripción:	Si se ha cambiado de página en el DataGrid2, por lo que se debe ejecutar alguno de los métodos expuestos en el item de dependencias funcionales, según la página sea: 11, 12, 13, 21 o 31 respectivamente.
Entrada:	(object source, System.Web.UI.WebControls.DataGridPageChangedEvent Args e)
Salida:	Void
Dependencias Funcionales:	pagina, Click_Rutas(), ClientesXRuta(), Factura_Cliente(), Click_Recaudador(), Clientes_Click()

7.4.1.3.1.2 Estados de la aplicación

Los estados de la aplicación se describen en la siguiente tabla:

Tabla 7.24: Estados de la aplicación WEB-REM versión 5 siguiendo EFT-SDM.

Descripción	Estado	Componente Asociado
Información Rutas	Página 11	LinkButton1
Información de clientes por ruta	Página 12	DataGrid1
Información Factura	Página 13	DataGrid1
Información Recaudadores	Página 21	LinkButton3
Información Clientes	Página 31	LinkButton4
Salir	No definido	LinkButton2

Estos se ven representados en el diagrama de estados de la **figura 7.37,** la cual representa un diagrama de estados de UML. Cada condition guard ([ ]) representa un enlace que gatilla la transición.

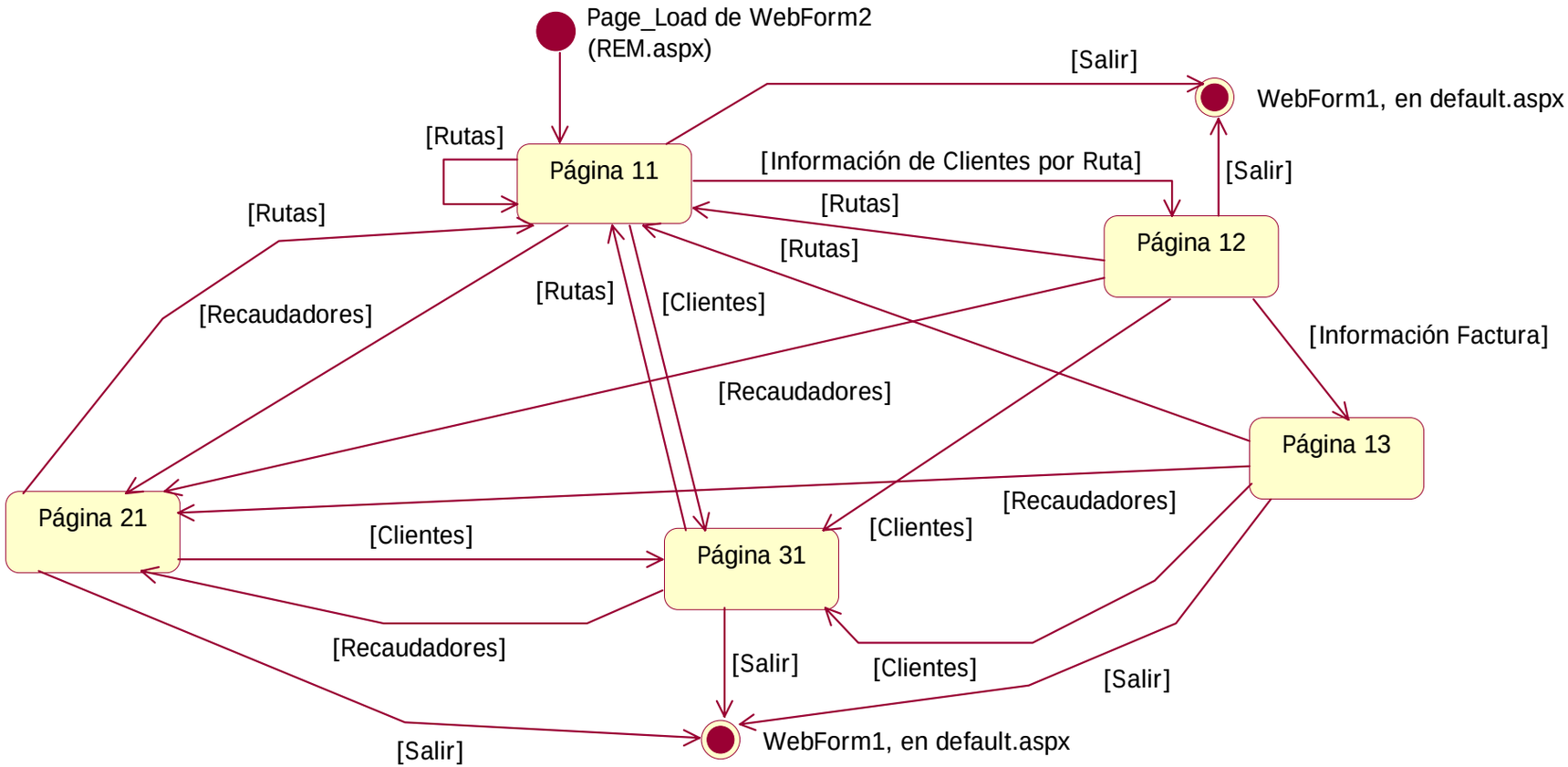


Figura 7.37: Diagrama de estados de la aplicación WEB-REM versión 5.

7.4.1.3.1.3 Diseño Base de datos

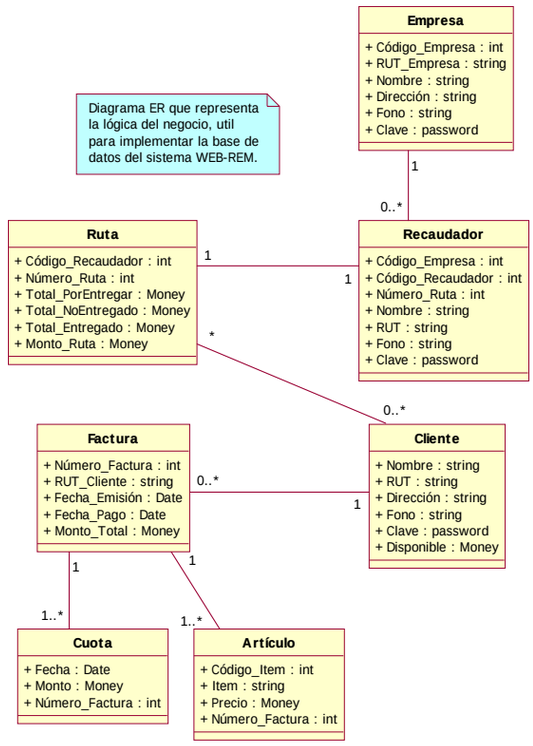


Figura 7.38: Modelo tentativo de la base de datos.

Tras algunas modificaciones al modelo de base de datos se obtuvo el siguiente diseño.

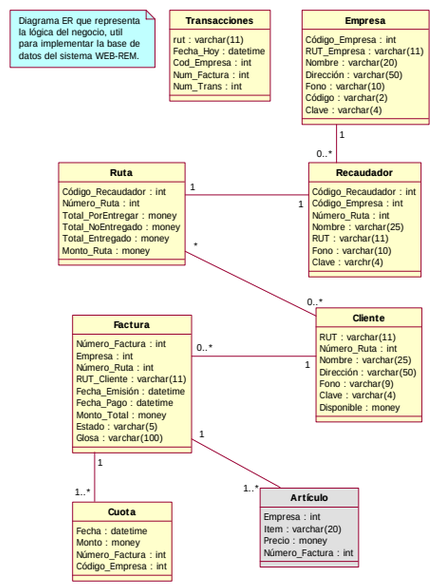


Figura 7.39: Modelo de la base de datos del sistema WEB-REM versión 5.

7.4.1.3.1.4 Tablas del modelo de base de datos

Las tablas utilizadas por el modelo de base de datos se describen a continuación. *Los rótulos de tablas de la presente sección 7.4.1.3.1.4, han sido omitidos en forma intencional para mejorar la legibilidad del documento.*

Artículo		
Campo	**Tipo**	**Largo**
Empresa	int	4
Item	varchar	20
Precio	money	8
Número_Factura	int	4

Cliente		
Campo	**Tipo**	**Largo**
RUT	varchar	11
Número_Ruta	int	4
Nombre	varchar	25
Dirección	varchar	50
Fono	varchar	9
Clave	varchar	4
Disponible	money	8

Cuota		
Campo	**Tipo**	**Largo**
Fecha	datetime	8
Monto	money	8
Número_Factura	int	4
Código_Empresa	int	4

Empresa		
Campo	**Tipo**	**Largo**
Código_Empresa	int	4
RUT_Empresa	varchar	11
Nombre	varchar	20
Dirección	varchar	50
Fono	varchar	10
Código	varchar	2
Clave	varchar	4

Factura		
Campo	**Tipo**	**Largo**
Número_Factura	int	4
Empresa	int	4
Número_Ruta	int	4
RUT_Cliente	varchar	11
Fecha_Emisión	datetime	8
Fecha_Pago	datetime	8
Monto_Total	money	8
Estado	varchar	3
Glosa	varchar	100

Recaudador		
Campo	**Tipo**	**Largo**
Código_Recaudador	int	4
Código_Empresa	int	4
Número_Ruta	int	4
Nombre	varchar	25
RUT	varchar	11
Fono	varchar	10
Clave	varchar	4

Ruta		
Campo	**Tipo**	**Largo**
Código_Recaudador	int	4
Número_Ruta	int	4
Total_PorEntregar	money	8
Total_NoEntregado	money	8
Total_Entregado	money	8
Monto_Ruta	money	8

Transacciones		
Campo	**Tipo**	**Largo**
rut	varchar	11
Fecha_Hoy	datetime	8
Cod_Empresa	int	4
Num_Factura	int	4
Num_Trans	int	4

7.4.1.3.1.5 Procedimientos almacenados

Los rótulos de tablas de la presente sección 7.4.1.3.1.5, han sido omitidos en forma intencional para mejorar la legibilidad del documento.

Nombre:	**sp_Clientes**
Descripción:	Dado el código de empresa @Código, se obtiene una lista de clientes para dicha empresa.
Entrada:	`(@Código VARCHAR(2))`
Salida:	`Cliente.Nombre 'Nombre', Cliente.RUT 'RUT',Cliente.Dirección 'Dirección', Cliente.Fono 'Fono', Cliente.Disponible 'Disponible'`
Dependencias Funcionales:	

Nombre:	**sp_clienteXruta**
Descripción:	Determina una lista de clientes para una ruta de una empresa determinada.
Entrada:	`(@Código_Empresa VARCHAR(2), @Número_Ruta VARCHAR(2))`
Salida:	`Cliente.Nombre 'Nombre', Factura.Número_Factura 'N. Factura', Factura.Estado 'Estado' ,Factura.Monto_Total 'Monto Total'`
Dependencias Funcionales:	

Nombre:	**sp_CuotasEmpresa**
Descripción:	Retorna una lista de pares ordenados (fecha, cuota), para una empresa dada.
Entrada:	`(@Código_Empresa VARCHAR(2))`
Salida:	Retorna lista de pares ordenados (fecha, cuota).
Dependencias Funcionales:	

Nombre:	**sp_CuotasFactura**
Descripción:	Retorna una lista de pares ordenados (fecha, cuota), para una empresa y un número de factura dados.
Entrada:	`(@Código_Empresa VARCHAR(2), @Número_Factura VARCHAR(20))`
Salida:	Retorna lista de pares ordenados (fecha, cuota).
Dependencias Funcionales:	
Nombre:	**sp_CuotasRutas**
Descripción:	Retorna una lista de pares ordenados (fecha, cuota), para una empresa y una ruta dada.
Entrada:	`(@Código_Empresa VARCHAR(2), @Número_Ruta VARCHAR(3))`
Salida:	Retorna lista de pares ordenados (fecha, cuota).
Dependencias Funcionales:	

Nombre:	sp_FacturaCliente
Descripción:	Retorna la información que está relacionada con un número de factura de una empresa determinada.
Entrada:	(@Código_Empresa VARCHAR(2), @Número_Factura INT)
Salida:	Factura.Número_Factura 'N. Factura', Cliente.Nombre 'Nombre', Cliente.RUT 'RUT', Cliente.Dirección 'Dirección', Cliente.Fono 'Fono', Factura.Fecha_Emisión 'Fecha Emisión'
Dependencias Funcionales:	

Nombre:	sp_GetMontoTotalRutasXEmpresa
Descripción:	Obtiene los montos totales por ruta para cada empresa. Se incluye Monto total, Monto total Entregado, Monto Total por Entregar, y Monto Total No Entregado.
Entrada:	(@Código VARCHAR(2))
Salida:	Monto, Entregado, Por Entregar, No Entregado
Dependencias Funcionales:	

Nombre:	sp_GetTotalEmpresa
Descripción:	Obtiene la suma de todos los montos que están en la tabla Cuota, y que pertenezcan a la empresa cuyo código es @Código.
Entrada:	(@Código VARCHAR(2))
Salida:	SUM(Cuota.Monto) 'Monto'
Dependencias Funcionales:	

Nombre:	sp_GetTotalFactura
Descripción:	Obtiene los montos totales para una factura de una empresa específica. Se incluye Monto total, Monto total Entregado, Monto Total por Entregar, y Monto Total No Entregado.
Entrada:	(@Código VARCHAR(2), @FACTURA VARCHAR(20))
Salida:	Monto, Entregado, Por Entregar, No Entregado
Dependencias Funcionales:	

Nombre:	sp_GetTotalRuta
Descripción:	Obtiene los montos totales para las rutas de una empresa específica. Se incluye Monto total, Monto total Entregado, Monto Total por Entregar, y Monto Total No Entregado.
Entrada:	(@Código VARCHAR(2), @RUTA VARCHAR(3))
Salida:	Monto, Entregado, Por Entregar, No Entregado
Dependencias Funcionales:	

Nombre:	sp_GetUltima_Transacción
Descripción:	Obtiene el número de la última transacción desde la tabla Transacciones.
Entrada:	
Salida:	num_trans
Dependencias Funcionales:	

Nombre:	sp_Insert_Cuota
Descripción:	Inserta valores en la tabla Cuota. Valores descritos en Entrada.
Entrada:	(@fecha VARCHAR(20), @monto VARCHAR(10), @num_factura VARCHAR(10), @Code_empresa VARCHAR(2))
Salida:	
Dependencias Funcionales:	

Nombre:	sp_Insert_Transacciones
Descripción:	Inserta una nueva fila a la tabla Transacciones.
Entrada:	(@RUT VARCHAR(11), @Code_Empresa VARCHAR(2), @Num_Factura VARCHAR(10), @Num_Trans Varchar(20))
Salida:	
Dependencias Funcionales:	

Nombre:	sp_No_Entregado
Descripción:	Actualiza la tabla Factura. Asigna el estado 'NEN' al campo Estado, para un número de factura perteneciente a una empresa específica.
Entrada:	(@Código_Empresa Varchar(2), @Num_Factura Varchar(20))
Salida:	
Dependencias Funcionales:	

Nombre:	sp_Recaudadores
Descripción:	Obtiene una lista a partir de la tabla Recaudador con los siguientes campos: Nombre, RUT y Fono.
Entrada:	(@Código VARCHAR(2))
Salida:	Recaudador.Nombre 'Nombre', Recaudador.RUT 'RUT', Recaudador.Fono 'Fono'
Dependencias Funcionales:	

Nombre:	sp_Reiniciar
Descripción:	1. Borra todos las filas de la tabla Transacciones en las que el campo rut sea distinto de null. 2. Borra todas las filas de la tabla Cuota. 3. Actualiza el campo Estado de todas las filas de la tabla Factura y los iguala a 'PEN' 4. Actualiza el campo Fecha_Pago de todas las filas de la tabla Factura y los iguala a: Fecha_Emisión + 45 5. Actualiza la tabla Cliente asignando al campo Disponible de todas sus filas el valor 200.000.
Entrada:	
Salida:	
Dependencias Funcionales:	

Nombre:	sp_rutasDeRecaudación
Descripción:	Obtiene el monto total, el monto total por entregar, el monto total entregado y el monto total no entregado para cada ruta de una empresa dada.
Entrada:	(@Código Varchar(2))
Salida:	'Por Entregar', 'Entregado', 'No Entregado', 'Monto Ruta'
Dependencias Funcionales:	

Nombre:	sp_UpDate_Cliente
Descripción:	Actualiza la tabla Cliente. El campo Disponible, contendrá el nuevo valor @dif, para un rut de cliente @rut_c
Entrada:	(@dif VARCHAR(20), @rut_c VARCHAR(11))
Salida:	
Dependencias Funcionales:	

Nombre:	sp_UpDate_Ruta
Descripción:	Actualiza el campo Estado de la tabla Factura con el valor 'ENT' para una factura de una determinada empresa.
Entrada:	(@empresa VARCHAR(10), @num_factura VARCHAR(20))
Salida:	
Dependencias Funcionales:	

Nombre:	sp_ValidaCliente
Descripción:	Obtiene la cantidad de dinero Disponible para un determinado rut de cliente y contraseña. (@rut_c, @pass_c)
Entrada:	(@rut_c VARCHAR(11), @pass_c VARCHAR(11))
Salida:	Cliente.Disponible
Dependencias Funcionales:	

Nombre:	sp_ValidaRecaudador
Descripción:	Valida la identidad de un recaudador mediante el código y clave proporcionados. Los nuevos códigos son de 6 dígitos (los 6 primeros dígitos del rut), la clave sigue siendo de 4 dígitos.
Entrada:	(@RUT VARCHAR(6), @Clave VARCHAR(4))
Salida:	
Dependencias Funcionales:	

Nombre:	sp_ValidaUsuario
Descripción:	Valida la identidad de un usuario mediante el código y clave proporcionados.
Entrada:	(@Código VARCHAR(2), @Clave VARCHAR(4))
Salida:	
Dependencias Funcionales:	

Nombre:	**sp_VerEmpresas**
Descripción:	Obtiene todos los campos de una empresa determinada
Entrada:	`(@Código VARCHAR(2), @Clave VARCHAR(4))`
Salida:	
Dependencias Funcionales:	

Nombre:	**sp_VerEstado**
Descripción:	Obtiene la tupla (estado, número factura), para una factura de una determinada empresa.
Entrada:	`(@num_factura VARCHAR(20), @Empresa VARCHAR(2))`
Salida:	`'estado', 'factura'`
Dependencias Funcionales:	
Nombre:	**sp_VerRuta**
Descripción:	Obtiene información de la factura de clientes en una ruta específica de una empresa.
Entrada:	`(@Código_Empresa VARCHAR(2),` `@Número_Ruta VARCHAR(2))`

Salida:	`Cliente.Nombre 'Nombre',`
	`Factura.Estado 'Estado',`
	`Factura.Número_Factura 'num_factura',`
	`Factura.Fecha_Emisión 'fecha_emision',`
	`Factura.Monto_Total 'monto',`
	`Factura.RUT_Cliente 'rut_c',`
	`Factura.Glosa 'glosa'`

Dependencias Funcionales:	

7.4.1.3.1.6 Modificaciones al sistema WAP-REM

Los siguientes archivos del prototipo WAP-REM han sido modificados para apoyar el uso de procedimientos almacenados a través del sistema administrador de bases de datos MSQL Server 7.

Tabla 7.25: Modificaciones a archivos del prototipo WAP-REM.

Archivo	Procedimientos Almacenados
procesarlogin_T.asp	**sp_ValidaRecaudador** **sp_VerRuta**
procesarlogin_c.asp	**sp_ValidaCliente** **sp_UpDate_Cliente** **sp_UpDate_Ruta** **sp_Insert_Cuota** **sp_GetUltima_Transacción** **sp_Insert_Transacciones** **sp_GetUltima_Transacción**
procesar_no_entregado.asp	**sp_ValidaRecaudador** **sp_No_Entregado**
facturas.asp	**sp_VerEstado**

No se han realizado mayores modificaciones en este nivel del sistema. Las interfaces gráficas de usuario no han sufrido ninguna modificación.

7.4.1.3.2 Workflow de implementación

Se realizan nuevas interfaces gráficas de usuario, ver figura 7.40.

Figura 7.40: Construcción de nuevas interfaces gráficas de usuario.

Además, se implementan los métodos de las clases. El objetivo es ir refinando el prototipo.

Como se ha dicho anteriormente, las interfaces gráficas de usuario del prototipo WAP-REM versión 5, no se han modificado. Esto debido a que se ha reutilizado la estructura del código ASP mezclado con WML que fue creado para la aplicación artesanal del sitio WAP. Lo anterior obedece a que no es posible utilizar elementos de la plataforma .NET en el desarrollo del sitio WAP, ya que los micro-browsers de los dispositivos celulares escogidos, no apoyan ASP.NET. Sin embargo, se ha actualizado el código para hacerlo más eficiente. Se han reconstruido las consultas y actualizaciones que se encontraban insertas en el código ASP, y han sido implementadas haciendo uso de procedimientos almacenados sobre el sistema administrador de bases de datos **MSQL Server 7**.

7.4.1.3.3 Workflow de prueba

Las siguientes figuras muestran un recorrido de prueba por la aplicación Web.

Figura 7.41: Página principal de la aplicación prototipo WEB - REM versión 5

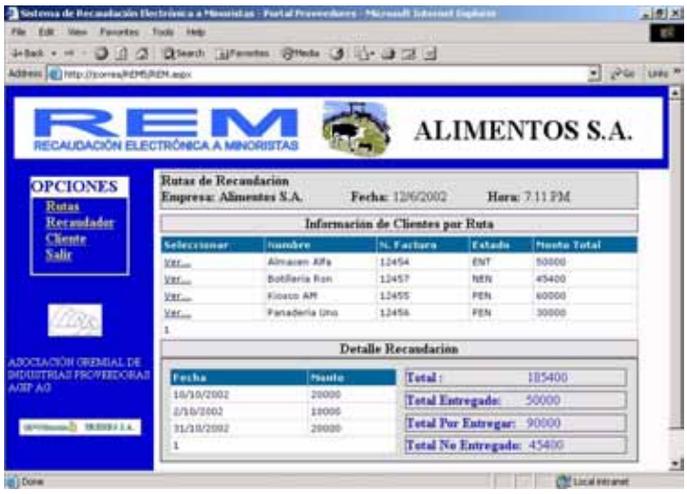


Figura 7.42: Página que despliega la información de clientes por ruta en prototipo REM 5.

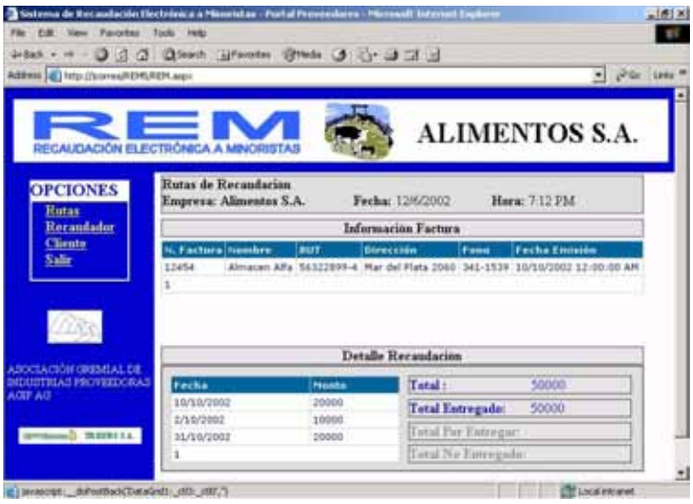


Figura 7.43: Página que despliega el detalle de la factura de un cliente en prototipo REM 5.

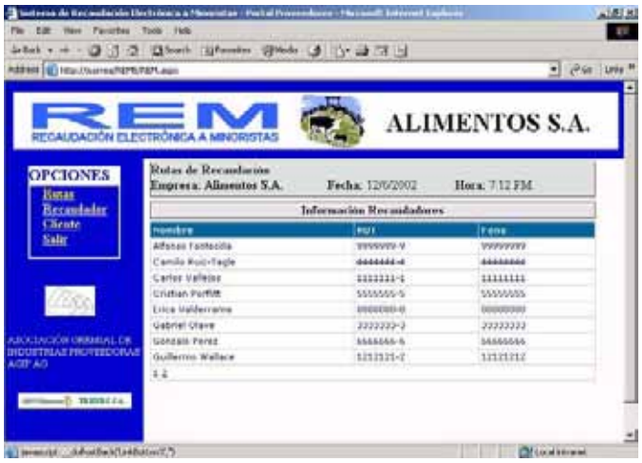


Figura 7.44: Página que despliega la información de los recaudadores.

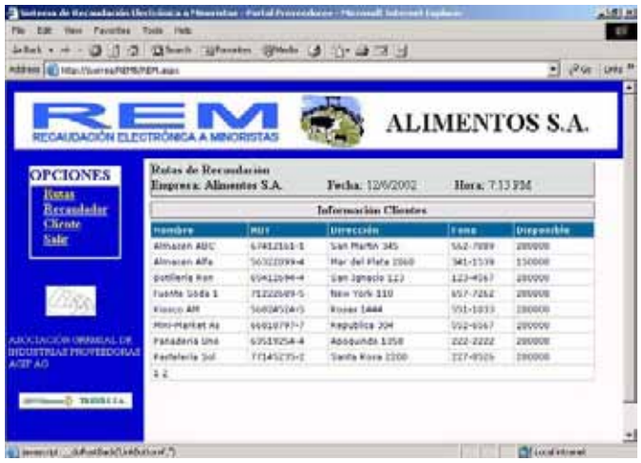


Figura 7.45: Página que despliega la información de los clientes.

Las transacciones han sido generadas mediante el sitio WAP-REM versión 5, modificado mediante la construcción de los procedimientos almacenados descritos anteriormente. Las interfaces gráficas de usuario para el sistema WAP no fueron modificadas por lo que desplegarán la misma secuencia de navegación típica de la **sección 7.3.4.1.2**.

7.4.1.4 Fase de transición

7.4.1.4.1 Workflow de despliegue

El workflow de despligue debe asegurar la correcta distribución de la aplicación web. Debido a la naturaleza de este tipo de aplicaciones, la distribución se realiza a través de una red de área local o a través de Internet. Debido a esto, se debe asegurar la accesibilidad de los clientes a la aplicación a través de la red, brindando el nivel de acceso adecuado a los servidores de prueba.

7.4.2 Próximas iteraciones

Es importante concluir que al final de esta primera iteración se tiene un prototipo funcional del sistema REM5. Sin embargo, es necesario seguir refinando dicho prototipo para obtener un producto de excelencia. Esto es, **calidad aceptable**, guiada según los requerimientos de usuario y niveles de complejidad del producto. De forma de brindar un producto competitivo con su pares en el mercado.

Por otro lado, el número de iteraciones será proporcional al grado de entendimiento de él o los problemas que resuelve el sistema software que está siendo desarrollado. El presente prototipo se ha planificado en tres iteraciones, según la experiencia en el desarrollo de este tipo de aplicaciones. Dentro de las próximas iteraciones, es necesario repetir el proceso de desarrollo poniendo énfasis en los diversos aspectos según se requiera. Nuevos requerimientos aparecerán para el producto software. Es posible anticipar requerimientos de seguridad y conectividad, los cuales no serán cubiertos en el presente trabajo. Sin embargo, se recomienda **[28]** en la bibliografía, como un muy buen punto de partida en esta materia.

La segunda y tercera iteración esperan las siguientes actividades :

- **Segunda Iteración**
- **Fase de inicio**
    - **Workflow de Requerimientos**
        - **Refinar Casos de uso de alto nivel WEB R.E.M.**
        - **Refinar Casos de uso expandidos WEB R.E.M.**
    - **Workflow de análisis y diseño**
        - **Refinar Modelamiento Web del sitio WEB R.E.M.**
    - **Workflow de administración de proyectos**
        - **Refinar Estimación de costos del prototipo WEB R.E.M. versión 5**
        - **Evaluación de riesgos del proyecto**
- **Fase de elaboración**
    - **Workflow de requerimientos**

- **Workflow de análisis y diseño**
  - **Ajustar diseño a requerimientos**
- **Workflow de implementación**
  - **Prototipado evolutivo**
- **Workflow de administración de proyectos**
  - **Ajustar planificación**

- **Fase de construcción**
  - **Workflow de análisis y diseño**
  - **Workflow de implementación**
  - **Workflow de prueba**
  - **Workflow de administración de proyectos**
- **Fase de Transición.**
  - **Workflow de prueba.**
    - **Prueba aplicación Web y distribución de ésta.**
      - **Prueba WAP.**
      - **Prueba WEB.**
        - **RED LAN.**
        - **RED Internet.**
- **Tercera Iteración.**
- **Fase de inicio .**
  - **Workflow de implementación.**
    - **Actualización de manuales de usuario.**
  - **Workflow de diseño.**
    - **Actualización del diseño del sistema.**
  - **Workflow de Administración de Proyectos.**
    - **Costo de componentes (Frecuencia versus Costo).**
      - **Mejores candidatos a reutilizar.**

        - **Costos de Reutilización.**

- **Fase de Elaboración.**
    - **Worflow de diseño** (según Worflow de Administración de proyecto fase anterior).
        - **Diseño para la reutilización de componentes.**
        - **Diseño y organización de librerías de código.**
    - **Workflow de implementación.**
        - **Prototipado de componentes reutilizables.**
- **Fase de construcción.**
    - **Workflow de implementación.**
        - **Implementación de componentes reutilizables.**
        - **Implementación de librerías de código.**
- **Fase de Transición.**
    - **Worflow de diseño.**
        - **Prueba y aseguramiento de la calidad.**

7.5 Conclusiones

En el presente capítulo se ha intentado plasmar la experiencia de desarrollo de las versiones DEMO del prototipo de Recaudación Electrónica a Minoristas, construidas desde dos enfoques distintos. Se ha podido comprobar cómo un proceso de desarrollo orientado a los casos de uso, puede mejorar aspectos de la administración de proyectos software, mediante estimaciones basadas en el diseño preliminar del sistema. También, se ha introducido una extensión al lenguaje UML, que permite modelar aplicaciones Web de una forma más eficiente que el UML tradicional.

El Capítulo 8, muestra el diseño de la metodología de desarrollo de aplicaciones sobre la plataforma Microsoft .NET Framework, construida gracias a la continua investigación en el área de Ingeniería de Software.

Capítulo 8 : Diseño de Metodología de desarrollo de aplicaciones sobre la plataforma Microsoft .NET Framework SDK

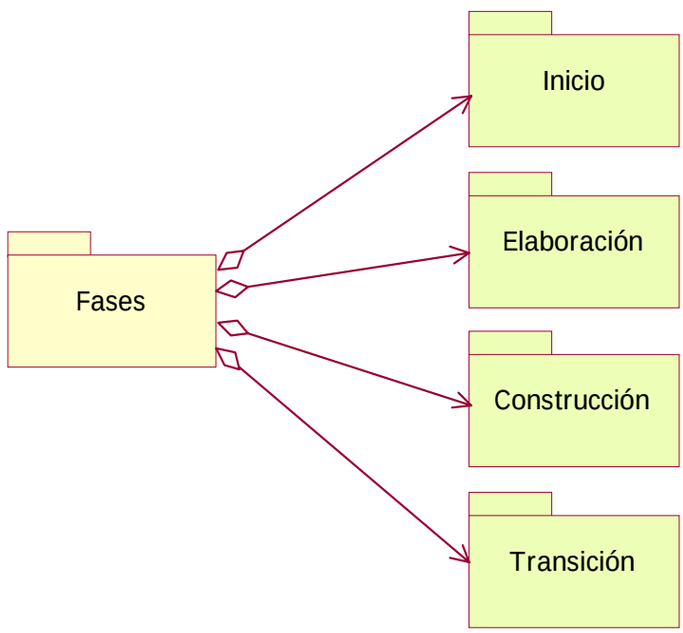
8.1 Workflows definidos por Metodología EFT-SDM

La metodología de desarrollo de aplicaciones sobre la plataforma Microsoft .Net Framework ha sido diseñada teniendo presente la siguiente estructura organizacional, ver tabla 8.1. Los workflows ahí expuestos, se ajustan a la realidad de las principales empresas de pequeña y mediana envergadura del ámbito nacional e internacional, que cuenten con al menos un área de desarrollo de software.

Tabla 8.1: Workflows participantes en la metodología EFT-SDM.

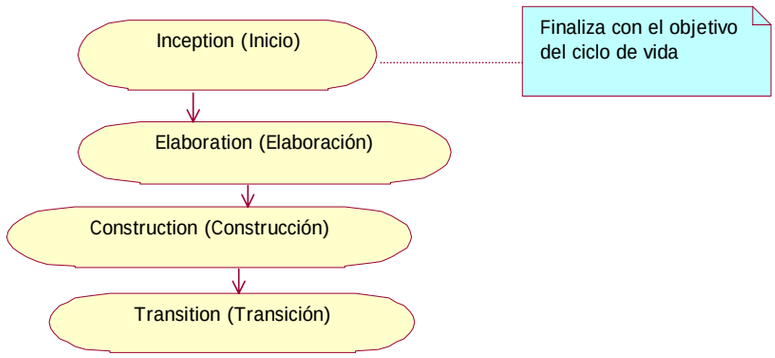
Workflow	Responsabilidades	# Mín. de personas
Administración de Proyectos	• **Modelamiento de negocios** • **Administración de proyectos**	**1**
Arquitectura de Software	• **Requerimientos** • **Análisis y Diseño** • **Prueba**	**1**
Desarrollo	• **Implementación** • **Despliegue**	**1**
Soporte	• **Entorno**	**1**

8.2 Fases del proceso de desarrollo

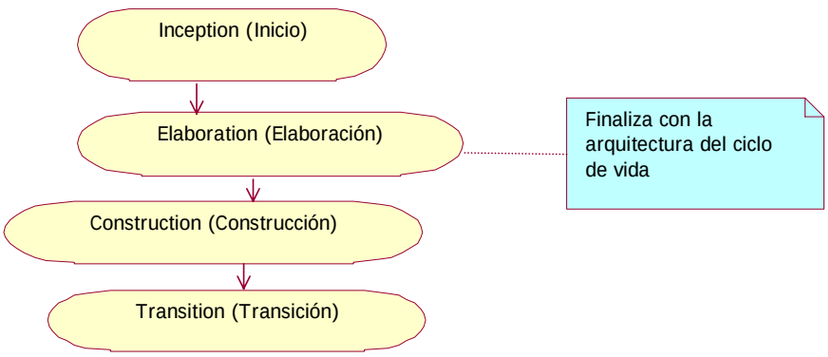


8.2.1 Secuencia e hitos de fases

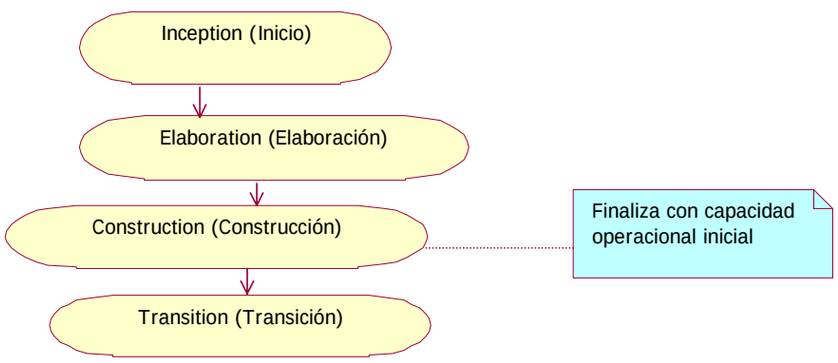
8.2.1.1 Fase de Inicio



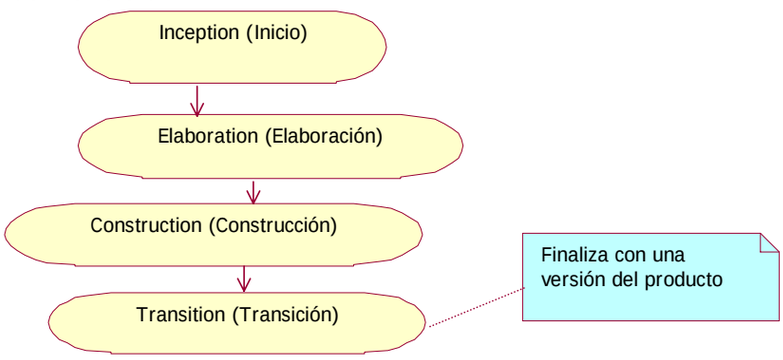
8.2.1.2 Fase de Elaboración



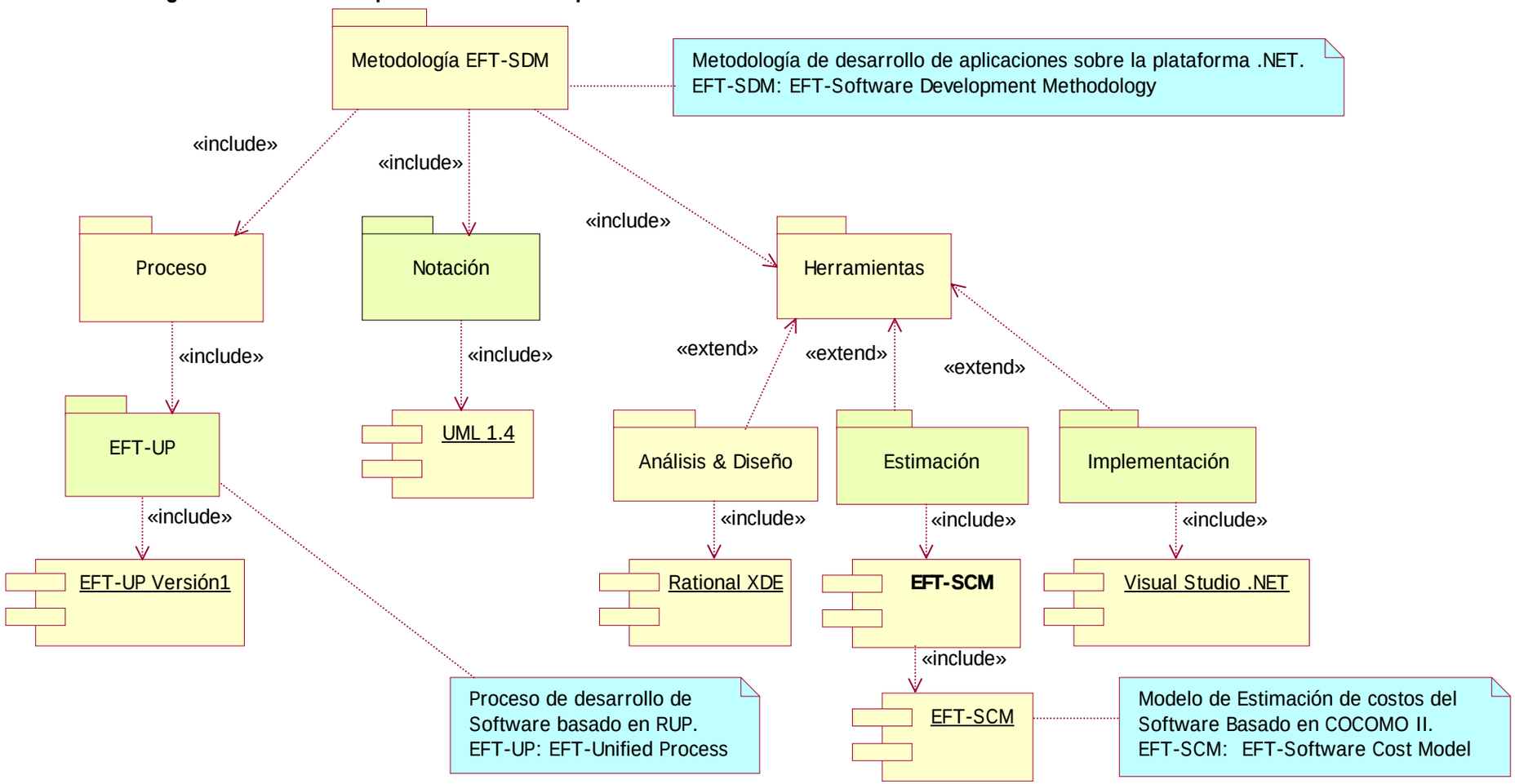
8.2.1.3 Fase de Construcción



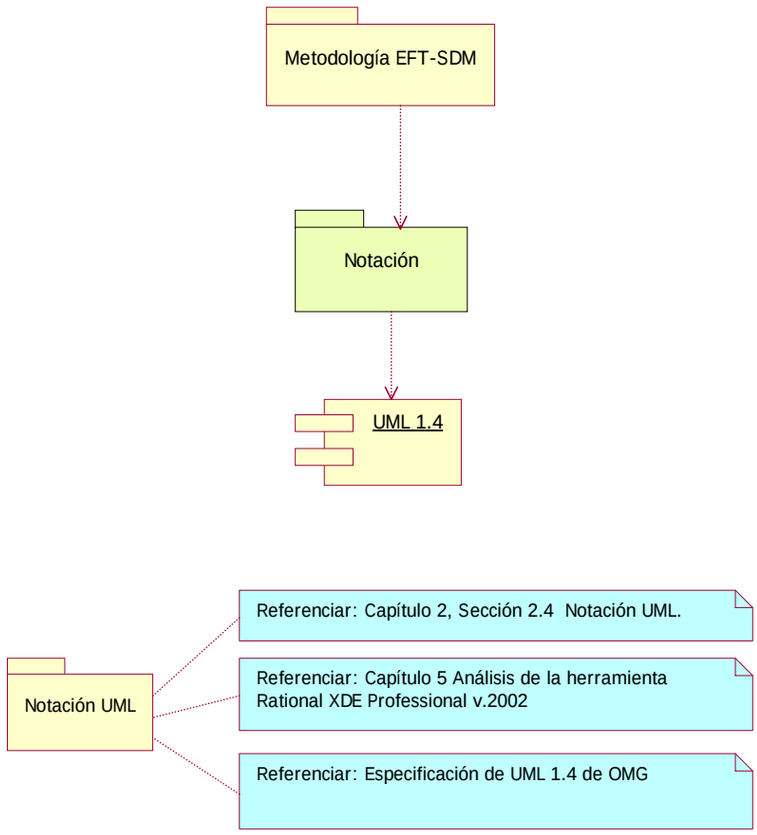
8.2.1.4 Fase de Transición



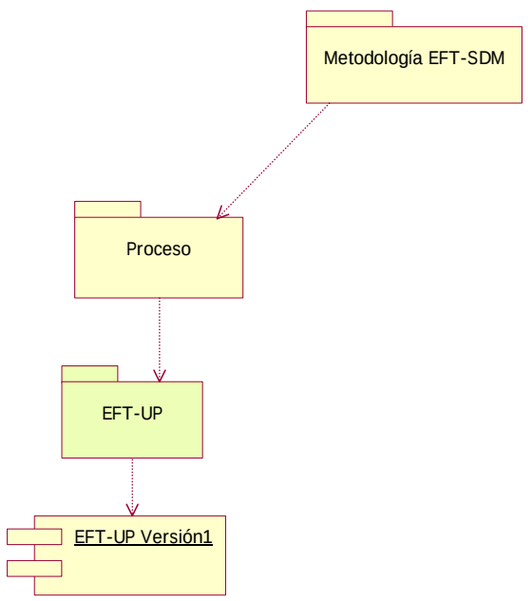
8.3 Metodología de desarrollo de aplicaciones sobre la plataforma Microsoft .Net



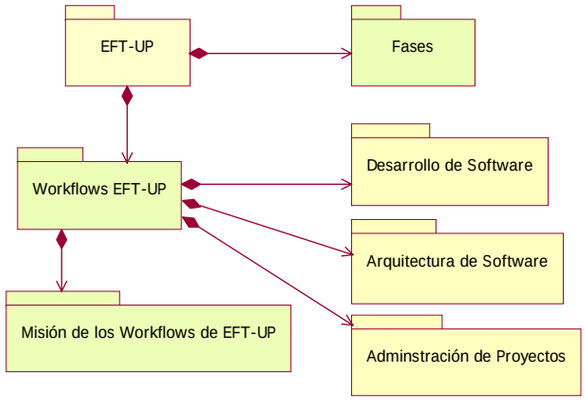
8.3.1 Notación



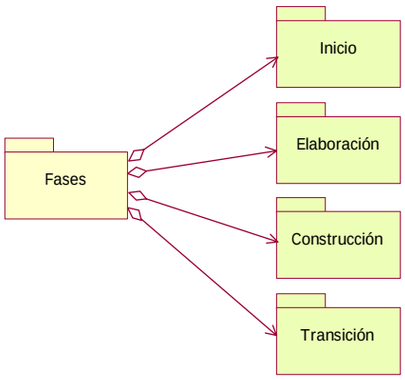
8.3.2 EFT-UP



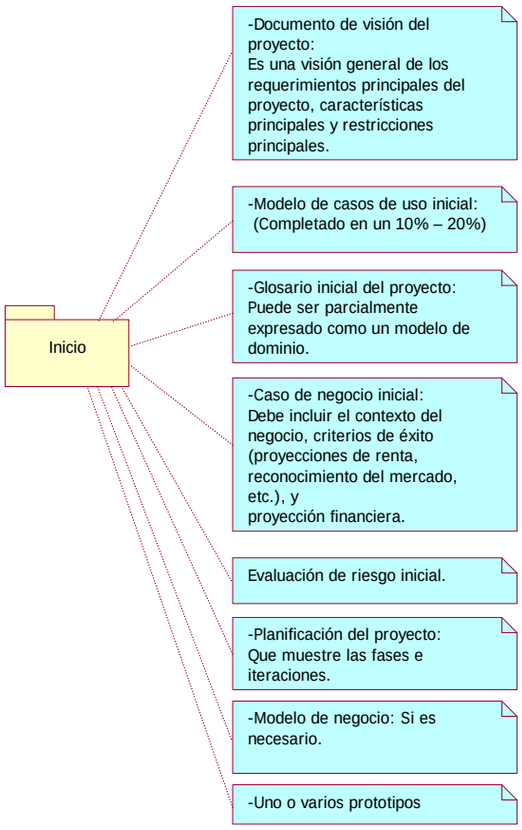
8.3.2.1 EFT-UP Versión 1



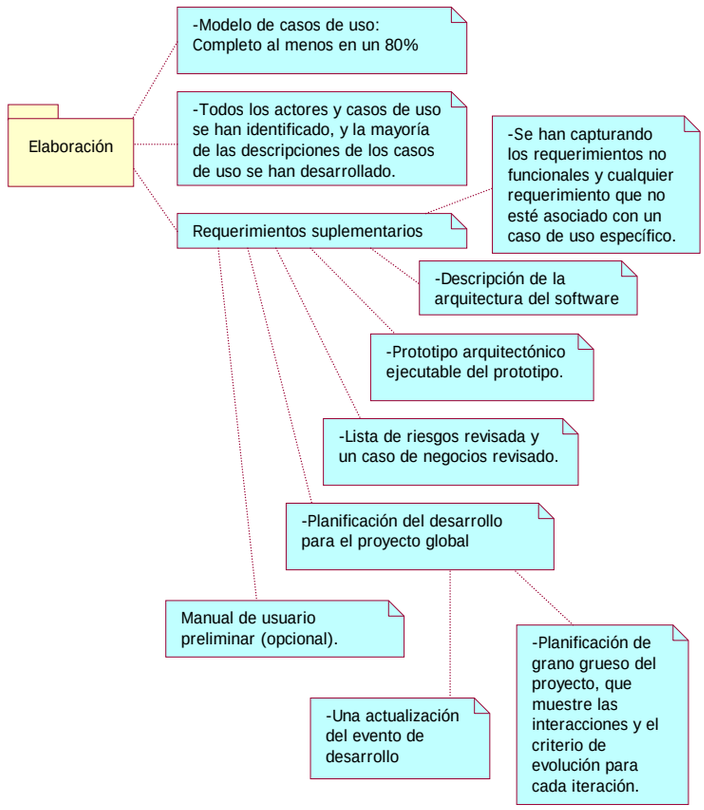
8.3.2.2 Fases EFT-UP Versión 1



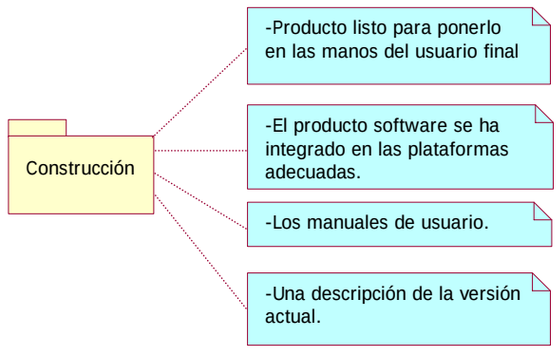
8.3.2.2.1 Resultados Fase de Inicio



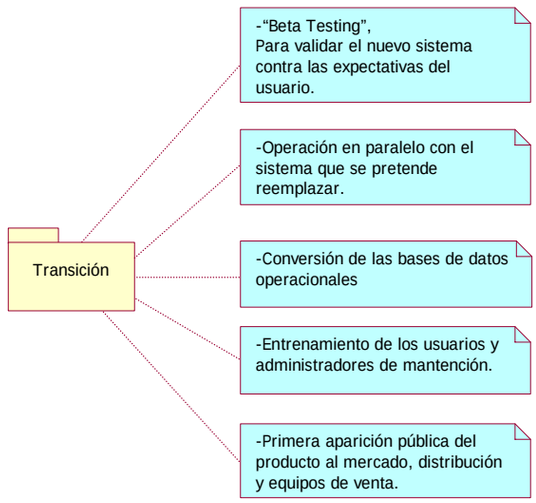
8.3.2.2.2 Resultados Fase de Elaboración



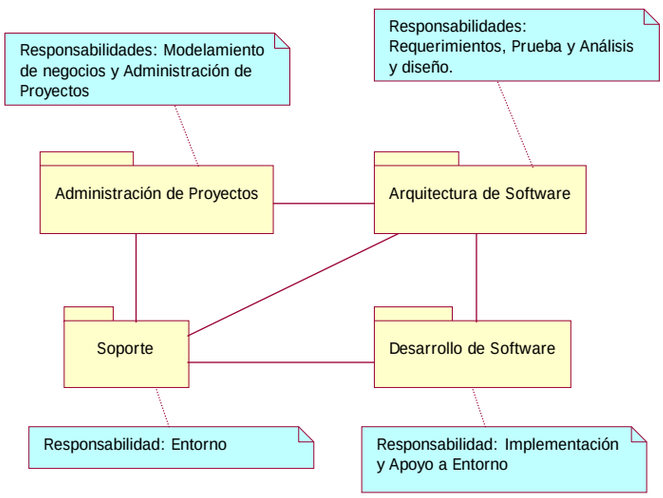
8.3.2.2.3 Resultados Fase de Construcción



8.3.2.2.4 Resultados Fase de Transición

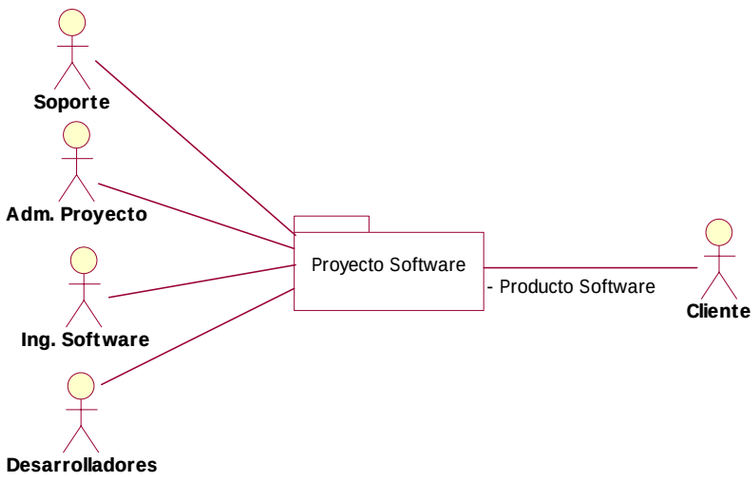


8.3.2.3 Workflows EFT- UP Versión 1

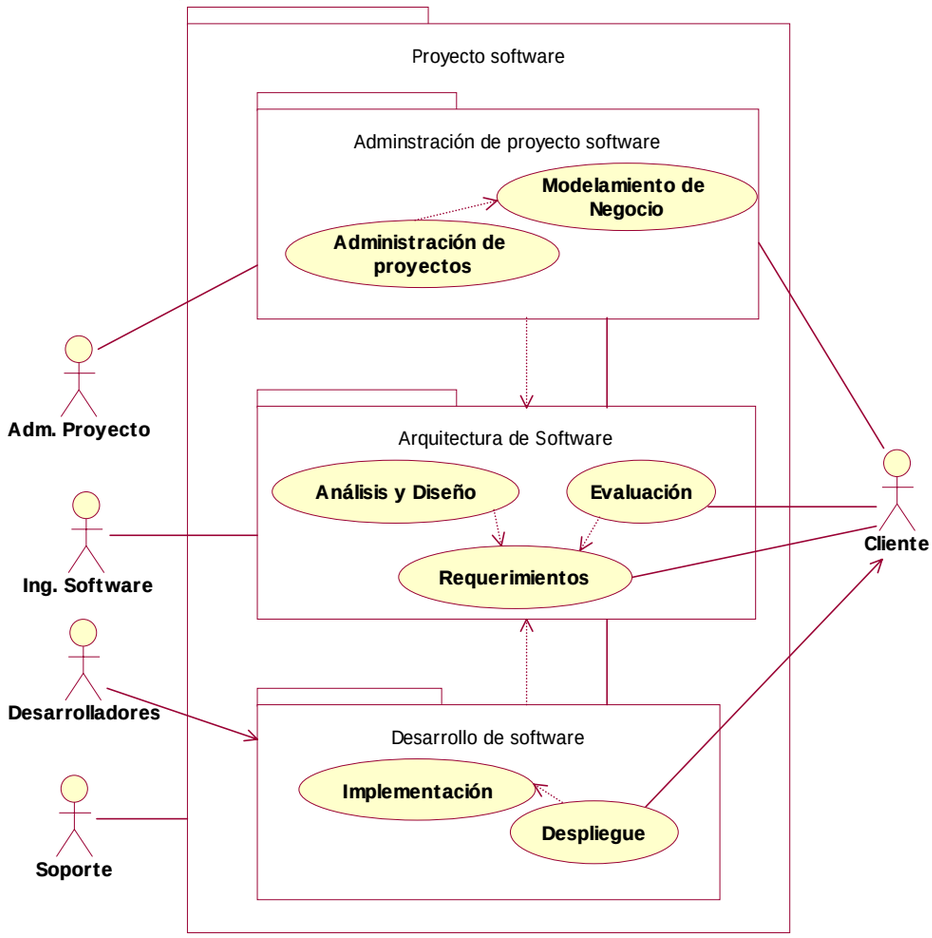


8.3.2.4 Misión de los Workflows de EFT- UP Versión 1

8.3.2.4.1 Misión global de los workflows de EFT - UP Versión 1

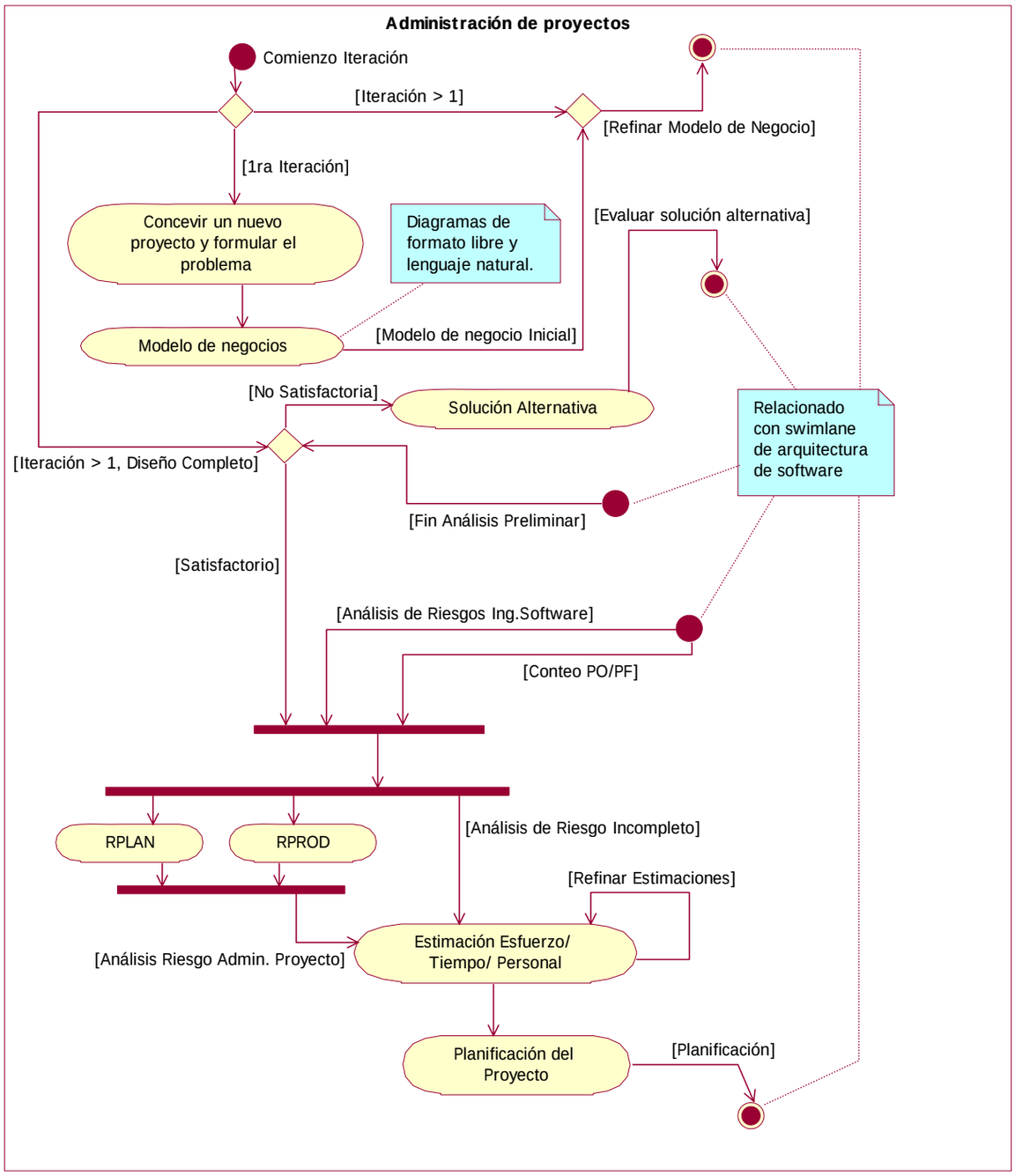


8.3.2.4.2 Misión específica de los workflows de EFT - UP Versión 1

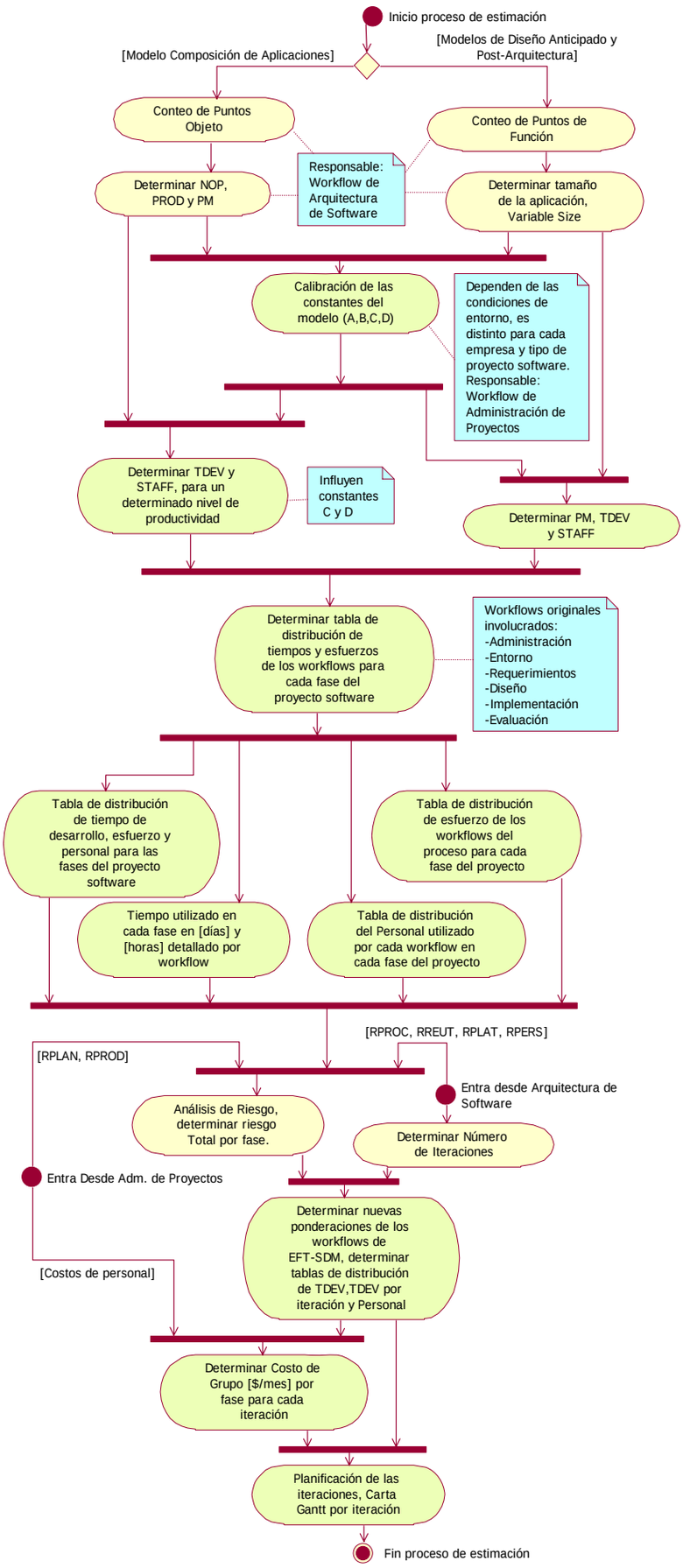


8.3.2.5 Diagramas de Actividad de los workflows del EFT-UP

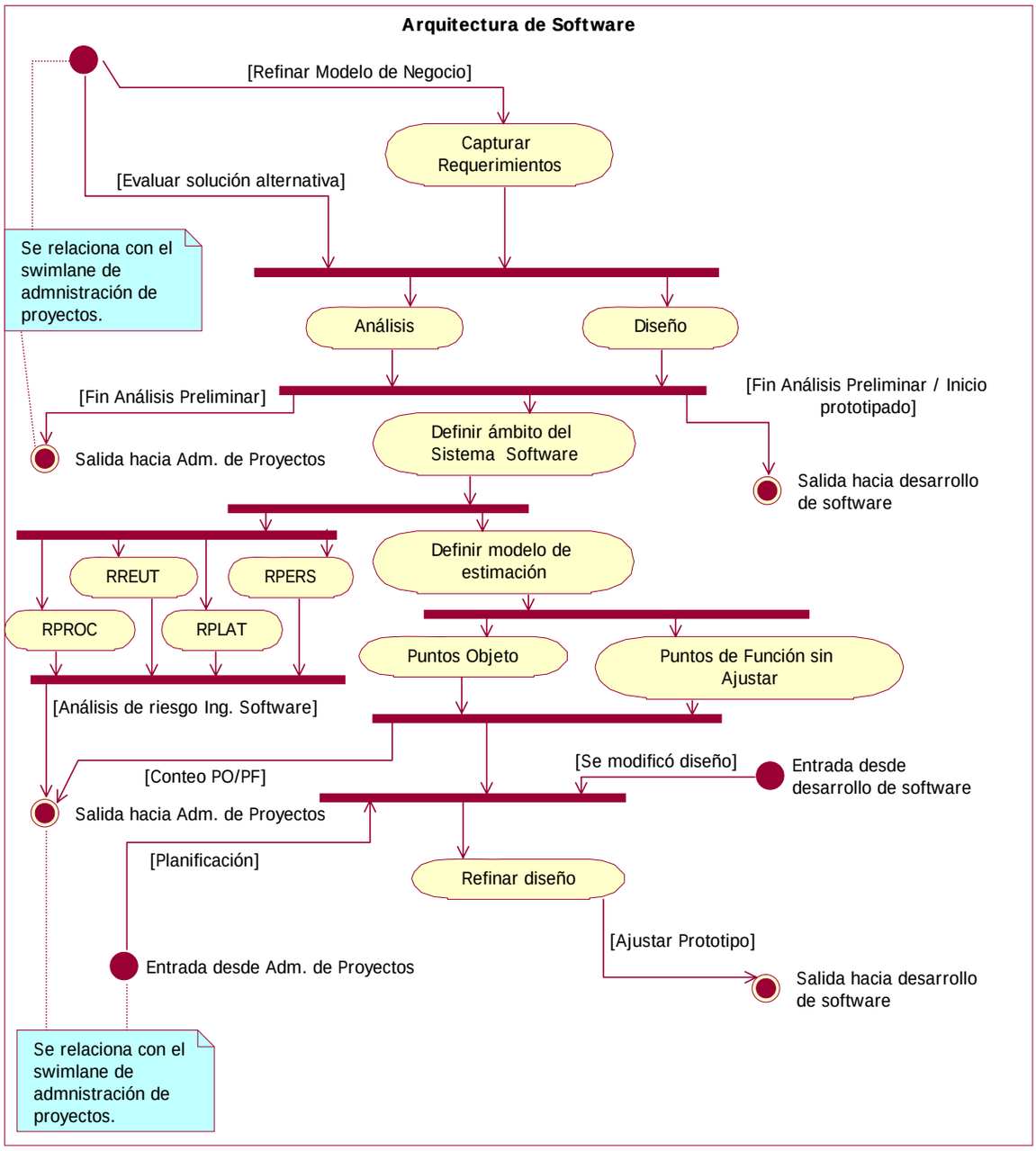
8.3.2.5.1 Workflow de Administración de Proyectos



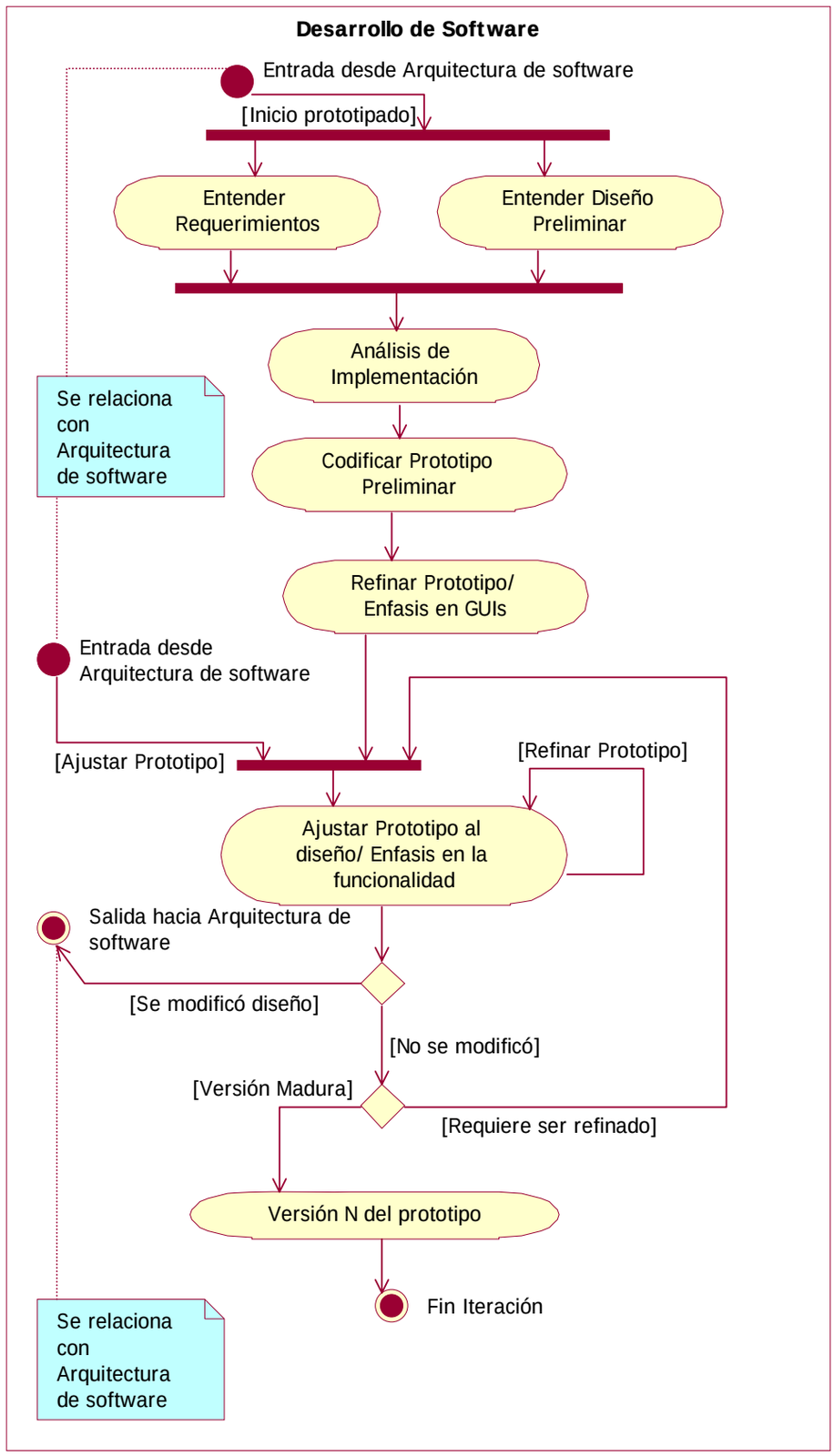
8.3.2.5.1.1 Proceso de estimación de EFT-SDM.



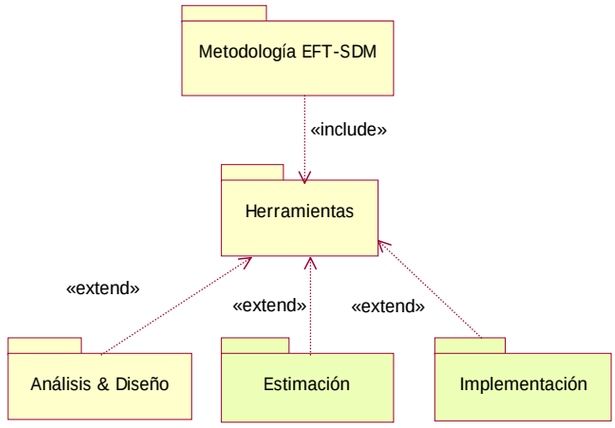
8.3.2.5.2 Workflow de Arquitectura de Software



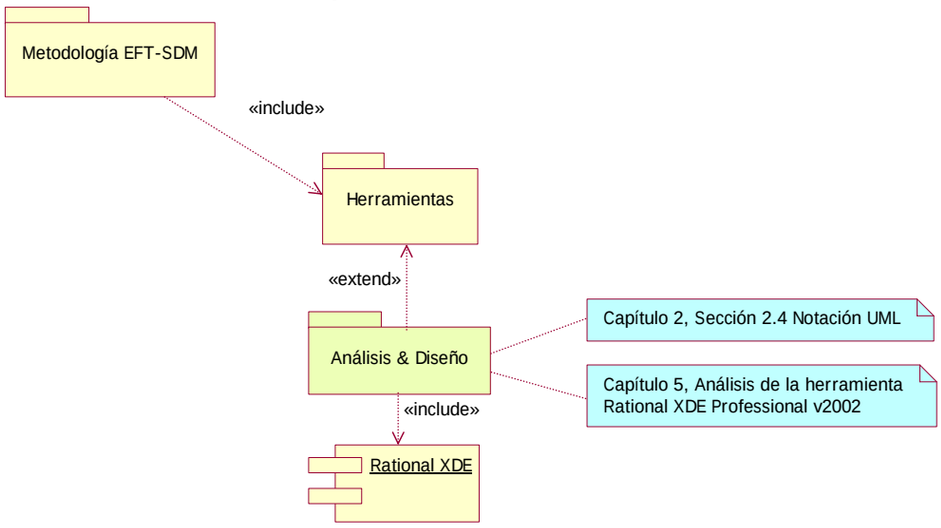
8.3.2.5.3 Workflow de Desarrollo de Software



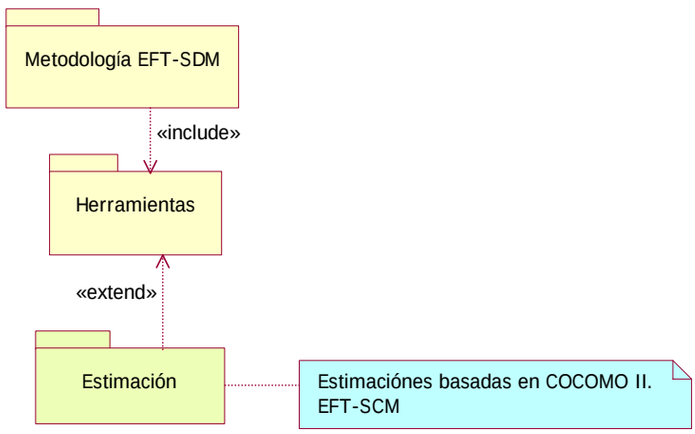
8.4 Herramientas de la metodología EFT - SDM

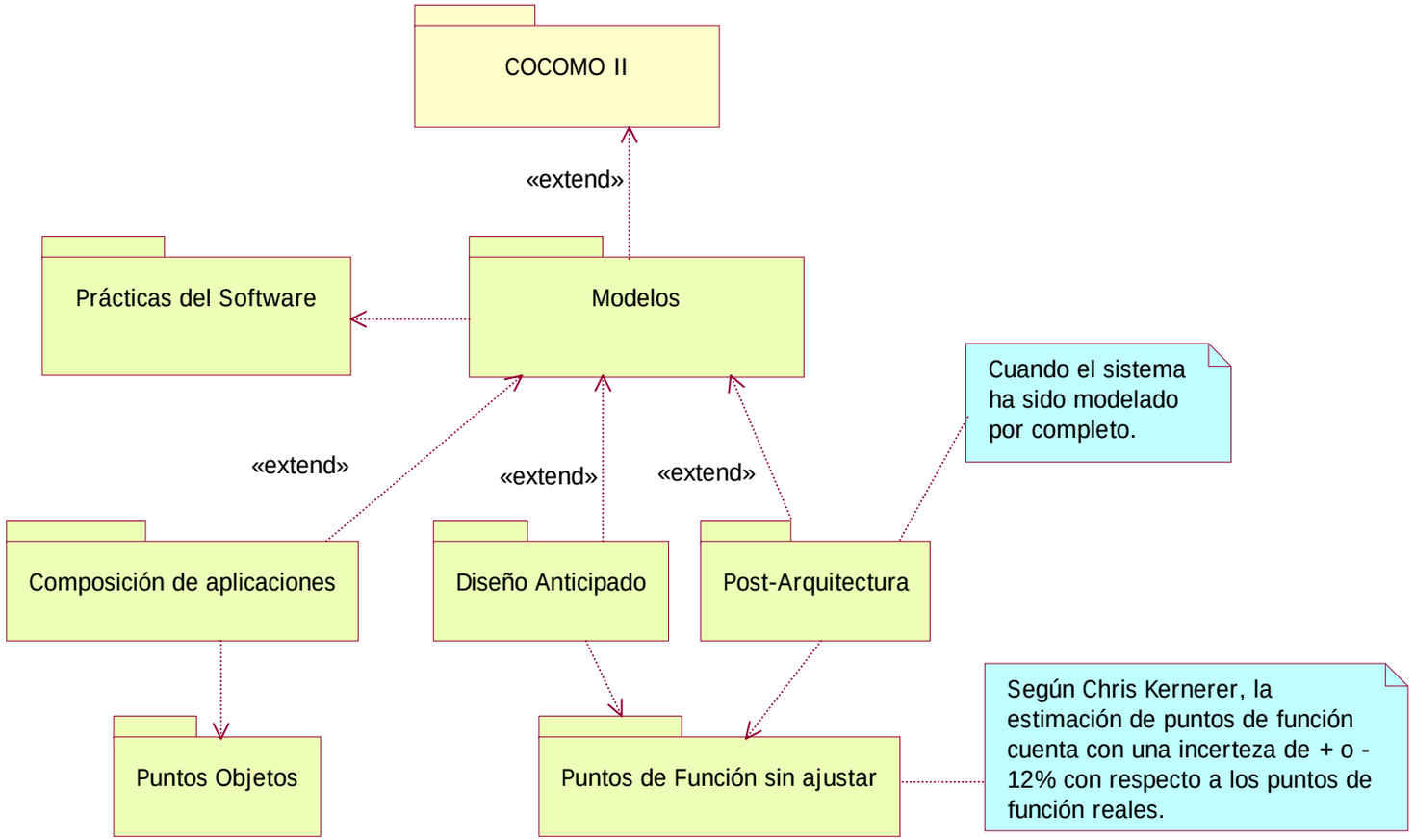


8.4.1 Herramientas de Análisis y Diseño



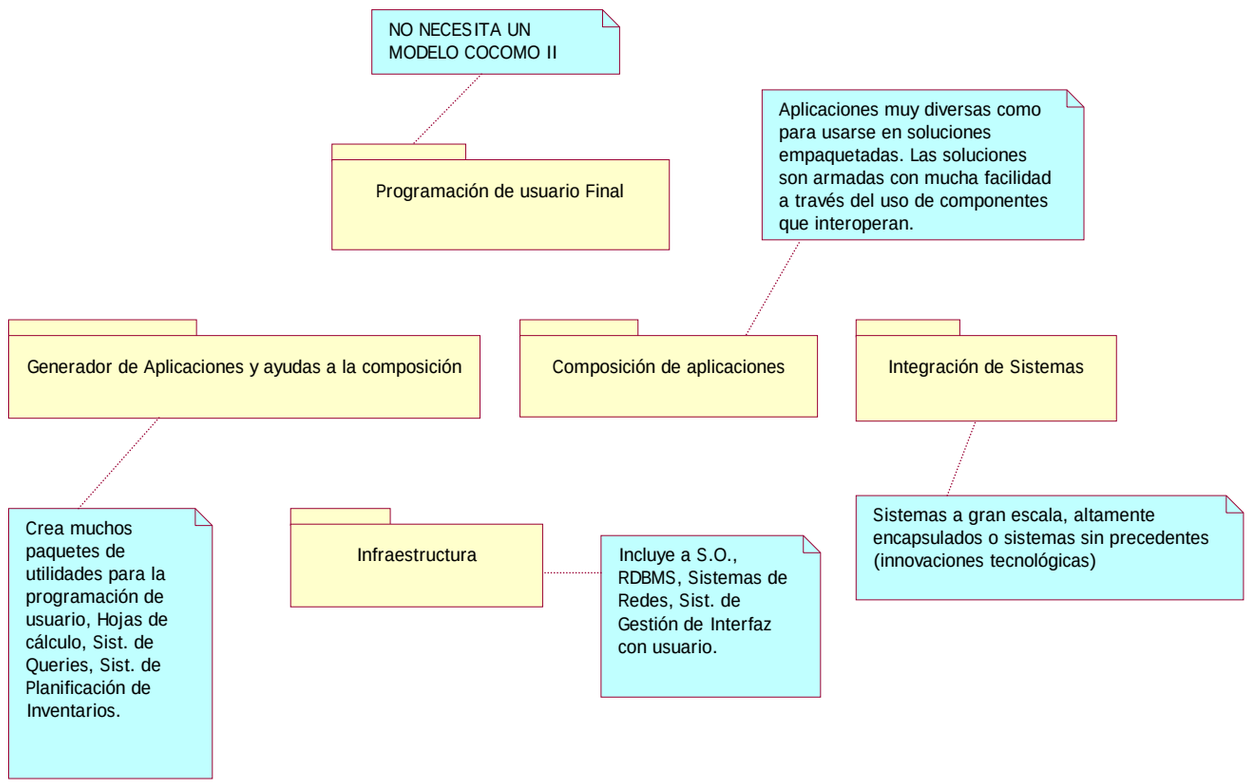
8.4.2 Herramientas de Estimación



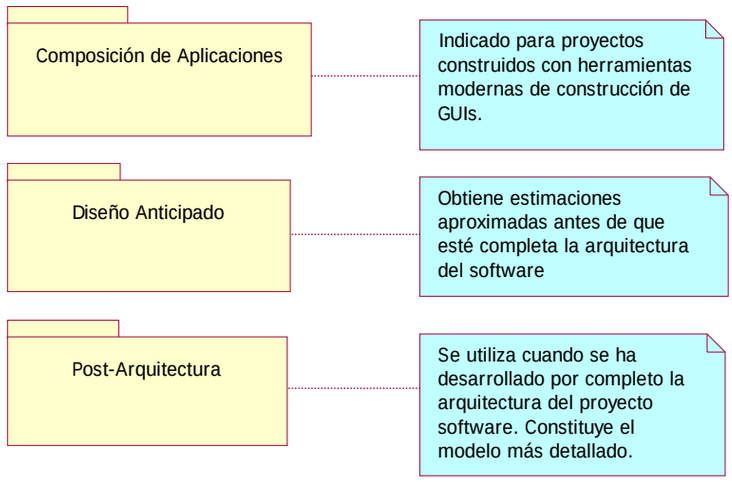

COCOMO II
«extend»
Prácticas del Software
Modelos
«extend»
«extend»
«extend»
Cuando el sistema ha sido modelado por completo.
Composición de aplicaciones
Diseño Anticipado
Post-Arquitectura
Puntos Objetos
Puntos de Función sin ajustar
Según Chris Kernerer, la estimación de puntos de función cuenta con una incerteza de + o - 12% con respecto a los puntos de función reales.

8.4.2.1 Futuras Prácticas del Software

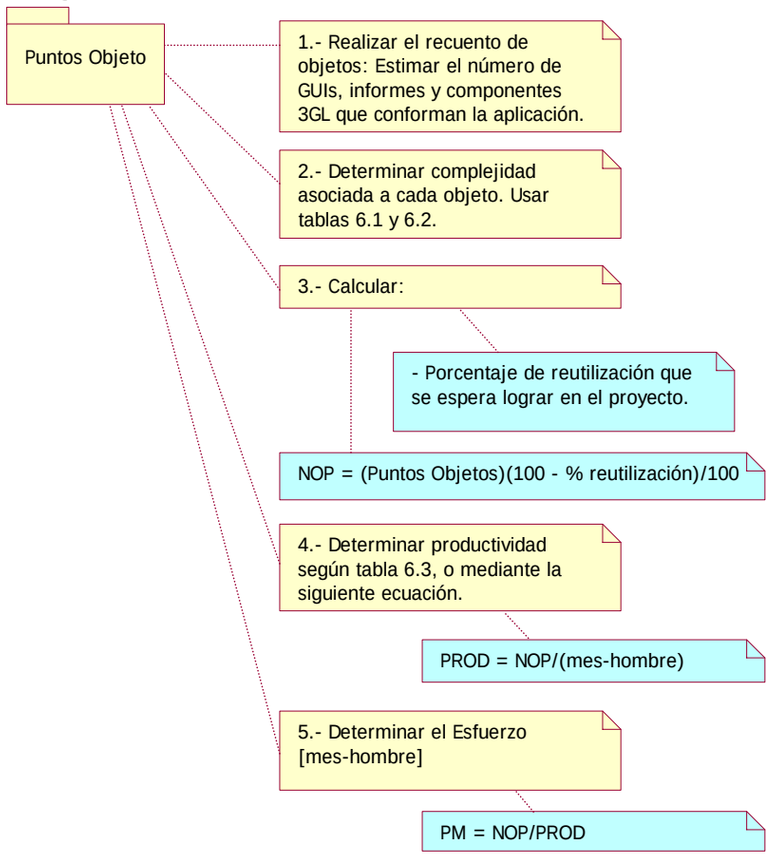
Modelo de futuras prácticas del Software - Previstas para el año 2005



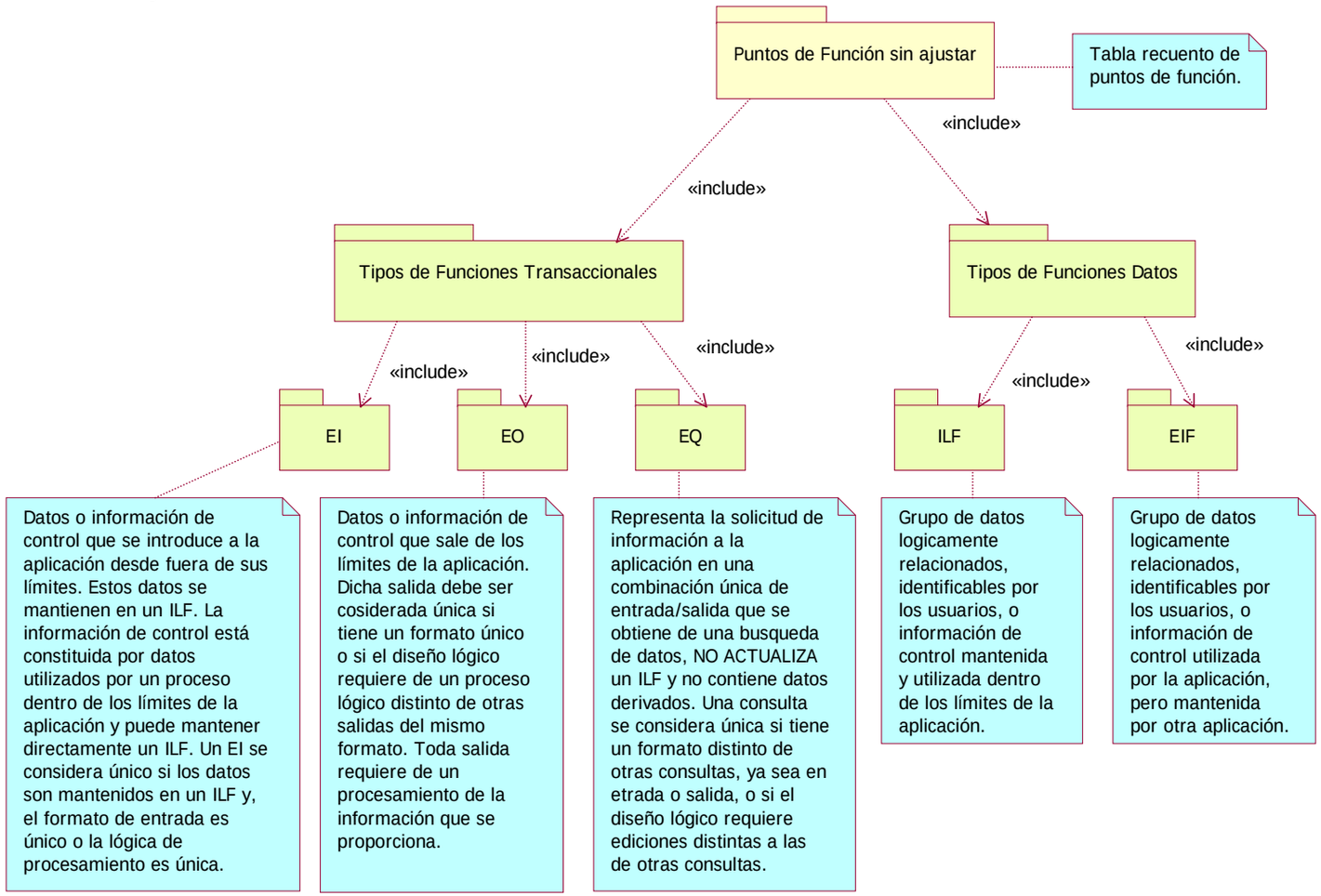
8.4.2.2 Modelos de Estimación



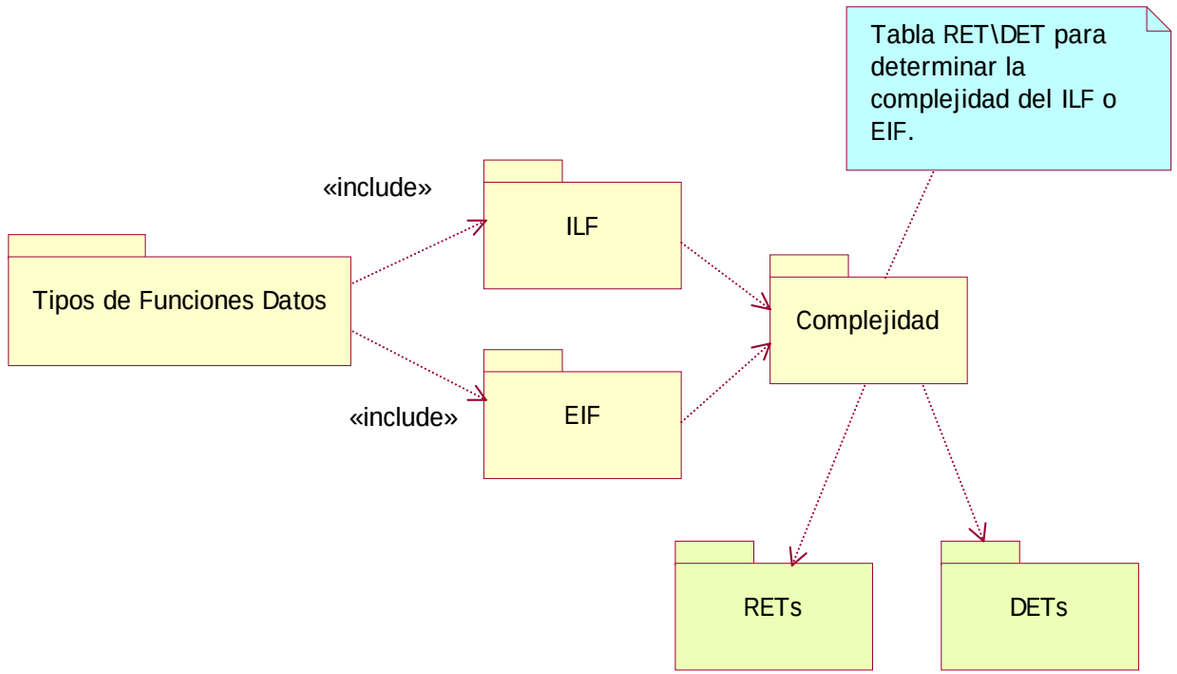
8.4.2.3 Puntos objeto



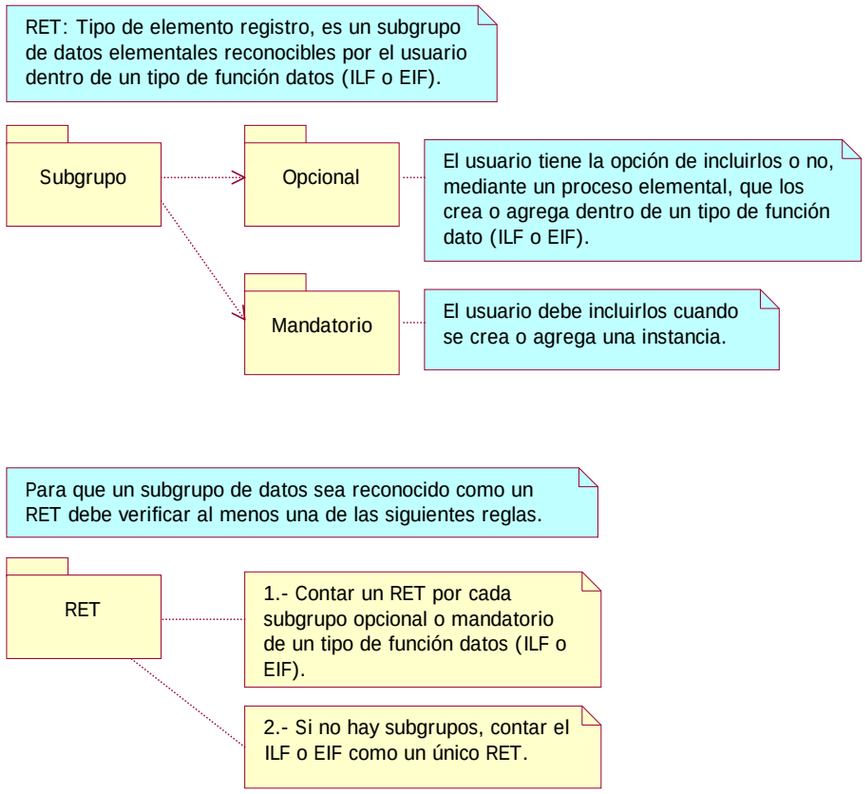
8.4.2.4 Puntos de Función sin ajustar



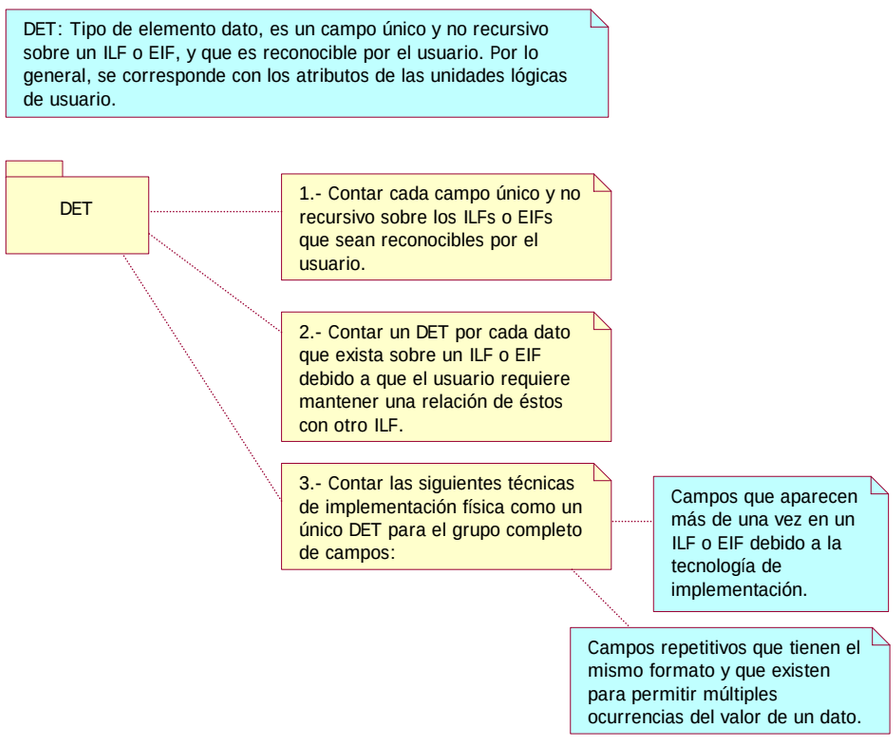
8.4.2.4.1 Tipos de funciones datos



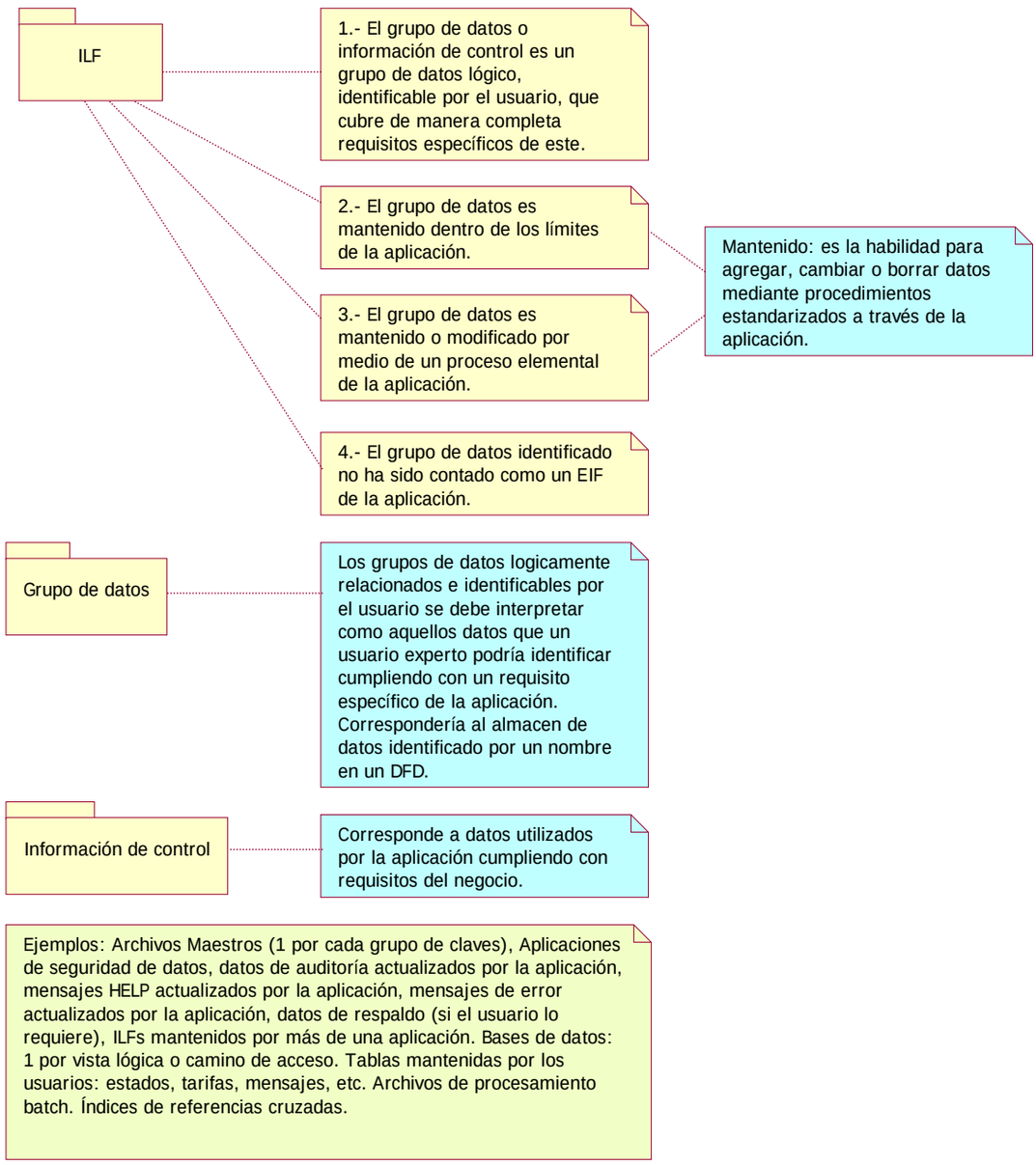
8.4.2.4.1.1 RETs



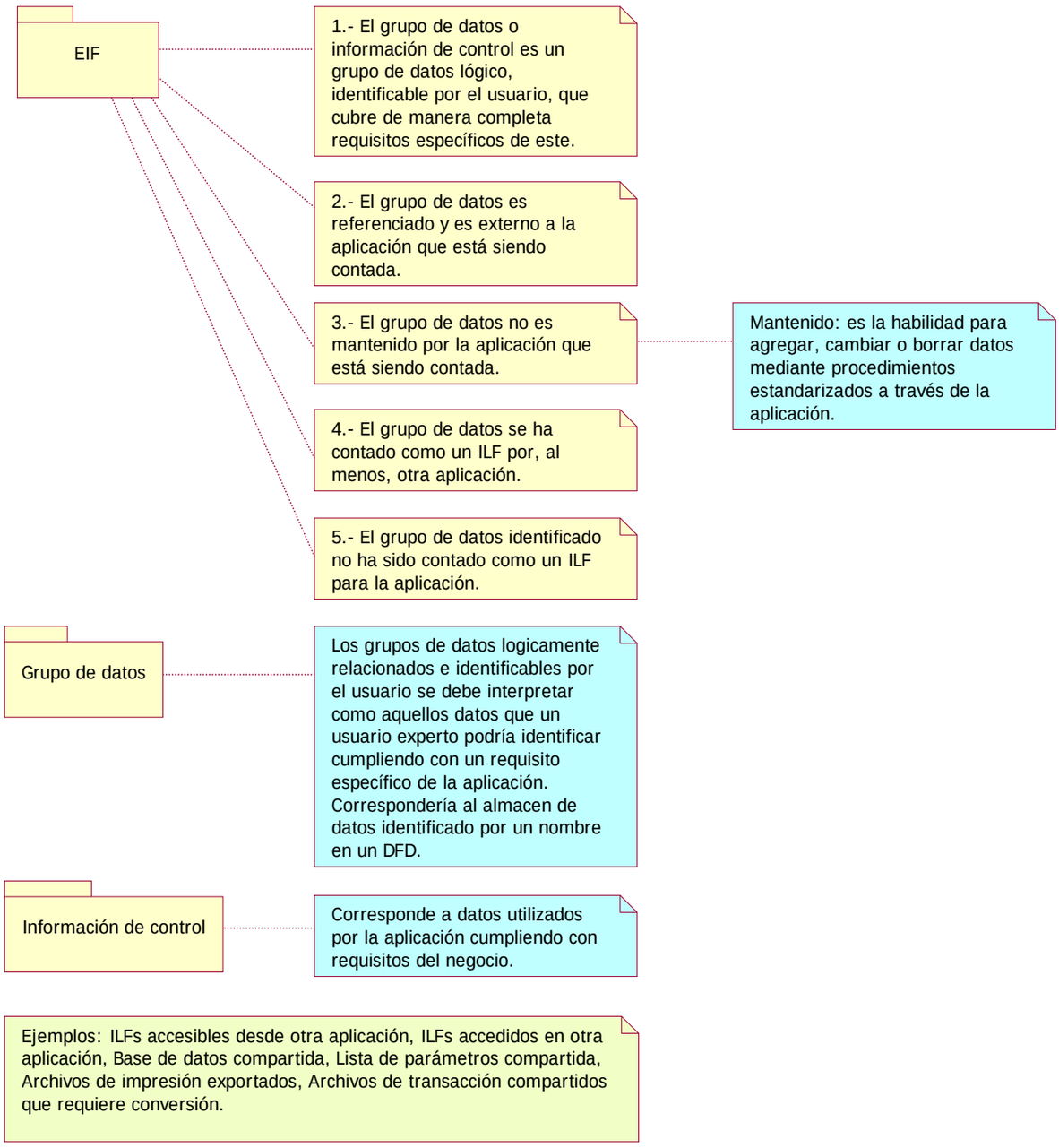
8.4.2.4.1.2 DETs



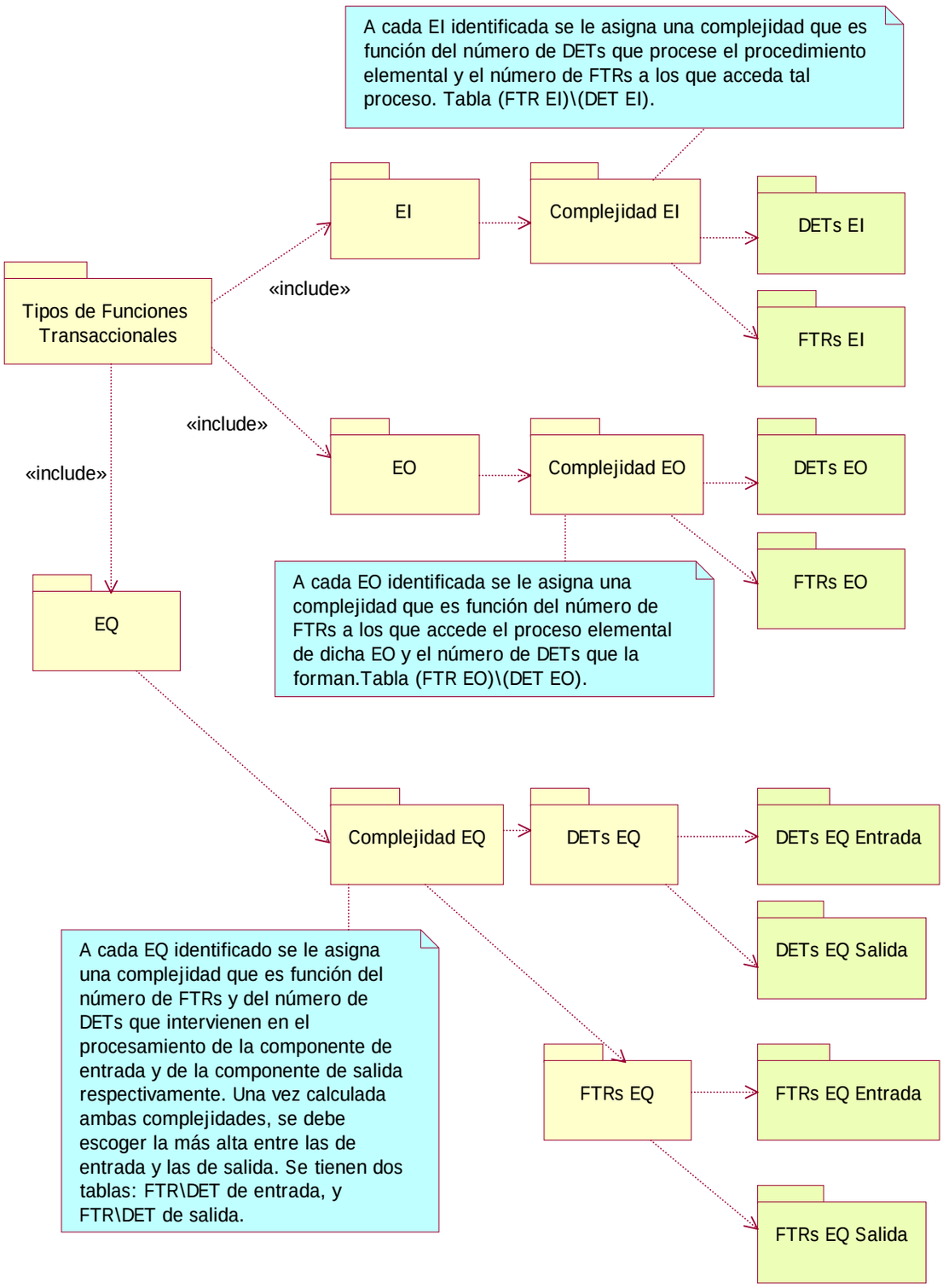
8.4.2.4.1.3 ILFs


ILF
1.- El grupo de datos o información de control es un grupo de datos lógico, identificable por el usuario, que cubre de manera completa requisitos específicos de este.
2.- El grupo de datos es mantenido dentro de los límites de la aplicación.
3.- El grupo de datos es mantenido o modificado por medio de un proceso elemental de la aplicación.
4.- El grupo de datos identificado no ha sido contado como un EIF de la aplicación.
Mantenido: es la habilidad para agregar, cambiar o borrar datos mediante procedimientos estandarizados a través de la aplicación.
Grupo de datos
Los grupos de datos logicamente relacionados e identificables por el usuario se debe interpretar como aquellos datos que un usuario experto podría identificar cumpliendo con un requisito específico de la aplicación. Correspondería al almacen de datos identificado por un nombre en un DFD.
Información de control
Corresponde a datos utilizados por la aplicación cumpliendo con requisitos del negocio.
Ejemplos: Archivos Maestros (1 por cada grupo de claves), Aplicaciones de seguridad de datos, datos de auditoría actualizados por la aplicación, mensajes HELP actualizados por la aplicación, mensajes de error actualizados por la aplicación, datos de respaldo (si el usuario lo requiere), ILFs mantenidos por más de una aplicación. Bases de datos: 1 por vista lógica o camino de acceso. Tablas mantenidas por los usuarios: estados, tarifas, mensajes, etc. Archivos de procesamiento batch. Índices de referencias cruzadas.

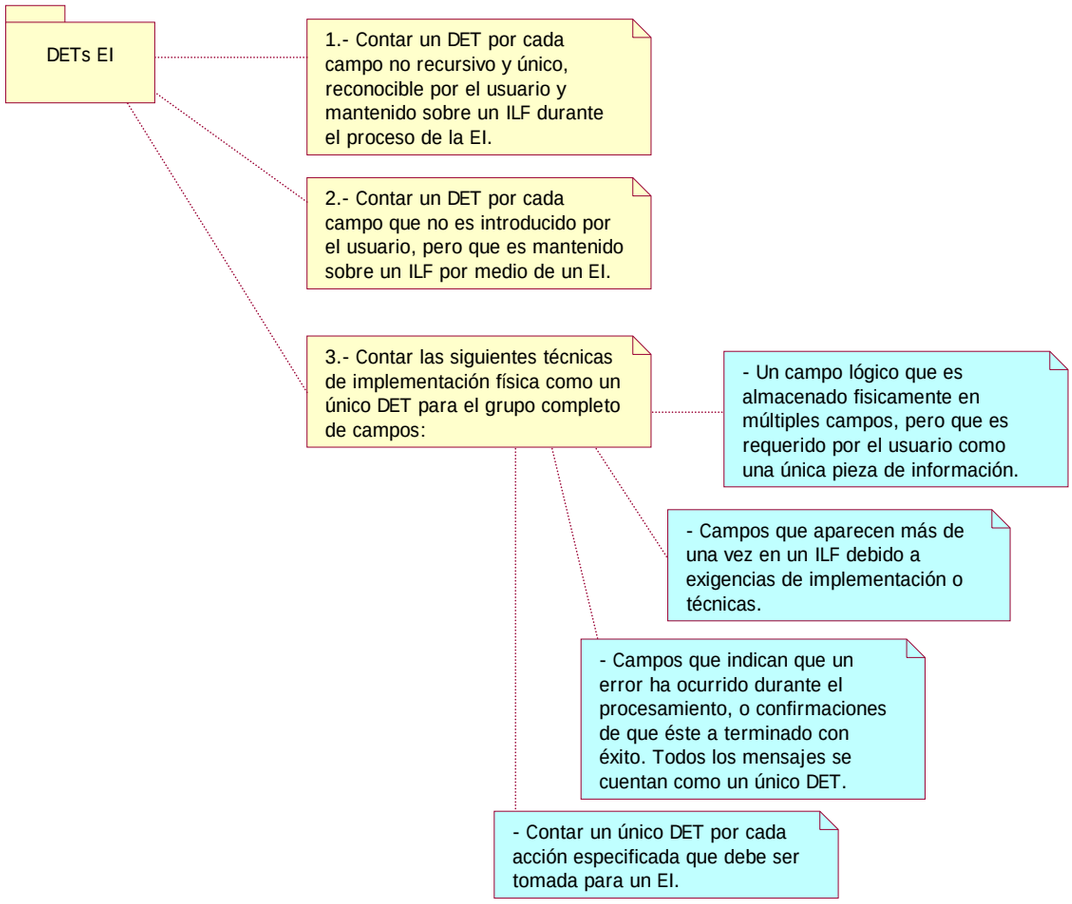
8.4.2.4.1.4 EIFs


EIF
1.- El grupo de datos o información de control es un grupo de datos lógico, identificable por el usuario, que cubre de manera completa requisitos específicos de este.
2.- El grupo de datos es referenciado y es externo a la aplicación que está siendo contada.
3.- El grupo de datos no es mantenido por la aplicación que está siendo contada.
Mantenido: es la habilidad para agregar, cambiar o borrar datos mediante procedimientos estandarizados a través de la aplicación.
4.- El grupo de datos se ha contado como un ILF por, al menos, otra aplicación.
5.- El grupo de datos identificado no ha sido contado como un ILF para la aplicación.
Grupo de datos
Los grupos de datos logicamente relacionados e identificables por el usuario se debe interpretar como aquellos datos que un usuario experto podría identificar cumpliendo con un requisito específico de la aplicación. Correspondería al almacen de datos identificado por un nombre en un DFD.
Información de control
Corresponde a datos utilizados por la aplicación cumpliendo con requisitos del negocio.
Ejemplos: ILFs accesibles desde otra aplicación, ILFs accedidos en otra aplicación, Base de datos compartida, Lista de parámetros compartida, Archivos de impresión exportados, Archivos de transacción compartidos que requiere conversión.

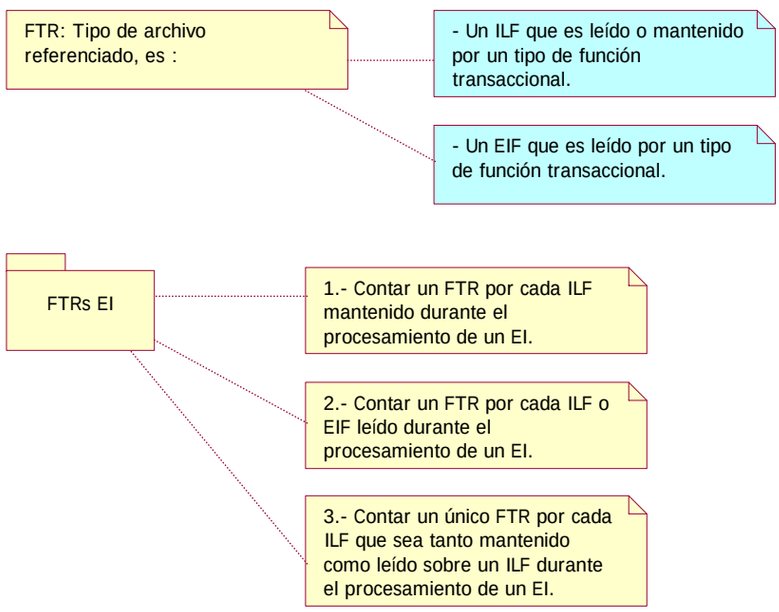
8.4.2.4.2 Tipos de Funciones Transaccionales



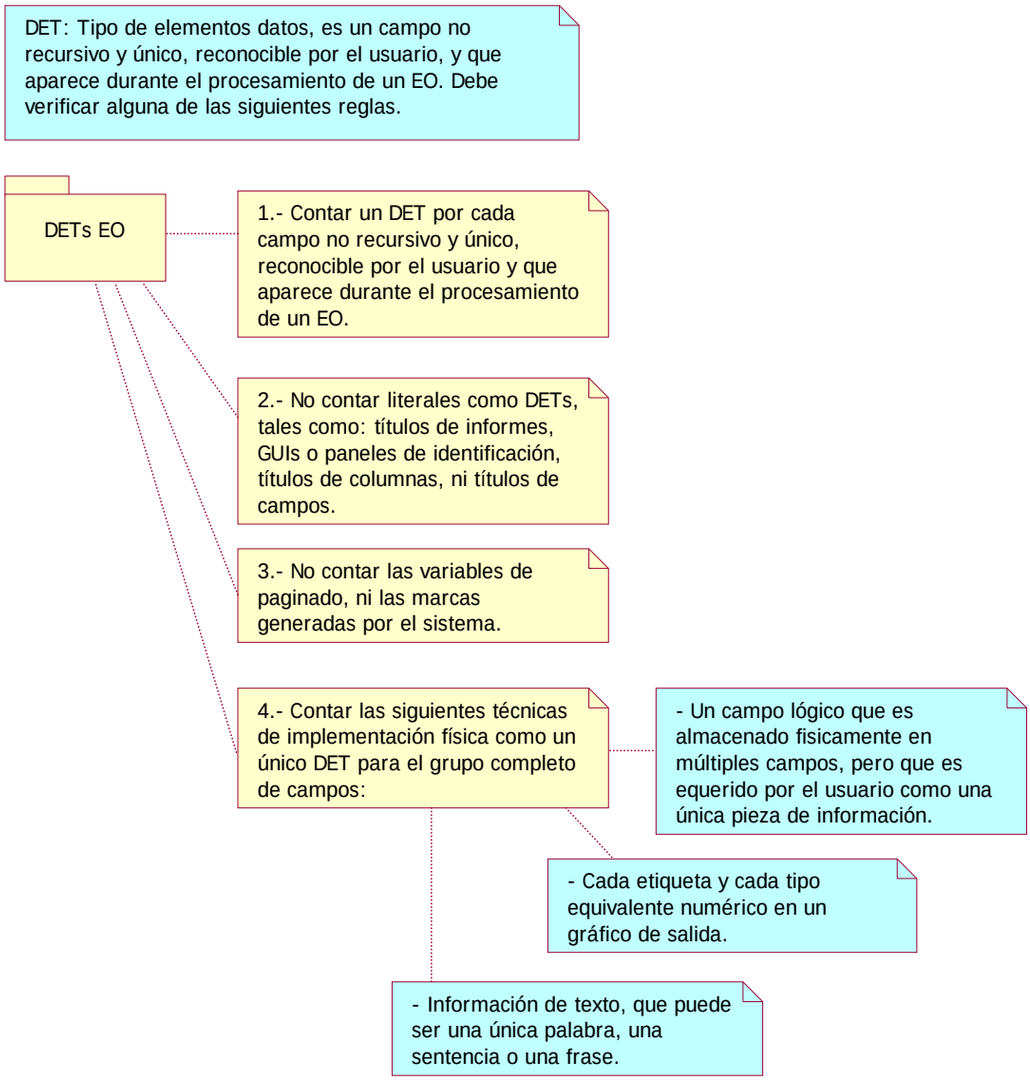
8.4.2.4.2.1 DETs EI



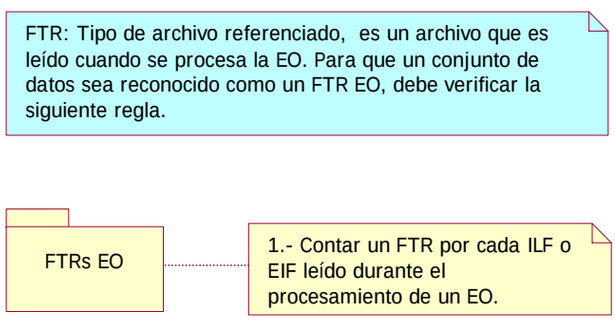
8.4.2.4.2.2 FTRs EI



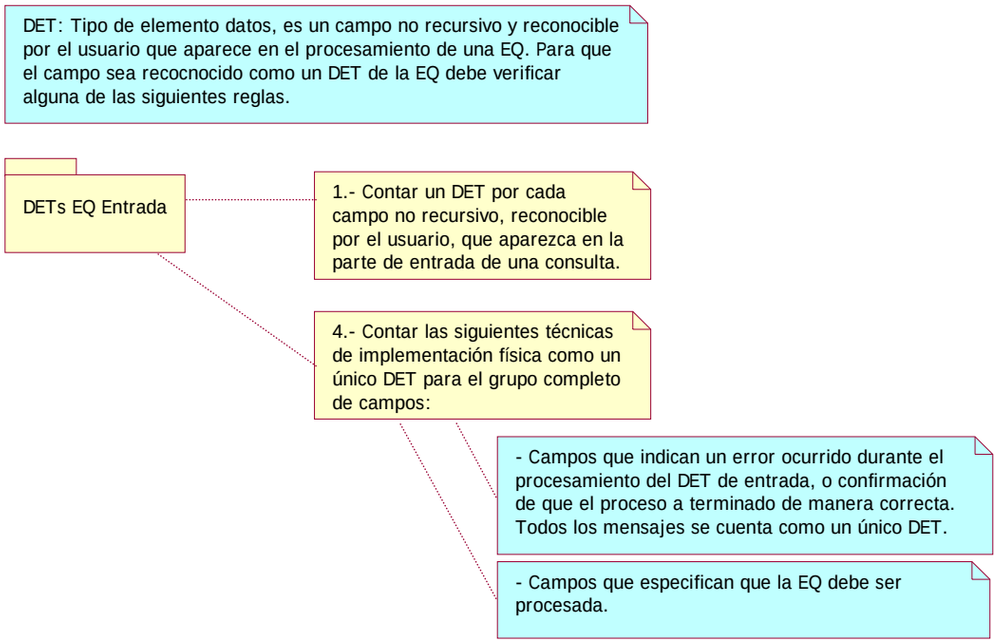
8.4.2.4.2.3 DETs EO



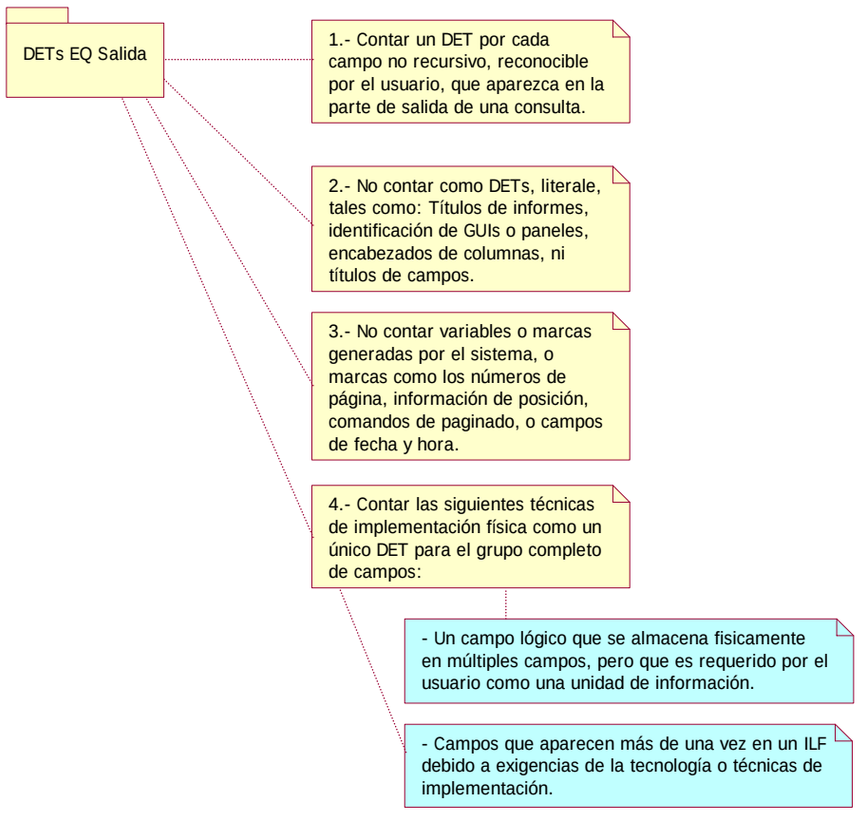
8.4.2.4.2.4 FTRs EO



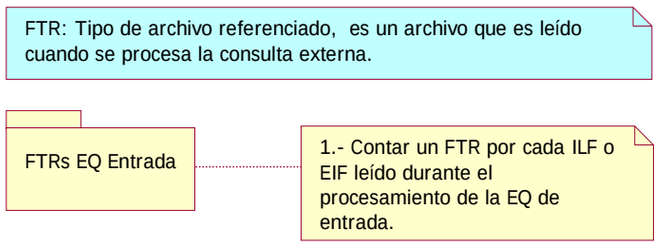
8.4.2.4.2.5 DETs EQ Entrada



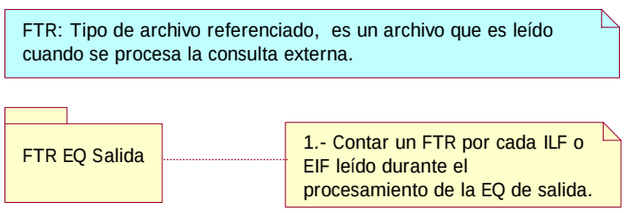
8.4.2.4.2.6 DETs EQ Salida



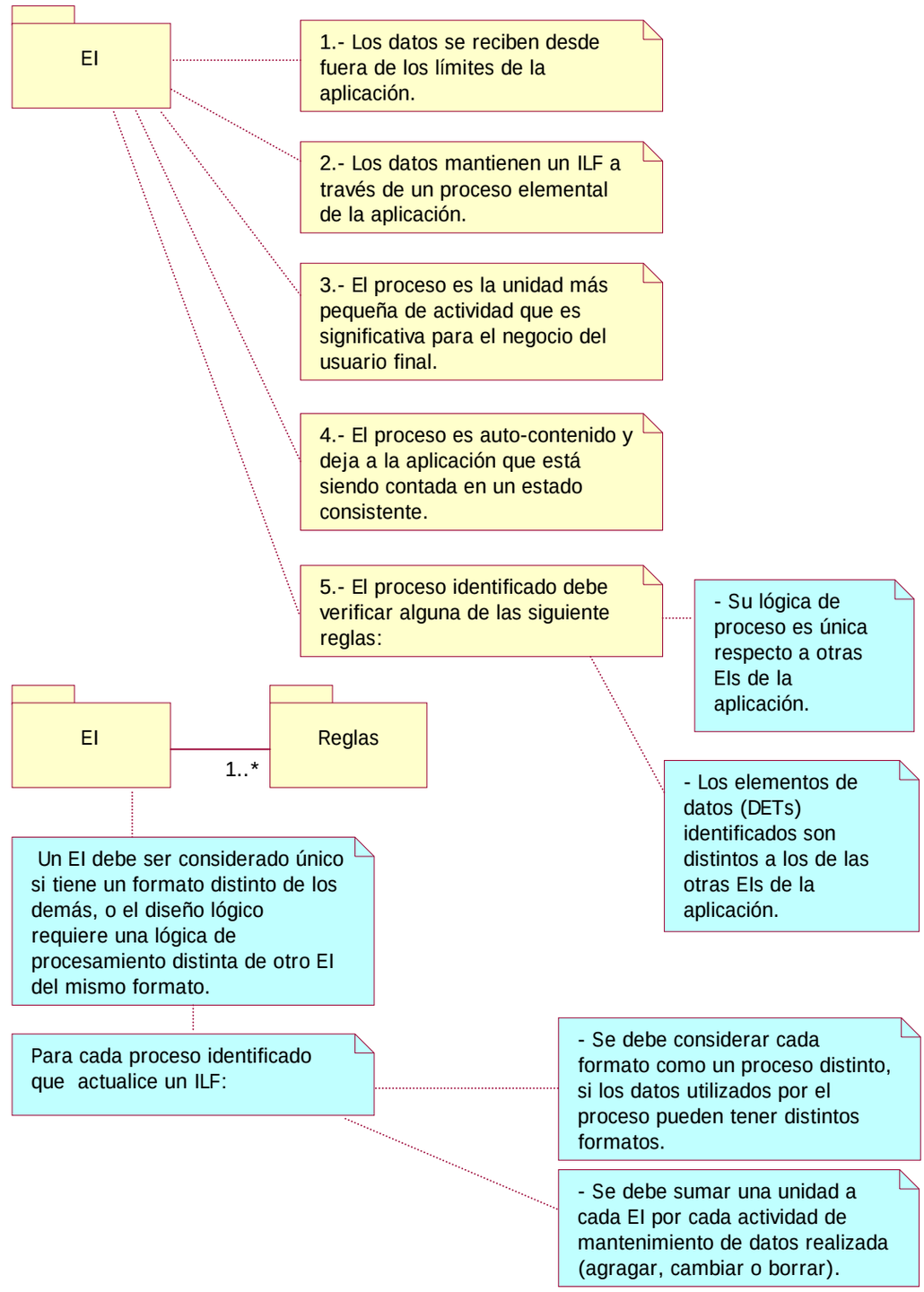
8.4.2.4.2.7 FTRs EQ Entrada



8.4.2.4.2.8 FTRs EQ Salida

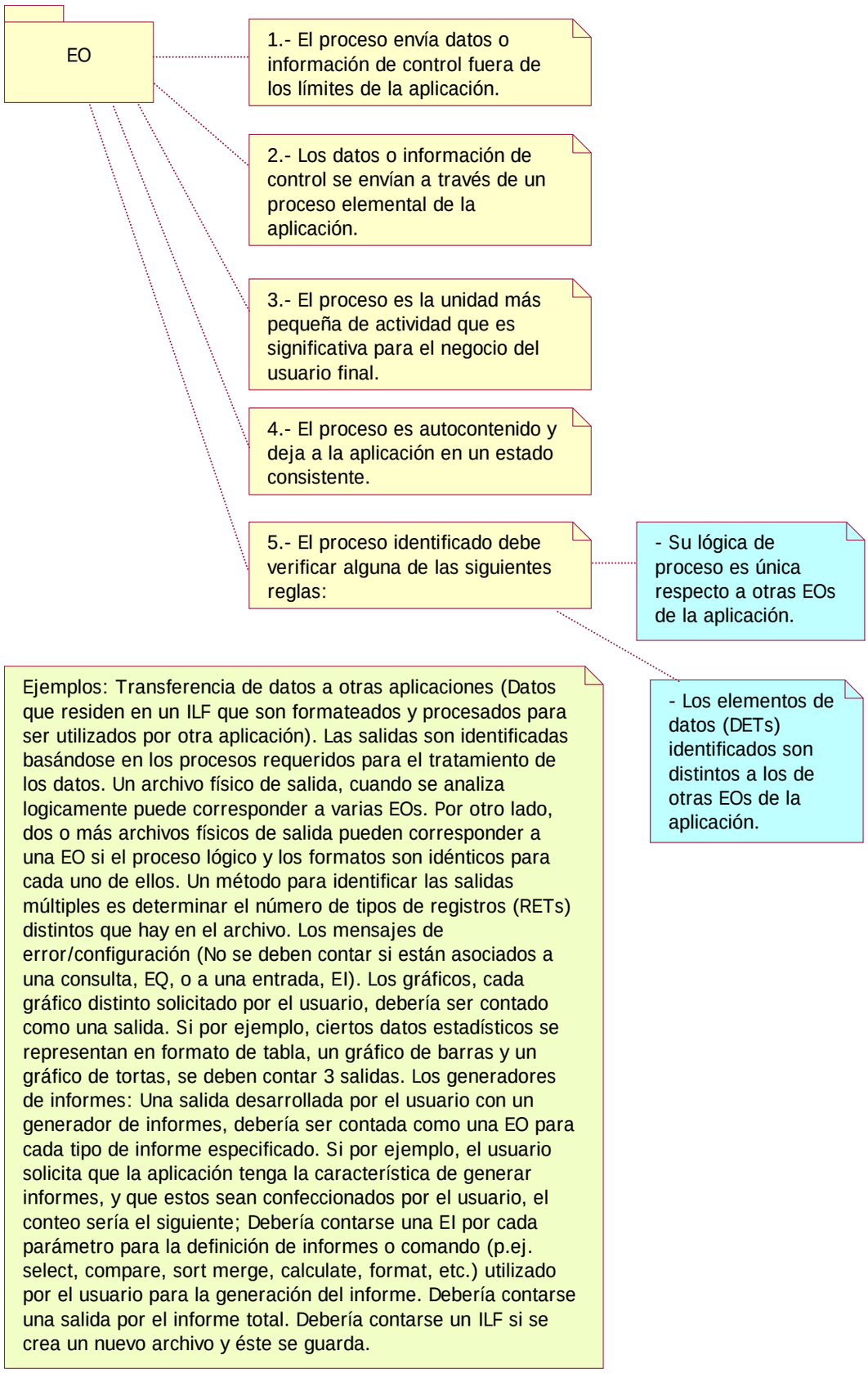


8.4.2.4.2.9 EIs

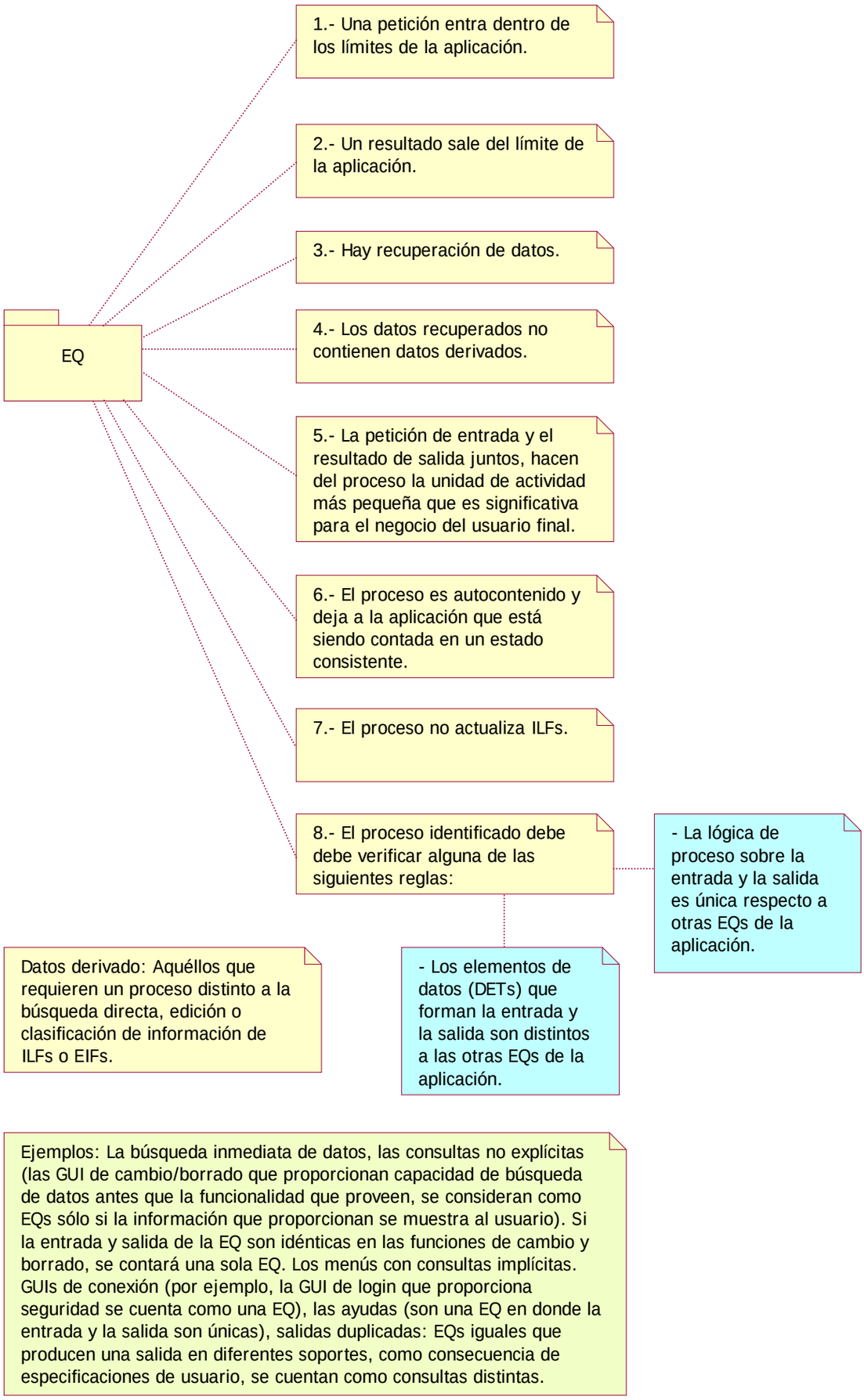


Ejemplos: Las transacciones (Datos introducidos para mantener ILFs), las GUIs de entradas (Se debe agragar una unidad a EI por cada función que mantiene un ILF, por ejemplo; si los datos en una GUI pueden agregar, cambiar y borrar información en un ILF, se deberían contar 3 EIs), Las entradas por lote (Por cada proceso único que mantiene un ILF, se debe sumar un EI por cada actividad que agregue, cambie o borre datos), un archivo físico de entrada (cuando se analiza logicamente, corresponde a una o varias EIs, dependiendo de los tipos de registros contenidos y del proceso requerido para su tratamiento. Es así como, dos o más archivos físicos de entrada pueden corresponder a un EI si el proceso lógico y el formato son idénticos para cada uno de los archivos), Las EIs duplicadas si distintos procesos de entrada solicitados expresamente por el usuario duplican una EI, son contados en forma independiente cada uno.

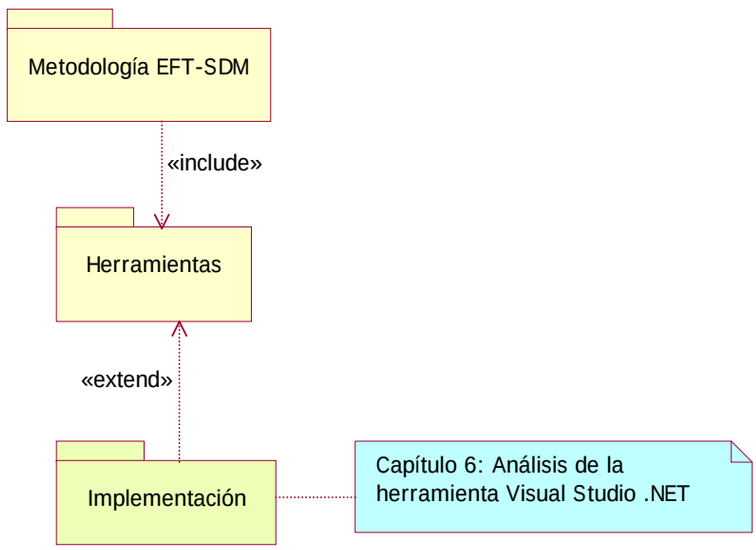
8.4.2.4.2.10 EOs


EO
1.- El proceso envía datos o información de control fuera de los límites de la aplicación.
2.- Los datos o información de control se envían a través de un proceso elemental de la aplicación.
3.- El proceso es la unidad más pequeña de actividad que es significativa para el negocio del usuario final.
4.- El proceso es autocontenido y deja a la aplicación en un estado consistente.
5.- El proceso identificado debe verificar alguna de las siguientes reglas:
- Su lógica de proceso es única respecto a otras EOs de la aplicación.
- Los elementos de datos (DETs) identificados son distintos a los de otras EOs de la aplicación.
Ejemplos: Transferencia de datos a otras aplicaciones (Datos que residen en un ILF que son formateados y procesados para ser utilizados por otra aplicación). Las salidas son identificadas basándose en los procesos requeridos para el tratamiento de los datos. Un archivo físico de salida, cuando se analiza logicamente puede corresponder a varias EOs. Por otro lado, dos o más archivos físicos de salida pueden corresponder a una EO si el proceso lógico y los formatos son idénticos para cada uno de ellos. Un método para identificar las salidas múltiples es determinar el número de tipos de registros (RETs) distintos que hay en el archivo. Los mensajes de error/configuración (No se deben contar si están asociados a una consulta, EQ, o a una entrada, EI). Los gráficos, cada gráfico distinto solicitado por el usuario, debería ser contado como una salida. Si por ejemplo, ciertos datos estadísticos se representan en formato de tabla, un gráfico de barras y un gráfico de tortas, se deben contar 3 salidas. Los generadores de informes: Una salida desarrollada por el usuario con un generador de informes, debería ser contada como una EO para cada tipo de informe especificado. Si por ejemplo, el usuario solicita que la aplicación tenga la característica de generar informes, y que estos sean confeccionados por el usuario, el conteo sería el siguiente; Debería contarse una EI por cada parámetro para la definición de informes o comando (p.ej. select, compare, sort merge, calculate, format, etc.) utilizado por el usuario para la generación del informe. Debería contarse una salida por el informe total. Debería contarse un ILF si se crea un nuevo archivo y éste se guarda.

8.4.2.4.2.11 EQs


1.- Una petición entra dentro de los límites de la aplicación.
2.- Un resultado sale del límite de la aplicación.
3.- Hay recuperación de datos.
EQ
4.- Los datos recuperados no contienen datos derivados.
5.- La petición de entrada y el resultado de salida juntos, hacen del proceso la unidad de actividad más pequeña que es significativa para el negocio del usuario final.
6.- El proceso es autocontenido y deja a la aplicación que está siendo contada en un estado consistente.
7.- El proceso no actualiza ILFs.
8.- El proceso identificado debe debe verificar alguna de las siguientes reglas:
- La lógica de proceso sobre la entrada y la salida es única respecto a otras EQs de la aplicación.
Datos derivado: Aquéllos que requieren un proceso distinto a la búsqueda directa, edición o clasificación de información de ILFs o EIFs.
- Los elementos de datos (DETs) que forman la entrada y la salida son distintos a las otras EQs de la aplicación.
Ejemplos: La búsqueda inmediata de datos, las consultas no explícitas (las GUI de cambio/borrado que proporcionan capacidad de búsqueda de datos antes que la funcionalidad que proveen, se consideran como EQs sólo si la información que proporcionan se muestra al usuario). Si la entrada y salida de la EQ son idénticas en las funciones de cambio y borrado, se contará una sola EQ. Los menús con consultas implícitas. GUIs de conexión (por ejemplo, la GUI de login que proporciona seguridad se cuenta como una EQ), las ayudas (son una EQ en donde la entrada y la salida son únicas), salidas duplicadas: EQs iguales que producen una salida en diferentes soportes, como consecuencia de especificaciones de usuario, se cuentan como consultas distintas.

8.4.3 Herramientas de Implementación



Capítulo 9 : Mejoras y palabras finales

9.1 Mejoras

Las mejoras más importantes que podrían ser realizadas al presente trabajo consisten en la implementación de las aplicaciones componentes enunciadas en la metodología de desarrollo de aplicaciones sobre la plataforma Microsoft .NET Framework SDK. Entre ellas se incluyen:

1. Refinar e implementar el proceso EFT-UP en base a la experiencia adquirida en el desarrollo de proyectos software.
2. Refinar e implementar el modelo de estimación de costos propuesto, EFT-SCM, cuyo diseño e interfaces gráficas de usuario han sido esbozadas y se exponen a continuación.

9.1.1 Diseño tentativo de la implementación de EFT-SDM

9.1.1.1 Interfaces gráficas de usuario

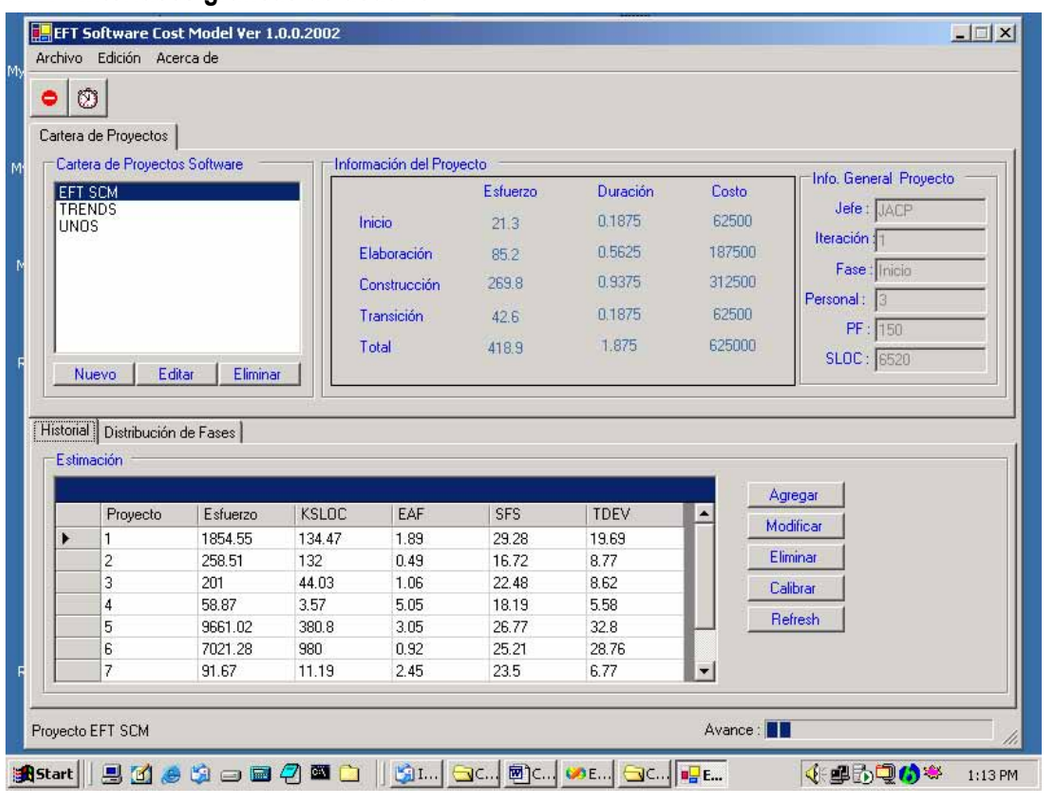


Figura 9.1: Diseño tentativo de GUIs de la implementación de EFT-SCM propuesta.

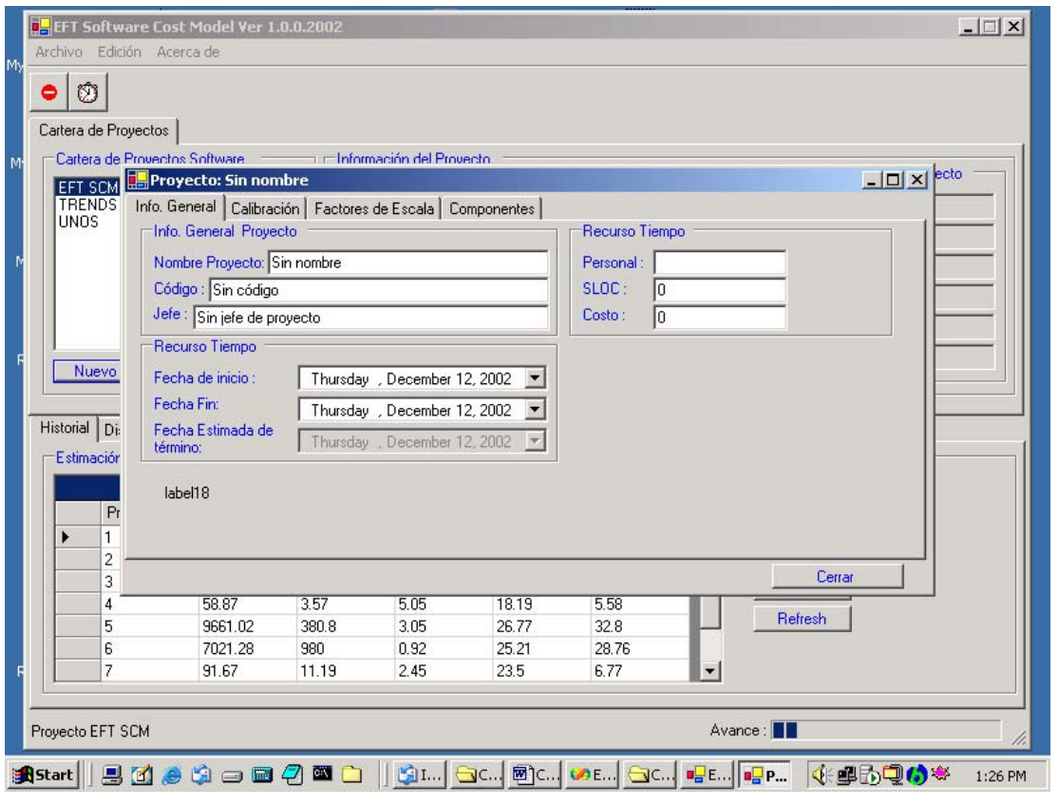


Figura 9.2: Diseño tentativo de GUIs de la implementación de EFT-SCM propuesta.

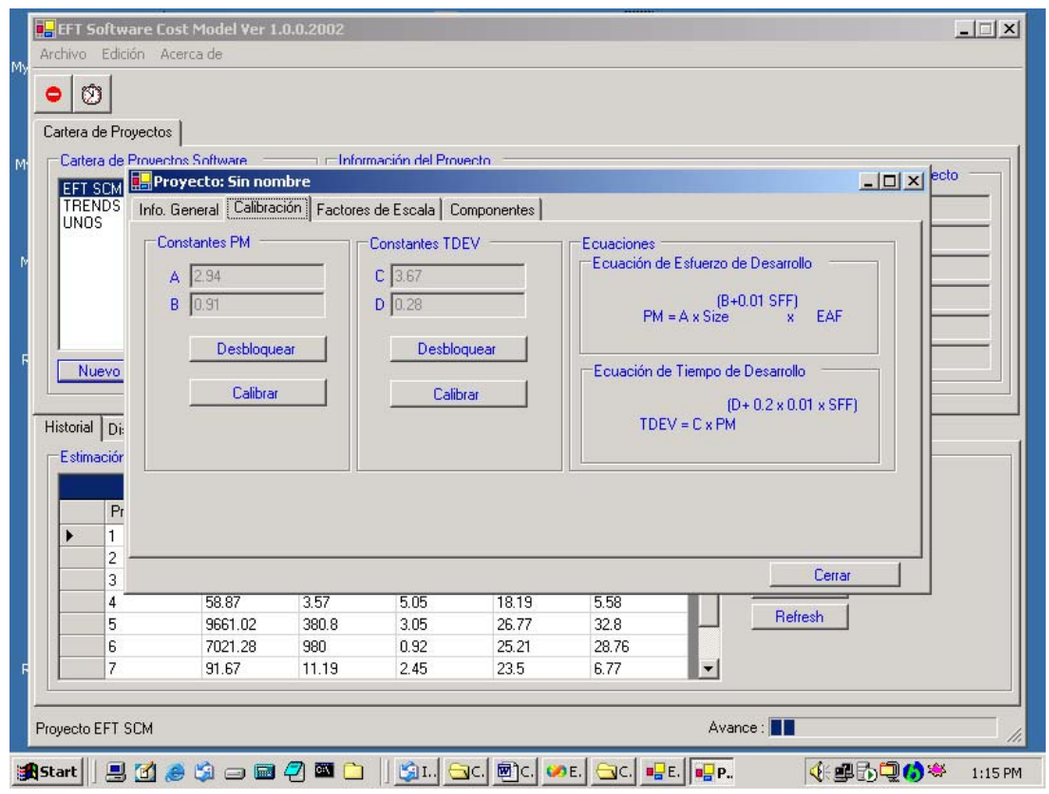


Figura 9.3: Diseño tentativo de GUIs de la implementación de EFT-SCM propuesta.

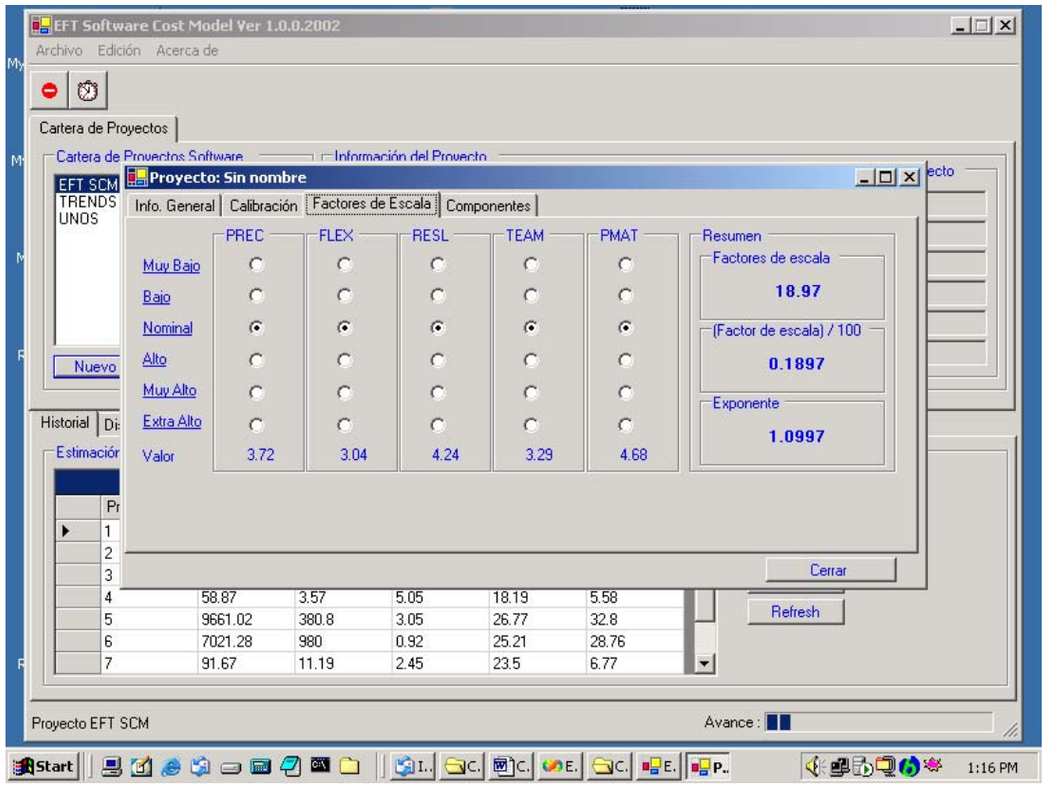


Figura 9.4: Diseño tentativo de GUIs de la implementación de EFT-SCM propuesta.

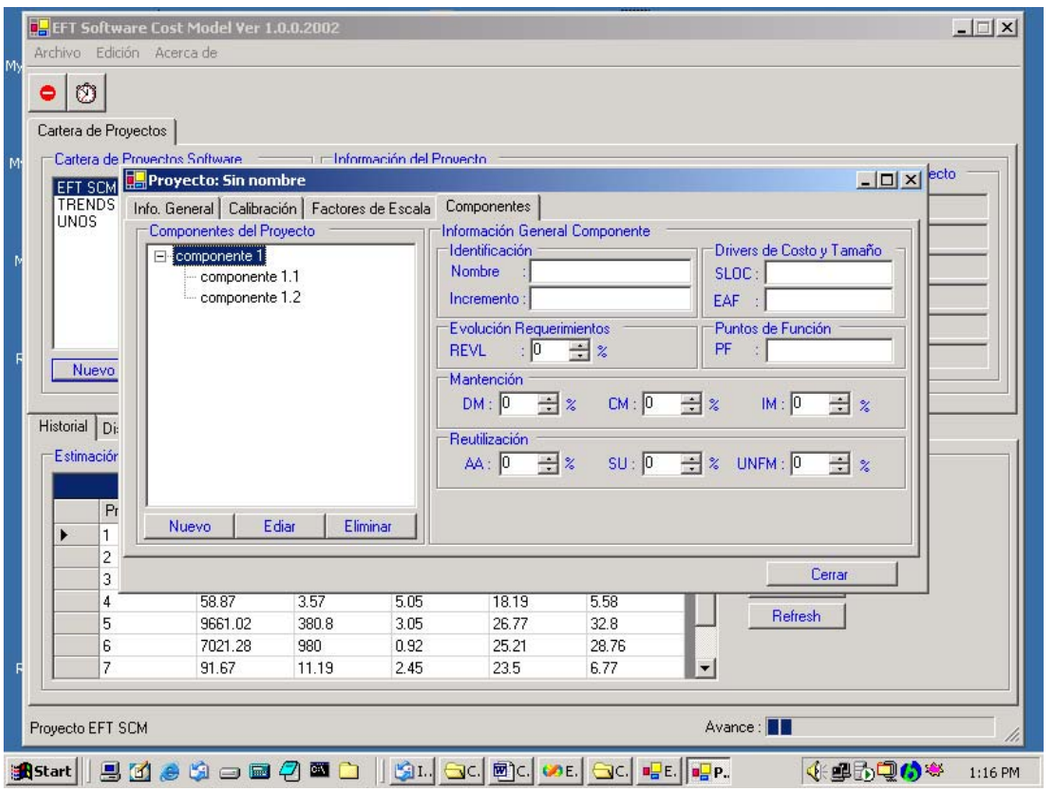


Figura 9.5: Diseño tentativo de GUIs de la implementación de EFT-SCM propuesta.

9.1.1.2 Diseño preliminar de implementación de EFT-SCM

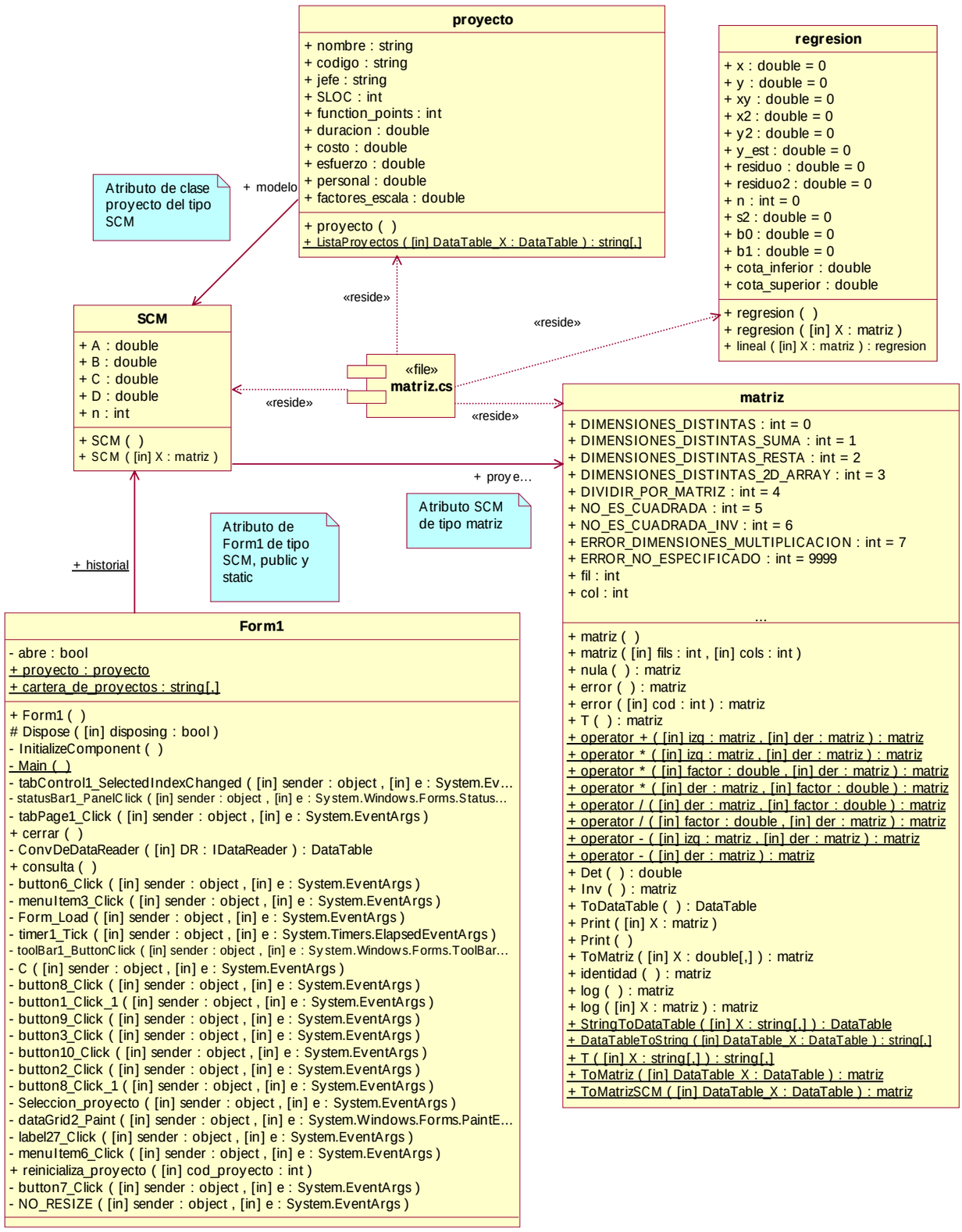


Figura 9.6: Diseño tentativo producto de la sincronización de código de Rational XDE.

9.1.2 Mejoras al prototipo de recaudación electrónica a minoristas

La principales mejoras al prototipo REM consisten en implementar la seguridad punta a punta de las transacciones efectuadas a través de éste. Esto se justifica demostrando por ejemplo cómo la confidencialidad de las transacciones puede ser violada a través de un programa analizador de redes o **packet sniffing,** tal como **Ethereal 0.9.7**, ver figura 9.7.

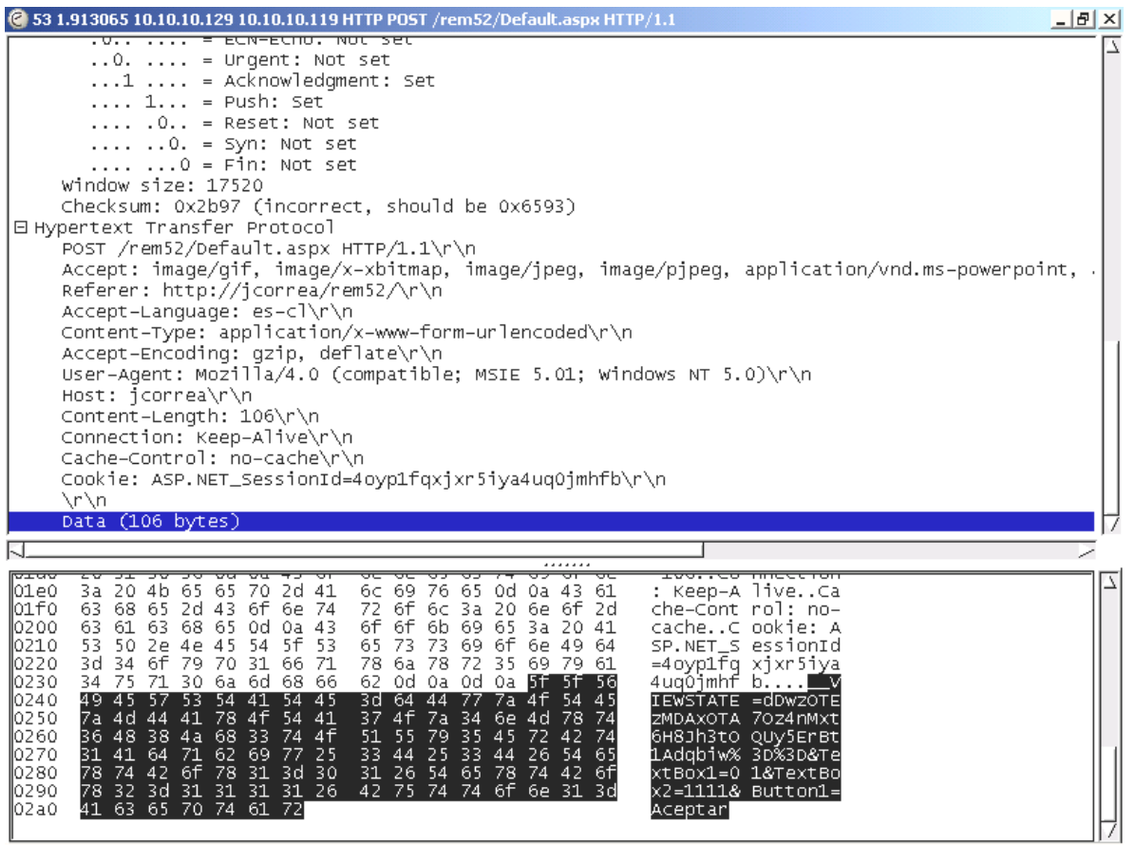


Figura 9.7: Ethereal 0.9.7 es capaz de ver la información de acceso de usuario.

En la Figura 9.7, es posible distinguir en las últimas tres líneas seleccionadas de la columna derecha, *TextBox1=01&TextBox2=1111&Button1=Aceptar*, que corresponde al código de usuario y a la contraseña respectivamente.

La solución propuesta es implementar un canal seguro usando SSL de punta a punta en el sistema REM. Lo anterior es posible gracias a que el Protocolo de Aplicaciones Inalámbrico, WAP, posee una capa llamada WTLS que permite implementar SSL sobre WAP. Los esquemas de seguridad disponibles se pueden encontrar en **[28]**.

Implementando el canal seguro a traves de SSL, el ataque realizado para probar la seguridad de las transacciones se vería como en la figura 9.8.

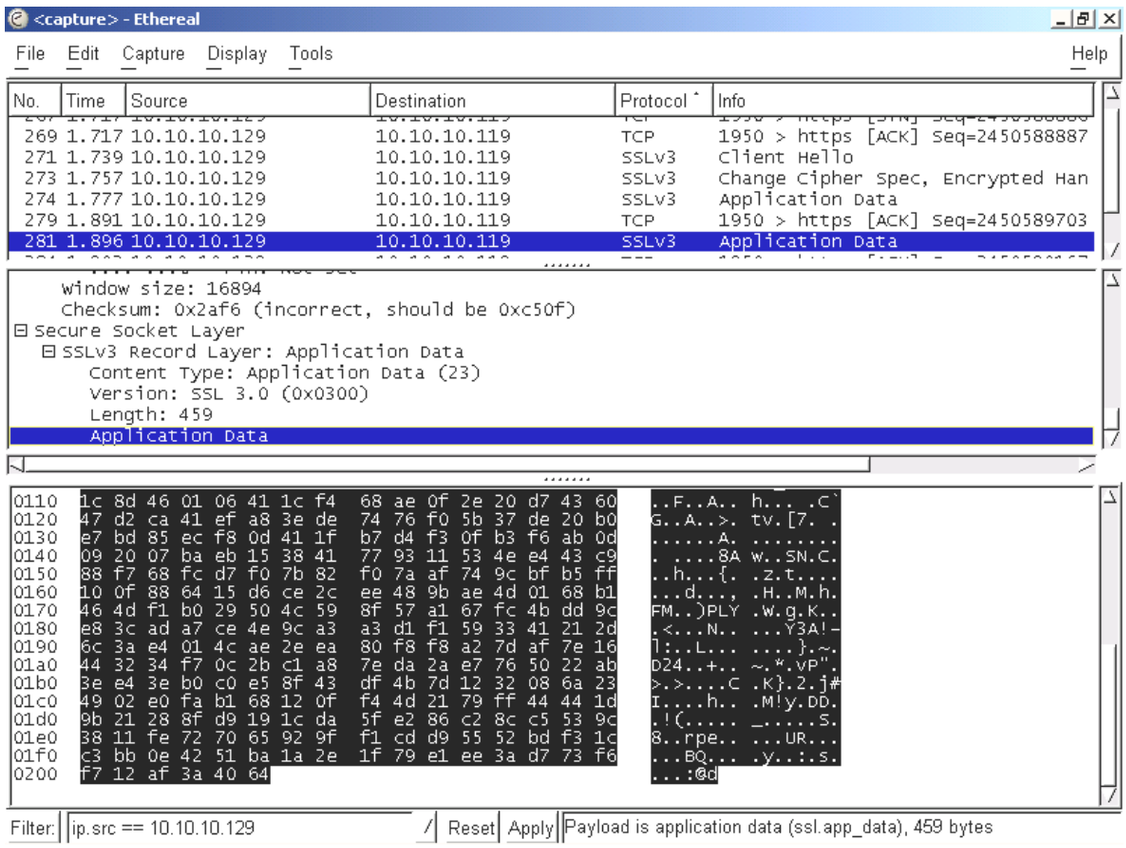


Figura 9.8: No es posible identificar la información confidencial del usuario, ésta viaja encriptada.

Las otras mejoras se encuentran dentro del ámbito de los requerimientos no funcionales, tales como arreglos estéticos de las interfaces gráficas de usuario.

9.2 Conclusión.

La instauración de una metodología de desarrollo de aplicaciones en las áreas de desarrollo de software se puede considerar crítica debido al poco conocimiento y difusión del tema y la diversidad de soluciones disponibles en el mercado. Soluciones que quizás no se ajusten al modelo de negocios de cualquier empresa. En el presente documento se ha visto que toda metodología esta constituida al menos por tres puntos importantes, estos son : La notación, el proceso y las herramientas. Sin estos, es imposible conceptualizar, o mucho menos, implementar una metodología de desarrollo de aplicaciones. Dentro de la notación, se han explorado diferentes aspectos del lenguaje de modelamiento unificado (UML), sus tipos de diagramas, y sus usos según sea el caso; además, en capítulos posteriores, se han integrado estos conocimientos a potentes herramientas de desarrollo de software asistido por computadora (CASE), como es el caso de Rational XDE Profesional y Visual Studio .NET sobre la plataforma Microsoft .NET Framework. Por otra parte, se ha estudiado en detalle el proceso RUP (Rational Unified Process), el cual provee de las directrices para la confección del nuevo proceso que acompañará a la metodología de desarrollo de software. Junto a este proceso, denominado EFT-UP, se desprende el uso de técnicas avanzadas de estimación de costos. Esto mediante herramientas tales como COCOMO II o Costar, que brindan información de vital importancia a la administración del proyectos software.

Se a tratado el desarrollo de una aplicación prototipo llamada REM (Recaudación electrónica a minoristas). Esta ha sido esbozada mediante la metodología de desarrollo de aplicaciones sobre la plataforma .NET Framework. Se ha intentado plasmar la experiencia de desarrollo de las versiones DEMO del prototipo de Recaudación Electrónica a Minoristas, construidas desde dos enfoques distintos. Se ha podido comprobar cómo un proceso de desarrollo orientado a los casos de uso, puede mejorar aspectos de la administración de proyectos software, mediante estimaciones basadas en el diseño preliminar del sistema. Por último, se muestra el diseño de la metodología de desarrollo de aplicaciones sobre la plataforma

Microsoft .NET Framework, La cual queda disponible para su posterior implementación según se requiera.

9.3 Palabras finales

Esta tesis, tal cual es, se presenta a vosotros examinada prolijamente, ha sido discutida con mis maestros y objeto de modificaciones en lo pertinente. No presumo ofrecer bajo estos respectos una obra perfecta; ninguna tal ha salido hasta ahora de las manos del hombre. Pero no temo aventurar mi juicio anunciando que la adopción de los postulados y principios expuestos con exhaustividad en este proyecto permitirá desvanecer mucha parte de las dificultades que ahora embarazan el desarrollo del mundo informático.

Otras materias complementan, sin duda, esta tesis, como por ejemplo, el modelamiento en UML; el empleo de los patrones de diseño; el uso de herramientas de estimación de costo y modelamiento visual, y todo lo relativo a la seguridad de las transacciones en aplicaciones inalámbricas, temas todos estos, que en esta oportunidad no ha sido posible abordar.

Así, entonces, la obra debe entenderse terminada en una primera gran etapa, su desarrollo posterior exigirá nuevos estudios, investigaciones y métodos. Con todo, estoy convencido que éste es el momento preciso de citar a Blaise Pascal: “Ya se han escrito todas las buenas máximas, sólo falta ponerlas en práctica”.

Capítulo 10 : Bibliografía

10.1 Libros y Papers

[1] Charte, Francisco, (2000) *C# y Microsoft .NET, EDICIONES ANAYA MULTIMEDIA.*

[2] FOWLER, MARTIN; SCOTT, KENDALL, (2000) UML DISTILLED SECOND EDITION , ADDISON-WESLEY

[3] Ana Ma Moreno S., (1999) , Estimación de Proyectos Software, Universidad Politécnica de Madrid.

[4] Letelier T., Patricio, (2002) , Desarrollo de Software Orientado a objeto usando UML, Universidad Politécnica de Valencia, Departamento Sistemas Informáticos y Computación.

[5] Rational Software Corporation White paper, (1998) , Rational Unified Process, Best Practices for software Development Teams, Rational Software.

[6] Sommerville, Ian, (2002), Ingeniería de Software, 6ta Edición, Pearson Educación.

[7] Larry O'brien & Bruce Eckel, (2002), Thinking in C#, Revisión 12, Prentice Hall.

[8] Rational Software Corporation, (2002), Rational XDE Professional v2002: Microsoft.NET Edition Evaluetors Guide, Rational Software.

[9] Rational Software Corporation, (2002), Rational XDE Professional v2002: Rational XDE Help Files, Rational Software.

[10] J.B. Dreger, (1989), "Function Point Analysis", Prentice-Hall.

[11] David Longstreet, (1989), "Function Points Analysis Training Course ", Longstreet Consulting Inc.

[12] Bruce Eckel, (2001), "Thinking in Patterns Problem-Solving Techniques using Java", MindView Inc.

[13] Barry Boehm, (2000), "COCOMO II Model Definition Manual Version 1.4", University of Southern California.

[14] Barry Boehm, (2000), "Calibrating the COCOMO II Post-Architecture Model", University of Southern California.

[15] Bradford K. Clark, (1997), "The effects of software process maturity on software development effort", University of Southern California.

[16] Sunita Chulani, (2000), "COCOMO II Calibration", IBM Research.

[17] Tim Menzies - Erik Sinsel, (2000), "Practical Large Scale What - if Queries: Case studies with with software risk assessment", NASA / West Virginia University.

[18] Barry Bohem et al 1998, (1998), "Calibration Results of COCOMO II.1997", University of Southern California - Center for Software Engineering.

[19] George C. Canavos, (1990), "Probabilidad y estadística, aplicaciones y métodos",McGraw Hill.

[20] B. Boehm, (1981), "Software Engineering Economics", Prentice Hall.

[21] R. Park, (1992),"Software Size Measurement: A Framework for Counting Source Statements", Software Engineering Institute.

[22] M. Ruhl y M. Gunn, (1991), "Software Reengineering: A Case Study and Lessons Learned", NIST Special Publication, Washington, DC.

[23] Banker R. D., Chang H., Kemerer C., (1994), "Evidence on Economies of Scale in Software Development", Information and Software Technology.

[24] Wap Forum, (2002), "Wireless Application Protocol WAP 2.0 Technical Whitepaper", Wap Forum.

[25] Jim Conallen, (1999), "Modeling Web Application Architectures with UML", Rational Software.

[26] James Rumbaugh, Ivar Jacobson, Grady Booch, (2000), "El lenguaje unificado de modelado. Manual de referencia", Pearson Educación S.A., Madrid.

[27] United Kingdom Software Metrics Association, (1998), "MK II Function Point Analysis; Counting Practices Manual, version 1.3.1", UKSMA, United Kingdom.

[28] Castro, J.C., Forné, J., (2000), "Acceso Seguro a Internet Móvil", Deprtamento de Matemática Aplicada y Telemática, Universidad Politécnica de Cataluña.

10.2 Direcciones en Internet

- Microsoft Solution Framework (MSF)

http://www.microsoft.com/framework/

- C# Programmer's Reference, MSDN:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csref/html/vcoricprogrammersreference.asp

- C# Programmer's Reference ; Documentación XML en MSDN

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csref/html/vcorixmldocumentation.asp

- Methods, Tools & Techniques:

http://www.itmweb.com/methodology/

- UML Resource Center, Rational

http://www.rational.com/uml/resources/documentation/index.jsp

- Web Services Description language

http://www.w3.org/TR/wsdl

- C# Overview

http://www.beust.com/cedric/csharp.html

- C# station

http://www.csharp-station.com/default.aspx

- Software Engineering Baselines

http://www.dacs.dtic.mil/techs/baselines/toc.html

- Software Metrics

http://www.literateprogramming.com/fmetrics.html

- Departamento Sistemas Informáticos y Computación, Universidad Politécnica de Valencia

http://www.dsic.upv.es/~uml/

Capítulo 11 : Glosario de Términos

11.1 Glosario de términos

Abstracto
Un elemento generalizable es abstracto si no puede ser directamente instanciado.

Abstracción (acto)
El término **abstracción** se refiere al acto de buscar similitudes a través de un conjunto de cosas centrándose en las características comunes que hacen a esas cosas diferentes de otra clase de cosas.

Acceso
El término **acceso** se refiere a una dependencia estereotipada dentro de la cual el paquete fuente tiene el derecho a referenciar el contenido del paquete destino.

Acción
Una **acción** es un cálculo atómico ejecutable el cual resulta en un cambio de estado o el retorno de un valor.

Activación
Una **activación** es una ejecución de una operación.
Una activación se representa en un diagrama de secuencia por un foco de control.

Actividad (UML)
Una actividad es una ejecución ininterrumpible dentro de una máquina de estado.

Actividad (Unified Process)
Dentro del Proceso Unificado y el RUP, una actividad es una unidad de trabajo tangible con una responsabilidad bien definida la cual realiza un trabajador.

Actor
El término **actor** es una palabra clave la cual se refiere a un conjunto de roles coherentes los cuales interpreta una entidad (humano o no humano) fuera del sistema que se está modelando cuando interactúa con uno o más casos de uso.

Arquitecto
El arquitecto es el trabajador que guía y coordina las actividades técnicas, y la producción de los elementos asociados a lo largo del proyecto. También establece la estructura global para las 5 vistas que comprometen la arquitectura.

Clase
Es una descripción de un conjunto de objetos, los cuales comparten los mismos atributos, operaciones, relaciones, y semántica. Estos objetos pueden representar cosas del mundo real o conceptuales.

Clase abstracta
Una **clase abstracta** es una clase que no puede tener instancias directas. Una clase abstracta puede tener operaciones concretas.

Clase activa
Una **clase activa** es una clase cuyas instancias son objetos activos.
Una clase activa, modela una familia de procesos o hilos.

Casos de uso abstractos
Un caso de uso abstracto es un caso de uso que nunca es instanciado directamente, pero existe para ser reutilizado por otros casos de uso

Classifier o Clasificador
Un classifier es un elemento del modelo que describe una característica de comportamiento o estructura.

Classifier role
Un classifier role es una ranura dentro de una colaboración que describe un rol que es jugado por un participante en esa colaboración.

Comportamiento
Se refiere a loa efectos observables de una operación o evento, incluyendo sus resultados.

Configuración de estado activo
El término **configuración de estado activo** se refiere al conjunto de estados activos que existen en un momento determinado dentro de una máquina de estado en particular.

Constructor
Es una operación, la cual se encuentra en el dominio de la clase, y que crea e inicializa una instancia de ésta (un objeto).

Diagrama
Es una representación gráfica de un conjunto de elementos del modelo. Esta representación es una proyección de esos elementos de un modelo dado.

Diagrama de actividad
Un **diagrama de actividad** es un diagrama que muestra los flujos entre actividades.
Un diagrama de actividad muestra un gráfico de actividad.

Directivas
Una directiva es una recomendación, regla o heurística que apoya a una o más actividades.

Elemento
Es un constituyente atómico de un modelo.

Elemento del modelo
Un elemento del modelo es un elemento que representa una abstracción del sistema que está siendo modelado.

Estado acción
Un **estado acción** es un estado cuyo propósito es ejecutar una acción, y luego, gatillar una transición hacia otro estado.

Estado activo
Un **estado activo** es un estado que actualmente es sostenido por un objeto dentro de la máquina de estado del objeto

Estado de actividad
Un **estado de actividad** es un estado cuyo propósito es ejecutar una actividad y luego gatillar una transición hacia otro estado.

Guard condition
Es una expresión booleana, la cual debe ser evaluada como verdadera, antes de gatillar la transición asociada a ésta.

Herencia
Se refiere al mecanismo por el cual elementos más específicos incorporan estructura y comportamiento definido por elementos más generales.

Línea base
Una línea base es una conjunto de elementos los cuales han sido revisados por los sostenedores apropiados del proyecto, y que en conjunto forman una base de acuerdos para la próxima evolución y desarrollo del sistema. Una línea base puede ser cambiada sólo a través de un procedimiento formal, tal como el cambio de requerimientos

Modelo
Se refiere a una abstracción de un sistema que está semánticamente completa. Representa una simplificación de la realidad de ese sistema.

Objeto activo
Un **objeto activo** es una instancia de una clase activa.
Un objeto activo posee un hilo y puede iniciar actividad de control.

OCL
Es un acronismo para Object Constraint Language, que es un lenguaje textual para realizar restricciones.

Operaciones abstractas
Una operación abstracta es una operación que nunca es instanciada.

Parámetro actual
El término **parámetro actual** es un sinónimo de argumento.

Pattern o Patrón
Es una colaboración parametrizada que representa una solución común a un problema común en un contexto dado.

Regla de las superclases abstractas
La **regla de las superclases abstractas** dice que las clases hojas deberían ser abstractas y que las clases que no son hojas deberían ser concretas. Esta “regla” es en realidad más un sólido principio de diseño. Bajo este esquema, una superclase abstracta, contiene todos los métodos a ser heredados, mientras la subclase concreta contiene versiones de esos métodos que son adaptados apropiadamente.

Secuencia de acción
Una **secuencia de acción** es una cadena de acciones que se realizan en forma ininterrumpida una tras otra.

Signature
Consiste en el nombre de la característica y parámetros

Workflow
Es una secuencia de actividades que producen un resultado de valor observable.

Workflow de análisis y diseño
Dentro del RUP, el propósito del workflow de análisis y diseño es traducir los requerimientos a una especificación que describe cómo debe estar construido e implementado el nuevo sistema.
El elemento principal del workflow de análisis y diseño es el modelo de diseño.

ANEXO 1: Herramientas basadas en UML

Company	Product	Version	Date	Platform	Price
Adaptive Arts	Simply Objects Standard	3.3.3	Mar-02	Windows	US$ 269
	forward engineering for Delphi, Smalltalk, Eiffel, Java, C++, Corba, VB				
	XMI, export diagrams as jpeg, png				
Adaptive Arts	Simply Objects Professional	3.3.3	Mar-02	Windows	US$ 1.999
	adds use case and interaction diagrams, report generator, multi-user				
Aonix	StP/UML	8.2	Jun-01	Windows, Unix	US$ 2.500
	multi-user repository, DOORS integration, report generation				
	Forte, Smalltalk, Java, C++				
Aonix	Select/Enterprise			Windows	n/a
	component repository, data modeling integration,				
	round-trip engineering for C++, Java, Forte, VB				
Arion Software	UML2COM	1	Feb-01	Windows	US$ 990
	integration tool for VC++/C++, add in for Rational Rose, COM+ code generator				
Artisan	Real-time Modeler	4.1	Aug-01	Windows	US$ 2.495
	real-time modeling, multi-user object repository				
Artisan	Real-time Studio Professional	4.1	Aug-01	Windows	n/a
	adds round-trip engineering for C, C++, Java,				
	DOORS integration, state machine animation				
Atos Origin	Delphia Object Modeler (D.OM)	3.2.6	Dec-00	Windows	US$ 0
	auto-generation of functional prototypes from UML models, XMI,				
	class and state diagrams, report generation				
BoldSoft	Bold for Delphi	3.1	Sep-01	Delphi	US$ 2.900
	OCL, forward engineering for Delphi, SQL generation, XMI import				
CanyonBlue	Cittera	1	Jun-01	Java VM	n/a
	web-based, collaborative, multi-user tool				
Computer Associates	AllFusion Component Modeler	4	Jan-00	Windows, Unix	n/a
	multiple code generations, object database repository, data modeling integration				
	formerly Paradigm Plus				
Dia	Dia	0.88	May-01	Linux	US$ 0
	Gnome Visio-like diagram tool with a UML template, export as SVG!				
Documentator	Documentator	4	Jan-02	Windows	US$ 89
	generate documentation from Rose 2000 to MS Word				
EctoSet	EctoSet Modeller	1.4	Feb-02	Windows	US$ 80
	Visio-like feel, scripting for code generation				
Eldean AB	ESS-Model	2	Oct-01	Windows	US$ 0
	generate class diagrams from Java and Kylix code, XMI export, HTML generation				
	open-source, GNU GPL license				
Elixir Technologies	Elixer CASE	1.2.4	Nov-99	Java VM	US$ 295
	auto-generation of sequence diagrams, metrics, OCL, XMI				
Embarcadero	Describe		Jul-01	Windows	n/a
	based on GDPro, integrates with leading Java IDE's, EJB Support				
Excel Software	Win A&D Standard	3.3	Jun-01	Windows	US$ 495
	CRC card support, component modeling				
Excel Software	QuickUML	1	Apr-01	Windows	US$ 495
	entry-level tool supports a core set of UML diagrams, XMI				
Excel Software	Win A&D Desktop	3.3	Jun-01	Windows	US$ 1.295

	adds forward-engineering for Java, Delphi, C++, scripting, state models				
Excel Software	Mac A&D Desktop	7.3	Jun-01	MacOS	US$ 1.295
	Macintosh version of Win A&D				
Fujaba	Fujaba	3	Jan-02	Java VM	US$ 0
	open-source GNU GPL license, forward and reverse engineering for Java				
	University of Paderborn project				
Gentleware	Poseidon for UML	1.2	Mar-02	Java VM	US$ 0
	based on ArgoUML, adds commercial support and services				
	integration with Forte for Java, OCL, SVG				
Honeywell	DOME	5.3	Mar-00	Smalltalk	US$ 0
	extensible notations, GNU GPL license, written in Smalltalk!				
	FTP site for exchanging models				
Hoora	HAT Professional	3.1	Mar-01	Windows	US$ 800
	supports HOORA process, C++ forward engineering, report generation				
	requirements management, Rose import				
I-Logix	Rhapsody Modeler	4	Sep-01	Windows	US$ 0
	real-time, C, C++, single-user, free starter version				
I-Logix	Rhapsody Solo	4	Sep-01	Windows	US$ 895
	real-time, C, C++, Java, single-user, XMI				
I-Logix	Rhapsody Development	4	Sep-01	Windows	n/a
	real-time, C, C++, Java, multi-user, XMI				
IBM	Visual Age Smalltalk UML Designer	5	Mar-00	Smalltalk	US$ 3.419
	UML Designer is an add-on to Visual Age Smalltalk Enterprise				
Ideogramic	Ideogramic UML	2.2	Mar-02	Windows	US$ 1.000
	innovative input scheme using gestures on large whiteboards				
	class, use-case, sequence diagrams, Java reverse engineering, XMI				
Jsequence	Jsequence		Mar-02	Java VM	US$ 199
	create sequence diagrams automatically from java source, XMI export				
Kennedy-Carter	iUML	2	Jan-01	Windows, Unix	US$ 3.000
	produce executable UML models using a formal action language,				
	includes code generator and simulator tools				
Magna Solutions	Silverrun Java Designer	2.7.5		Java VM	US$ 0
	forward and reverse engineer Java code to class diagrams				
Mega International	Mega Suite	5	Oct-00	Windows	n/a
	supports Delphi, Forte, Java, VB, XML				
MetaCase Consulting	MetaEdit+	3		Windows, Unix	US$ 4.500
	multi-user object repository, customizable meta-tool, report generation;				
	Java, C++, Smalltalk, IDL, Delphi, SQL				
Metamill Software	Metamill	2	Nov-01	Windows	US$ 85
	low-cost tool, index file based shared repository				
	round-trip engineering for Java, C++, C#				
MicroTOOL	ObjectiF	4.6		Windows	n/a
	VB scripting, C++, Java, IDL, XMI, integration with JBuilder				
Microgold Software	WithClass Professional	2000	Mar-01	Windows	US$ 495
	multiple code generation, Python scripting!				
	now supports C#				
Microgold Software	WithClass Enterprise	2000	Mar-01	Windows	US$ 800
	adds VBA support for scripting				
Microsoft	Visual Studio .NET Enterprise Architect	1	Feb-02	Windows	US$ 2.500
	supports 8 UML diagrams through Visio integration				
Microsoft Visio	Visio 2002 Professional	2002	Mar-01	Windows	US$ 499

	C++, VB reverse engineering, MS Visual Studio				
ModelMaker Tools	ModelMaker	6.1	Dec-01	Delphi	US$ 269
	forward and reverse engineering for Delphi, GOF design patterns				
Modelistic	Modelistic	1	Aug-00	Java VM	US$ 299
	round-trip engineering for Java				
Mountfield Computers	mDes	5.2.4	Feb-02	Java VM	US$ 65
	supports all 9 UML 1.3 diagrams, free for non-commercial use				
	forward and reverse engineering for Java, Python				
	XMI export, HTML generation, SVG!				
No Magic	MagicDraw UML Standard	5.1	Mar-02	Java VM	US$ 299
	supports all 9 UML 1.3 diagrams, Swing GUI, HTML generation,				
	read Rose models, XMI, SVG, XSLT, EJB				
No Magic	MagicDraw UML Professional	5.1	Mar-02	Java VM	US$ 699
	adds forward and reverse engineering for Java, C++, IDL				
Novosoft	Novosoft UML Library	1.4	Dec-01	Java VM	US$ 0
	open-source library which supports the UML 1.3 metamodel				
	persistence using XMI, integrated with ArgoUML				
Novosoft	FL	0.5.5	Feb-02	Java VM	n/a
	develop Java object persistence from class diagrams				
	Rational Rose add-in, supports OQL, supports major DBMS				
OTW Software	Object Technology Workbench Private	2.4	Apr-00	Windows	US$ 795
	round-trip engineering for Java, C++, Delphi				
	supports CORBA-IDL, SQL-DDL, patterns, repository, HTML				
OTW Software	Object Technology Workbench Team	2.4	Apr-00	Windows	US$ 1.495
	adds team-based repository				
OWiS Software (Germany)	OTW - Object Technology Workbench	2.4	Jan-00	Windows	n/a
	round-trip engineering for Java, C++, support for patterns, repository, data modeling				
Object Domain Systems	Object Domain Standard	3	Nov-01	Java VM	US$ 495
	forward and reverse engineering for C++, Java, Python,				
	Python scripting, educational pricing available				
Object Domain Systems	Object Domain Professional	3	Nov-01	Java VM	US$ 995
	adds multi-user repository, round-trip engineering for Java, HTML generation				
Object Insight	JVISION	2.1	Oct-01	Java VM	US$ 499
	reverse-engineering of Java, integration with Visual Café, HTML generation				
	9 UML diagrams plus robustness diagram				
Object Plant	Object Plant	3.0.1	Feb-02	MacOS	US$ 35
	shareware, class, state, use case diagrams, C++, Java				
Plastic Software	Plastic	3	Jan-01	Java VM	US$ 297
	forward and reverse engineering for Java, HTML generation, model validation				
Popkin	System Architect	8.5	Oct-01	Windows	n/a
	round-trip engineering for Java, C++, VBA				
	data modeling, Microsoft repository support, scripting, DOORS support				
Pragsoft Corporation	UML Studio	6.2	Feb-02	Windows	US$ 500
	forward and reverse engineering for C++, Java, IDL, scripting tools, auto-save!				
Project Technology	BridgePoint	5	Apr-01	Windows, Unix	n/a
	UML models compiled to executable code, supports Shlaer-Mellor method				
	model verification through animation				

ProxySource	ProxyDesigner	1	Dec-00	Windows	US$ 0
	publish UML models directly to online forums				
Rational	Rose Modeler	2001	Nov-00	Windows	US$ 1.829
	base model				
Rational	Rose Professional	2001	Nov-00	Windows	US$ 2.442
	adds round-trip engineering, repository support, data modeling;				
	Java, C++, and VB versions sold separately				
Rational	Rational XDE Professional .NET Edition	2002	Feb-02	Windows	US$ 3.595
	C#, fully integrated with Visual Studio .NET, patterns support, round-trip engineering				
Rational	Rational XDE Professional Java Edition	2002	Feb-02	Windows	US$ 3.595
	C#, fully integrated with IBM Websphere Studio and Eclipse				
	J2EE support, patterns support, round-trip engineering				
Rational	Rose Enterprise	2001	Nov-00	Windows	US$ 4.290
	adds web publishing, CORBA-IDL,				
	integration w/ ClearCase (version control) and MS VisualStudio (VB, C++ only)				
Rational	Rose Real Time	2001	Nov-00	Windows	n/a
	real-time modeling based on ObjecTime technology				
Sodifrance	Scriptor	2.5	Oct-01	Java VM	US$ 4.000
	meta-generator providing the capability to build your own specific code generator				
	reads XMI files				
Softeam	Objecteering Personal Edition	5.1.0	May-01	Windows, Unix	US$ 0
	free, base version; includes XMI support				
Softeam	Objecteering Personal Edition / Java	5.1.0	May-01	Windows, Unix	US$ 859
	adds round-trip engineering for Java				
Softeam	Objecteering Project Edition	5.1.0	May-01	Windows, Unix	US$ 1.569
	full-featured product without multi-user repository support				
Softeam	Objecteering Enterprise Edition	5.1.0	May-01	Windows, Unix	US$ 2.569
	adds parameterized code generation, multi-user repository,				
	data modeling, design patterns, metrics				
Softera	SoftModeler Standard	3.5	Dec-01	Java VM	US$ 245
	base model				
Softera	SoftModeler Professional	3.5	Dec-01	Java VM	US$ 495
	adds EJB support, round-trip synchronization, XMI export				
Softera	SoftModeler Enterprise	3.5	Dec-01	Java VM	US$ 995
	Adds multi-user, shared repository, model simulation				
	sequence-diagram animation				
Sparx Systems	Enterprise Architect Professional	3.1	Jan-02	Windows	US$ 149
	use cases, contracts (pre/post conditions), round-trip engineering for C++, C#, Java, VB.Net				
	multi-user, project estimation, excellent, free UML tutorial				
	XMI import/export				
Sybase	PowerDesigner	9	Dec-01	Windows	US$ 5.990
	object/relational design using class diagrams, repository support				
	includes use case and sequence diagrams				
TNI	OpenTool	3.2	Jan-01	Windows, Unix	n/a
	forward engineering for C++, Java, Smalltalk, reverse engineering for Java,				
	multiple documentation generations				
Tassc	Estimator / manager	4.1	Mar-01	Windows	US$ 4.255

	project estimation and scheduling tool based on OO criteria,				
	interfaces to Rational Rose, Select Enterprise, Together Control Center				
Telelogic	ObjectGeode	4.1	Jun-99	Windows, Unix	n/a
	real-time modeling, multiple RTOS targets, generates C, C++				
Telelogic	Tau UML Suite	4.3	Sep-01	Windows	n/a
	real-time modeling, UML to SDL translation, XMI				
	acquired COOL:Jex from Sterling Software, DOORS from QSS				
Tigris	ArgoUML	0.81	Oct-00	Java VM	US$ 0
	open-source project, written in Java, run-time model critique, OCL, XMI				
TogetherSoft	Together CommunityEdition	6	Apr-02	Java VM	US$ 0
	simultaneous round-trip engineering for Java, C++, class diagrams only				
TogetherSoft	Together Solo	6	Apr-02	Java VM	US$ 3.495
	adds complete UML diagram support, HTML generation, code debugger, refactoring				
TogetherSoft	Together Control Center	6	Apr-02	Java VM	US$ 5.995
	adds EJB development and deployment support, GOF design patterns, VB, .NET, C#				
Tri-Pacific Software	Rapid RMA	5.2	Nov-01	Java VM	n/a
	simulation and test for real-time UML				
	integrates with Rational's Rose RealTime and I-Logix Rhapsody				
Unimodeler	Unimodeler	1	Mar-02	Linux	US$ 0
	supports all 9 UML diagrams, GTK (Gnome) based, postscript printing				
Visual Object Modelers	Visual UML Standard Edition	2.9	Mar-02	Windows	US$ 495
	C++, C# and Java code generation and reverse engineering, data modeling				
Visual Object Modelers	Visual UML Standard Edition for VB	2.9	Mar-02	Windows	US$ 795
	adds VB support, VB round-trip engineering				
Visual Object Modelers	Visual UML Plus Edition for VB	2.9	Mar-02	Windows	US$ 995
	includes VBA, MS Repository 2.0 support				
Visual Paradigm	Visual Paradigm for UML	1	Dec-01	Java VM	n/a
	Java forward engineering, HTML and PDF generation				
	site has feature demo using viewlets				
WebGain	StructureBuilder Enterprise	4.5.4	Jun-01	Java VM	US$ 4.995

ANEXO 2: Namespaces de la plataforma .Net Framework

Espacio de nombres de las clases de la plataforma Microsoft .NET Framework y C#.

Namespaces

Microsoft.CSharp
Microsoft.VisualBasic
Microsoft.Win32
System
System.CodeDom
System.CodeDom.Compiler
System.Collections
System.Collections.Specialized
System.ComponentModel
System.ComponentModel.Design
System.ComponentModel.Design.Serialization
System.Configuration
System.Configuration.Assemblies
System.Data
System.Data.Common
System.Data.OleDb
System.Data.SqlClient
System.Data.SqlTypes
System.Diagnostics
System.Diagnostics.SymbolStore
System.DirectoryServices
System.Drawing
System.Drawing.Design
System.Drawing.Drawing2D
System.Drawing.Imaging
System.Drawing.Printing
System.Drawing.Text
System.Globalization
System.IO
System.IO.IsolatedStorage
System.Messaging
System.Messaging.Design
System.Net
System.Net.Sockets
System.Reflection
System.Reflection.Emit
System.Resources
System.Runtime.CompilerServices
System.Runtime.InteropServices
System.Runtime.InteropServices.Expando
System.Runtime.Remoting
System.Runtime.Remoting.Activation
System.Runtime.Remoting.Channels
System.Runtime.Remoting.Contexts
System.Runtime.Remoting.Lifetime
System.Runtime.Remoting.Messaging
System.Runtime.Remoting.Metadata
System.Runtime.Remoting.Metadata.W3cXsd2001
System.Runtime.Remoting.Proxies
System.Runtime.Remoting.Services
System.Runtime.Serialization
System.Runtime.Serialization.Formatters
System.Runtime.Serialization.Formatters.Binary
System.Security
System.Security.Cryptography
System.Security.Cryptography.X509Certificates
System.Security.Permissions
System.Security.Policy
System.Security.Principal
System.Text
System.Text.RegularExpressions
System.Threading
System.Timers
System.Web
System.Web.Caching
System.Web.Configuration
System.Web.Handlers
System.Web.Hosting
System.Web.Mail
System.Web.Security
System.Web.SessionState
System.Web.UI
System.Web.UI.HtmlControls
System.Web.UI.WebControls
System.Web.Util
System.Windows.Forms
System.Windows.Forms.ComponentModel.Com2Interop
System.Windows.Forms.Design
System.Windows.Forms.PropertyGridInternal
System.Xml
System.Xml.Schema
System.Xml.Serialization
System.Xml.XPath
System.Xml.Xsl

ANEXO 3: Acerca de patrones (Patterns)

Un Patrón consiste en un nombre que se asocia a la descripción de un problema y una solución que pueden aplicarse a un nuevo contexto. La estructura de un patrón es la siguiente:

- **Nombre de Patrón:**
- **Solución:**
- **El problema que resuelve:**

Los patrones se aplican durante la creación de los **diagramas de interacción** y, por lo general, llevan al descubrimiento de nuevos objetos del dominio.

Los patrones se han dividido en patrones de análisis y patrones de diseño. Estos últimos son los utilizados con más frecuencia para acelerar el desarrollo de proyectos informáticos.

Los patrones de diseño, a su vez se han catalogado en los siguientes 3 grupos:

1. **Creational Pattern (Patrones Creacionales)**
    - Abstract Factory
    - Builder
    - Factory Method
    - Prototype
    - Singleton
2. **Structural Patterns (Patrones Estructurales)**
    - Adapter
    - Bridge
    - Composite
    - Decorator
    - Facade
    - Flyweight
    - Proxy

3. **Behavioral Patterns (Patrones de comportamiento)**
    - Chain of responsibility
    - Command
    - Interpreter
    - Iterator
    - Mediator
    - Memento
    - Observer
    - State
    - Strategy
    - Template Method
    - Visitor

Constituyendo así, los 23 patrones más utilizados en la industria del desarrollo de software. Rational XDE ha incorporado estos patrones y la posibilidad de aplicarlos a los modelos.

Ejemplo de patrón:

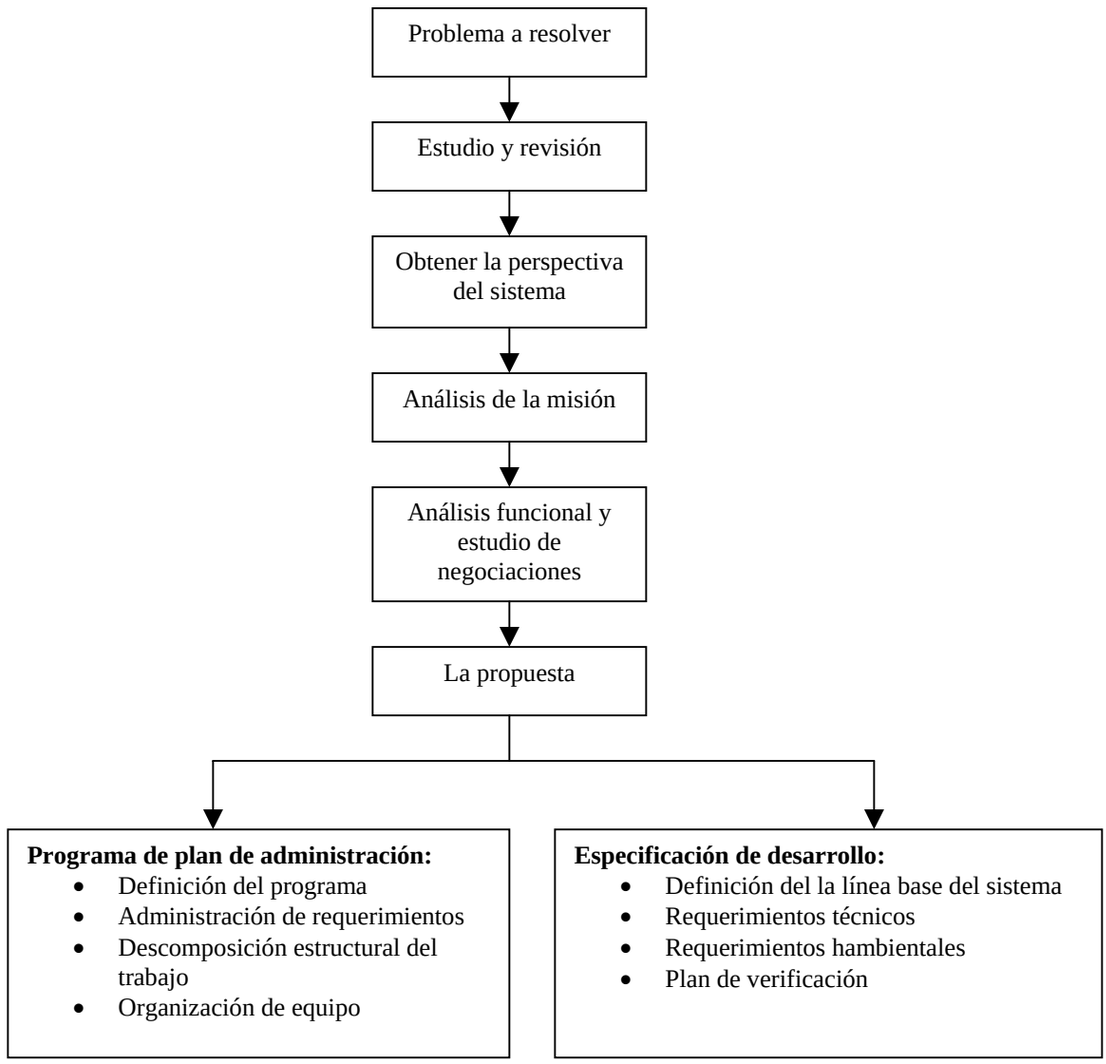
Singleton pattern (GoF)

Problema: Se permite exactamente **una instancia** de una clase. Los objetos requieren de un punto de acceso global y único.

Solución: Definir un método estático de la clase que retorna el singleton. El método estático del singleton debe ser llamado “getInstance()”.

Un singleton se representa en un diagrama de interacción haciendo uso de un estereotipo **<<singleton>>**. Al hacer esto, no es necesario mostrar explícitamente el mensaje “getInstance()” antes de enviar un mensaje a la instancia Singleton.

ANEXO 4: Proceso de Ingeniería Up-Front



ANEXO 5: Modelo de Ciclo de vida en espiral

El modelo en espiral del proceso del software, fue propuesto originalmente por Barry Boehm (1988). Cada ciclo del espiral se divide en 4 sectores:

1. **Definición de objetivos:** En esta fase del proyecto, se definen los objetivos específicos. Se identifican las restricciones del proceso y el producto, y se estipula un plan detallado de administración. Se identifican los riesgos del proyecto. Dependiendo de esos riesgos, se planifican estrategias alternativas.

2. **Evaluación y reducción de riesgos:** Se lleva a cabo un análisis detallado para cada uno de los riesgos del proyecto. Se definen los pasos para reducir dichos riesgos.

3. **Desarrollo y validación:** Después de la evaluación de riesgos, se elige un modelo para el desarrollo del sistema. Entre los modelos a elegir, se encuentran el prototipado evolutivo, en modelo en cascada, etc.

4. **Planificación:** El proyecto se revisa y se toma la decisión de si se debe continuar con el próximo ciclo del espiral. Si se decide continuar, se debe planificar la siguiente fase del proyecto.

Una diferencia importante entre el modelo en espiral y los otros modelos del proceso software, es que éste considera de una forma explícita el riesgo. En el modelo en espiral, no existen fase fijas como la especificación o el diseño. Este modelo puede contener otros modelos. Por ejemplo, la construcción de un prototipo se utiliza para resolver las dudas en los requerimientos y así reducir el riesgo. Esto se puede continuar con un desarrollo en cascada. El desarrollo formal de un sistema, se puede utilizar para aquellas partes que requieren un alto grado de seguridad. La figura A8.1, muestra el modelo en espiral de Barry Boehm.

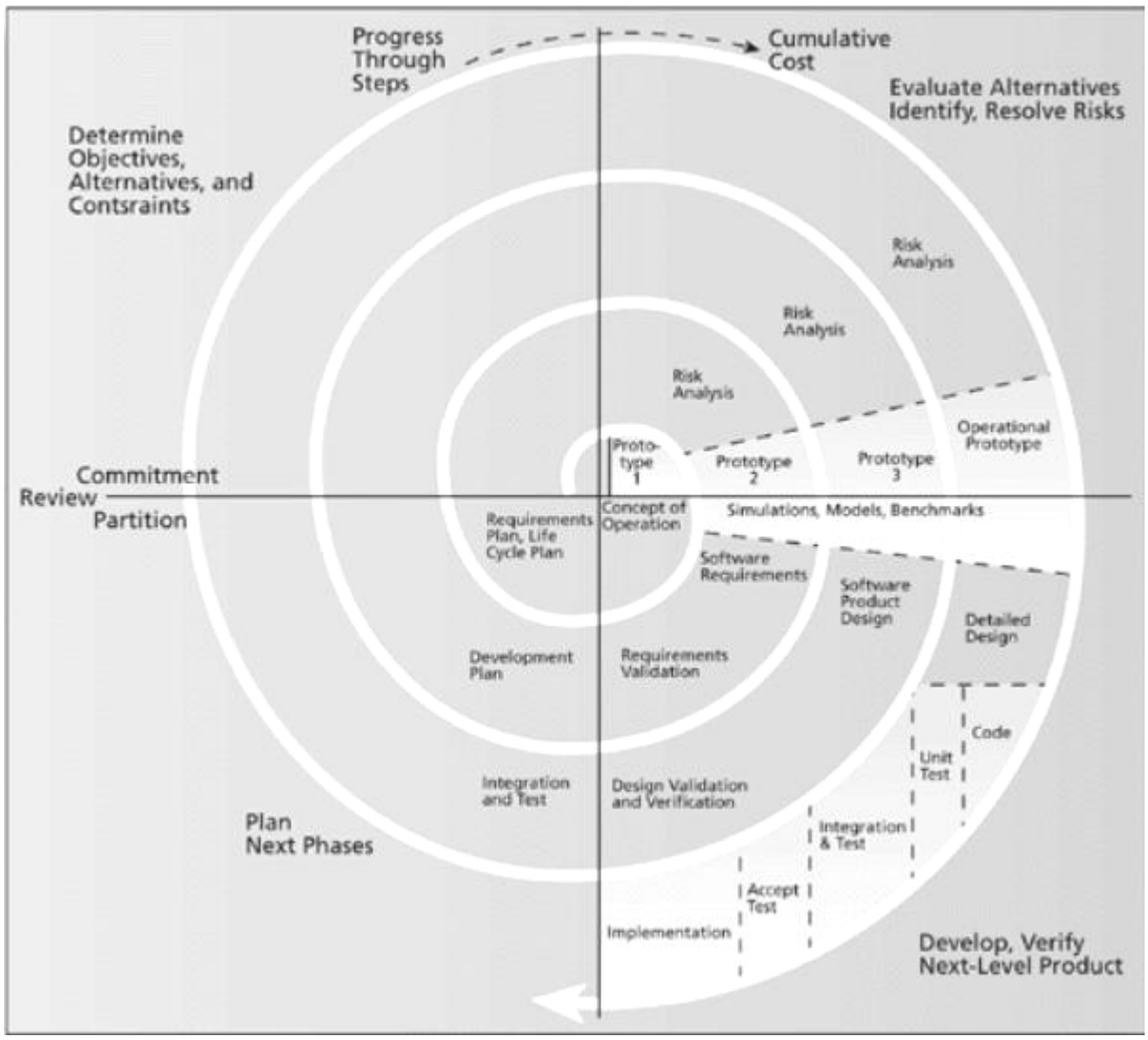


Figura A8.1: Modelo en espiral de Barry Boehm. Extraído desde PMBOK Guide 2000.

ANEXO 6: Generaciones de lenguajes de programación

Los lenguajes de programación se dividen en 2 categorías fundamentales :

- *Bajo nivel*: Son dependientes de la máquina, están diseñados para ejecutarse en un determinado computador. A esta categoría pertenecen las 2 primeras generaciones.

- *Alto Nivel*: Son independientes de la máquina y se pueden utilizar en una variedad de computadores. Pertenecen a esta categoría la tercera y la cuarta generación. Los lenguajes de más alto nivel no ofrecen necesariamente mayores capacidades de programación, pero si ofrecen una *interacción programador/computador más avanzada*. Cuanto más alto es el nivel del lenguaje, más sencillo es comprenderlo y utilizarlo.

Cada generación de lenguajes es más fácil de usar y más parecida al lenguaje natural que sus predecesores. Los lenguajes posteriores a la cuarta generación se conocen como *lenguajes de muy alto nivel*. Son lenguajes de muy alto nivel los generadores de aplicaciones y los naturales.

En cada nuevo nivel se requieren menos instrucciones para indicar al computador que efectúe una tarea particular. Pero los lenguajes de alto nivel son sólo una ayuda para el programador. Un mayor nivel significa que son necesarios menos comandos, debido a que cada comando o mandato de alto nivel reemplaza muchas instrucciones de nivel inferior.

1. **Primera Generación - Lenguaje de máquina**: Empieza en los años 1940-1950. Consistía en sucesiones de dígitos binarios. Todas las instrucciones y mandatos se escribían valiéndose de cadenas de estos dígitos. Aún en la actualidad, es el único lenguaje interno que entiende el computador; los programas se escriben en lenguajes de mayor nivel y se traducen a lenguaje de máquina.

2. **Segunda Generación – Lenguajes ensambladores**: Fines de los ’50. Se diferencian de los lenguajes de máquina en que en lugar de usar códigos binarios, las instrucciones se representan con símbolos fáciles de reconocer,

conocidos como *mnemotécnicos*. Aún se utilizan estos lenguajes cuando interesa un nivel máximo de eficiencia en la ejecución o cuando se requieren manipulaciones complejas. Al igual que los lenguajes de máquina, los lenguajes ensambladores son únicos para un computador particular. Esta dependencia del computador (procesador) los hace ser lenguajes de bajo nivel.

3. **Tercera Generación**: años ’60. Los lenguajes de esta generación se dividen en tres categorías, según se orienten a:

   - **Procedimientos**: Requieren que la codificación de las instrucciones se haga en la secuencia en que se deben ejecutar para solucionar el problema. A su vez se clasifican en científicos (por ejemplo: FORTRAN), empresariales (por ejemplo COBOL), y de uso general o múltiple (como BASIC). Todos estos lenguajes permiten señalar *cómo* se debe efectuar una tarea a un nivel mayor que en los lenguajes ensambladores. Hacen énfasis en los procedimientos o la matemática implícita, es decir en *lo* que se hace (la acción).

   - **Problemas**: Están diseñados para resolver un conjunto particular de problemas y no requieren el detalle de la programación que los lenguajes orientados a procedimientos. Hacen hincapié en la entrada y la salida deseadas.

   - **Objetos**: El énfasis se hace en el *objeto* de la acción. Los beneficios que aportan estos lenguajes incluyen una mayor productividad del programador y claridad de la lógica, además de ofrecer la flexibilidad necesaria para manejar problemas abstractos de programación.

4. **Cuarta Generación:** su característica distintiva es el énfasis en especificar *qué* es lo que se debe hacer, en vez de cómo ejecutar una tarea. Las

especificaciones de los programas se desarrollan a un nivel más alto que en los lenguajes de la generación anterior. La característica distintiva es ajena a los procedimientos, el programador no tiene que especificar cada paso para terminar una tarea o procesamiento. Las características generales de los lenguajes de cuarta generación son:

- Uso de frases y oraciones parecidas al inglés para emitir instrucciones;
- no operan por procedimientos, por lo que permiten a los usuarios centrarse en lo que hay que hacer no en cómo hacerlo;
- Al hacerse cargo de muchos de los detalles de cómo hacer las cosas, incrementan la productividad.

Hay dos tipos de lenguajes de cuarta generación, según se orienten:

- **A la producción:** Diseñados sobre todo para profesionales en la computación.
- **Al usuario:** Diseñados sobre todo para los usuarios finales, que pueden escribir programas para hacer consultas en una base de datos y para crear sistemas de información.

ANEXO 7: Checklist para el conteo de líneas de código fuente.

La siguiente lista pretende definir qué debe ser considerado como una línea de código fuente y qué no. La lista ha sido extraída desde la referencia bibliográfica perteneciente al modelo COCOMO II, y tiene su origen en el Software Engineering Institute, USC.

Checklist de definición para el conteo de líneas de código fuente			
Nombre de la definición: Definición básica de una línea de código fuente **Creador :** COCOMO II			
Unidad de medida:	Líneas físicas de código fuente.		
	Líneas lógicas de código fuente.		X
Típo de declaración			
Cuando una línea o declaración contenga más de un tipo, éste debe ser clasificado como el tipo con la mayor precedencia.			
	Orden de precedencia	**Incluir**	**Excluir**
1.- Ejecutable	1	X	
2.- No Ejecutable			
3.- Declaraciones	2	X	
4.- Directivas de compilador	3	X	
5.- Comentarios			
6.- En su propia línea	4		X
7.- En líneas con código fuente	5		X
8.- Banners y espaciadores no blancos	6		X
9.- En blanco (Comentarios vacios)	7		X
10.- Líneas en blanco	8		X
Cómo es producido		**Incluir**	**Excluir**
1.- Programada		X	
2.- Generada con generadores de código fuente			X
3.- Convertida con traductores automáticos		X	
4.- Copiada o reutilizada sin cambios		X	
5.- Modificada		X	
6.- Eliminada			X
Origen		**Incluir**	**Excluir**
1.- Trabajo nuevo: sin existencia previa		X	
2.- Trabajo Previo: Tomado o adaptado de			
3.- Una versión previa, compilación o release		X	X
4.- COTS, que no sean librerias			X
5.- GFS, que no sean librerias			X
6.- Otro producto			X
7.- Una librería de apoyo de lenguaje provista por un tercer vendedor (Sin modificaciones)			X
8.- Un sistema operativo o utilidad provista por un tercer vendedor.			X
9.- Una librería de apoyo de lenguaje local o modificada o un sistema operativo.			X
10.- Otra librería comercial			X
11.- Una librería de reutilización (Software diseñado para ser reutilizado)		X	
12.- Otro componente de software o librería		X	

Checklist de definición para el conteo de líneas de código fuente			
Nombre de la definición: Definición básica de una línea de código fuente **Creador :** COCOMO II			
Uso		**Incluir**	**Excluir**
1.- En o como parte de un producto principal		X	
2.- Externo a o en apoyo a el producto principal			X
Entrega		**Incluir**	**Excluir**
1.- Entregado:			
2.- Entregado como fuentes		X	
3.- Entregado en forma compilada o ejecutable, pero no en fuentes			X
4.- No entregado:			
5.- Bajo control de configuración			X
6.- No bajo control de configuración			X
Funcionalidad		**Incluir**	**Excluir**
1.- Operativa		X	
2.- No operativa (Muerta, bypaseado, no utilizado, no referenciado, o inaccesible)			
3.- Funcional (código desechado en forma intencional, reactivado para propósitos especiales)		X	
4.- No funcional (presente en forma no intencional)			X
Replicaciones		**Incluir**	**Excluir**
1.- Líneas de código fuente maestras (originales)		X	
2.- Replicación física de las sentencias maestras, almacenadas en el código maestro		X	
3.- Copias insertadas, instanciadas, o expandidas cuando se compila o linkea (linking)			X
4.- Replicas postproducción – como en sistemas distribuidos, redundantes o reparametrizados.			X
Estado de desarrollo	Cada sentencia tiene un único estado, por lo general, éste es el de su unidad padre.		
		Incluir	**Excluir**
1.- Estimado o planificado			X
2.- Diseñado			X
3.- Codificado			X
4.- Prueba de la unidad completada			X
5.- Integrado dentro de componentes			X
6.- Revisión de pruebas completa (Test Readiness Review)			X
7.- Pruebas del software completas			X
8.- Prueba del sistema completa		X	

Checklist de definición para el conteo de líneas de código fuente			
Nombre de la definición: Definición básica de una línea de código fuente **Creador :** COCOMO II			
Lenguaje	Lista cada lenguaje fuente en una línea distinta.		
		Incluir	**Excluir**
1.- Separar totales para cada lenguaje		X	
Clarificaciones (General)		**Incluir**	**Excluir**
1.- Nulls, continues, y no-ops		X	
2.- Sentencias vacias, por ejemplo ";;" y símbolos de punto y coma en líneas separadas			X
3.- Sentencias que instancíen generalizaciones		X	
4.- Begin...end y los símbolos {...} usados como sentencias ejecutables.		X	
5.- Begin...end y los símbolos {...} que delimitan los cuerpos de los subprogramas			X
6.- Expresiones lógicas usadas como condiciones de prueba.			X
7.- Expresiones de evaluación utilizadas como argumentos de subprogramas			X
8.- Símbolos de Fin que terminan con sentencias de ejecución.			X
9.- Símbolos de Fin que terminan con declaraciones o cuerpos de subprogramas			X
10.- Then, else y tokens que representen ramificaciones.			X
11.- Sentencias Elseif		X	
12.- Palabras claves tales como: procedure, interface, implementation		X	
13.- Labels en sus propias líneas			X
Clarificaciones (Específicas al lenguaje)			
C y C++ (Aplicables a C#)		**Incluir**	**Excluir**
1.- Sentencia Nula, por ejemplo: ";" por si solo para indicar un cuerpo vacio			X
2.- Sentencias de expersión (terminadas en punto y coma)		X	
3.- Expresiones separadas por punto y coma, como en una sentencia "for".		X	
4.- Senetencias de bloque, como por ejemplo: {...} sin un terminador punto y coma		X	
5.- ";" en una línea por si solo cuando forme parte de una declaración			X
6.- ";" en una línea por si solo cuando forme parte de una sentencia ejecutable.			X
7.- Sentencias compiladas condicionalmente (#if, #ifdef, #ifndef)		X	
8.- Sentencias de preprocesador distintas de #if, #ifdef y #ifndef		X	

ANEXO 8: Características generales de las aplicaciones

El conteo de puntos de función sin ajustar debe calibrarse según 14 elementos que dependen del entorno. Estos son:

1. <u>Comunicación de Datos</u>: los datos o información de control que utiliza la aplicación, se envía o recibe a través de sus características de comunicación.

0 Aplicación es batch exclusivamente.
1-2 Impresión o entrada de datos remota.
3-5 Teleproceso (TP) interactivo.
3 TP interface a un proceso batch.
5 La aplicación es interactiva predominantemente.

2. <u>Función Distribuida</u>. "Distribuida" significa que los componentes (o los datos) de la aplicación están distribuidos en dos o más procesadores diferentes (esto también incrementa el factor anterior).

0 La aplicación no ayuda a la trasferencia de datos o a la función de procesamiento entre los componentes del sistema.
1 La aplicación prepara datos para el usuario final de otro procesador.
2-4 Los datos se preparan para trasferencia, se trasfieren y se procesan en otro componente del sistema.
5 Las funciones de procesamiento se realizan dinámicamente en el componente más apropiado del sistema.

3. <u>Rendimiento</u>: referido a la importancia de respuesta dentro de todo el sistema

0-3 Análisis y diseño de las consideraciones del rendimiento son estándar. No se precisan requerimientos especiales por parte del usuario.
4 En la fase de diseño se incluyen tareas del análisis del rendimiento para cumplir los requerimientos del usuario.
5 Además se utilizan herramientas de análisis del rendimiento en el diseño, desarrollo e instalación.

4. <u>Configuración utilizada masivamente</u>: se refiere a la importancia del entorno. Es decir, si hay restricciones de memoria o del hardware.

0-3 La aplicación corre en una máquina estándar sin restricciones de operación.
4 Restricciones de operación requieren características específicas de la aplicación en el procesador central.
5 Además hay restricciones específicas a la aplicación en los componentes distribuidos del sistema.

5. <u>Tasas de Transacción</u>: una alta llegada de transacciones provoca problemas más allá de los de la característica 3

0-3 Las tasas son tales que las consideraciones de análisis de rendimiento son estándares.
4 En la fase de diseño se incluyen tareas de análisis de rendimiento para verificar las altas tasas de transacciones.
5 Además se utilizan herramientas de análisis del rendimiento.

6. <u>Entrada On-Line de datos</u>

0-2 Hasta el 15% de las transacciones tienen entrada interactiva.
3-4 15% al 30% tienen entrada interactiva.
5 30% al 50% tienen entrada interactiva.

7. <u>Diseño para la eficiencia de usuario final</u>

0-3 No se especifican requerimientos especiales.
4 Se incluyen tareas de diseño para la consideración de factores humanos.
5 Además se utilizan herramientas especiales o de prototipado para promover la eficiencia.

8. <u>Actualización On-Line</u>

0 Nada.
1-2 Actualización on line de los archivos de control. El volumen de actualización es bajo y la recuperación fácil.
3 Actualización on line de la mayoría de los archivos lógicos internos.
4 Además es esencial la protección contra la pérdida de datos.
5 Además se considera el costo de recuperación de volúmenes elevados.

9. <u>Complejidad del procesamiento</u>: esto es, complejidad interna más allá de la media en lo referente a la entrada, salida o lógica de procesamiento

¿Qué características tiene la aplicación?

- mucho procesamiento matemático y/o lógico.
- procesamiento complejo de las entradas.
- procesamiento complejo de las salidas.
- muchas excepciones de procesamiento, muchas transacciones incompletas y mucho reprocesamiento de las transacciones.
- procesamiento de seguridad y/o control sensitivo.

0 No se aplica nada de esto.
1 Se aplica alguna cosa.
2 Se aplican dos cosas.
3 Se aplican tres cosas.
4 Se aplican cuatro cosas.
5 Se aplica todo.

10. Utilizable en otras aplicaciones: el código se diseña para que sea compartido o utilizable por otras aplicaciones (no confundir con 13).

0-1 Una aplicación local que responde a las necesidades de una organización usuaria.
2-3 La aplicación utiliza o produce módulos comunes que consideran más necesidades que las del usuario.
4-5 Además, la aplicación se "empaquetó" y documentó con el propósito de fácil
Reutilización.

11. Facilidad de Instalación

0-1 No se requieren características especiales de conversión e instalación por parte del usuario.
2-3 Los requerimientos de conversión e instalación fueron descritos por el
usuario y se proporcionaron guías de conversión e instalación.
4-5 Además se proporcionaron y probaron herramientas de conversión e
Instalación.

12. Facilidad de Operación

0 No se especifican por parte del usuario consideraciones específicas
de operación.
1-2 Se requieren, proporcionan y prueban procesos específicos de arranque, backup y recuperación.
3-4 Además la aplicación minimiza la necesidad de actividades manuales,
tales como instalación de cintas y papel.
5 La aplicación se diseña para operación sin atención.

13. Puestos Múltiples.

0 El usuario no requiere la consideración de más de un puesto.
1-3 Se incluyeron necesidades de varios puestos en el diseño.
4-5 Se proporciona documentación y plan de apoyo para soportar la aplicación en varios lugares.

14. Facilidad de Cambio: esfuerzo específico de diseño para facilitar cambios futuros.

0 No hay requerimientos especiales del usuario para minimizar o facilitar
el cambio.
1-3 Se proporciona capacidad de consulta flexible.
4-5 Datos importantes de control se mantienen en tablas que son actualizadas por el usuario a través de procesos on-line interactivos.

Para calcular el total de puntos de función ajustados, se utilizarán las siguientes ecuaciones.

$$VFA = 0{,}65 + 0{,}01 \cdot \sum_{i=1}^{14} C_i$$

$$Puntos\ de\ función\ Ajustados = (Puntos\ de\ función \sin ajustar) \cdot VFA$$

ANEXO 9: Evaluación de niveles de KPAs

Areas de Proceso Claves (KPAs)		Casi Siempre (100%)	Frecuentemente (75%)	En la mitad (50%)	Ocacionalmente (25%)	En pocas ocaciones (1%)	No se aplica	No se sabe
1	Gestión de requisitos							
2	Planificación de proyectos software							
3	Seguimiento de proyecto software							
4	Gestión de subcontrato software							
5	Aseguramiento de la calidad software							
6	Gestión de la configuración software							
7	Focos de proceso de organización							
8	Definición de proceso de organización							
9	Programa de formación							
10	Gestión del software integrado							
11	Ingeniería de producto software							
12	Coordinación intergrupos							
13	Informes detallados							
14	Gestión de proceso cuantitativo							
15	Gestión de calidad software							
16	Prevención de defectos							
17	Gestión de cambio de tecnología							
18	Gestión de cambio de proceso							

ANEXO 10: Estimación de tamaño de la base de datos

Recomendaciones para estimar el tamaño de un base de datos:

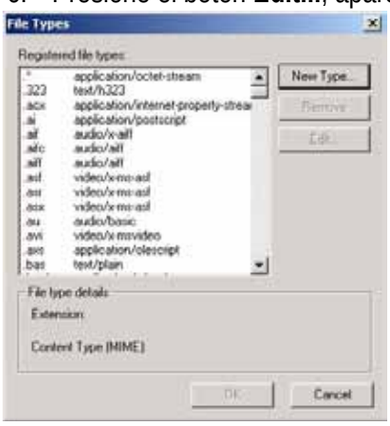
- Para cada tabla, encontrar cuantos bytes usará cada fila en promedio. Es fácil calcular el tamaño de una columna fija, pero calcular el espacio de campos variables es mucho más difícil. Una de las maneras de hacer esto, es obtener algo de la información real que contendrá el campo de largo variable, de forma que basado en esta información se pueda obtener un promedio del número de bytes que contendrá dicha columna. Una vez que se ha estimado el tamaño típico de cada columna, se puede calcular el tamaño de cada fila en la tabla.
- El punto anterior es un buen comienzo. Sin embargo, por lo general subestimará la cantidad de espacio que se necesitará. Además de la información real, también se debe estimar el tipo y el número de índices que se usarán para cada tabla. Los índices pueden ocupar una gran cantidad de espacio, por lo que se debe estimar cuánto espacio pueden ocupar. Esto dependerá del tipo de índice, es decir, si éste es **clustered** o **non-clustered**, el número de índices y el ancho del índice.
- Además de estimar el tamaño de los índices, se deben calcular el tamaño del índice Fillfactor y Pad que son utilizados cuando los índicies son creados. Estos afectan la cantidad de espacio vacío que queda en un índice, y este espacio vacío debe ser considerado en las estimaciones.
- Otro factor que afecta cuanto espacio toma almacenar información en una tabla, es la cantidad de filas que caben en una página de datos de 8K de SQL Server. Dependiendo del tamaño de cada fila, es probable que no todo el espacio en una página de datos sea utilizado. Esto también debe ser contabilizado cuando se estime el tamaño de página.
- Así como las tablas y sus índices asociados ocupan la mayoría del espacio físico en la mayoría de las bases de datos, hay que mantener en mente que cada objeto en SQL Server ocupa espacio por lo que debe ser contabilizado.

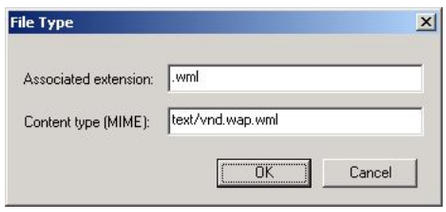
Para mayor información, se recomienda el artículo de MSDN:
"Estimating the Size of a Table".
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/createdb/cm_8_des_02_92k3.asp

ANEXO 11: Configuración Servidor WEB con servicios WML

Configuración del servidor Web, Internet Information Server, para que apoye los servicios WAP.

1. Desde el **Panel de Control**, abrir la carpeta de herramientas administrativas (Administrative Tools).
2. Abrir el administrador de servicios Internet (**Internet Services Manager**). Aparecerá la ventana de Internet Information Services.
3. Hacer click con el botón derecho del mouse sobre el servidor que debe estar WAP-enabled (habilitado para servicios WML).
4. Seleccionar **Properties**.
5. Aparecerá la ventana de propiedades del servidor seleccionado. Dentro de ésta, hay un panel con el siguiente rótulo: **Computer MIME Map**.
6. Presione el botón **Edit...**, aparecerá la ventana rotulada como: **File Types**.





7. Presione **New Type**.
8. Ingrese **.wml** en la caja de texto correspondiente a la extensión asociada.
9. Ingrese **text/vnd.wap.wml** en la caja de texto correspondiente a tipo de contenido (mime).
10. Repetir los pasos del 7 al 9 para cada uno de los siguientes archivos, ver tabla al final.

Al concluir esta serie de pasos se obtendrá el servidor Internet Informaction Server con Servicio WML.

Associated Extension	Content Type (MIME)
.wmlc	application/vnd.wap.wmlscript
.wmls	text/vnd.wap.wmlscript
.wmlsc	application/vnd.wap.wmlscriptc
.wbmp	image/vnd.wap.wbmp

ANEXO 12: Modelamiento de aplicaciones WEB usando UML.

Cuando se intentan modelar aplicaciones Web con el lenguaje de modelamiento unificado, se vuelve aparente que alguno de sus componentes no encajen de una forma natural entre los elementos del modelo estándar de UML. Para poder realizar el modelamiento, es necesario extender dicho lenguaje de modelamiento unificado.

Se entenderá por una aplicación Web un sistema Web, el que típicamente está compuesto por: un servidor Web, una red, el protocolo HTTP, y un navegador de internet o Browser. En dicho sistema, el usuario ingresa información, ya sea a través de la navegación o explícitamente; y ésta afecta el estado del modelo de negocio. Esta breve definición, intenta establecer que una aplicación Web es un sistema software, con estados y que sus interfaces gráficas de usuario se entregan en su mayor parte a través de dicho sistema.

La arquitectura típica de un sistema Web es un sistema servidor. Una de las ventajas más notables de una aplicación Web es su despliegue, ya que éste por lo general, significa configurar algunas variables del servidor.

La diferencia entre una aplicación Web y un sitio Web, incluso uno dinámico, involucra su uso. Las aplicaciones Web implementan la lógica de negocio, y su uso cambia el estado del negocio. Esto es importante, ya que define el objetivo del esfuerzo de modelamiento. Las aplicaciones Web ejecutan la lógica del negocio, y debido a esto, los modelos más importantes del sistema, se centran en la lógica de negocio y el estado del negocio, y no en los detalles de presentación. La presentación es importante, sin embargo, es necesario lograr una clara separación entre ésta y la lógica de negocio. Si los detalles de presentación fueran complejos o relevantes, deberían ser modelados, pero no como una parte integral del modelo de la lógica de negocio. Además, los recursos humanos que trabajan en la presentación, tienden a preocuparse de los aspectos artísticos y no tanto en la implementación de las políticas del negocio.

Una metodología asociada al desarrollo de sistemas Web es RMM (Relationship Management Methodology). Dicha metodología se utiliza en el diseño, construcción y mantenimiento de sistemas basados en Web Internet / Intranet. Consiste en un proceso iterativo que va descomponiendo los elementos visuales en una página Web, y sus asociaciones a entidades de bases de datos. Se centra en reducir el costo de mantenimiento de sitios web dinámicos conducidos por bases de datos.

RMM no es suficiente para el modelamiento de aplicaciones Web, esto debido a que las aplicaciones Web se centran en la lógica del negocio, e involucran una gran cantidad de tecnologías para implementar dicha lógica de negocio. Entre las tecnologías utilizadas frecuentemente se incluyen los script de cliente, tales como applets y controles

ActiveX. Además, las aplicaciones Web se pueden utilizar como un mecanismo de entrega para un sistema de objetos distribuido. Las aplicaciones más sofisticadas, también hacen uso de múltiples instancias de navegador (o Web Browser) o Frames en el cliente, las cuales establecen y mantienen sus propios mecanismos de comunicación. También es necesario considerar los scripts de servidor, que se ejecutan tras el servidor Web, y que pueden implementar gran parte de la lógica de negocio.

Hay que recordar que un sistema software tiene múltiples modelos, cada uno de los cuales representa un punto de vista diferente con un nivel de abstracción y detalle distintos. El nivel de abstracción y detalle adecuados para un modelo, dependerán de los elementos y actividades en el proceso de desarrollo de software.

Los modelos ayudan a entender el sistema simplificando algunos detalles. Las aplicaciones Web, al igual que otros sistemas software, se representan por lo general por un conjunto de modelos, tales como: modelo de casos de uso, modelo de implementación, modelo de despliegue, modelo de seguridad, etc. Un modelo adicional, utilizado exclusivamente por los sistemas web, es el denominado Site Map, el cual es una abstracción de las páginas web y de las rutas de navegación a través del sistema.

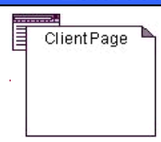
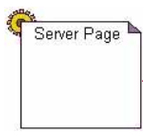
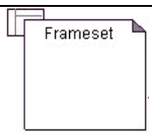
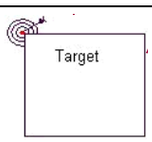
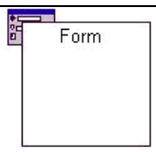
Es importante determinar qué debe ser modelado y qué no. Por ejemplo, *no es importante modelar los detalles internos del servidor ni del cliente*. En cambio, **es importante modelar:** las páginas, sus enlaces, todo el contenido dinámico que participó en elaborar la página, y el contenido dinámico de las páginas una vez en el cliente. Por lo tanto, deberían ser modeladas las páginas, los enlaces, y el contenido dinámico en el cliente y en el servidor.

Es importante asociar estos componentes presentes en los sistemas web a elementos del modelo. Por ejemplo, los enlaces (links) se asocian de una forma natural a los elementos de asociación en el modelo. Los enlaces, representan un camino de navegación entre una página y otra. Las páginas, por ejemplo, se asocian con las clases en una vista lógica del modelo. Entonces, **los métodos de esta clase, se asociarán directamente con los scripts de dicha página**. **Y cualquier variable con visibilidad en los scripts de la página Web se asociará directamente a atributos en la clase que la representa**.

En el modelamiento de páginas WEB, cada **página dinámica**, es decir, cualquier página cuyo contenido se determina en tiempo de ejecución, debe ser construida (build) a través de una **página de servidor** (**<<Server Page>>**). Cada **página de cliente** se construye como mucho con una **página de servidor**, sin embargo, es posible que una **página de servidor** construya muchas **páginas de cliente**.

Es posible concluir que **cada página de servidor representa una clase en la lógica de negocio de la aplicación WEB**, es decir, si se lleva al plano de la implementación, representa, por ejemplo, a una clase escrita en C# que está contenida en un archivo con extensión **cs**. Por otro lado, una página de cliente, contendrá el diseño, el cual se relaciona directamente, en el mismo plano de implementación, con una página en ASP.NET, que está almacenada en un archivo con extensión **aspx**.

A continuación, se exponen los 5 elementos estereotipados más comunes en el modelamiento de aplicaciones WEB, los cuales se asocian todos a una clase determinada. Luego, se describen las asociaciones dirigidas estereotipadas más importantes, y por último se muestran algunos ejemplos de arquitecturas típicas en el modelamiento de aplicaciones Web.

Estereotipo	Vista Lógica	Descripción
 **ClientPage**	«clientpage» **ClientPage**	**Representa a una página de cliente**. Las clases en la vista lógica del modelo, poseen métodos que se asocian con los scripts de la página de cliente y atributos que representan las variables de scripts con visibilidad en esta página.
 **ServerPage**	«serverpage» **ServerPage**	**Representa a una página de servidor**. Las clases en la vista lógica del modelo poseen métodos que se asocian con los scripts de servidor y sus atributos representan variables globales para dichos métodos. Las páginas de servidor se modelan creando relaciones a recursos del servidor (Componentes de capas intermedias, componentes de acceso a base de datos, sistema operativo del servidor, etc).
 **Libro**	«frameset» **Login**	Representa a un elemento del modelo contenedor que se asocia directamente con los FrameSets de HTML. Permite que múltiples páginas estén activas y visibles para el usuario en un momento determinado. Contiene páginas de clientes y Targets.
 **Contenido**	«target» **Contenido**	Es un frame específico o una instancia de un web browser que es referenciada por otra página de cliente. Se referencia a través de una restricción de la forma: {Target=Contenido}, en una asociación dirigida estereotipada como: <<enlace dirigido>> o <<targeted link>>.
 **ClienteNuevo**	«form» **ClienteNuevo**	Representa al mecanismo típico para entrada de datos desde una página Web. Una form puede contener otros elementos de entrada de datos, tales como: <input>[41], <select>, <textarea>

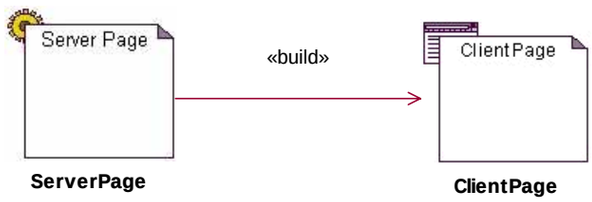
[41] **<input>:** Se encuentra sobrecargado, ya que incluye a los siguientes elementos: text, checkbox, radio button, button, image, hidden, entre otros.

Estereotipos de las asociaciones dirigidas presentes en el modelamiento de aplicaciones Web.

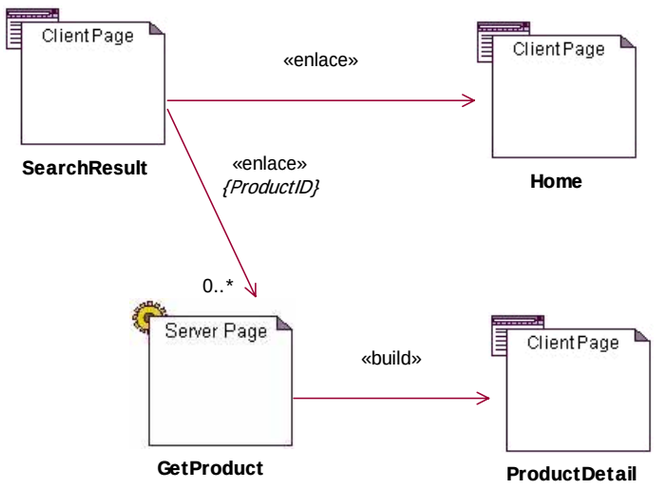
Estereotipo	Descripción
<<build>> o <<builds>>	La asociación dirigida va desde una página de servidor y apunta a una página de cliente. Significa que la página de servidor participa en la construcción de la página de cliente.
<<enlace>>, <<link>> o <<links>>	La asociación dirigida siempre va desde una página de cliente a otra página de cliente o a una página de servidor. Significa que existe un enlace o camino de navegación entre dichos elementos. Se pueden pasar parámetros a través de un enlace.
<<enlace dirigido>> o <<targeted link>>	La asociación dirigida implica que la página a la que se apunta será desplegada en el elemento Target al cual se hace referencia a través de la restricción: {Target=Contenido} .
<<redirects>>	Asociación dirigida que permite que una página de servidor redireccione el flujo de procesamiento a otra página de servidor.
<<submits>>	Asociación dirigida que permite enviar la información recolectada por los elementos de una form a una página de servidor.

Ejemplos de arquitecturas típicas en el modelamiento de aplicaciones WEB

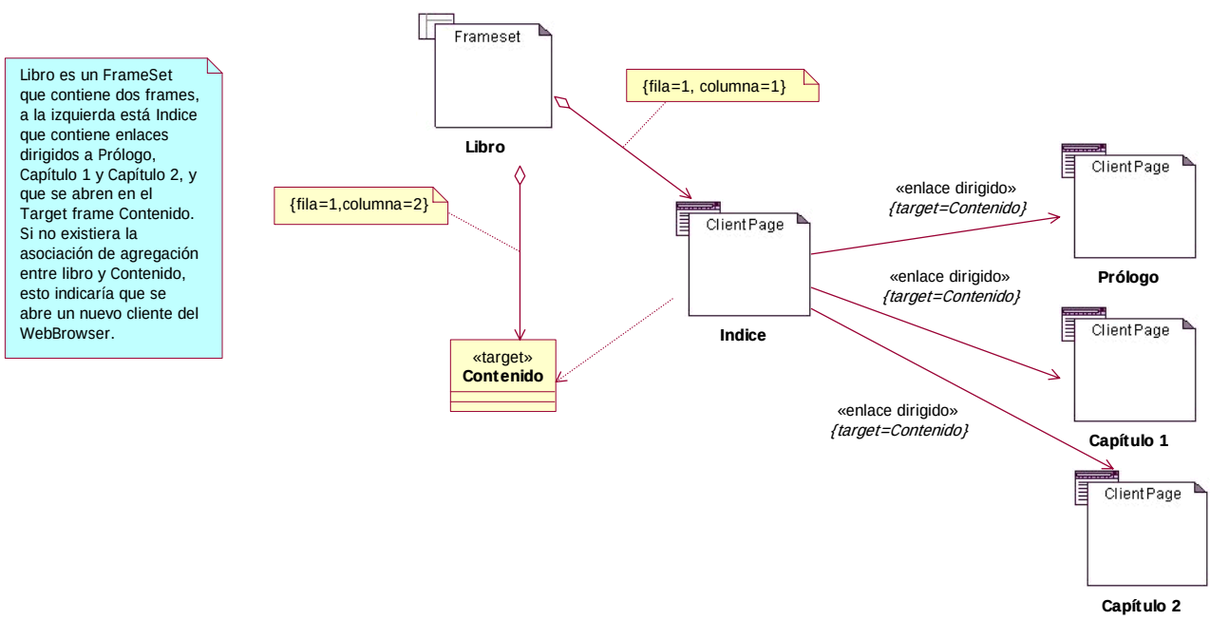
ServerPage Construye ClientPage



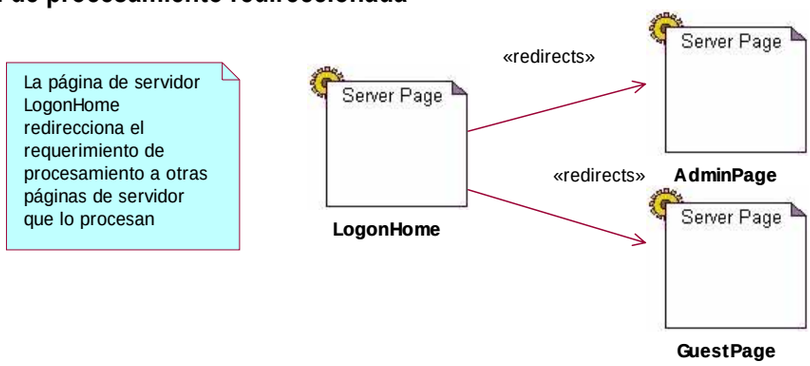
Interacción Entre un ClientPage y un ServerPage



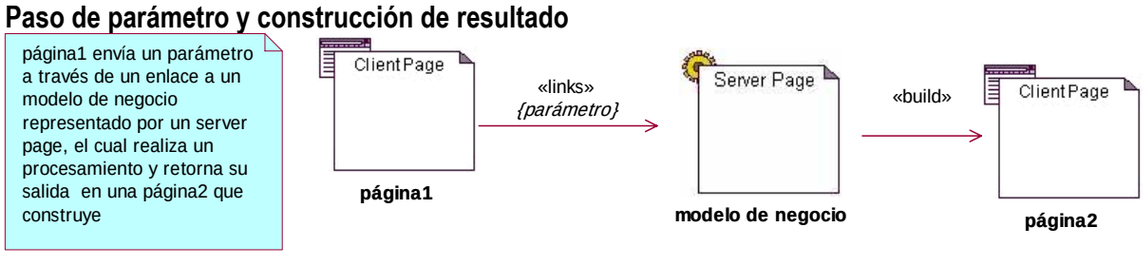
Página Web con dos Frames verticales



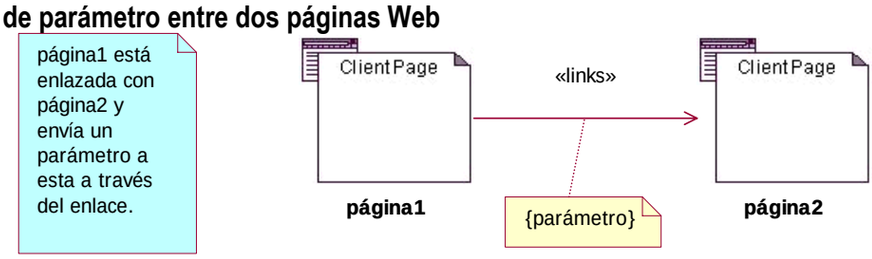
Solicitud de procesamiento redireccionada



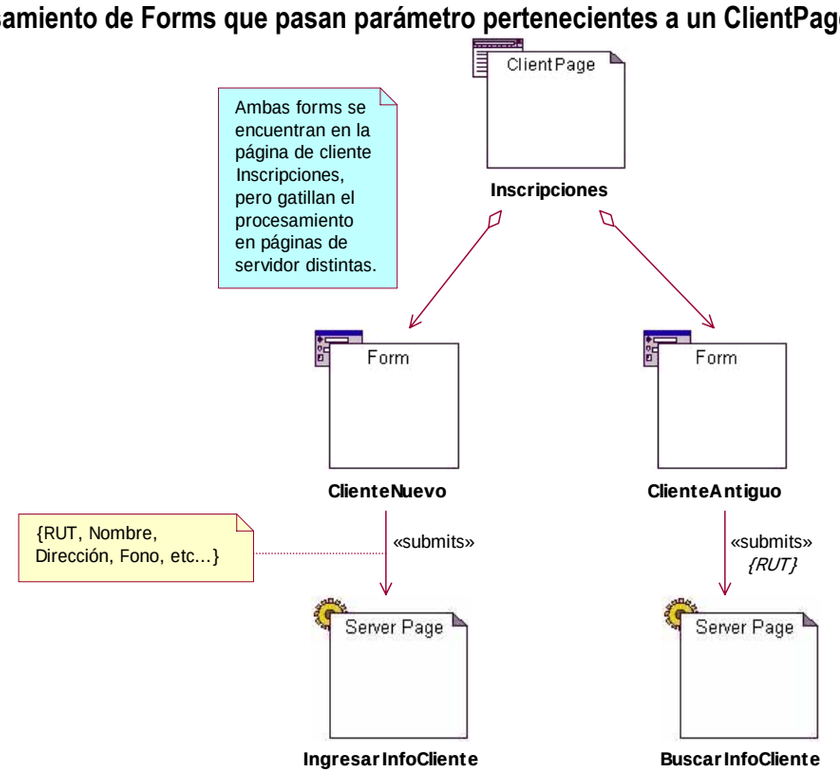
Paso de parámetro y construcción de resultado



Paso de parámetro entre dos páginas Web



Procesamiento de Forms que pasan parámetro pertenecientes a un ClientPage



ANEXO 13: Carta Gantt primera iteración prototipo REM

Planificación para la primera iteración del proceso de desarrollo del prototipo REM 5

Carta Gantt: (En documento impreso. Biblioteca Miraflores, Universidad Austral de Chile)