UNIVERSIDAD NACIONAL DE ASUNCIÓN

FACULTAD POLITÉCNICA

MIDS: MODELO INICIAL DE DESARROLLO DE SOFTWARE

UNA METODOLOGÍA ÁGIL COMO INICIACIÓN EN EL USO DE MODELOS DE PROCESOS DE SOFTWARE

ENRIQUE BAÑUELOS GÓMEZ

2007

UNIVERSIDAD NACIONAL DE ASUNCIÓN

FACULTAD POLITÉCNICA

MIDS: MODELO INICIAL DE DESARROLLO DE SOFTWARE

UNA METODOLOGÍA ÁGIL COMO INICIACIÓN EN EL USO DE MODELOS DE PROCESOS DE SOFTWARE

ENRIQUE BAÑUELOS GÓMEZ

Proyecto de Trabajo de Grado presentado en conformidad a los requisitos para obtener el grado de Ingeniero en Informática.

Profesor Tutor: Prof. M. Sc. Luis Gilberto Salinas.

San Lorenzo, 2007

Agradecimientos

En primer lugar, a los Familiares y Amigos que han brindado su apoyo incondicional, a nivel personal, a lo largo del desarrollo del presente trabajo.

Al Profesor Tutor Luis Gilberto Salinas, por su esfuerzo y dedicación como orientador y revisor de esta tesis.

A los compañeros Alberto Amadeo, por proveer un servidor público de gestión de versiones para el Caso de Estudio; y Ellen Méndez, por haber trabajado en el desarrollo del mismo, realizando un aporte de vital importancia.

Por último, a todas las personas que ayudaron en la realización de la encuesta sobre el estado del desarrollo de software en el país. Incluyendo a las personas que respondieron a los cuestionarios, como a las que ayudaron a difundir los mismos.

Resumen

El desarrollo de software ya tiene años de trayectoria como una actividad comercial en el país. Sin embargo, hasta ahora no se han realizado esfuerzos para evaluar la situación de los mismos en comparación a estándares de calidad del área.

El presente trabajo, como uno de sus aportes, presenta una evaluación de los entornos de desarrollo de software locales acerca de prácticas básicas de Ingeniería de Software que se utilicen actualmente. En base al resultado de esta evaluación, como segundo aporte, se propone el "Modelo Inicial de Desarrollo de Software MIDS" como un modelo de procesos de software sencillo que pueda ser adoptado por las organizaciones locales y que permita la mejora de sus procesos.

El MIDS se basa en Metodologías Ágiles y propone un conjunto de prácticas y documentaciones mínimo como para poder ser implementado en entornos que no tienen definidos sus procesos, pero que a su vez introduzcan la disciplina necesaria para el desarrollo de software. Sus principales ventajas son el establecimiento de una metodología de trabajo que incluye prácticas de desarrollo y mecanismos de interacción con los clientes, utilización de programación en pares para determinadas situaciones, pruebas de unidad, gestión de versiones y estándares de codificación.

Para validar las ventajas que el MIDS ofrece, se realizó un caso de estudio aplicando el modelo propuesto al desarrollo de un software de gestión.

Abstract

Software development already has years of trajectory as a commercial activity in the country. However, until now efforts have not been made to evaluate the situation of such in comparison to standards of quality of the area.

The present work, as one of its contributions, presents an evaluation of the local environments of software development, about basic practices of Software Engineering that are used at the moment. Based on the results of this evaluation, as second contribution, the "Initial Model for Software Development MIDS" is proposed as a simple software process model that can be adopted by local organizations allowing the improvement of its processes.

The MIDS is based on Agile Methodologies and proposes a minimum set of practices and documentations, being able to be implemented in environments that they do not have defined his processes, and introducing as well the necessary discipline for software development. The main advantages are the establishment of a work methodology that includes practices of development and mechanisms of interaction with the clients, use of pair programming for certain situations, unit testing, version management and the use of coding standards.

In order to validate the advantages that the MIDS offers, a case of study was made applying the proposed model to the development of a management software.

Tabla de Contenido

Lista de Tablas y Figuras

FIGURAS:

TABLAS:

Capítulo 1
Introducción

En nuestro país son cada vez más las entidades que cuentan con la necesidad de sistemas de software que automaticen el manejo de su información, lo que conlleva a una mayor demanda de productos y a su vez a una mayor cantidad de empresas que se dedican a desarrollar sistemas.

Sin embargo, la simple presencia de un sistema informático no es garantía de una mayor eficiencia en las actividades de una entidad. Es importante cuestionar el nivel de calidad de los sistemas desarrollados para que realmente puedan ser útiles para las entidades que deseen implementarlos. Uno de los factores que determina la calidad de un producto software es el *proceso* que se siguió para la producción del mismo.

1.1 Conceptos Básicos

Un proceso de software es un conjunto de actividades y resultados asociados que producen un producto software [Sommerville2005]. Básicamente, existen cuatro actividades comunes para cualquier proceso de software que son:

- *Especificación del software* donde se definen las características y restricciones del producto a ser desarrollado.
- *Desarrollo del software* donde se realiza el diseño y codificación.
- *Validación del software* donde se realizan diversos tipos de pruebas para asegurar que el software cumple con lo especificado.
- *Evolución del software* que abarca los cambios que se realizan sobre el software para adaptarlo a los cambios requeridos por los clientes o por otros factores.

Por otro lado, la calidad del software puede definirse como la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente.

En esencia, un software debe ser *usable*, *confiable*, *eficiente* y *mantenible*. Por lo tanto, para poder producir un software de calidad, deben definirse cómo se llevarán a cabo las actividades definidas y cómo se interrelacionarán, de manera que el mismo posea los atributos citados.

Es allí donde juega su papel la Ingeniería del Software, que es la que se encarga de la aplicación de enfoques sistemáticos, disciplinados y cuantificables para el desarrollo, operación y mantenimiento del software [IEEE1990].

La misma define *modelos de procesos* de software como representaciones abstractas de procesos de software. Pueden incluir definición de las actividades, el orden entre las mismas, los resultados de cada etapa, y los roles y responsabilidades. La mayoría de los modelos de procesos se basan en tres modelos generales o paradigmas de desarrollo que son el *enfoque en cascada*, el *enfoque de desarrollo iterativo* y la *Ingeniería del Software Basada en Componentes (CBSE)* [Sommerville2005].

1.2 Planteamiento

Cabe señalar que un modelo de procesos nunca es adecuado para todos los escenarios de desarrollo. Los tres enfoques generales mencionados anteriormente tienen ventajas que los hacen ideales para ciertas situaciones, así como también presentan desventajas para otras.

Entonces, es importante describir las características de la situación actual del desarrollo de software en nuestro país, para poder así decidir que paradigma o modelo de procesos es el más aplicable a los proyectos de desarrollo locales.

Para poder caracterizar la situación, es importante separar a la misma en dos partes. La primera está relacionada con los productos desarrollados, y la otra con el proceso de desarrollo.

Analizando lo relativo al producto, se pueden observar que la mayoría de los sistemas desarrollados a nivel local son de gestión, contabilidad, o dicho de manera más genérica, sistemas de Planeamiento de Recursos Empresariales (ERP por sus siglas en inglés); normalmente desarrollados a medida. Como estos sistemas están sujetos a las empresas que los solicitan, o los desarrollan, y éstas a su vez se desenvuelven en un ambiente de

negocios cambiantes, es lógico suponer que los requerimientos serán tan variables como el entorno mismo en el cual se desempeñarán.

En cuanto a lo que respecta al proceso, lo importante es evaluar que tan disciplinado y medible es el proceso de desarrollo. Verificar de qué manera, o hasta qué punto están definidas las actividades a lo largo del proceso de desarrollo. De la misma manera, ver cómo están definidos los resultados de cada actividad, y cómo éstos son utilizados en las actividades posteriores. Por último, también se debe evaluar la definición de los roles y responsabilidades, así como la interacción entre los mismos.

El resultado de este análisis, en sus dos partes, debería arrojar una serie de rasgos que deben ser contrastados con las ventajas y desventajas de los distintos tipos de modelos de procesos, para poder evaluar la aplicabilidad de los mismos a la realidad de nuestro país.

Por lo tanto, con este trabajo se pretende contribuir a la mejora de procesos en nuestro país, en base a la propuesta de un modelo de procesos que permita una planificación realista, y lograr un producto de calidad en base a un proceso de desarrollo disciplinado y medible.

1.3 Objetivos

En base a lo expuesto como planteamiento, se pueden extraer los siguientes objetivos de este trabajo de tesis.

1.3.1 Objetivo Principal

Presentar un modelo de procesos capaz de adecuarse a la realidad del desarrollo de software en nuestro país, de manera a lograr que los productos desarrollados sean de calidad, y que el proceso de desarrollo sea lo suficientemente disciplinado para lograr cumplir razonablemente con los tiempos de entrega.

1.3.2 Objetivos específicos

- Realizar una investigación y una posterior comparación entre distintos paradigmas de desarrollo.

- Comparar las características (ventajas y/o desventajas) de cada paradigma con los rasgos de los entornos de desarrollo del país, para definir el paradigma más apropiado.

- Proponer una metodología o modelo de procesos, basado en el paradigma seleccionado, que tenga en cuenta todas las características de nuestros entornos de desarrollo de software, para que la misma pueda ser adoptada por nuestras empresas que se dedican al desarrollo de software.

- Mejorar la calidad de los productos de software desarrollados en el país, en base a la aplicación de la metodología propuesta.

- Definir detalladamente las actividades, resultados de cada actividad, interrelación entre las mismas, y los roles y responsabilidades del modelo, para que su implementación lleve a las organizaciones a tener un proceso disciplinado de desarrollo del software.

- Lograr un equilibrio entre rigurosidad y flexibilidad, de manera a lograr que la metodología propuesta pueda ser adoptada por empresas que no tengan definidos sus procesos.

- Servir como primer paso hacia la adopción de modelos de procesos, y que esto pueda permitir con el tiempo, que las empresas aspiren a certificaciones internacionales de calidad del software.

1.4 Hipótesis

En base a los objetivos, se puede construir la hipótesis del presente trabajo, por analogía sustantiva [Sierra2003], de la siguiente manera. Sean

- **ME** los modelos de procesos existentes para el desarrollo de software.

- **MP** el modelo de procesos propuesto.

- **PS** proyectos de desarrollo software.

- **E** un entorno de desarrollo.

Si distintos **ME** dan buenos resultados para distintos **PS** y **E** de acuerdo a sus características respectivamente, **entonces**, la combinación de las características de los **ME** en un **MP** dará también un buen resultado para **PS** en **E** locales, siempre que la combinación se haya hecho teniendo en cuenta las características de los **E** locales.

1.5 Metodología Utilizada

"*Se entiende por metodología al proceso de investigación que permite sistematizar los métodos y las técnicas a utilizar, facilitando el descubrimiento de conocimientos seguros y confiables para la resolución de problemas*" [Martínez2007a].

La metodología utilizada para este trabajo es el *método de investigación científica* que puede verse como una serie ordenada de operaciones [Sierra2003]:

- Enunciar preguntas bien formuladas.
- Formular conjeturas contrastables con la experiencia, para responder a las preguntas.
- Derivar consecuencias lógicas de las preguntas.
- Seleccionar las técnicas para contrastar las conjeturas formuladas.
- Someter a su vez a contrastación las técnicas seleccionadas para comprobar su relevancia y veracidad.
- Llevar a cabo la contrastación e interpretar sus resultados.
- Determinar los dominios en los cuales valen las conjeturas y las técnicas.
- Formular nuevos problemas generados por la investigación.

Este método tiene su punto de partida en bases teóricas previas, descubriendo luego problemáticas de la realidad para formular conjeturas (hipótesis) de posibles soluciones. Mediante la experiencia, se toman los datos suficientes para lograr conclusiones sobre las conjeturas realizadas previamente, en base a inducciones y/o deducciones.

La metodología planteada es la recomendada para este tipo de trabajos [Sierra2003], ya que como se puede apreciar, la serie de etapas que propone permite llegar a conclusiones válidas, en un dominio establecido, a partir de preguntas y/o problemas, y conjeturas que respondan a dichos problemas, con la experiencia como aval de dichas hipótesis.

Por lo tanto, en base a lo expuesto en un principio como fundamentación y a las características de la metodología, se adopta el *método de investigación científica* para este trabajo, ya que su correcta utilización podrá darle la credibilidad suficiente que se necesita.

1.6 Estructura del Documento

El presente trabajo tendrá dos partes básicas. La primera se encuadra en lo que es la base teórica de la propuesta incluyendo los capítulos 2 y 3. La segunda parte tomará como base los resultados de la primera para efectivamente describir la metodología propuesta.

Dentro de la primera parte, en el capítulo 2, se verá el estudio de los tres paradigmas ya mencionados. En el capítulo 3 se analizará a un paradigma "innovador" que son las llamadas "Metodologías Ágiles", contrastándolo con los paradigmas del capítulo 2.

Dentro de la segunda parte, en el capítulo 4 se mostrarán la aplicabilidad de los paradigmas de desarrollo expuestos en la primera parte a los entornos de desarrollo locales; para dar así una visión general de la propuesta. En los capítulos 5, 6 y 7 se detallarán cada una de las fases del modelo propuesto; en el capítulo 8 se describirá el desarrollo utilizado como Caso de Estudio, y por último, en el capítulo 9 se presentarán las Conclusiones y Trabajos Futuros.

Capítulo 2
Paradigmas de Desarrollo de Software

Como se había definido previamente, un modelo de proceso es una representación abstracta de un proceso de software. Define cómo se llevan a cabo las actividades necesarias para producir software y cómo se relacionan entre sí para determinar lo que se conoce como el ciclo de vida del software [Ghezzi1991].

Así también, se mencionó lo cambiante que puede ser el proceso de desarrollo de software, debido a diversos motivos como cambios en los requerimientos, en las tecnologías, en el personal, etc. Esto hizo que a lo largo del tiempo, surjan diversos enfoques para el desarrollo del software.

En este capítulo se describirán tres modelos generales (o paradigmas). Estos son:

- El modelo en cascada (*Waterfall*): Representa a las actividades fundamentales del desarrollo de software como etapas claramente separadas, y con un flujo lineal entre las mismas.

- Desarrollo Evolutivo: Entrelaza las actividades de especificación, desarrollo y validación. Se crea una versión inicial en base a las especificaciones, y luego el software se va refinando basándose en las peticiones del cliente, hasta que el resultado satisfaga al cliente.

- Ingeniería de Software Basada en Componentes (*CBSE*): Se basa en la existencia de componentes reutilizables, y se enfoca en la integración de los mismos para producir software, antes que en el desarrollo de cero de software.

Estos tres modelos de procesos genéricos se utilizan ampliamente en la práctica actual. No se excluyen mutuamente, y a menudo pueden utilizarse juntos según el tipo de proyecto que se esté llevando a cabo [Sommerville2005]. Justamente, al terminar el capítulo, se verá el *Rational Unified Process (RUP)*, como un ejemplo de modelo híbrido.

2.1 El Modelo en Cascada (*Waterfall*)

El modelo en cascada se volvió popular en la década de 1970, y permanece en los libros de Ingeniería de Software actuales y en las prácticas industriales estándares [Ghezzi1991]. Es más, en la última versión de la Guía del Cuerpo de Conocimiento de la Ingeniería de Software (*SWEBOK*) [IEEE2004], las cinco primeras áreas de conocimiento están organizadas en la misma secuencia que un "flujo tradicional en cascada", evidenciando la vigencia del modelo.

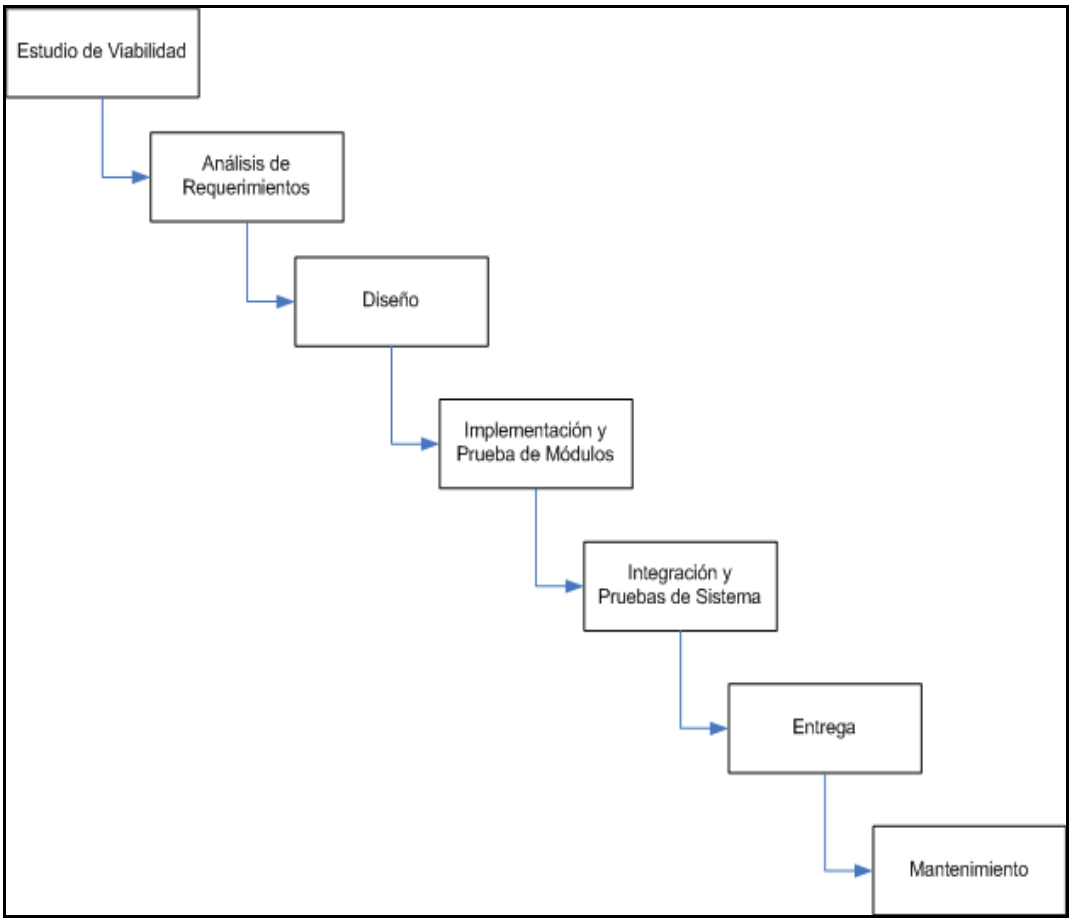

Fig. 2.1. Proceso en Cascada. [Ghezzi1991]

Como se puede ver en la figura, el proceso está estructurado como una cascada de fases donde las salidas de una fase constituyen las entradas de la siguiente. Dichas salidas son normalmente documentaciones que sirven como guía para el planeamiento y desarrollo de

la fase posterior. El proceso de la figura es un ejemplo representativo, ya que el modelo puede tener variantes de acuerdo al proyecto o a la organización que lo implemente.

Las organizaciones que decidan adoptar el modelo deben definir claramente las salidas de cada etapa, y también métodos para producir dichas salidas. Independientemente de cómo se realice esta definición, la filosofía es siempre la del flujo lineal de las fases identificadas (ejemplificadas en la Fig. 2.1).

En base al ejemplo, se describirán brevemente las mismas para tener una idea más clara de cómo se desarrolla un proceso en cascada.

2.1.1 Estudio de Viabilidad

Esta fase representa al estudio previo al proyecto en el cual se determinan los costos y beneficios del desarrollo del mismo. Este estudio puede ser más, o menos crítico, dependiendo de la naturaleza del proyecto; pero en todos los casos, la meta de esta fase es la producción de un documento que detalle los resultados obtenidos.

Dicho documento debería contener:

- Una definición del problema,
- Alternativas de solución con sus respectivos beneficios, y
- Los recursos necesarios, los costos, y las posibles fechas de entrega para cada una de las alternativas.

Es aquí donde los ingenieros deben hacer valer sus conocimientos y experiencias, ya que por un lado, este análisis no debe durar mucho; pero por otro lado, mientras más tiempo se invierta en el análisis, mejores estimaciones podrán hacerse.

2.1.2 Análisis de Requerimientos

En esta etapa se identifican las cualidades que debe tener la aplicación, y se redacta un documento que especifica los resultados del análisis de requerimientos.

Las cualidades mencionadas anteriormente, pueden clasificarse de la siguiente manera [Ghezzi1991]:

- *Requerimientos Funcionales:* Que describe lo que el sistema debe hacer.

- *Requerimientos No Funcionales:* Son los que se refieren al rendimiento, confiabilidad, robustez, calidad de las interfases gráficas, entre otros.

- *Requerimientos del Proceso de Desarrollo y Mantenimiento:* Como controles de garantía de calidad, o el establecimiento de prioridades sobre los requisitos funcionales.

La manera en la que se especifican los requerimientos es relativa a la organización. Se deben definir estándares sobre la forma y contenido del documento, notaciones a utilizar y validaciones sobre el mismo.

2.1.3 Diseño

Una vez más, el propósito principal de la tercera etapa dentro del modelo *Waterfall*, es la producción de un documento con la descripción de la arquitectura del sistema.

El mismo debería abarcar algunos, o todos los siguientes ítems de diseño [Sommerville2005] y [Ghezzi1991]:

- *Diseño arquitectónico:* Descripción modular del sistema en términos de composición e interrelación.

- *Especificación abstracta:* Para cada módulo, se produce una especificación que incluya, por ejemplo, los servicios que ofrece, las restricciones para su funcionamiento, y cualquier otra información pertinente sobre el módulo.

- *Diseño de la interfaz:* Es un diseño complementario a la especificación abstracta, ya que en este caso, se diseña y documenta la interfaz de cada módulo con los demás.

- *Diseño de componentes:* Se asignan los servicios dentro de cada módulo, a componentes del mismo y luego se diseñan las interfases de los mismos.

- *Diseño de las estructuras de datos y de algoritmos:* Se documentan las estructuras de datos a ser utilizadas en el sistema y por último, los algoritmos que implementan los servicios especificados.

De acuerdo a lo que se esté especificando y al tipo de sistema a desarrollar, se puede elegir entre una diversa variedad de notaciones de diseño para documentar los resultados de esta etapa. Se pueden citar como ejemplos: el Lenguaje Universal de Modelado (UML), Diagramas de Flujos de Datos (DFD), Máquinas de Estados Finitos (FSM), Redes de Petri, Diagramas de Entidad Relación (ER), entre otras.

2.1.4 Implementación y Prueba de Módulos

En el modelo en Cascada, esta es la primera etapa en donde se codifica. A diferencia de las anteriores etapas, la salida en esta etapa es un conjunto de módulos probados [Ghezzi1991].

Los módulos son implementados siguiendo cuidadosamente las especificaciones citadas en la etapa anterior, y dependiendo de la organización, puede estar definido el uso de estándares generales de codificación y pruebas, así como también la creación de estándares propios que se adecuen a las características de la organización.

2.1.5 Integración y Pruebas del Sistema

Una vez concluida la construcción de los módulos, se procede a integrarlos en base a las interrelaciones entre los mismos, identificadas en la fase de diseño. Esto no se lleva a cabo de una manera repentina (conocida como pruebas de Big-Bang), sino incremental o progresivamente [Ghezzi1991].

Sin embargo, es importante tener en cuenta que en algunos casos, esta etapa no estará separada de la anterior ya que de acuerdo a los resultados del diseño, un módulo puede necesitar de otros dos (por ejemplo) para poder ser probado efectivamente. Esto a su vez, está realizando implícitamente pruebas de integración sobre los tres módulos.

Independientemente de la separación de esta etapa con la anterior, luego de finalizar las pruebas de integración, se realizan pruebas sobre el sistema completo, que es considerado como la salida de esta etapa.

2.1.6 Entrega

En esta etapa, se realizan las pruebas de aceptación por parte de los usuarios, para que una vez aprobadas las mismas, el sistema sea puesto en producción.

2.1.7 Mantenimiento

A partir de la puesta en marcha, o utilización real del sistema, pueden surgir motivos por los cuales sea necesario modificar el sistema. Básicamente, los motivos pueden ser corregir errores que no fueron detectados anteriormente en el sistema, adaptar al sistema a cambios que hayan surgido en el ambiente en donde se desenvuelve, y mejorar o agregar funcionalidades al sistema.

El mantenimiento es el conjunto de actividades realizadas después de la implementación del sistema para realizar los cambios necesarios sobre el mismo.

2.1.8 Análisis del Modelo en Cascada

Como este modelo es el primero en haber surgido formalmente, será analizado como para que los demás modelos a presentarse puedan ser contrastados con este modelo inicial.

Sus dos principales aportes son los siguientes [Ghezzi1991]:

- Imponer disciplina, planeamiento y gestión al desarrollo del software.
- La filosofía de que el desarrollo del software debe ser pospuesto hasta que los objetivos estén bien definidos y entendidos.

Además, introdujo el concepto de documentación del software, ya que las salidas de varias de sus etapas son documentos de especificación para la siguiente etapa. Por último, como el modelo en cascada fue el que identificó todas las etapas involucradas en el desarrollo del software, sigue siendo utilizado como referencia o punto de partida (como es el caso de este trabajo) para el estudio de los procesos de software.

Así como realizó aportes, el modelo en cascada también tiene sus limitaciones. La disciplina del modelo está basada en su rigidez de etapas. Esto se refiere a que una etapa no comienza hasta que haya concluido completamente la anterior. Como resultado de esta característica del modelo, surgen las siguientes dificultades [Ghezzi1991]:

- Estimación precisa de uso de recursos.

- Por más precisa que sea la especificación inicial, o lo fielmente que se haya implementado el software en base a las especificaciones, no se puede garantizar que el mismo cumpla con las expectativas de los clientes con el paso del tiempo que dure la implementación.

- En las etapas del modelo, no se hace énfasis en el diseño para el cambio, lo que dificulta su utilización en la implementación de sistemas que se desenvuelven en ambientes que pueden ser cambiantes (situación que se da habitualmente).

- Por más que el hecho de la introducción del concepto de documentación es algo positivo, el exceso puede hacer que el proceso de desarrollo se torne burocrático.

En base a estas características positivas y negativas del modelo, se presentarán los demás paradigmas de desarrollo de software.

2.2 Desarrollo Evolutivo

En base a las limitaciones encontradas sobre el modelo en cascada, sobre todo el largo tiempo entre la especificación y la entrega, surge la idea de que el desarrollo de software debe consistir en una serie de pasos que van expandiendo incrementalmente la funcionalidad del mismo. La dirección que va tomando el proceso de desarrollo se va determinando mediante la experiencia de uso de cada incremento [Ghezzi1991]. A este tipo de modelos de procesos se lo conoce como enfoques incrementales o evolutivos.

Además, se debe tener en cuenta que existen diversos factores que harán que los requerimientos cambien constantemente a lo largo del ciclo de vida del software como el surgimiento de nuevas funcionalidades requeridas, o pedidos de cambios de lo que ya se implementó. Estos son ejemplos de situaciones que no solo cambian al sistema final, sino que también afectan la gestión misma del proyecto.

Otro factor de ejemplo que no necesariamente involucra a los clientes es el cambio de tecnologías utilizadas en el desarrollo que afectan al proceso en sí como herramientas de diseño, lenguajes de programación, y otros [Sommerville2005]. Por lo tanto, existen

suficientes razones para adoptar un enfoque del tipo evolutivo, ya que este enfoque permite, mediante sus pasos o ciclos, adaptarse mejor a los posibles cambios.

Básicamente, un enfoque evolutivo (como se puede ver en la Fig. 2.2), consiste en tres etapas que van repitiéndose cíclicamente hasta que se logre una conformidad entre las partes. En la primera, se realiza un planeamiento de lo que se entregará en el incremento de software. En el caso del primer ciclo, este planeamiento podría incluir bosquejo de planificación de todo el proyecto.

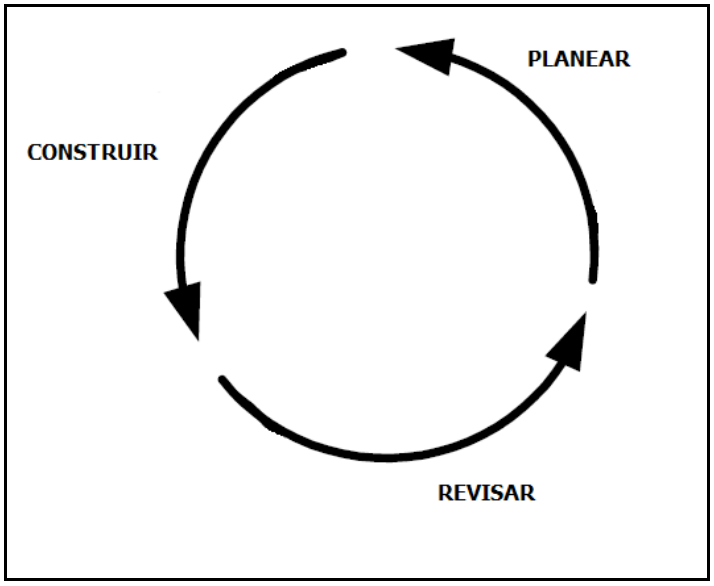

Fig. 2.2. Ciclo Evolutivo. [Highsmith1997]

La segunda fase es la implementación del incremento, siendo la tercera la de revisión. En esta tercera etapa es en donde, luego del uso del incremento por parte de los clientes, se miden los beneficios y el rendimiento en general para que estos resultados sean la base del planeamiento de la siguiente etapa.

Es importante destacar, que este tipo de ciclos podría llevar a un desarrollo caótico si no se tiene cuidado de mantener la disciplina y alguna noción de lineamiento general del proyecto. Por lo tanto, es importante utilizar técnicas de gestión que garanticen el desarrollo controlado del proyecto.

Como ejemplos de modelos de procesos basados en el enfoque evolutivo, pueden mencionarse el Desarrollo Incremental y el Modelo en Espiral. Además, como se verá en el siguiente capítulo, el enfoque evolutivo sirve como base del enfoque conocido como Metodologías Ágiles.

2.3 Ingeniería de Software Basada en Componentes (*CBSE*)

Este paradigma está basado en una de las características deseables del software: la reutilización. Los sistemas se construyen en base a componentes diseñados para ser reutilizables o sistemas por sí mismos (COTS – *Components of the Shelf*).

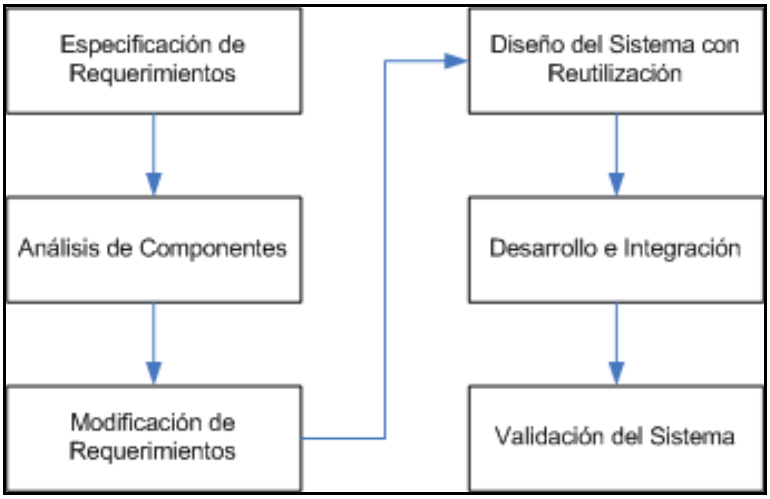

Fig. 2.3. Proceso CBSE. [Sommerville2005]

Las etapas de especificación de requerimientos y validación del sistema pueden verse de manera similar a otros paradigmas. En donde se distingue este paradigma es en las etapas intermedias [Sommerville2005]:

- *Análisis de Componentes:* Es la etapa en la que se comparan los requerimientos de la primera etapa con los componentes que puedan cumplir las funcionalidades.

- *Modificación de Requerimientos:* Se revisan los requerimientos en base a la comparación de la etapa anterior, puesto que las funcionalidades de los componentes no concordarán totalmente con los requerimientos iniciales. Se analiza entonces, la posibilidad de adaptar los requerimientos a los componentes existentes. Si no es posible la modificación, se buscan soluciones alternativas.

- *Diseño del Sistema con Reutilización:* Se diseña o reutiliza un marco de trabajo para combinar los componentes que serán utilizados.

- *Desarrollo e Integración:* Lo importante en esta etapa es que las dos actividades no se toman como separadas, sino que se realizan de manera conjunta. Parte del desarrollo mismo del sistema, es la integración de COTS, u otros componentes.

Este enfoque ofrece principalmente, la ventaja de reducir el desarrollo, y a su vez, los costos, riesgos y tiempos. Como desventaja de este enfoque, se puede mencionar que las modificaciones en los requerimientos para adaptarse a los componentes podría dar como resultado, un sistema que no cumpla con los requerimientos reales del usuario [Sommerville2005].

2.4 Rational Unified Process (RUP)

Como se mencionó al comienzo del capítulo, el RUP se incluye como un ejemplo de modelo híbrido, ya que combina ideas algunos de los paradigmas mencionados anteriormente.

El RUP es un modelo de procesos para sistemas Orientados a Objetos diseñado como complemento al UML [Kruchten2000]. Posee cuatro fases principales que pueden verse en la Fig. 2.4.

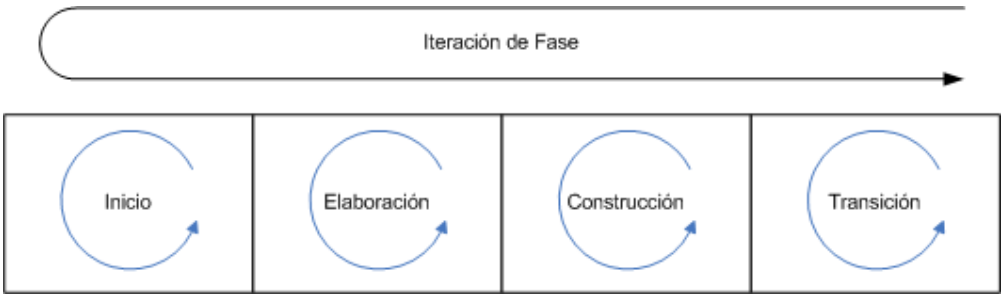

Fig. 2.4. Fases del RUP. [Sommerville2005]

- *Inicio:* Se establecen los objetivos del proyecto y se definen las necesidades de todas las personas involucradas en el proyecto (*stakeholders*). También pueden considerarse ya en esta etapa los casos de uso críticos, posible arquitectura del sistema y estimaciones preliminares de tiempo y costos [Abrahamsson2002].

- *Elaboración:* En esta etapa se establece formalmente la arquitectura del sistema. En base a esta definición, se estabilizan los estudios previos sobre los costos, tiempos y riegos del sistema para obtener como resultado el plan del proyecto.

- *Construcción:* Se llevan a cabo el diseño, la programación y las pruebas. Al final de esta etapa se tiene como resultado un sistema operativo y con la documentación necesaria como para ser entregada a los usuarios finales [Sommerville2005].

- *Transición:* Basada en las respuestas de los clientes, se realizan entregas subsecuentes para corregir problemas, agregar funcionalidades o terminar funcionalidades que fueron pospuestas por algún motivo en la fase previa [Abrahamsson2002].

Las etapas dentro del RUP no son muy distintas (si son seguidas secuencialmente) a las de un modelo en cascada. Sin embargo, RUP utiliza iteraciones en dos niveles. Dentro de cada fase los resultados pueden obtenerse de forma iterativa. Además, el conjunto entero de las fases puede ser visto como un proceso iterativo [Sommerville2005].

Además de la especificación del proceso, el RUP cita buenas prácticas de ingeniería de software que deben utilizarse durante el proceso [Sommerville2005]:

- *Desarrollo iterativo:* El software debe realizarse en cortos incrementos para tratar de detectar riesgos lo antes posibles.

- *Gestión de requerimientos:* Se debe documentar los requerimientos del cliente y mantener actualizados los mismos según haya cambios. También se debe realizar análisis de impacto antes de aceptar cambios.

- *Arquitecturas basadas en componentes:* Intentar encapsular módulos en componentes de manera que sean más fáciles de manejar. Además, un buen diseño puede arrojar como resultado componentes reutilizables que reducirán el trabajo en futuros desarrollos.

- *Modelado visual del software:* Se recomienda la utilización de modelos gráficos como el UML para el diseño de vistas estáticas y dinámicas del software.

- *Verificación de calidad del software:* Mediante pruebas en cada iteración, se debe asegurar que el software cumple con todos los estándares organizacionales.

- *Control de cambios:* Se deben manejar los cambios mediante herramientas de gestión de este tipo. Mediante esta gestión se pueden obtener datos útiles sobre el ciclo de vida del software, que sirven para determinar la madurez del mismo en base a la frecuencia y tipos de cambios [Abrahamsson2002].

2.5 Resumen

En este capítulo, se pudo ver a grandes rasgos tres enfoques o paradigmas de desarrollo de software, analizando sus principales características y sus posibles ventajas y/o desventajas resultantes de su uso. Además, se presentó al RUP como modelo que intenta combinar las ventajas de estos tres enfoques para lograr un paradigma de desarrollo, que aunque no sea aplicable a todo tipo de proyectos, realizó aportes de importancia a la ingeniería de software, como por ejemplo la inclusión de la utilización del software por parte de los usuarios como parte del proceso [Sommerville2005].

Así, es posible tener una idea más clara de lo que es la ingeniería del software y de cómo se modelan los procesos de software, con el objetivo de lograr un producto de calidad mediante un proceso de desarrollo medible y controlable.

Capítulo 3
Metodologías Ágiles (MAs)

Los tres paradigmas comunes de desarrollo de software analizados en el capítulo anterior: modelo en cascada, desarrollo evolutivo y CBSE, serán llamados en adelante como Modelos o Metodologías Tradicionales (MTs).

Luego del análisis sobre las MTs, se pueden extraer características comunes como el planeamiento estricto, la clara separación de sus etapas, rigurosa definición de roles y la documentación detallada de los resultados.

Las mencionadas características hacen que las MTs sean efectivas en proyectos de gran tamaño en tiempo y recursos. Sin embargo, no se adecuan a proyectos con cambios de requerimientos constantes o a aquellos que exigen tiempos de entrega cortos, escenarios que son comunes en la actualidad. Esto motivó el surgimiento de modelos alternativos a las MTs, conocidos como Metodologías Ágiles (MAs), enfocadas específicamente a proyectos de pequeño y mediano porte, simplificando el proceso de desarrollo sin perder las prácticas esenciales que garanticen las calidades del producto.

En febrero de 2001 nace el término de Metodologías Ágiles, ya que antes de esta fecha ya existían metodologías de este tipo (como se verá más adelante), pero que necesitaban mayor reconocimiento y difusión. Se creó desde entonces la *Agile Alliance* [AgileAlliance s.a.], que es la organización dedicada a promover los conceptos de desarrollo ágil y su adopción.

3.1 Valores Ágiles

La Agile Alliance en el llamado Manifiesto Ágil [AgileManifesto2001] establecen los valores para los modelos ágiles de desarrollo de software:

- *Individuos y sus Interacciones sobre el proceso y las herramientas:* Se pretende poner énfasis en el equipo de desarrolladores y el relacionamiento entre ellos. En general, valorar el factor humano por sobre procesos rígidos y herramientas de desarrollo. Muchas veces se comete el error de construir primero el entorno y

esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.

- *Software en funcionamiento sobre documentación comprensiva:* Se establece como objetivo principal el desarrollo continuo de software funcional probado. Nuevos incrementos de software son entregados en intervalos frecuentes de dos semanas a tres meses, y la documentación generada es la mínima necesaria. Esto impulsa al equipo a mantener el código lo más simple y documentado posible.

- *Colaboración de los clientes sobre la negociación de contratos:* Se da preferencia a la colaboración de los clientes en el proceso de desarrollo, por sobre contratos estrictos a seguir; aunque no signifique que los contratos no deban estar bien definidos y que son más importantes a medida que los proyectos sean de mayor envergadura. Desde el punto de vista del negocio, se pretende con esta colaboración reducir el riesgo del no cumplimiento de los requerimientos establecidos en el contrato.

- *Responder a los cambios sobre el seguimiento de un plan:* se trata de crear conciencia tanto en los desarrolladores como en los representantes de los clientes de que a lo largo del proceso, emergerá la necesidad de ajustes en los requerimientos, o simplemente cambios de cualquier índole (tecnológicos, en el equipo, etc.). Esto prepara a las partes a hacer cambios y establece que los contratos deben ser formulados de manera a soportar estos cambios.

Teniendo en cuenta estos valores, podemos decir que un modelo de procesos es ágil cuando es incremental (entregas de software constante), sencillo (se mantiene la simplicidad tanto de la solución como del proceso de desarrollo), cooperativo (trabajo en conjunto entre desarrolladores y clientes), y adaptativo (capaz de hacer cambios de último momento sobre cualquier parte del software, previo estudio de factibilidad).

Para ver mejor cómo son algunas de las MAs, en las siguientes cuatro secciones se describirán los modelos de Adaptive Software Development, Extreme Programming, Scrum y Test Driven Development.

3.2 Adaptive Software Development (ASD)

ASD fue desarrollada por John Highsmith en el año 2000 [Highsmith2000]. Pregona el desarrollo incremental e iterativo con prototipado constante. La idea fundamental del ASD es balancearse en el borde del caos. Con esto, se quiere mostrar que se debe encontrar un equilibrio entre crear un ambiente de trabajo que favorezca la creatividad e innovación y proveer las pautas de trabajo suficientes para evitar que el proceso de desarrollo sea caótico (sin un control de qué procesos se realizan y cómo se los están realizando) [Highsmith1997].

3.2.1 Modelo de Procesos

Como se puede ver en la Fig. 3.1, el modelo de procesos básico del ASD se basa en ciclos de tres etapas.

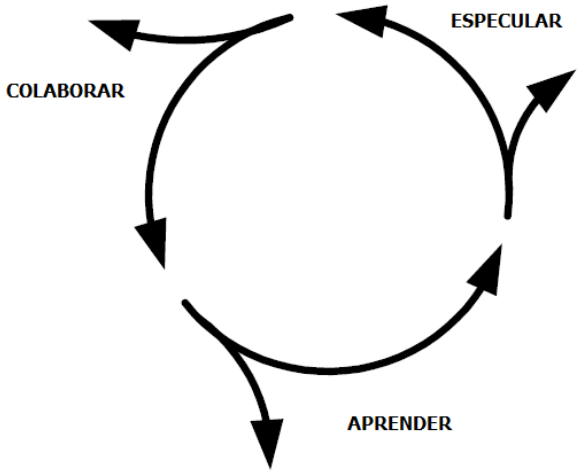

Fig. 3.1. Modelo de Procesos Básico del ASD. [Highsmith1997]

La primera de estas tres etapas es nombrada “Especular” para resaltar el hecho de que no se siguen planes estrictos, más bien se especula con los factores (herramientas, equipo de desarrollo, clientes, requerimientos, plazos y otros) que influyen en el desarrollo de cada ciclo para programar las actividades. La siguiente etapa “Colaborar”, es donde se lleva a cabo el desarrollo del incremento de software del ciclo y el nombre de la etapa también pretende destacar otro valor del ASD que es el trabajo en equipo. Por último se tiene la

etapa de “Aprendizaje” en la que se obtiene la retroalimentación (feedback) para el siguiente ciclo.

En la Fig. 3.2 se puede ver de una manera un poco más detallada el modelo de procesos. La Iniciación del Proyecto define la misión del mismo. Las facetas importantes de la misión están definidas en tres ítems: el documento de la visión del proyecto, una hoja de datos del proyecto y una línea de resultados (*outline*) de especificaciones del producto. También se define en esta fase la agenda y los objetivos globales.

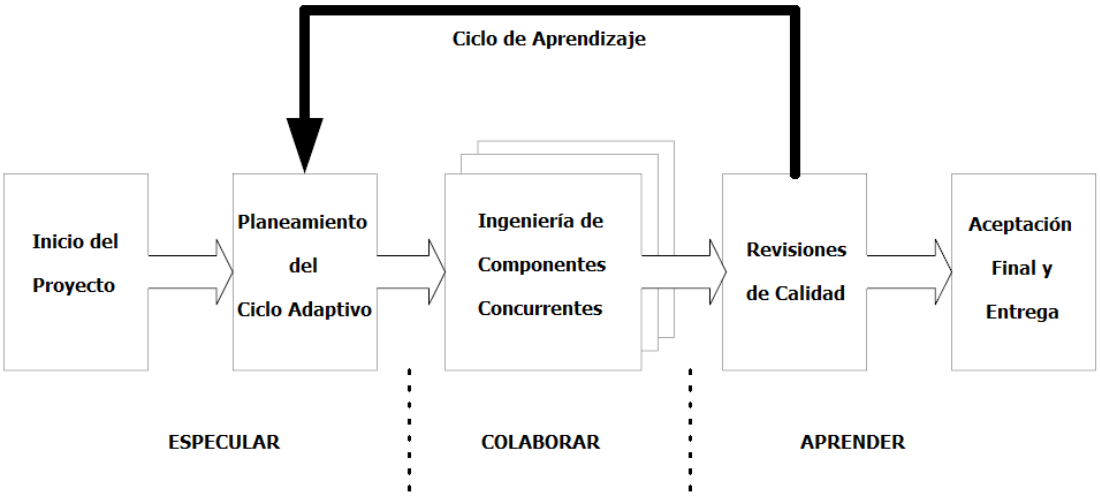

Fig. 3.2. Modelo de Procesos ASD. [Abrahamsson2002]

En base al resultado de la Iniciación del Proyecto, comienza el proceso cíclico con el Planeamiento del Ciclo Adaptativo que normalmente dura entre cuatro y ocho semanas [Abrahamsson2002]. En esta etapa se analizan los componentes que serán implementados en la siguiente fase.

La Ingeniería de Componentes Concurrentes representa al desarrollo. No se especifica en detalle cómo se lleva a cabo esta etapa. Lo que sí establece es que se pueden desarrollar distintos módulos de manera concurrente, siempre que esto sea posible.

La base para los siguientes ciclos se obtiene mediante repetidas revisiones de calidad que tienen el objetivo de demostrar la funcionalidad del software desarrollado en el ciclo. Pero como estas revisiones son pocas, se prevé mayor presencia del cliente por medio de sesiones JAD (Joint Application Development) [Highsmith2000]. JAD es básicamente un

taller en el cual los desarrolladores y los clientes discuten sobre las características principales del software y que sirve para mejorar la comunicación entre el cliente y los desarrolladores. ASD no fija un momento específico en el cual se deben realizar sesiones JAD pero se señala que son particularmente importantes al principio del proyecto [Highsmith1997].

La parte final es la aceptación y entrega del producto. No se especifica como esto se realizará pero si se destaca la importancia de capturar las lecciones aprendidas. Los postmortems de los proyectos son críticamente importantes.

3.2.2 Prácticas y Adopción

ASD nombra específicamente solo tres prácticas cotidianas de trabajo de desarrollo de software. Éstas son el desarrollo iterativo, planeamiento basado en componentes y revisiones por grupos enfocados al cliente. El problema con ASD es que deja muchos detalles abiertos, no indica que prácticas deben realizarse, sino más bien indica ejemplos de qué se podría hacer.

Justamente, esto hace que ASD no sea una de los modelos más adoptados entre las MAs. Sin embargo, no quiere decir que no se pueda usar ASD pero para ser adoptada como modelo de procesos debe complementarse con algún otro modelo.

Otro punto destacable de ASD es que la filosofía que promueve concuerda con los valores de las MAs, por lo que puede servir como base para el estudio de las mismas y también provee un importante marco teórico sobre temas como el desarrollo de sistemas complejos, trabajo en equipo y agilidad en la cultura administrativa.

3.3 Extreme Programming (XP)

XP surge como para sobreponerse a las dificultades de los largos ciclos de desarrollo de las MTs [Beck1999a]. Fue publicado por Kent Beck en 1999 [Beck1999a] [Beck1999b]. Las prácticas propuestas por XP, de manera individual, no son del todo novedosas, sin embargo, la manera en la que fueron integradas para formar una nueva metodología fue lo innovador. El término “extremo” viene por el hecho de aplicar sus prácticas de una manera extrema.

3.3.1 Modelo de Procesos

Como se puede apreciar en la Fig. 3.3, XP divide el ciclo de vida del software en seis fases. La primera es la llamada Fase de Exploración, en la que se redactan las "Historias" del cliente. Cada una de estas historias describe una característica deseada en el software. Al mismo tiempo, el equipo de desarrollo se familiariza con las tecnologías a ser utilizadas durante el desarrollo del proyecto.

La fase de Planeamiento establece las prioridades a las Historias, y se llega a un acuerdo del subconjunto de las mismas que serán implementadas en la siguiente iteración. Luego de esto, se realiza una estimación de esfuerzos para poder establecer los tiempos.

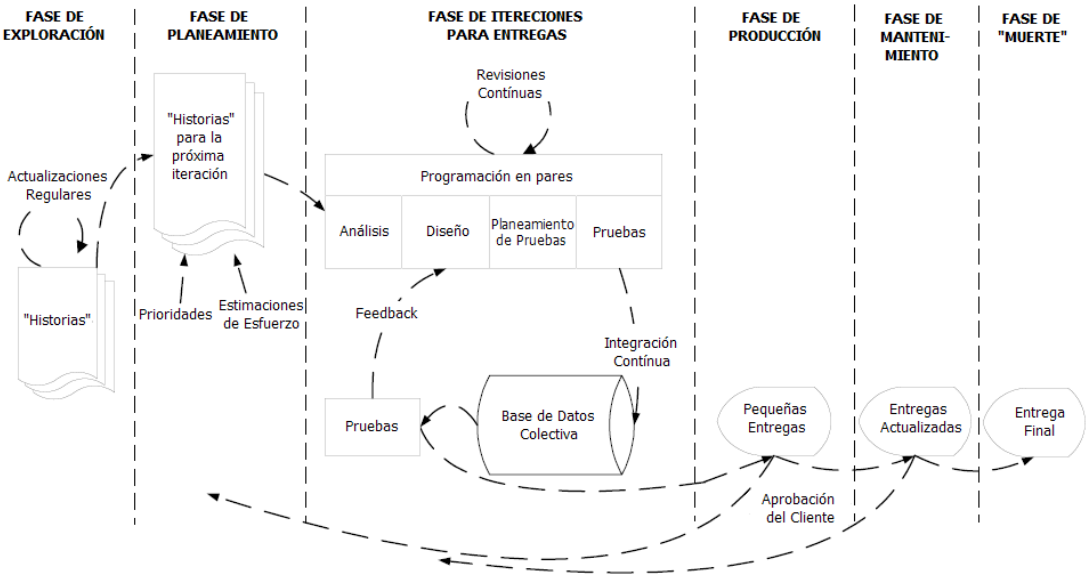

Fig. 3.3. Modelo de Procesos de XP. [Abrahamsson2002]

La fase de Iteraciones para Entregas dura entre una a cuatro semanas. La primera de las iteraciones crea la arquitectura del sistema. Al final de cada iteración, los clientes participan en el desarrollo de las pruebas sobre el incremento desarrollado. Y también el cliente participa en la elección de las Historias que serán implementadas en cada ciclo.

Para que el software pase a la fase de Producción debe ser sometido a un nuevo conjunto de pruebas funcionales y no funcionales (rendimiento). De surgir en esta etapa nuevos cambios, se debe decidir si los mismos serán incluidos en la entrega actual. En caso

contrario, se documentan los cambios para ser implementados en la fase siguiente que es la de Mantenimiento.

Por último, la llamada fase de Muerte, es el fin del proyecto. Se da cuando no existen más historias a ser implementadas. Es en este momento donde se escriben todas las documentaciones necesarias del software.

3.3.2 Prácticas

Entre las prácticas sugeridas por XP, se pueden destacar [Abrahamsson2002] [Hunt2006]:

- *Planning Game (juego de planificación):* Interacciones entre clientes y desarrolladores. El cliente establece la prioridad de cada Historia. Los programadores estiman el esfuerzo asociado a cada Historia. Se ordenan las historias de usuario según prioridad y esfuerzo, y se define el contenido de la entrega o iteración, apostando por enfrentar lo de más valor y riesgo cuanto antes. Este "juego" se realiza durante la planificación de la entrega, en la planificación de cada iteración o si es necesario, en cualquier otra etapa durante el desarrollo del proyecto.

- *Small Releases (entregas pequeñas):* Nuevas versiones de software deben pasar a la fase de Producción por lo menos cada dos o tres meses. Estas versiones no tendrán todas las funcionalidades, pero si deben tener un subconjunto de las mismas que sean de valor para el negocio.

- *Diseño Simple:* Se pone énfasis en diseñar soluciones lo más sencillas posibles que se puedan implementar en un momento dado. Se eliminan complejidades innecesarias y código extra, y (por ejemplo) se definen la menor cantidad de clases posible.

- *Refactoring (Refactorización):* Consiste en la reestructuración continua sobre el sistema para remover duplicación, mejorar las comunicaciones, simplificar el código y agregar flexibilidad. Complementa a la práctica de Diseño Simple.

- *Pruebas:* La producción de código está dirigida por las pruebas de unidad. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas

constantemente ante cada modificación del sistema. Los clientes escriben las pruebas funcionales para cada Historia que deba validarse.

- *Programación en Pares:* Dos personas escriben código en una sola máquina. Entre las ventajas de esta práctica, pueden citarse la detección de errores de código mientras son introducidos, continuo intercambio de ideas y transferencia de conocimientos por parte de los programadores y la mejora de la dinámica del equipo.

- *Propiedad Colectiva del Código:* Cualquier programador puede cambiar cualquier parte del código en cualquier momento. Esta práctica motiva a todos a contribuir con nuevas ideas, evitando a la vez que algún programador sea imprescindible para realizar cambios en alguna porción de código.

- *Integración Continua:* Una nueva porción de código es integrada a la base de código colectiva tan pronto como esté lista. Dicha porción es aceptada una vez que se corran todas las pruebas y hayan concluido sin encontrar errores.

- *40 horas por semana:* Como máximo de horas de trabajo semanales. No se permiten dos semanas consecutivas con más de cuarenta horas. Si esto ocurre, se trata como un problema que tiene que ser resuelto.

- *Clientes In Situ:* Representantes de los clientes deben estar disponibles a tiempo completo para el equipo de desarrolladores. Si esto no es posible, se deben buscar estrategias que garanticen un alto grado de comunicación entre ambas partes.

- *Estándares de Codificación:* Deben existir reglas de codificación y ser seguidas por los programadores. Esto debe permitir comunicación a través del código.

3.3.3 Adopción

XP es una de las metodologías ágiles más documentadas, razón por la cual es por amplio margen la más adoptada en el mercado ágil. Esto puede decirse en base a la lista de los artículos publicados por cada metodología de la *Agile Alliance*[1], se puede ver claramente que XP es la que acapara la atención. Estos artículos tratan tanto de investigaciones como

[1] http://agilealliancebeta.org/library

reportes de su adopción. También en la lista de grupos de usuarios de la misma entidad[2] se puede ver la superioridad amplia de XP sobre otras MAs. Por último, una encuesta que revela que aproximadamente el 38% de las empresas que se consideran ágiles utilizan XP [Reynoso2004].

3.4 Scrum

Basado en una publicación de 1986 por H. Takeuchi y I. Nokata [Takeuchi1986], Scrum fue tomado como una MA en el libro de K. Schwaber y M. Beedle en el 2002 [Schwaber2002]. El nombre de la metodología proviene de la estrategia del deporte Rugby, que denota el hecho de "traer un balón que estaba fuera del juego, de vuelta al mismo, mediante el trabajo en equipo".

Esta metodología no define técnicas específicas de desarrollo de software en la fase de implementación, más bien se concentra principalmente en como los miembros de un equipo deben actuar en conjunto para producir software flexible a posibles cambios.

3.4.1 Modelo de Procesos

Debido al origen del nombre de la metodología, los nombres de las fases también tienen una cierta connotación deportiva. Como se aprecia en la Fig. 3.4, Scrum define tres fases: El Pre-Juego, el Juego (Desarrollo) y el Post-Juego (*Pregame*, *Game* y *Postgame* en inglés).

En la primera fase, se observan dos actividades. El Planeamiento se define un documento llamado Product Backlog List, que podría traducirse como la "columna vertebral" del producto y que contiene todos los requerimientos del software conocidos hasta ese momento. Luego, estos requerimientos son priorizados y evaluados en cuanto a esfuerzo requerido. Se puede ver que se tienen previstas constantes actualizaciones sobre el documento, tanto en los requerimientos como en las prioridades y estimaciones de esfuerzo. También, como otra actividad dentro de la primera fase, se encuentra el Diseño de la Arquitectura del sistema, en base a los ítems actuales del Backlog List.

[2] http://www.agilealliance.org/resources/usergroups

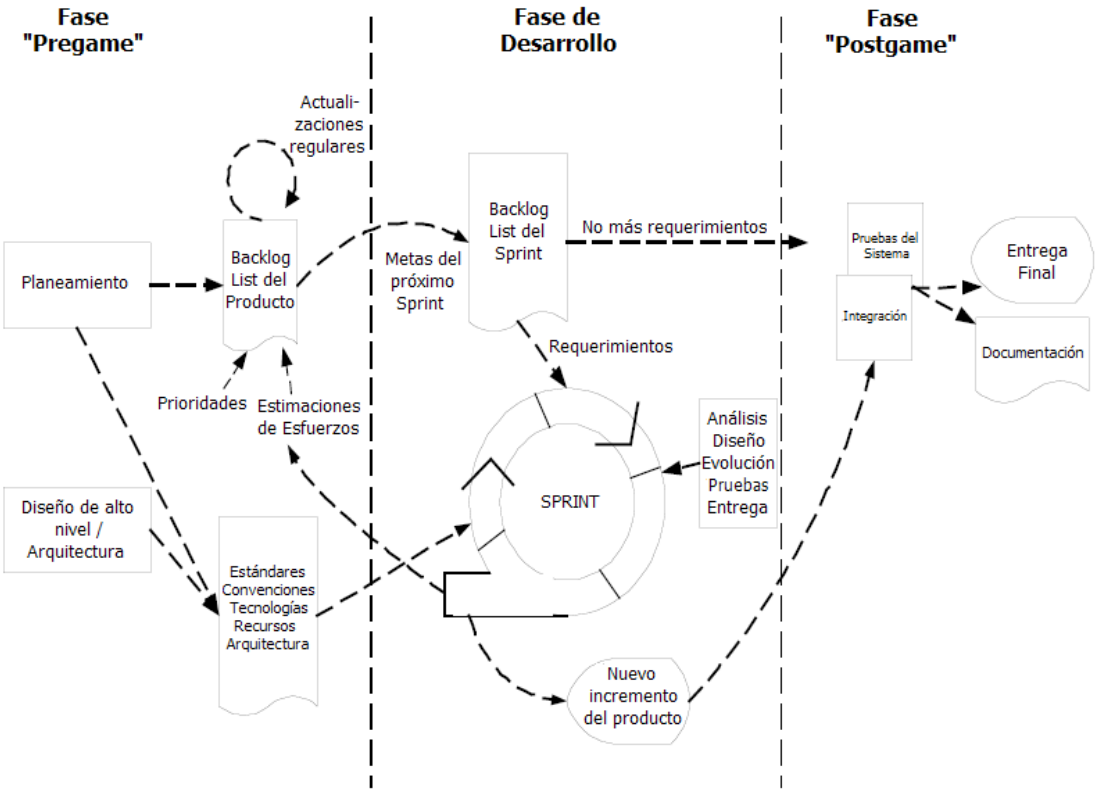

Fig. 3.4. Modelo de Procesos de Scrum. [Abrahamsson2002]

En la segunda fase, el desarrollo se lleva a cabo mediante ciclos llamados Sprints (serán descritas más adelante en la sección 3.4.2). Cada Sprint incluye las fases tradicionales de desarrollo: requerimientos (subconjunto del Backlog List), análisis, diseño, implementación y pruebas. Normalmente, cada ciclo debe durar treinta días. Los equipos de desarrollo deben ser menores a diez personas, y si es necesario, se pueden formar varios equipos.

La tercera y última parte, contiene la entrega final del producto. Se llega hasta aquí cuando hay un acuerdo de que no hay más requerimientos a ser implementados. Entonces, se llevan a cabo las pruebas de integración de lo último que se desarrolló, las pruebas integrales del sistema y las documentaciones correspondientes.

3.4.2 Prácticas y Adopción

Las prácticas más importantes de Scrum son [Abrahamsson2002]:

- *Backlog List:* Define todas la características necesarias que debe exhibir el producto final en base al conocimiento actual. Esta práctica incluye tareas para la creación del documento, para mantener su consistencia a la hora de agregar, eliminar, especificar, actualizar y priorizar su contenido.

- *Sprint:* Son los ciclos de desarrollo en Scrum. El equipo escoge un subconjunto de ítems del Backlog List de acuerdo a las prioridades. Cada Sprint dura treinta días. También se puede destacar que al equipo se le da la autoridad de hacer lo que necesite con tal de completar con los objetivos del Sprint, dentro de las restricciones de presupuesto, tiempos, funcionalidad y calidad del proyecto.

Como Scrum no requiere ni especifica ninguna práctica de desarrollo, puede ser adoptado como modelo de gerenciamiento sobre cualquier práctica de ingeniería utilizada en la organización.

Justamente por esta razón, a nivel de investigación, se están realizando esfuerzos para integrar XP y Scrum, argumentando que Scrum provee un marco de gestión del proyecto sobre las prácticas bien definidas de XP [Abrahamsson2002].

3.5 Test Driven Development (TDD)

En un principio, TDD puede verse simplemente como la idea de escribir las pruebas antes que escribir el programa. Pero como consecuencia de esto, los desarrolladores deben pensar en cómo se debe usar el programa (o mejor dicho, su propósito) y luego en cómo implementarlo [Steindl s.a.]. Por lo tanto, se puede afirmar que TDD no es solo una técnica de pruebas, sino también una técnica para el análisis y diseño del software [George2002].

3.5.1 Modelo de Procesos

El modelo de procesos separa las pruebas en pruebas de unidad y pruebas de aceptación. Para las pruebas de unidad, se asume la utilización de herramientas del tipo *xUnit* (por ejemplo JUnit) para realizar las mismas. Sin herramientas que automaticen este proceso sería casi imposible aplicar el modelo porque requiere pruebas de unidad para todo el sistema [TDD s.a.].

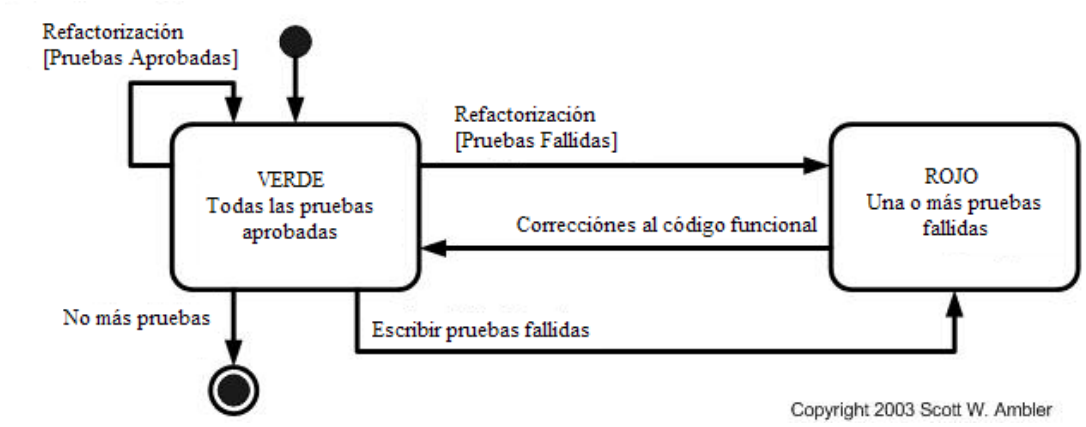

Fig. 3.5. Pruebas de Unidades en TDD. [TDD s.a.]

En este proceso no se comienza diseñando, sino pensando en las pruebas, es decir, en la funcionalidad. Nuevo código funcional solo se escribe cuando falla alguna prueba [Steindl s.a.]. Las pruebas se corren muy a menudo para determinar si se alcanzaron los objetivos de la unidad y para poder capturar errores lo antes posible. El lapso de tiempo entre la escritura de nuevo código y la refactorización (definida ya como práctica en XP, sección 3.3.2), debe ser bien corto.

Para el segundo tipo de pruebas, las de aceptación, también se sigue la filosofía de escribir las pruebas antes de comenzar, para poder escribir las pruebas de unidades en correspondencia con las pruebas de aceptación.

También deben ejecutarse las pruebas de aceptación durante los ciclos de desarrollo para poder entender mejor el progreso del proyecto y al final del ciclo para verificar el cumplimiento de las funcionalidades que debieron ser construidas.

3.5.2 Prácticas y Adopción

Las dos principales prácticas que se promueven son la de la escritura de pruebas antes de la escritura del código que implementa el sistema y la refactorización. Además de estas prácticas, TDD asume el uso de herramientas (disponibles gratuitamente para distintos lenguajes) para automatizar las pruebas.

No es común el uso de TDD como una metodología independiente, pero sin embargo sirve como soporte en las etapas de desarrollo de otras metodologías, en especial XP, entre cuyas prácticas se encuentran las dos prácticas propuestas por TDD.

3.6 Evaluación de las MAs

Habiendo analizado sus valores, y tres ejemplos de modelos de procesos ágiles, se pueden destacar un conjunto de ventajas y desventajas de este tipo de metodologías.

Tabla 3.1. Comparación entre características de las MTs y las MAs [Letelier2006].

Metodologías Tradicionales (MTs)	Metodologías Ágiles (MAs)
Planeamiento estricto.	Planeamiento Adaptativo.
Clara separación entre sus etapas.	Etapas solapadas.
Documentación detallada.	Reducción de la documentación al mínimo necesario.
Rigurosa definición de roles.	Pocos roles y roles flexibles.
Se espera que no ocurran cambios de gran impacto.	Se aceptan todo tipo de cambios.
Énfasis en el proceso.	Énfasis en las personas.
Existe un contrato prefijado.	Contrato flexible, pero difícil de definir.
Interacción con el cliente a través de reuniones.	Interacción continua con el cliente mediante clientes in situ.

3.6.1 Ventajas

Desde el punto de vista de las entregas rápidas, se pueden destacar [Poole2006]:

- Iteraciones en ciclos cortos que permiten correcciones y verificaciones más rápidamente.
- Límites de tiempo para cada ciclo de una a ocho semanas.
- Adaptable al surgimiento de nuevos riesgos.

También, se puede destacar que todas las MAs, se orientan a las personas favoreciendo el trabajo en equipo y tomando a los individuos como el principal factor de éxito de los proyectos.

Todas estas características permiten sobreponerse a varias de las críticas que se hacen a las MTs, y hacen que sean una solución ideal para la implementación de sistemas de pequeño y mediano porte de una manera que permite adaptarse a los cambios.

3.6.2 Desventajas

Si bien las ventajas que ofrecen las MAs podrían hacer pensar que son la alternativa ideal para el desarrollo de software, se destacan también una serie de desventajas:

- Su filosofía de poder adaptarse al cambio en vez de seguir estrictamente un plan puede llevar al caos en el proceso de desarrollo.
- Difícil de mantener el interés de los clientes que participan en el proceso de desarrollo.
- Definir las prioridades de los cambios pueden ser difíciles de establecer cuando existen múltiples personas o empresas involucradas en el proyecto.
- La definición de los contratos puede ser un problema.

Por lo tanto, las MAs no siempre son la mejor solución a la hora de optar por un modelo de procesos, ya que sobre todo en sistemas de gran tamaño, las dificultades mencionadas pueden hacer que el proyecto fracase.

Capítulo 4
Propuesta: MIDS

Para poder dar una propuesta de modelo de procesos, adecuada a los entornos de desarrollo de nuestro país, es fundamental describir las características de los mismos. De esta manera, se podrán contrastar las características de nuestros entornos con las ventajas y desventajas de los modelos de procesos analizados anteriormente (capítulos 2 y 3) para formular un modelo de procesos simple, que pueda ser adoptado fácilmente por las empresas que se dediquen al desarrollo de software.

A su vez, se pretende que la adopción del modelo propuesto permita con el tiempo, que las empresas hayan adoptado el modelo puedan optar en por el uso de cualquier otro modelo de procesos existente, de manera a aspirar a certificaciones internacionales de calidad como el CMMi [Sei s.a.] (Capability Maturity Model for Software), el estándar ISO-15504 [Iso2006] y otros.

A continuación, se comenzará analizando lo que es un entorno de desarrollo de software inmaduro, que se tomará como punto de partida para el estudio de los entornos locales ya que es el nivel inicial de los estándares de calidad. Luego se contrastarán las características de un entorno inmaduro, con las características locales. Los datos locales fueron extraídos en base a la realización de una encuesta.

Más adelante, en base a los resultados de las comparaciones, se evaluará la aplicabilidad de los modelos de procesos estudiados a los entornos locales. Y, por último, se dará una visión general del modelo de procesos propuesto.

4.1 Entornos Inmaduros de Desarrollo de Software

En un entorno inmaduro, los procesos de desarrollo son generalmente improvisados por los programadores y sus gerentes a lo largo del proyecto. Aún si existe una especificación del proceso de desarrollo, ésta no es seguida de manera rigurosa. Se los conocen como reaccionarios, es decir, están acostumbradas a resolver crisis inmediatas (apagar incendios). Las estimaciones de tiempo no son precisas, normalmente se exceden los plazos, y cuando existen fechas límites estrictas, o no se las cumplen o la calidad del producto final se ve bastante afectada.

Otra característica importante de estos entornos, es que no existen bases objetivas para medir la calidad del software ni para resolver problemas del producto o del proceso. Por lo tanto, la calidad del producto es difícil de predecir. Las actividades destinadas a garantizar la calidad como las revisiones y pruebas, se reducen o incluso se eliminan cuando los proyectos se retrasan.

Se puede destacar también la carencia o insuficiencia de documentaciones, tanto a nivel de los procesos (si es que están definidos) como a nivel del software resultante; lo cual tiene incidencia directa en distintos aspectos como en la contratación de nuevas personas en el caso de los procesos, o la mantenibilidad del software en segundo caso.

En las especificaciones del Nivel 1 (Inicial) del CMM en su versión 1.1 [Paulk1993], se caracteriza a este tipo de entornos de la siguiente manera:

- Ambiente inestable para el desarrollo y mantenimiento del software.
- El éxito depende de las capacidades individuales del gerente y de los programadores. El cambio de personal puede tener drásticas consecuencias en el desempeño del equipo.
- Los procesos son impredecibles porque son constantemente modificados a medida que el trabajo progresa (*ad hoc*).

Por lo tanto, habiéndose señalado estas características, se puede decir que como su éxito depende de individuos altamente competentes (llamados habitualmente héroes), la selección, contratación, capacitación y continuidad del personal competente son los aspectos críticos de las organizaciones que se encuentran en este nivel.

Evidentemente, todo lo expresado arroja como conclusión que una organización que pretenda el éxito como desarrolladora de software, no puede encontrarse con estas características. Una organización con estas características debería buscar medios viables para lograr reducir o eliminar cada una de las falencias en sus procesos y poder tener un marco de trabajo estable que le permita producir productos de software con un modelo de procesos definido y aplicado.

4.2 Entornos Locales vs. Entornos Inmaduros

Para recabar los detalles sobre los entornos de desarrollo locales, se realizó una encuesta de diez preguntas. Se expondrán en primer lugar los fundamentos de las mismas, para luego pasar a los resultados obtenidos, el cálculo del margen de error y por último a las conclusiones obtenidas de la realización de la encuesta.

4.2.1 Lista y Fundamentos de las Preguntas

1. ¿Están bien definidos los roles dentro de la organización?
2. ¿Existe una definición de cómo debería llevarse a cabo el proceso de desarrollo de software? ¿Se utiliza alguna metodología de desarrollo?
3. ¿Se utiliza alguna herramienta de manejo de versiones de código (CVS, Subversion)?
4. ¿Se mide la calidad del software desarrollado? ¿De qué manera?
5. ¿Se utilizan herramientas para automatizar las pruebas (xUnit)?
6. ¿Se utiliza alguna herramienta de seguimiento de errores (Bugzilla)?
7. ¿Existe documentación del software y/o del proceso? ¿De qué tipo?
8. ¿Con qué frecuencia se realizan reuniones con los clientes para analizar el estado de los proyectos?
9. ¿Se realiza una planificación adecuada de los proyectos?
10. ¿Con qué frecuencia se necesitó de horas extras para cumplir con plazos de entrega?

Las primeras dos preguntas relevan datos relacionados con la organización, en cuanto a designación de responsabilidades. Estas preguntas sirven para verificar si el desarrollo de software posee las características mencionadas en el primer párrafo de la sección 4.2, como procesos improvisados, o desarrollo reaccionario.

La tercera pregunta está específicamente relacionada con la manera en que las empresas gestionan los códigos fuentes de los sistemas que desarrollan. Los códigos fuentes de los sistemas desarrollados son de vital importancia, y por lo tanto su gestión debería ser controlada.

La cuarta y quinta pregunta están relacionadas con lo mencionado en entornos inmaduros sobre la carencia de métodos claros para definir la calidad del producto.

Las preguntas seis, siete y ocho están fuertemente relacionadas con la etapa de mantenimiento del software. Para que un software tenga alta mantenibilidad es necesaria una buena comunicación entre desarrolladores y clientes; y una buena documentación.

Por último, las preguntas nueve y diez se relacionan a la frecuencia de retrasos que se incurren en los proyectos desarrollados por la organización.

4.2.2 Resultados

Los resultados fueron obtenidos de una muestra de 20 personas encuestadas, que se dedican al desarrollo de software como parte de una organización cuya finalidad principal es la de producción de software.

La selección de las personas se basó en la consulta a egresados y alumnos de carreras relativas al desarrollo sobre sus actividades laborales. Así se obtuvieron candidatos a responder las encuestas que ofrezcan resultados significativos, a quienes se les proveyó de los cuestionarios.

Los resultados de la primera pregunta (sobre la definición de roles) es que en un 75% de los casos, los roles están bien definidos. Esto significa que en general existe una delimitación entre las autoridades de la organización, a nivel de jefes y desarrolladores.

En cuanto a tener definida una metodología, sólo el 15% respondió de manera afirmativa la pregunta. Del restante 85%, el 60% no tiene definida la metodología de trabajo y el 25% la tiene definida, pero no la utiliza estrictamente.

Como se describió en los fundamentos de las preguntas, estas dos primeras nos dan la pauta de que en nuestros entornos puede que existan los roles y modelos, pero sin embargo

no son cumplidos, por lo que se puede concluir que en cuanto a la gestión de los proyectos se trabaja de manera reaccionaria. Es decir, se trabaja en reacción a los problemas que vayan surgiendo a medida que avanza el proyecto.

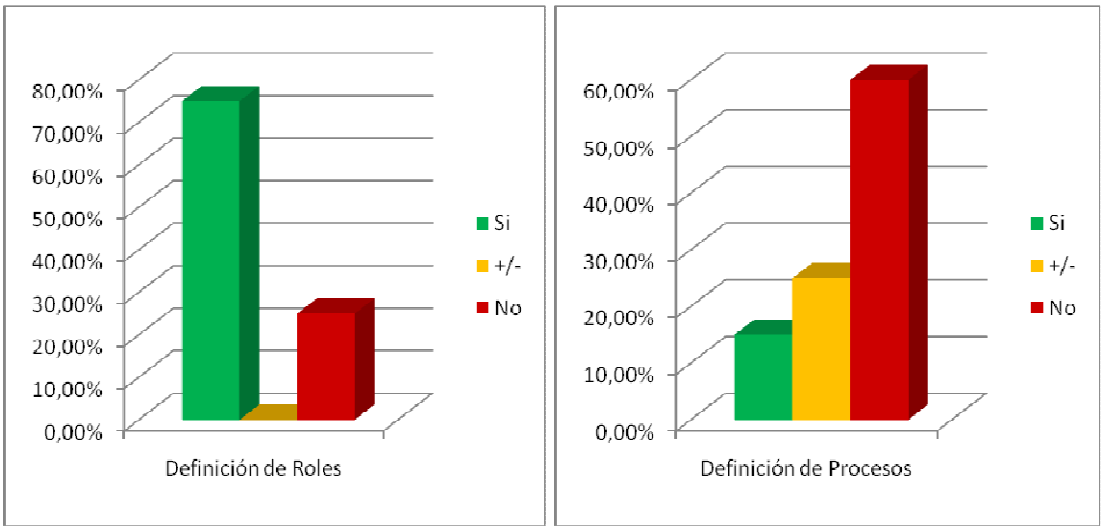

Fig. 4.1. Definición de Roles y Procesos.

En el caso de la tercera pregunta, existe un 40% que utiliza herramientas de gestión de versiones para mantener el historial de los códigos fuente de la organización. El 60% restante de los encuestados, manifestaron no utilizar herramientas de este tipo.

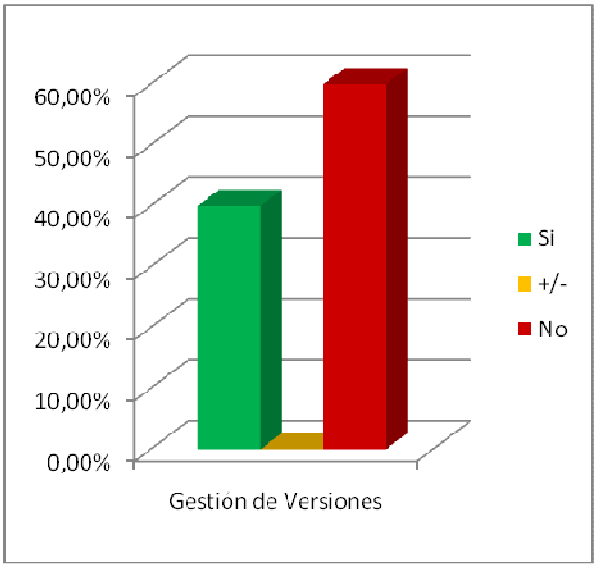

Fig. 4.2. Utilización de Sistemas de Gestión de Versiones.

En cuanto a las preguntas cuatro y cinco, relacionadas con la metodología y rigurosidad de las pruebas realizadas al software producido, se destaca que solo el 10% utiliza una metodología de prueba establecida, mientras que el restante 90% simplemente deja las pruebas a cargo de la persona que escribe el código. Por otra parte un 15% afirma que utiliza herramientas de automatización de pruebas en algunas secciones críticas del sistema desarrollado, pero no como parte de una metodología de trabajo.

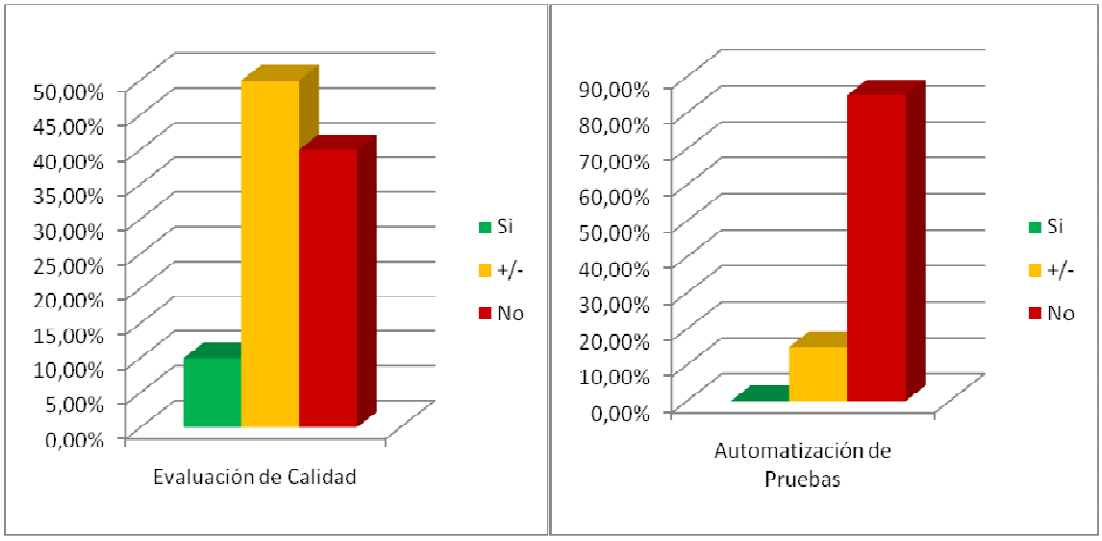

Fig. 4.3. Medición de Calidad y Automatización de Pruebas.

Esto arroja como conclusión que es alto porcentaje de organizaciones que no definen claramente cómo se evalúa la calidad del producto desarrollado.

Las tres siguientes preguntas, relacionadas con el trato hacia los clientes y el mantenimiento del software, arrojaron los siguientes resultados:

- En cuanto a la utilización de una herramienta de seguimiento de errores, el 15% las utiliza en todos sus proyectos, otro 10% las utiliza en algunas ocasiones y el restante 75% no las utiliza.

- Con respecto a la documentación existente, un 15% utiliza rigurosamente la documentación, otro 20% no utiliza documentación alguna y el restante 65% utiliza la documentación solo en algunos proyectos, o partes de proyectos, de acuerdo a la necesidad.

- En lo relacionado a la frecuencia de reuniones entre desarrolladores y clientes, las respuestas fueron muy variables. Se pueden citar distintas frecuencias como dos veces por semana, una vez por semana, dos veces al mes, y una vez al mes. En ciertos casos se especificó que a principios del proyecto era mucho mayor la frecuencia y que luego de la entrega del producto se reducía a lo estrictamente necesario, es decir, cuando surgían fallas o pedidos de cambios.

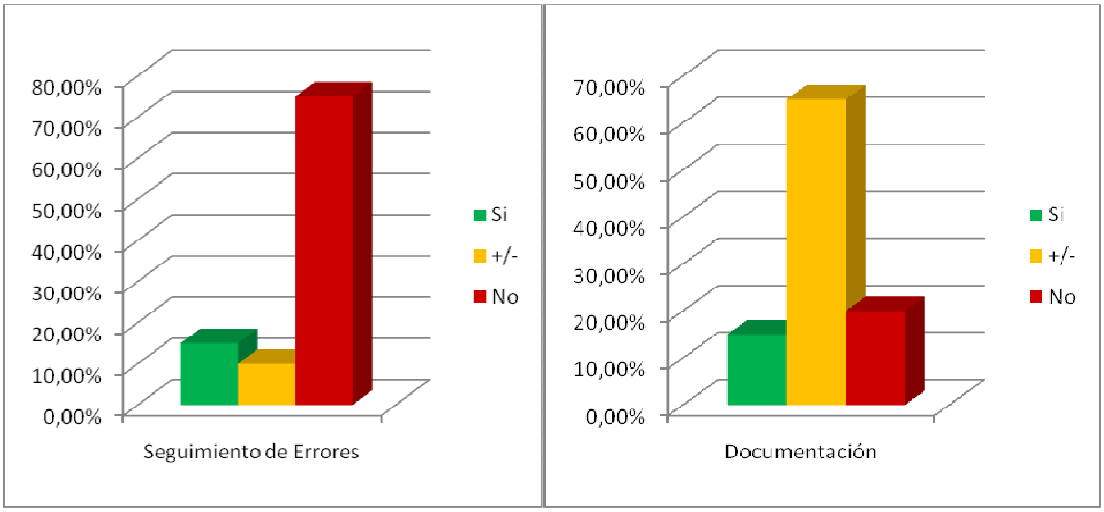

Fig. 4.4. Herramientas de Seguimiento de Errores y Documentación de Sistemas.

En base a estas respuestas se pueden deducir falencias en cuanto a la documentación y al trato con el cliente, ya que en la mayoría de los casos no están definidas por la organización las pautas mediante las cuales se actuará en cada caso. Ambas actividades tienden a ser realizadas solo en los casos estrictamente necesarios.

Por último, las preguntas nueve y diez, que están relacionadas a la estimación de recursos, dan como resultado que un 10% realiza una estimación inicial y planificación del proyecto, un 60% no lo hace, y el restante 30% lo hace pero la planificación no es seguida por el equipo de desarrollo de manera estricta. En la mayoría de los casos, se destacó también la constante necesidad de horas extras para cumplir con los plazos establecidos.

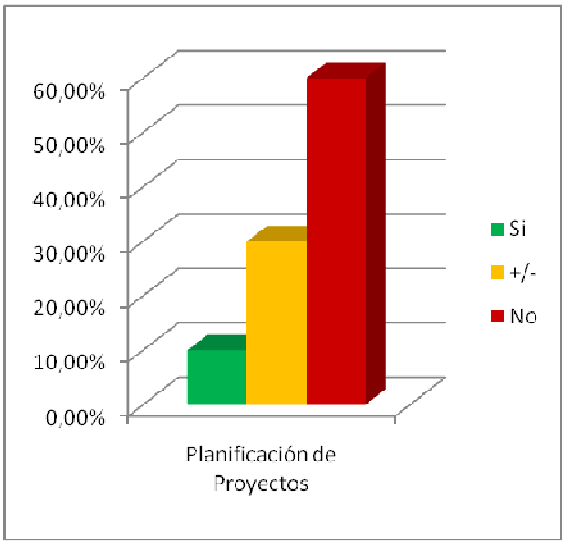

Fig. 4.5. Planificación Adecuada de Proyectos.

4.2.3 Cálculo del Margen de Error de la Encuesta

Para calcular el margen de error de la encuesta, se tomará como base la siguiente fórmula [Oncins s.a.] [Fernández1996]:

$$n = \frac{NZ_{\alpha}^{2} pq}{d^{2}(N-1) + Z_{\alpha}^{2} pq} \qquad (4.1)$$

El significado de las variables es el siguiente:

- n: Tamaño de la muestra.
- N: Tamaño de la población.
- Z_{α}: Coeficiente del nivel de confianza. Para una confianza del 95%, el valor del coeficiente será de 1,96.
- p: Proporción de la variable estudiada.
- q: 1 – p.
- d: Intervalo de confianza (margen de error).

Para calcular el margen de error, se despeja la Fórmula 4.1 para obtener:

$$d = \sqrt{\frac{\left(\frac{N}{n} - 1\right) Z_{\alpha}^{2} pq}{N - 1}} \qquad (4.2)$$

Así, quedan por establecer los valores para *p* y *N* en base a los datos de la encuesta. En el primer caso, la variable estudiada es la situación del desarrollo local. El promedio de respuestas favorables es de aproximadamente 20% por lo que *p* valdrá 0,2 y *q* 0,8.

Para el valor del tamaño de la población, se tuvo en cuenta el resultado de un censo de empresas realizado por la Red de Inversiones y Exportaciones (REDIEX) en la que se registraron 130 (ciento treinta) empresas desarrolladoras de software [Rediex2007].

Ya con todos los valores establecidos, se puede obtener el margen de error a partir de la Fórmula 4.2 de la siguiente manera,

$$d = \sqrt{\frac{\left(\frac{N}{n} - 1\right) Z_{\alpha}^{2} pq}{N - 1}} = \sqrt{\frac{\left(\frac{130}{20} - 1\right) * 1{,}96^{2} * 0{,}2 * 0{,}8}{130 - 1}} \approx 0{,}16 = 16\% \qquad (4.3)$$

Así, se puede ver que el margen de error resultante es del ±16%. Esto significa que el máximo error posible en un porcentaje resultante de alguna de las respuestas de la encuesta podría ser de 16% más o 16% menos.

4.2.4 Conclusiones

Como se mencionó en la sección anterior, el promedio de respuestas con rasgos positivos es aproximadamente del 20%. Tomando el peor de los casos, en el que el margen de error sea del 16% a favor de los rasgos positivos, no se llega a un 40% de promedio general.

Por lo tanto, se concluye que en un alto porcentaje, los entornos de desarrollo locales se asemejan a los entornos inmaduros de desarrollo de software descritos en la sección anterior.

4.3 MAs y Entornos Inmaduros

Como ya se mencionó, las MAs surgieron como respuesta a las limitaciones de las MTs. Sin embargo, luego de analizar las características de las MAs y de los entornos inmaduros

de desarrollo, se puede ver en la Tabla 4.1, que las ventajas ofrecidas por las MAs pueden ser notablemente útiles para aplicarlas en entornos inmaduros.

Tabla 4.1. Relación entre MAs y Entornos Inmaduros de Desarrollo.

Características de un Entorno Inmaduro	**Características de las MAs**
Falta de disciplina en el desarrollo del software.	Iteraciones cortas acostumbran al equipo a adquirir disciplina en el desarrollo.
Carencia de Documentación.	Documentar lo mínimo necesario.
Problemas de personal, dependencia de ciertos individuos altamente capaces.	Estilo de trabajo de equipo ayuda a nivelar a los miembros del equipo.
Falta de garantías de la calidad del producto.	Mediante las pruebas continuas al final de cada ciclo, se garantiza un aceptable nivel de calidad.
Falta de comunicación entre clientes y desarrolladores.	Trabajo colaborativo con los clientes.

Primero, con iteraciones cortas de desarrollo se logra disciplinar progresivamente al equipo. Como las MAs dan prioridad a los individuos y a su relacionamiento, por sobre los procesos, la implementación de estos ciclos no supondrá un cambio radical en el actuar de los miembros del equipo, sin embargo estarán acostumbrándose a trabajar con ciertos límites de tiempo (tiempo límite de cada iteración).

La falta de documentación también está cubierta por las MAs de una manera que no será tampoco un cambio brusco en el actuar de los desarrolladores. Porque se insta a mantener el código lo más simple y documentado posible y porque la documentación en papel se reduce a lo estrictamente necesario (sin eliminarla).

Los problemas de personal en entornos inmaduros pueden ser solucionados con un buen uso (por ejemplo) de la programación en pares. Mediante esta práctica, se logra equilibrar de manera razonable el conocimiento del equipo para no tener que depender de programadores o gerentes “héroes”.

También, en los principios de las MAs, se encuentra el hecho de entregar frecuentemente software en funcionamiento probado. Esto ayuda a sobreponerse de a poco a los problemas de la gestión de la calidad del software.

Por último, el beneficio del trabajo colaborativo con los clientes y el de la adaptabilidad a los cambios, ayudará a cumplir con los requerimientos establecidos, a obtener retroalimentación del software producido, a crear la conciencia en los clientes de que el proceso de producción de software es cambiante y a estar preparados para la ocurrencia de los cambios de requerimientos.

Se puede entonces decir que las MAs son potencialmente aplicables a entornos inmaduros.

4.4 Descripción General de la Propuesta: MIDS

De los análisis realizados en este capítulo, se han extraído dos conclusiones determinantes para elegir un modelo de procesos propuesto para su aplicación en entornos locales de desarrollo:

- En un alto porcentaje, las características de los entornos locales son similares a las de un entorno inmaduro de desarrollo de software.

- Las MAs son potencialmente aplicables a entornos inmaduros.

Por lo tanto, el modelo propuesto tendrá como base la filosofía de las MAs, para poder aprovechar las ventajas que podría ofrecer al aplicarse en los entornos locales.

El nombre elegido para el modelo propuesto es el de "Modelo Inicial para el Desarrollo de Software MIDS". Se remarca que es un modelo inicial porque su principal objetivo es ser aplicado en entornos inmaduros de desarrollo y también porque puede ser un primer paso hacia la adopción de otros modelos de procesos, como por ejemplo alguna MA.

Constará de las tres etapas básicas que se repetirán cíclicamente para construir software de modo iterativo, como todas las MAs. Como en un principio se señaló en los objetivos se pretende proponer una metodología lo suficientemente sencilla para ser utilizada, encontrando el equilibrio con la rigurosidad necesaria en cualquier metodología de trabajo.

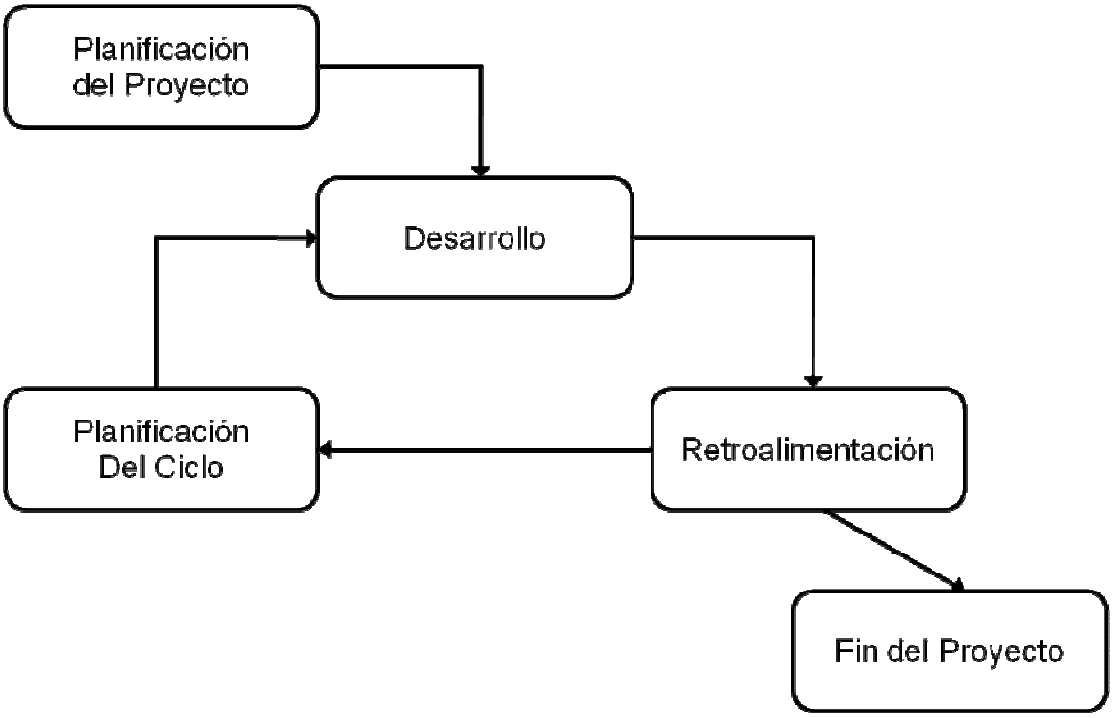

Fig. 4.6. Modelo de Procesos del MIDS.

La Figura 4.6 nos muestra el modelo de procesos del MIDS en base a las etapas básicas descritas anteriormente. Si bien se pueden ver cinco etapas, las de planificación del proyecto y del ciclo forman parte de una sola etapa que es la de "Planificación". Por último, "Fin del Proyecto" es un hito al cual se llega cuando se logre un acuerdo final entre clientes y desarrolladores, y cuando ya no existan funcionalidades a ser implementadas.

Por lo tanto, habiendo demostrado la dependencia básica de etapas del MIDS, en cada uno de los siguientes tres capítulos se analizarán las tres etapas del modelo propuesto de la siguiente manera:

- *Planificación,* en el capítulo 5 (incluyendo la planificación del proyecto y del ciclo).
- *Desarrollo,* en el capítulo 6, y
- *Retroalimentación,* en el capítulo 7.

Capítulo 5
Planificación

La etapa de planificación del modelo propuesto tiene dos escenarios básicos:

- *Planificación del proyecto:* Este caso es el del inicio del proyecto. En primer lugar, se deberán recabar los datos sobre las reglas del negocio y llegar a un acuerdo entre clientes y desarrolladores acerca de las funcionalidades y el alcance del sistema. En base a este acuerdo se podrá determinar el tamaño del equipo y el diseño de la arquitectura del sistema.

- *Planificación de ciclos:* Este es el caso general de lo que se hará al principio de cada uno de los restantes ciclos a lo largo del proceso de desarrollo del sistema. Se seleccionarán las actividades y/o tareas pendientes (nuevas funcionalidades, correcciones o mejoras en el sistema) para asignarlas a desarrolladores que deberán cumplirlas durante el transcurso del ciclo.

Para ambos casos, se deberán redactar documentos que delineen el trabajo a realizarse con los responsables para cada actividad. Esta documentación debe ser breve, ya que como Metodología Ágil, debe reducirse la documentación a lo estrictamente necesario.

A lo largo de este capítulo se verán tres aspectos relativos a esta etapa, que definirán cómo se llevará a cabo la misma, especificando en qué casos los conceptos se relacionan a la planificación del primer ciclo o a la de los ciclos restantes.

5.1 Tamaño del Equipo y Roles

El modelo propuesto, tomando como ejemplo a las demás MAs, puede ser utilizado por equipos de desarrollo de entre **uno** a **ocho** integrantes.

Los roles definidos son:

- *Líder del Proyecto:* Es la persona encargada de tomar las decisiones con respecto a la gestión del proyecto. Este rol puede ser exclusivo, o puede ser mixto. Al decir mixto, se quiere decir que puede ser el líder del proyecto, y a su vez un *desarrollador.*

- *Desarrollador:* Se encarga de la codificación y documentación del sistema.

- *Representante del cliente:* Es la persona designada por los clientes para interactuar con el equipo de desarrollo ya sea para especificar funcionalidades o para validar los resultados.

5.2 Documentación

Teniendo en cuenta que el modelo propuesto se enmarca dentro de la filosofía "ágil", la documentación debe reducirse a lo estrictamente necesario. Así, los documentos a utilizarse serán dos obligatorios y uno opcional:

- Documento de Especificación de Requerimientos.

- Especificación de Casos de Usos.

- Documento de Solicitud y Seguimiento de Cambios (opcional).

El Documento de seguimiento de cambios, que es opcional, será analizado en el Capítulo 7 que describe la etapa de mantenimiento. También en el mencionado capítulo se mencionará la alternativa a la utilización de dicho documento. En las siguientes dos secciones se analizarán los dos primeros documentos.

5.2.1 Documento de Definición de Requerimientos

Para realizar un proyecto de software, deben establecerse pautas y delineamientos que definirán los resultados esperados al final del proyecto. El objetivo del Documento de Definición de Requerimientos, el de servir como base para el acuerdo inicial entre los clientes y los desarrolladores sobre lo que se espera del sistema. La redacción de este documento es la parte más importante de la planificación del primer ciclo, definida anteriormente.

El contenido del documento se detallará a continuación [Ghezzi1991], para poder entender todo lo que abarca el mismo y cuáles de los roles son los involucrados en su redacción.

1. *Introducción:* La introducción del documento es redactada por el líder del proyecto para dar un panorama general de las características del proyecto a los clientes en cuanto reciban la copia del documento para su aprobación.

   1.1. *Propósito del Documento:* En este apartado, brevemente se explicarán los objetivos del documento que ya se han explicado anteriormente.

   1.2. *Ámbito de la Aplicación:* Aquí se explicará la percepción, por parte del equipo de desarrollo del entorno en donde se ejecutará la aplicación. Así se podrá justificar el por qué de cualquier propuesta de solución descrita más adelante.

   1.3. *Definiciones y Siglas:* Se listarán los términos y acrónimos con sus respectivos significados para hacer más fácil la lectura del resto del documento.

2. *Descripción General del Sistema:* En este apartado, que también es redactado por el líder del proyecto, se definirán aspectos generales relativos a la solución propuesta. Dichos aspectos se clasifican en:

   2.1. *Función y Propósito:* Se describen los objetivos del sistema en cuanto a funcionalidad.

   2.2. *Alcance de la Aplicación:* Se delimita el alcance en términos de qué procesos organizacionales serán automatizados o asistidos por el sistema. También puede ser resumido a nivel de áreas que abarcará. Por ejemplo, para un sistema empresarial podría abarcar la contabilidad, la gestión de recursos humanos, etc.

   2.3. *Restricciones Generales:* Las restricciones generales son aquellas consideraciones que se deben tener en cuenta para el correcto funcionamiento del sistema propuesto, tanto en hardware, software, telecomunicaciones, y recursos humanos.

   2.4. *Descripción del Modelo:*

2.4.1. *Sistema:* Brevemente se describirá el modelo que será utilizado en el sistema (posible arquitectura). Esto puede ayudar a entender mejor el por qué de las restricciones generales, ya que algunas pueden ser resultado de la arquitectura elegida.

2.4.2. *Proceso:* Se deberá especificar el modelo de procesos a utilizarse para poner en claro la metodología de trabajo. Esto es importante de definir inicialmente, para que sirva como conformidad de ambas partes sobre el cómo se desarrollará el proyecto.

3. *Especificación de Requerimientos:* En esta sección ya se describirá en mayor detalle el sistema propuesto. Será redactada por el líder del proyecto como resultado de sus reuniones con los representantes del cliente. El modelo propuesto adopta como método de captación de requerimientos las reuniones con los representantes de los clientes. Es importante que el líder del proyecto deje en claro que los representantes deberían ser personas que conocen suficientemente el funcionamiento operacional de la organización cliente.

3.1. *Requisitos funcionales:* Es el listado de las funciones y/u operaciones que deberá llevar a cabo el sistema propuesto.

3.1.1. *Definición de Roles:* Antes de detallar las funcionalidades, se deben definir los roles de los usuarios del sistema, para poder luego asociar las funcionalidades con cada rol.

3.1.2. *Descripción de Funcionalidades:* Aquí se detallan cada una de las funcionalidades del sistema. La descripción no debe ser mayor de un párrafo, ya que debe simplemente explicar el objetivo de la funcionalidad y para qué roles estará disponible. Alternativamente, se podrían agrupar por roles las descripciones.

3.2. *Requisitos No Funcionales:* Ya culminada la especificación de requerimientos funcionales, se deben describir otro tipo de restricciones que no son necesariamente una funcionalidad en el sistema. Éstos son:

3.2.1. *Seguridad:* Se establecerán los requerimientos de seguridad sobre los datos del sistema en base a los roles. Y también en cuanto a planes de contingencia, copias de seguridad, etc.

3.2.2. *Mantenibilidad:* Se describirá brevemente cómo el sistema soportará la posibilidad de ser corregido y ampliado a lo largo del tiempo.

3.2.3. *Hardware y Software:* Descripción de requisitos mínimos y recomendados para la utilización del sistema. Dependiendo del modelo, se debe también aclarar la conectividad necesaria y el uso o no de servidores para la aplicación (con sus respectivos requerimientos).

3.2.4. *Otros:* Cualquier otro requerimiento específico del sistema puede ser detallado en este apartado.

4. *Especificaciones de Entrega:* Por último, se describirá cómo se haría la entrega del software. Esto debe incluir un cronograma tentativo detallado, con todas las actividades relacionadas a la implementación del sistema, incluyendo alternativas de acción si es que se ofrece más de una solución, o cursos de acción alternos en el cronograma. También debería incluir tentativas de precios en base a cada alternativa propuesta (en caso de que se especifique más de una). Esta sección será redactada por el líder del proyecto.

5. *Anexos:* En caso de que existan adjuntarse modelos de formularios u otros tipos de documentos que sirvan de modelo para la implementación de algún requisito funcional. Los mismos son facilitados por los clientes e incluidos en la versión final del documento por el líder del proyecto a modo de conformidad.

Así, con la estructura del documento, se puede ver claramente qué se especifica y quiénes lo harán. Sin embargo, como comentarios finales sobre el documento, el modelo propuesto estipula que la versión final de este documento debe ser tomada como base del acuerdo contractual que pueda existir entre las partes.

Otra aclaración importante está relacionada con posibles cambios que puedan realizarse sobre el documento a lo largo del proyecto. El mecanismo de cambio de este documento a lo largo del proyecto debería detallarse en la Sección *ii.iv.ii* del mismo.

5.2.2 Especificación de Casos de Uso

Una vez que se haya llegado a un acuerdo en base al Documento de Especificación de Requerimientos el líder del proyecto deberá seleccionar los casos de uso a implementarse en el ciclo de desarrollo que comenzará.

Esta selección se debe hacer teniendo en cuenta las limitaciones de tiempo establecidas por el modelo que son de **cuatro** semanas por ciclo, exceptuando el primer ciclo que podría durar hasta **seis** semanas. Por ejemplo, para el primer ciclo se deberían seleccionar casos de uso básicos (inicio de sesión, gestión de usuarios u otros) ya que debe implementarse la base del sistema (si no se reutilizará de otros proyectos).

Si bien, la redacción de este documento se entrelaza con la etapa de desarrollo que se analizará en el siguiente capítulo, su formato y contenido que están basados en UML [Herrera1999], serán descritos a continuación:

1. *Enumeración de los actores:* Al comenzar cada ciclo (en especial el primero), se toma la lista de roles definidas en el documento de especificación de requerimientos para definir los actores del sistema. En un principio, cada rol sería un actor, pero a lo largo del proceso de desarrollo los roles podrían diversificarse o unificarse.

   Opcionalmente, se podrán hacer los diagramas de UML de actores en base a las relaciones que puedan existir entre los mismos.

   Cualquier cambio en los actores, debe ser comunicado a los clientes. El líder del proyecto (encargado de la redacción de esta sección) deberá decidir en base a la proximidad de la siguiente entrega, y a la criticidad de los cambios; si comunica a los clientes antes de realizar el cambio, o si lo hace en el momento de la siguiente entrega.

2. *Enumeración de Casos de Uso por Actores:* Luego de la definición de los actores, se citará por cada uno de los mismos, la lista de casos de uso que se le permitirán utilizar. Opcionalmente, se podrán realizar los diagramas de UML correspondientes. Esta sección también es redactada por el líder del proyecto.

3. *Descripción de los casos de uso:* Las dos secciones anteriores eran más bien generales. Esta sección es redactada por los desarrolladores responsables de la implementación de cada caso de uso y está compuesta de las siguientes sub-secciones.

   3.1. *Breve Descripción:* En un párrafo de dos o tres oraciones se debe resumir el objetivo del caso de uso, en términos de quién lo usa y cómo se realiza la interacción entre el actor y el sistema. Los detalles internos del sistema se omiten para que la especificación sea legible también por los clientes.

   3.2. *Flujo de Eventos:*

      3.2.1. *Precondiciones:* Indica bajo qué condiciones y/o circunstancias se inicia el caso de uso.

      3.2.2. *Flujo Principal:* Se indica en párrafos, paso a paso, cómo el actor interactúa con el sistema en base a datos de entrada y a respuestas del sistema.

      Cuando se llega a algún punto en el que el actor deba optar por más de una alternativa se deben referenciar a los distintos subflujos que se definirán más adelante.

      3.2.3. *Subflujos:* Se enumeran y se describen de la misma forma que en el flujo principal, los distintos subflujos que se pueden presentar en el caso de uso.

      3.2.4. *Flujos alternos:* El formato y descripción es similar al de los subflujos, pero la ocurrencia de un flujo alterno está dada por los casos de excepción que puedan darse a lo largo del proceso de intercambio de datos entre el actor y el sistema.

Cuando en alguna de las dos descripciones anteriores se puede dar un flujo alterno, se debe hacer referencia a las descripciones de esta sección.

4. *Definición de Relaciones entre los Casos de Uso:* A medida que avancen las descripciones de casos de uso se pueden empezar a detallar gráficamente las relaciones entre los mismos. Esta tarea corresponde al líder del proyecto que irá elaborando y modificando los gráficos a medida que vaya progresando el proyecto.

Como se explicó, este documento irá creciendo a medida que avance el proyecto, así que el uso de una herramienta que permita gestionar las versiones del mismo, y el trabajo concurrente serían de vital utilidad para su gestión.

5.3 Estimación de Recursos

Como ya se pudo ver, el modelo propuesto está basado en especificaciones de casos de uso por lo que una elección natural para estimar los recursos necesarios para el proyecto es la Estimación por Puntos de Casos de Uso, propuesto por Gustav Karner en su tesis en 1993 (Universidad de Linkoping) y supervisado por Ivar Jacobson en Objectory AB [Wikipedia s.a.b].

Este método de estimación de esfuerzos se basa en las especificaciones de casos de uso que no estarán disponibles en un primer momento ya que éstas se irán redactando a medida que los casos de uso se vayan implementando. De todas formas, con la información contenida en el Documento de Especificación de Requerimientos se puede realizar el proceso de estimación de manera aproximada. Los pasos para realizar estos cálculos serán descritos en las subsecciones siguientes [Peralta s.a.].

5.3.1 Puntos de Caso de Uso Sin Ajustar

Primero se calcula lo que se llama Puntos de Casos de Uso Sin Ajustar (UUCP) mediante la siguiente fórmula:

$$UUCP = UAW + UUCW \qquad (5.1)$$

La primera de las dos variables del lado derecho de la ecuación (UAW) es el Factor de Peso de los Actores. Para hallar este valor se lista a todos los actores del sistema y a cada uno se le asigna un valor numérico de acuerdo a los siguientes criterios:

- Uno: Cuando el actor es otro sistema que interactúa con el sistema a desarrollar mediante una interfaz de programación.

- Dos: Cuando el actor es otro sistema que interactúa con el sistema a desarrollar mediante un protocolo o interfaz de texto.

- Tres: Una persona que interactúa con el sistema mediante una interfaz de usuario.

El valor de UAW es la sumatoria de los valores numéricos establecidos a cada actor.

El segundo valor del lado derecho (UUCW) es el de Peso de Casos de Uso Sin Ajustar. También se asignan valores numéricos por cada caso de uso de acuerdo a la siguiente especificación, en donde transacciones significa secuencias de actividades atómicas:

- Simple (5): El caso de uso contiene una a tres transacciones.

- Medio (10): El caso de uso contiene cuatro a siete transacciones.

- Complejo (15): El caso de uso contiene más de siete transacciones.

Como en el caso de UAW, UUCW es la sumatoria de los valores asignados.

5.3.2 Puntos de Casos de Uso Ajustados

Una vez hallados los UUCP, se calculan los Puntos de Casos de Uso Ajustados (UCP) en base a la siguiente fórmula:

$$UCP = UUCP \times TCF \times EF \qquad (5.2)$$

Los valores TCF y EF son los Factores de Complejidad Técnica del Sistema, y Ambientales respectivamente. Ambos se calculan con las fórmulas:

$$TCF = 0.6 + 0.01 \times \Sigma [Peso(i) \times Valor\ asignado(i)] \qquad (5.3)$$

$$EF = 1.4 - 0.03 \times \Sigma [Peso(i) \times Valor\ asignado(i)] \qquad (5.4)$$

Para dejar más claro el funcionamiento de las Fórmulas 5.3 y 5.4, se presentarán las tablas que se utilizan para el cálculo de estos dos factores.

Tabla 5.1. Factores de Complejidad Técnica (TCF).

Factor	Descripción	Peso	Valor
T01	Sistema Distribuido	2	
T02	Objetivo de performance o tiempo de respuesta	1	
T03	Eficiencia del usuario final	1	
T04	Procesamiento interno complejo	1	
T05	El código debe ser reutilizable	1	
T06	Facilidad de instalación	0,5	
T07	Facilidad de uso	0,5	
T08	Portabilidad	2	
T09	Facilidad de cambio	1	
T10	Concurrencia	1	
T11	Incluye objetivos especiales de seguridad	1	
T12	Provee acceso directo a terceras personas	1	
T13	Se requieren facilidades especiales de entrenamiento a usuarios	1	

En la Tabla 5.1 se enumeran los factores técnicos, cada uno con su correspondiente peso. El valor asignado lo determina la persona que está haciendo la evaluación y debe estar entre cero y cinco de acuerdo a la relevancia de cada ítem, donde cero es irrelevante y 5 muy importante. Por ejemplo, si es un sistema de control de stock quizás el T02 no sea tan relevante por lo que se le podría asignar un valor de dos.

Una vez asignados todos los valores para todos los T(i), se aplica la Fórmula 5.3 para obtener el valor final de TCF.

Tabla 5.2. Factores del Ambiente (EF).

Factor	Descripción	Peso	Valor
E01	Familiaridad con el modelo de proyecto utilizado	1,5	
E02	Experiencia en la aplicación	0,5	
E03	Experiencia en orientación a objetos	1	
E04	Capacidad del analista líder	0,5	
E05	Motivación	1	
E06	Estabilidad de los requerimientos	2	
E07	Personal de medio tiempo	-1	
E08	Dificultad del lenguaje de programación	-1	

Los EF tratan de capturar las habilidades del grupo de desarrollo dentro de la estimación, ya que obviamente esto influye en el tiempo necesario para el desarrollo del sistema. Como con los TCF, el valor asignado puede estar entre cero y cinco. Ingresando estos valores en la Fórmula 5.4, se obtiene el valor EF.

Ya con TCF y EF fácilmente se obtienen los Puntos de Casos de Uso (UCP).

5.3.3 Estimación de Esfuerzo

Como ya se ha terminado el cálculo de UCP, lo que queda es "traducir" el valor obtenido a una métrica de esfuerzos, que es en este caso Horas Hombre, siguiendo los siguientes pasos:

- Se contabilizan cuántos factores de ambiente poseen un valor menor a tres para los factores E01 al E06.

- Se contabilizan cuántos factores de ambiente poseen un valor mayor a tres para los factores E07 y E08.

- Si la suma de estos dos conteos es menor o igual a dos, cada punto de caso de uso tomará 20 horas/hombre.

- Si la suma vale tres o cuatro, cada punto de caso de uso tomará 28 horas/hombre.

- Si la suma es igual o mayor a cinco, el proyecto se considera riesgoso y se sugiere realizar cambios en el diseño.

5.3.4 Aclaraciones sobre los UCP

Como se mencionó al principio de la explicación del cálculo de UCP, para el momento de la estimación de recursos, no se tendrán las especificaciones completas de los casos de uso. Sin embargo, la "Definición de Roles", la "Descripción de Funcionalidades" y los "Requisitos No Funcionales" serán suficientes para una estimación aproximada.

Capítulo 6
Desarrollo

La etapa del modelo que se presentará en este capítulo, es la que será descrita en mayor detalle en cuanto a prácticas a ser utilizadas. De esta manera, se logrará la disciplina suficiente para lograr la calidad del software desarrollado.

La etapa de desarrollo se repite en cada ciclo del modelo y abarca desde el momento en que se haya decidido el subconjunto de casos de uso a ser implementado hasta que la implementación de los mismos sea terminada y probada adecuadamente.

Esto deja abiertas las siguientes preguntas:

- ¿Cuánto tiempo debería durar esta etapa?
- ¿Cómo se debe llevar a cabo el proceso de desarrollo?
- ¿Cómo definir si la implementación del ciclo está adecuadamente probada?

Las respuestas a estas tres preguntas definirán las características de la etapa de desarrollo.

6.1 Duración

En la primera vez que se llegue a esta etapa, habrá más trabajo que en las siguientes repeticiones por lo que la duración debe ser mayor de lo que sería para los siguientes ciclos.

Como la arquitectura del sistema debe ser implementada, para que la primera entrega posea funcionalidades que el cliente pueda probar se necesitará un trabajo extra. Un rango de tiempo para la duración recomendable sería el de **cuatro** a **seis** semanas. Así, se prevé un lapso razonable para la implementación de la base del sistema y de un conjunto de funcionalidades para ser entregadas.

Para las siguientes repeticiones, la duración no debe exceder las **cuatro** semanas debido a que se ha comprobado con otras MAs que es tiempo suficiente para entregar incrementos de software útiles para los clientes. Es importante que se tenga un tiempo límite para cada iteración por que ayuda a mantener la disciplina de desarrollo del equipo. Por otra parte, si

el trabajo previsto se termina antes de la conclusión de las cuatro semanas, se debe proceder a la entrega del incremento cuanto antes.

6.2 Cómo desarrollar

Habiéndose explicado los motivos de la duración de cada iteración, queda ahora empezar a definir cómo se efectuará el desarrollo en sí a partir del inicio al fin de cada ciclo.

El líder del proyecto deberá elegir los procesos o funcionalidades que serán implementados desde el Documento de Especificación de Requerimientos (Sección 5.2.1) o las modificaciones que deban realizarse de acuerdo sus prioridades (Sección 7.2).

También, el líder del proyecto deberá asignar a responsables por cada ítem seleccionado de manera a que cada persona del equipo sepa cuál es su rol dentro del ciclo.

Los ítems seleccionados son detallados en definiciones de Casos de Uso por parte de sus responsables (o en caso de modificaciones se someten a revisión sus especificaciones), de tal manera a tener una mejor idea de cómo deben ser implementados o modificados y para dimensionar mejor que tan crítico es el ítem dentro del funcionamiento del sistema.

A partir de este momento se tienen claros los objetivos del ciclo y se comienza la codificación. Durante el desarrollo de esta actividad el modelo adopta el uso de prácticas y herramientas que serán descritas más adelante para garantizar la mejora en la calidad del código producido.

Una vez que un desarrollador del equipo termina la implementación del o los casos de uso a su cargo, debe enviar (en caso de no utilizar una herramienta que permita el trabajo concurrente sobre el documento de especificación de casos de uso) la documentación de vuelta al líder del proyecto para comunicar la conclusión de la implementación.

6.2.1 Sistema de Gestión de Versiones

Este tipo de herramientas permiten el manejo de distintas revisiones sobre la misma unidad de información [Wikipedia s.a.a], que en este caso son los códigos fuente del sistema a desarrollarse. Permiten que los programadores puedan editar el código simultáneamente en vez de que deban coordinar entre si las modificaciones para evitar conflictos. Las

herramientas asumen el trabajo de mezclar todos los cambios y de mantener un seguimiento de cualquier conflicto al integrar las distintas revisiones [Bar2003].

Como ejemplos de estas herramientas se pueden citar a *Revision Control System (RCS), Concurrent Version System (CVS), Microsoft Visual Source Safe, Subversion (SVN),* entre otros. Se utilizará en adelante a *CVS* como ejemplo para explicar mejor la mecánica del funcionamiento de una herramienta de gestión de versiones y así poder resaltar las ventajas que ofrecen.

6.2.1.1 Funcionamiento

Se utiliza un modelo cliente-servidor con un esquema de trabajo conocido como **copia-modificación-mezcla** que funciona como sigue [Bar2003]:

1. Una vez que se almacena el código inicial en el servidor cada desarrollador obtiene una copia. A esta operación se la conoce como *check out*.

2. Concurrentemente, cada desarrollador edita libremente su copia local. Esto es posible teniendo en cuenta que como cada uno tiene su copia independiente, no se interfieren ni bloquean entre sí.

3. Cuando un desarrollador desea que sus cambios pasen al servidor *CVS* procede a realizar una operación conocida como *commit*, que confirma sus cambios con un comentario que explica la naturaleza o el propósito de los cambios que están siendo confirmados.

4. Mientras, los otros desarrolladores pueden consultar al servidor si la copia que almacena (copia maestra) ha sido cambiada. Si hay cambios, *CVS* actualiza automáticamente la copia local de quien haya realizado la consulta. Esta operación es conocida como *update*.

Como todos los desarrolladores trabajan como iguales, la decisión de cuándo realizar las operaciones de *update* y *commit* depende de un acuerdo entre los programadores. Aún así, existe una convención común de siempre actualizar (realizar un *update*) antes de comenzar a realizar cambios importantes y confirmar los cambios (realizar un *commit*) cuando los

cambios están terminados y probados. Siguiendo estas convenciones, la copia que está en el servidor estará siempre en un estado "ejecutable".

Lo que falta aclarar en el funcionamiento general del *CVS* es el caso en el que dos de los integrantes del equipo realizan cambios sobre la misma área de texto y desean confirmar sus cambios. El primero que intente confirmar sus cambios lo logrará sin inconvenientes. Cuando el segundo quiera hacer lo mismo recibirá un aviso del servidor de que hay una versión más nueva de la que posee localmente y por lo tanto tendrá que realizar una actualización (*update*).

Al realizar la actualización, *CVS* notificará la ocurrencia de un *conflicto*. Colocará marcas reconocibles de texto para distinguir cuales son las partes conflictivas indicando los conjuntos de cambios almacenados en el servidor y en la copia local. Si es posible el desarrollador resuelve los conflictos y así podrá confirmar sus cambios. De lo contrario, debe ponerse en contacto con la persona que confirmó los primeros cambios para discutir la manera de solucionar el conflicto. El *CVS* solo puede alertar de la existencia del conflicto. Las personas son las encargadas de resolverlos.

Otras funcionalidades que ofrece el *CVS* es la creación de etiquetas (*tags*) y de bifurcaciones (*branches)*. En el primer caso se permite guardar el estado de todas las revisiones del proyecto para poder volver a ese estado en el futuro. Por ejemplo se podría hacer una etiqueta que guarde el estado de cada incremento que se entrega a los clientes. De esta manera, es fácil "volver en el tiempo" a la versión que se le entregó al cliente en caso de problemas a solucionarse.

El segundo caso es la creación de bifurcaciones. Que permiten separar el proyecto en dos vertientes paralelas sin que los cambios realizados en una afecten a la otra. Esto puede ser útil en complemento con la utilidad de las etiquetas. Suponiendo que los clientes detectan un error en una entrega, realizan el reporte correspondiente a los desarrolladores. Durante este tiempo el desarrollo continuó, por lo que los clientes no están utilizando la versión del software que poseen los desarrolladores y puede darse el caso de que el error reportado haya desaparecido o que de alguna forma el error vuelva a quedar inadvertido por algún cambio que evite su ocurrencia.

Allí, se puede crear una bifurcación al desarrollo actual a partir de la etiqueta de la entrega a los clientes y solucionar el error sobre este "proyecto paralelo". Una vez que se haya probado la solución, se puede realizar una mezcla con el desarrollo actual y así tener la solución del error reportado integrada al desarrollo actual del sistema.

6.2.1.2 Ventajas

La ventaja principal del uso de una herramienta de Gestión de Versiones es la de permitir el trabajo concurrente de personas sobre un mismo proyecto de software evitando inconvenientes como la falta de coordinación en el trabajo en equipo y la pérdida de información del aporte de alguno de los desarrolladores.

Además permite mantener un histórico del desarrollo del proyecto y con la buena utilización de todas sus características se pueden lograr ventajas como las mencionadas en la utilización de etiquetas y bifurcaciones.

Así, se llega a la conclusión de que las ventajas ofrecidas por este tipo de herramientas las hacen indispensables para el desarrollo de software en equipo.

6.2.2 Estándares de Codificación

Siguiendo la filosofía ágil que es la que guía al modelo propuesto, la reducción de la documentación generada por el proceso de desarrollo es un factor importante a la hora de eliminar el trabajo innecesario y así poder llevar a cabo un desarrollo que realice entregas útiles en cortos plazos.

Esto no quiere decir que el software no debe ser documentado sino más bien unificar las tareas (normalmente separadas en otros modelos de procesos no ágiles) de codificación y documentación.

Por lo tanto, la utilización de estándares y/o convenciones (internacionales u organizacionales) de programación y documentación de código es crucial para que distintas personas puedan codificar sobre cualquiera de los componentes del sistema sin la necesidad de consultar al autor. Y además, en el caso de seguir con los estándares en la documentación (comentarios) del código se pueden generar mediante herramientas

existentes manuales de referencia en formato HTML (ejemplos: *Javadoc*, comentarios XML de *.Net*).

Con respecto al contenido, los comentarios (que son la documentación del código) deben estar centrados en los detalles técnicos de la implementación de manera que cualquier integrante del equipo de desarrollo sea capaz entender el funcionamiento y de realizar modificaciones sobre cualquier parte del sistema.

La documentación de los aspectos técnicos no solo ayuda en lo mencionado anteriormente, sino que también facilita la reutilización de código ya que con una buena documentación se ayuda a los miembros del equipo a tomar decisiones sobre qué módulos reutilizar en un próximo proyecto con un menor esfuerzo (sin tener que analizar por completo el código fuente).

6.2.3 Programación en Pares

Esta práctica consiste simplemente en hacer que dos programadores trabajen juntos en la misma computadora. Extreme Programming (XP) es una de las MAs que más utiliza esta práctica estableciendo que todas las líneas de código útil deben ser producidas por pares de programadores.

Entre los principales beneficios de su utilización se pueden destacar los siguientes [Langr2005] [Hunt2006]:

- Produce una mejor cobertura del código. Una continua rotación de parejas de programadores favorece a que cada uno vaya adquiriendo más conocimiento del sistema.
- Minimiza la dependencia de personal.
- Eleva los conocimientos y las habilidades del equipo.
- Acorta la curva de aprendizaje del nuevo personal.
- Ayuda a apegarse a estándares.
- Mejora el trabajo en equipo.

Estas ventajas ayudan a mejorar varios de los aspectos mencionados sobre entornos inmaduros de desarrollo de software en el Capítulo 4 como la dependencia de personal y también permite asegurar el cumplimiento de objetivos del modelo como la utilización de estándares de programación. Su utilización podría ser considerada indispensable.

Sin embargo, si se trata de aplicar la programación en pares de la manera en la que XP lo hace a los entornos de desarrollos locales se podría encontrar resistencia ya que se puede llegar a pensar que sería una pérdida de recursos tener a dos personas trabajando en una misma computadora y por lo tanto, una pérdida de tiempo. Además, supondría un cambio radical en la manera de trabajar que nuevamente sería visto como una pérdida de tiempo por los costos que el cambio conllevaría.

Teniendo en cuenta esta resistencia al cambio, el modelo propone el uso de la programación en pares en las situaciones en las que se pueda aprovechar la ventaja de la nivelación de conocimiento de los integrantes del equipo, como por ejemplo cuando una nueva persona se acopla al proyecto o cuando se nota algún desnivel en la productividad de algunos de los miembros del equipo.

El líder del proyecto, al notar a un miembro del equipo que se ajuste a una de las dos circunstancias citadas anteriormente, decidirá con cuál de los demás integrantes del equipo estará trabajando en par y durante cuánto tiempo. Este tiempo no debe exceder a dos semanas porque esto equivaldría a la mitad de la duración de un ciclo.

De esta manera se logrará que la persona de menores conocimientos vaya adquiriendo las capacidades necesarias durante el tiempo que trabaje en par.

6.2.4 Pruebas de Unidad

El modelo requiere el uso de herramientas que automaticen las pruebas de unidad. En la Sección 6.3 se explicarán los motivos de utilización de estas herramientas y la manera y ocasiones en las que se deben usar. Ejemplos de estas herramientas son la librería JUnit para Java, la NUnit para C#, Microsoft Visual Test, PyUnit, HTMLUnit, entre otras varias que pueden ser gratuitas o propietarias.

6.3 Pruebas

Al llegar al final de un ciclo de desarrollo se debe tener preparado un incremento de software que esté en correcto funcionamiento de manera que pueda ser entregado a los clientes para obtener una retroalimentación de su parte y poder hacer los cambios y mejoras que sean pertinentes en las siguientes entregas.

Decir "en correcto funcionamiento" implica que los desarrolladores deben realizar pruebas en el software de tal manera a garantizar este correcto funcionamiento en el mayor grado posible. Mayor grado posible es importante acotar debido a que las pruebas solo pueden *demostrar la presencia de errores pero nunca demostrar su ausencia* [Dahl1972].

Las pruebas pueden dividirse en tres niveles según lo que se esté probando [IEEE2004]:

- *Pruebas de Unidad:* Verifican el funcionamiento de manera aislada de los componentes que puedan ser probados independientemente.

- *Pruebas de Integración:* Verifican el funcionamiento de las interacciones entre los distintos componentes del sistema.

- *Pruebas de Sistema:* Verifican el funcionamiento del sistema como tal. Como en los anteriores niveles ya se descubrieron (en su mayoría) los errores funcionales, este nivel se asocia más bien con la verificación de los requerimientos no funcionales.

A continuación se describirá como deben llevarse a cabo estos tres niveles de prueba.

6.3.1 Pruebas de Unidad

Como se mencionó anteriormente, el uso de herramientas para la automatización de pruebas es requerido por el modelo. Gracias a estas herramientas, el programador puede escribir diversos fragmentos de código que realicen los procesos que se desean probar. Esto significa que el programador puede escribir los distintos casos de prueba que necesite y ejecutarlos para recibir un informe de los casos en los que las pruebas arrojaron errores y los casos en los que no se encontraron fallas.

La principal ventaja de esta automatización es que los casos de prueba podrán ejecutarse en cualquier momento a lo largo del proceso de desarrollo. Al principio se podría pensar que diseñar y escribir código extra que realice pruebas podría resultar en una pérdida de tiempo. Pero a medida que avanza el proyecto, la posibilidad de poder realizar pruebas de manera automática puede compensar el tiempo utilizado en el diseño y codificación de las pruebas.

También es importante destacar como otra ventaja la ayuda que proveen para las pruebas de regresión. Pruebas de regresión son aquellas que intentan descubrir fallas en el sistema luego de modificaciones que vayan surgiendo a lo largo de su desarrollo. Las pruebas de unidad automatizadas permiten realizar las pruebas de regresión sin necesitar demasiado tiempo. Los casos de prueba necesarios están disponibles y listos para ejecutarse.

Estas ventajas hacen que el modelo adopte el uso de este tipo de herramientas. Sin embargo todavía no se ha especificado en qué momentos o bajo qué criterios deberían utilizarse.

El enfoque de pruebas utilizado por XP y Test Driven Development (TDD) supondría nuevamente un cambio radical en cuanto a la manera de trabajar actual en nuestros entornos. Como se mencionó en el Capítulo 3, se deben escribir las pruebas antes de la codificación misma. Lo que se sugiere a cambio de este enfoque es la utilización de pruebas de unidad automatizadas luego de la implementación de los casos de uso.

Una vez terminada la implementación del caso de uso se deben evaluar en base a la cantidad de datos de entrada los posibles casos de prueba de manera a determinar si merece la implementación de sus pruebas. Como ya se cuentan con las especificaciones del caso de uso (Sección 5.2.2) este modelo propone los siguientes criterios para la automatización de las pruebas de unidad:

- Cuando la suma de cantidades de subflujos y flujos alternos sea mayor a 5.

- Cuando la cantidad de datos de entrada requeridos sea mayor o igual a 10.

- Cuando el líder del proyecto (mediante iniciativa propia o por sugerencia de algún otro miembro del equipo) considere que el caso de uso es crítico en base a las reglas del negocio.

El líder del proyecto podrá decidir modificar estos valores en base a su experiencia, pudiendo reducir el uso de pruebas automatizadas solamente a los casos de usos que puedan considerarse críticos o expandir su uso a medida que el equipo de desarrollo se familiarice con esta práctica.

6.3.2 Pruebas de Integración y de Sistema

Para los dos siguientes niveles de pruebas (de integración y del sistema) son los programadores los que los deben realizarlas en un tramo final (uno a cinco días) antes de la entrega a los clientes. De esta manera se pretende utilizar el sistema lo suficiente como para minimizar las posibilidades de errores una vez que se realice la entrega a los clientes.

Dependiendo del tamaño del equipo, el líder del proyecto deberá designar a las personas que se encargarán de dichas pruebas. Desde este momento se vuelve importante el uso de otra herramienta propuesta por el modelo que es la de un Sistema de Seguimiento de Errores que se analizará en el siguiente capítulo. Este tipo de herramientas permiten llevar un registro de los errores o fallas encontradas.

En caso de encontrarse errores, el programador que los haya encontrado debe evaluar si es factible solucionarlo. Si es así, puede hacerlo teniendo en cuenta la realización de pruebas de regresión para cerciorarse de que sus cambios no hayan afectado el correcto funcionamiento del resto del sistema.

Si el programador cree que el tiempo no es suficiente, debe comunicar al líder del proyecto para que tome una decisión en base a la gravedad del problema. Si es un problema menor se podría simplemente entregar el sistema tal como estaba planeado avisando al cliente del error esperado. En caso contrario, si el problema limita considerablemente el incremento a ser entregado al cliente se debe negociar una prórroga que no debería exceder a una semana.

Capítulo 7
Retroalimentación

Como último paso a la hora de definir el modelo de procesos queda la elaboración de un mecanismo para que los clientes puedan emitir juicios críticos sobre cada incremento de software que reciben.

Para ello se necesita definir un medio de comunicación entre desarrolladores y clientes. A continuación se verán las posibilidades propuestas por el modelo a la hora de llevar a cabo esta última etapa del ciclo y los mecanismos para mantener un registro de los cambios que se vayan realizando a la aplicación.

7.1 Reuniones y/o Negociaciones con los Clientes

Como se fue detallando en el capítulo anterior, se realizan ciclos de desarrollo de los cuales se obtienen como resultados "incrementos" de software que serán evaluados por los clientes para correcciones y/o mejoras en los siguientes ciclos.

Sin embargo, esto no es suficiente a la hora de garantizar que el sistema de software final deje conformes a los clientes. O tal vez el sistema lo haga a expensas del tiempo debido a que muchos ciclos serán necesarios para ir puliendo cada objeción y/o detalle requerido por los clientes.

Por eso las MAs, queriendo innovar en este aspecto, proponen que exista un representante de los clientes acompañando permanentemente al equipo de desarrollo (*on-site customer*). Así se busca una colaboración que garantice que el desarrollo del sistema esté basado en la especificación de requerimientos y que permita a los clientes ofrecer detalles importantes sobre las reglas del negocio a lo largo de todo el proceso.

Como también sucedió en la etapa de desarrollo vista en el capítulo anterior, intentar aplicar un enfoque de clientes *in situ* (como lo proponen las MAs) sería muy difícil teniendo en cuenta el nivel de interacción que existe actualmente. Proponer este nivel de interacción como parte del modelo podría convertir al MIDS en inutilizable.

Por otro lado, no se puede prescindir de la importante colaboración del cliente por lo mencionado anteriormente. Por lo tanto el modelo propuesto trata de encontrar un balance

entre estas posibles situaciones estableciendo la realización de reuniones periódicas entre los representantes de los clientes y los del equipo de desarrollo.

Dichas reuniones deben realizarse **una vez por semana**, salvo en el caso del transcurso del primer ciclo de desarrollo, ya que los esfuerzos de desarrollo de esta etapa estarán más centrados en los detalles técnicos. En el caso especial del primer ciclo se propone **una reunión** con los clientes antes de la primera entrega.

En estas reuniones los desarrolladores dan un informe general de lo que se está desarrollando y realizan una comparación del avance en función a los requerimientos iniciales. Allí se podrán sugerir cambios que de ser necesarios y aprobados por ambas partes, deben registrarse en el Documento de Especificación de Requerimientos. También allí se decidirá si los cambios propuestos se verán implementados en el incremento del ciclo actual o en el del próximo ciclo.

También deben aprovecharse estas reuniones para dar cualquier explicación concerniente al uso del sistema (especialmente en las reuniones previas a una entrega) para que el incremento que reciban los clientes pueda ser puesto a prueba sin mayores inconvenientes.

Así, con este esquema sencillo se podrá obtener información importante por parte de los clientes para que el desarrollo vaya en la dirección correcta hacia la satisfacción de los requerimientos.

Como observación final se puede mencionar que para los casos en que clientes y desarrolladores no se encuentren en la misma ciudad y/o país (escenarios de exportación de software) se debe utilizar un mecanismo de comunicación que permita realizar en intercambio de información entre clientes y desarrolladores con la misma periodicidad propuesta anteriormente.

7.2 Documentación

En la sección anterior se ha establecido ya cómo se llevará a cabo la interacción entre desarrolladores y clientes a lo largo del proceso de desarrollo. Ahora queda por definir los procedimientos mediante los cuales los clientes (o también los mismos desarrolladores) deberán realizar sus pedidos de mantenimiento.

Como ya se explicó en el Capítulo 2, el mantenimiento es el conjunto de cambios que se realizan sobre el sistema una vez que se lo utilice realmente (puesta en producción). No obstante, en las MAs el mantenimiento no está separado del desarrollo porque los pedidos de cambios pueden surgir como resultado de la primera entrega. Y mientras continúa el desarrollo de nuevas características se puede estar realizando mantenimientos sobre las que ya han sido desarrolladas. Esto significa que desde un principio se debe establecer un mecanismo mediante el cual se puedan realizar los pedidos de cambios.

El modelo propuesto ofrece dos alternativas. La primera es el uso de una documentación especial que no solo permita la realización de los mencionados pedidos sino también su seguimiento. Seguimiento se refiere a poder saber en cualquier momento luego de la realización del pedido quién es el responsable de la implementación de los cambios solicitados y si ya han sido solucionados.

La segunda alternativa es la utilización de herramientas (software) que permitan realizar este seguimiento ya sea localmente (en el sitio de trabajo de los desarrolladores) o vía red. Esta segunda alternativa es la que inspiró el formato del documento escrito de la primera opción [Bugzilla2007] y será analizada en la siguiente sección. Ahora se describirá el formato del documento llamado: "Documento de Solicitud y Seguimiento de Cambios" que consta de las siguientes partes:

1. *Autor*: Es el nombre, organización y cargo de la persona que origina el documento. Es decir, la persona que encuentra la falla o que solicita una nueva funcionalidad. El autor normalmente será un representante de los clientes pero también puede ser un miembro del equipo de desarrollo que por algún motivo note la ocurrencia de un error que no pueda solucionar en ese momento.

2. *Producto y Módulo*: Especifica en qué sistema y/o módulo se encuentra el error. De igual manera se procede para los casos de nuevas funcionalidades señalando dónde deben ser añadidas. La lista de módulos puede encontrarse en el documento de especificación de requerimientos.

3. *Versión*: La versión del sistema que está siendo utilizada por el autor. Aquí es importante mencionar que se debe definir un sistema de numeración de versiones al principio del proyecto.

4. *Tipo de Cambio*: Se dará una clasificación a los cambios de acuerdo a valores estándares sugeridos por Bugzilla, que es una herramienta del tipo de las que se analizarán en la siguiente sección [Wikipedia s.a.c]:

   4.1. *Bloqueador*: inhibe la continuidad de desarrollo o pruebas del programa.

   4.2. *Crítico*: Caída de la aplicación, pérdida de datos o fuga de memoria severa.

   4.3. *Mayor*: pérdida mayor de funcionalidad como menús inoperantes, datos de salida extremadamente incorrectos o dificultades que inhiben parcial o totalmente el uso del programa.

   4.4. *Normal*: Una parte menor del componente no es funcional.

   4.5. *Menor*: Una pérdida menor de funcionalidad o problemas que pueden ser subsanados por otras funcionalidades similares en el sistema.

   4.6. *Trivial*: Un problema estético como puede ser una falta de ortografía o un texto desalineado.

   4.7. *Mejora*: Solicitud de una nueva característica o funcionalidad.

   Los valores pueden variar de acuerdo al criterio de los desarrolladores pero una vez establecidos para un proyecto, no se deben cambiar.

5. *Resumen*: En caso de un error, se debe describir brevemente la ocurrencia del mismo y luego citar los pasos que llevaron a dicha situación. En caso de una nueva funcionalidad, se debe describir de la misma forma que en el documento de especificación de requerimientos lo que se supone que debe realizar esta nueva funcionalidad.

6. *Responsable*: La persona en el equipo de desarrollo que se encargará de la implementación del pedido. El líder del proyecto debe designar al responsable

en base a factores como la experiencia del programador o a la selección de la persona que desarrolló el módulo en cuestión. No obstante, cualquier integrante del equipo de desarrollo debe ser apto para ser responsable de cualquier pedido.

7. *Prioridad*: En base al tipo de cambio se debe establecer un valor de prioridad, en términos de alta, media o baja (o de otros niveles a definirse por los desarrolladores). Con este valor el responsable debe decidir en qué momento implementar los cambios.

8. *Estado*: Como se señaló en un principio, el objetivo no es solo la realización del pedido sino también su seguimiento. Es por eso que este campo será una lista de estados que se irán agregando a medida que vayan sucediendo con la fecha y hora correspondiente. La lista de valores sugeridos también fueron extraídos de la lista de estados de Bugzilla [Bugzilla2007]. Éstos son:

   8.1. *Nuevo*: Estado inicial. El pedido de cambio podría volver a este estado inicial en el caso de que se deba reasignar el pedido a otra persona.

   8.2. *Asignado*: Cuando uno de los desarrolladores se hace responsable de la implementación del pedido.

   8.3. *Resuelto*: Cuando el desarrollador termina la implementación de los cambios.

   8.4. *Verificado*: Cuando se realizan las validaciones para sobre los cambios, para asegurar su calidad. En caso de no aprobar las pruebas requeridas, se vuelve a trabajar hasta llegar de nuevo al estado Resuelto.

   8.5. *Reabierto*: Cuando se han aprobado las validaciones pero los cambios no satisfacen totalmente los requerimientos solicitados. De igual manera que cuando no se aprueba la etapa Verificado se vuelve a trabajar hasta llegar al estado Resuelto.

   8.6. *Cerrado*: Cuando la verificación es aprobada y los requerimientos son satisfechos completamente.

Al igual que con los tipos de cambios, estos valores citados son una sugerencia. Pueden ser establecidos por la organización pero no deben ser cambiados durante el desarrollo del proyecto.

9. *Observaciones*: Cualquier comentario que el responsable quiera hacer constar por escrito a lo largo de los diversos estados por los que vaya pasando el cambio.

Ya expuesto el formato, se puede notar que mediante este documento se podrá solicitar el pedido de mantenimiento y a su vez consultar el estado de implementación del mismo. Sin embargo, existen ciertos puntos por aclarar.

Primero, es importante que se trabaje siempre sobre una versión electrónica de los documentos de cambios y que se almacenen en algún servidor (por ejemplo reutilizando el servidor de gestión de versiones que debe existir en el modelo). Así se puede consultar en cualquier momento dado el estado actual del pedido. En caso de recibir por escrito el pedido por parte de los clientes, el líder del proyecto deberá generar la versión electrónica del documento para que a partir de ese momento se trabaje siempre con una versión electrónica.

Luego, se debe establecer quiénes redactarán el documento. El autor es el encargado de redactar las primeras cinco partes del documento y enviarlo al líder del proyecto. Este se encarga de la sexta parte que es la asignación del responsable. Por último, es el responsable el que se encarga de la actualización de las últimas tres partes del documento que son la prioridad (utilizada por él para medir sus prioridades en cuanto a su trabajo), los estados por los que pasa el pedido y las observaciones.

Por último, como ya se había mencionado dentro de las descripciones de las partes respectivas, los valores de tipos de cambios, prioridades y estados son sugerencias que pueden ser modificadas por decisión de la organización que adopte este esquema. Sin embargo, se recomienda que una vez iniciado un proyecto no se realicen cambios sobre los mismos para mantener la consistencia de los datos históricos.

7.3 Comunicación Vía Red

Como se había mencionado en el capítulo 5 de planificación acerca de los documentos de la metodología, el Documento de Solicitud y Seguimiento de Cambios es opcional para adecuarse a la filosofía ágil de minimización de la documentación escrita.

La alternativa a la documentación es el uso de herramientas de seguimiento (*bug tracking systems o BTS*) cuyo objetivo es justamente la automatización del proceso descrito en la sección anterior [Wikipedia s.a.c]. El modelo propuesto se basó en el funcionamiento de Bugzilla para la definición del formato de la documentación.

A continuación se dará una idea general de las características del Bugzilla en términos de funcionalidades para entender mejor cómo se podría utilizar una herramienta de este tipo en vez de la documentación descrita anteriormente.

7.3.1 Bugzilla

Bugzilla es una herramienta *BTS* basada en una arquitectura *web*. Utiliza un servidor HTTP (por ejemplo Apache) y un gestor de base de datos (MySql o PostgreSQL) para cumplir sus funciones. Lanzado como software de código abierto por *Netscape Communications* en 1998, y está disponible actualmente de forma gratuita bajo Licencia Pública de Mozilla [Bugzilla s.a.].

Permite organizar en múltiples formas los defectos de software, permitiendo el seguimiento de múltiples productos con diferentes versiones a su vez compuestos de múltiples componentes. Permite además categorizar los defectos de software de acuerdo a su prioridad y severidad, así como asignarles versiones para su solución [Bugzilla2007].

También permite anexar comentarios, propuestas de solución, responsables a quienes asignar la solución y el tipo de resolución que tuvo el defecto. Todo esto se realiza llevando un seguimiento de fechas en las cuales sucede cada evento y enviando mensajes de correo electrónico a los interesados en el defecto (opcionalmente) [Bugzilla2007].

7.3.1.1 Cuentas de Usuarios

Se permite la creación de usuarios y grupos de usuarios. También se permite definir qué grupos de usuarios podrán crear, editar o consultar errores. De esta manera se pueden crear

perfiles para los clientes y desarrolladores y poder llevar a cabo el procedimiento de pedidos de cambios.

7.3.1.2 Datos del Error (Bug)

Permite la carga de diversos datos relacionados con el error (pedido de cambio en la terminología del modelo propuesto) además de los mencionados anteriormente en el Documento de Solicitud y Seguimiento de Cambios. Entre éstos, se pueden destacar los siguientes:

- *Palabras clave*: Este campo sirve para categorizar los errores de manera a poder ser accedidos con facilidad por medio de búsquedas.
- *Plataforma*: En el caso que se necesite saber sobre qué plataforma ocurrió el error.
- *Versión Objetivo (Target Milestone)*: Se señala el número de versión en la que debe estar solucionado el error.
- *Lista CC*: La lista de usuarios que deben recibir un correo si el error sufre cambios.
- *Archivos adjuntos*: Se da la posibilidad de adjuntar archivos que contengan información adicional como por ejemplo, casos de pruebas.
- *Dependencias*: Se permite señalar si la solución de un error está supeditada a la solución de otros errores anteriores.

También se debe mencionar que la herramienta permite la inclusión de campos personalizados en caso de que la organización quiera incluir alguna información adicional a cada pedido de cambio.

7.3.1.3 Búsquedas

Ofrece dos formas de búsqueda de errores, la primera es similar a los buscadores *web* como *Google* y la posibilidad de búsquedas avanzadas con distintos tipos de consultas como por ejemplo una lista de los errores cuya prioridad hayan cambiado en los últimos tres días.

7.3.1.4 Reportes y gráficos

Bugzilla permite la creación de reportes para conocer el estado actual de los errores en la base de datos. Se pueden crear tablas eligiendo dos campos (de los datos del error) como ejes X e Y además de criterios de búsqueda para limitar la cantidad de errores y así desplegar la información. Por ejemplo, se podría elegir Producto como X y Estado como Y para visualizar un reporte de cuántos errores están en cada estado por cada producto.

7.3.1.5 Otras funcionalidades

Bugzilla ofrece más funcionalidades que ya no serán descritas en este documento pero que pueden ser consultadas en el sitio oficial y en los manuales que se pueden obtener con cada versión de la herramienta. También allí se encontrará la información técnica para su instalación y operación.

7.3.2 Otras Herramientas

Otras herramientas de Seguimiento de Errores pueden ser utilizadas para automatizar el proceso de pedidos y seguimiento de errores. Sin embargo, queda a cargo de la organización que las elige la revisión de qué tanto se adecua la herramienta a los procedimientos definidos por el modelo propuesto.

Entre las herramientas *BTS* se pueden mencionar a las siguientes (con el tipo de licencia y el autor) [Wikipedia s.a.d]:

- *JIRA (Propietario - Atlassian Software Systems).*
- *Mantis Bug Tracker (GPL – Victor Boctor).*
- *Eventum (GPL – MySQL AB).*
- *GNATS (GPL – Free Software Foundation).*

7.4 Notas Finales

Se ha descrito en este capítulo cómo llevar a cabo el mantenimiento. Las reuniones son requeridas para ajustarse al modelo propuesto. Sin embargo, para los pedidos y seguimiento de los cambios que se realizan sobre el sistema existen dos opciones que son

la de Documentación Escrita y la del uso de Herramientas de Seguimiento de Errores. Se recomienda el uso de Herramientas para minimizar la documentación escrita total del proyecto. Sin embargo, el Líder del Proyecto podrá optar por el uso de la Documentación Escrita si no posee tiempo suficiente para la instalación y configuración de las Herramientas o si no se posee la infraestructura necesaria en términos de conectividad entre desarrolladores y clientes.

Capítulo 8
Caso de Estudio

Para validar el modelo propuesto, se lo utilizó en el desarrollo de un sistema de Gestión de Consultorios Odontológicos (GCO) que será presentado como Caso de Estudio de este trabajo. El mismo sigue en etapa de desarrollo.

Los resultados presentados son todas las documentaciones resultantes de los dos primeros ciclos de desarrollo.

El tamaño del equipo de desarrolladores es de dos personas:

- Enrique Bañuelos Gómez (Líder del Proyecto).
- Ellen Méndez (Estudiante de Ingeniería Informática, Facultad Politécnica, Universidad Nacional de Asunción).

La persona que representa al consultorio cliente es la Dra. Estela Poggi. Participó en la captación de requerimientos y en las reuniones semanales que el modelo establece.

En el Anexo A se puede apreciar el Documento de Especificación de Requerimientos (DER), que se tomó como base para la estimación de Puntos de Casos de Uso explicada en el Capítulo 5, que se expondrá a continuación.

8.1 Estimación de Esfuerzo

Como también se había explicado en el Capítulo 5, la estimación de esfuerzos se hará en base a los requerimientos funcionales y no a la especificación de casos de uso, ya que las últimas se van escribiendo a medida que se implementan.

8.1.1 Puntos de Casos de Uso sin Ajustar

Las siguientes tablas son el resultado del primer paso (Sección 5.3.1) de la estimación en donde se hallan los pesos de los actores (UAW) y el de los posibles casos de uso (UUCW), tomando como bases los requerimientos funcionales (RF) del DER.

Tabla 8.1. Pesos de Actores (UAW)

Actor	*Valor*
Actor 1 (Administrador)	3 (Complejo)
Actor 2 (Doctor)	3 (Complejo)
Actor 3 (Secretario)	3 (Complejo)

$$UAW = 3 + 3 + 3 = 9 \quad (8.1)$$

Tabla 8.2. Pesos de Casos de Uso (UUCW)

Casos de Uso	*Simple*	*Medio*	*Complejo*
RF1	1		
RF2	1		
RF3	1		
RF4	1		
RF5	1		
RF6		1	
RF7	1		
RF8	1		
RF9	1		
RF10	1		
RF11	1		
RF12	1		
RF13	1		
RF14	1		
RF15	1		
RF16	1		
RF17	1		
RF18	1		
RF19	1		
RF20	1		
SUMATORIA	19	1	0

$$UUCW = 19 \times 5 + 1 \times 10 + 0 \times 15 = 105 \quad (8.2)$$

Por lo tanto, el resultado valor de los casos de uso sin ajustar (UUCP) es,

$$UUCP = UAW + UUCW = 9 + 105 = 114 \quad (8.3)$$

8.1.2 Puntos de Casos de Uso Ajustados (UCP)

Cómo se vio en la Fórmula 5.2 de la Sección 5.3.2, para calcular los UCP se deben hallar los valores de Complejidad Técnica (TCF) y los de Factores del Ambiente (EF). Las tablas que demuestran los resultados de estos valores se presentan a continuación.

Tabla 8.3. Factores de Complejidad Técnica (TCF)

Factor	Descripción	Peso	Valor
T01	Sistema Distribuido	2	0
T02	Objetivo de performance o tiempo de respuesta	1	3
T03	Eficiencia del usuario final	1	4
T04	Procesamiento interno complejo	1	2
T05	El código debe ser reutilizable	1	2
T06	Facilidad de instalación	0,5	2
T07	Facilidad de uso	0,5	4
T08	Portabilidad	2	0
T09	Facilidad de cambio	1	4
T10	Concurrencia	1	3
T11	Incluye objetivos especiales de seguridad	1	0
T12	Provee acceso directo a terceras personas	1	0
T13	Se requieren facilidades especiales de entrenamiento a usuarios	1	2

$$TCF = 0.6 + 0.01 \times \Sigma [Peso(i) \times Valor\ asignado(i)] = 0,83 \qquad (8.4)$$

Tabla 8.4. Factores del Ambiente (EF)

Factor	Descripción	Peso	Valor
E01	Familiaridad con el modelo de proyecto utilizado	1,5	3
E02	Experiencia en la aplicación	0,5	0
E03	Experiencia en orientación a objetos	1	5
E04	Capacidad del analista líder	0,5	3
E05	Motivación	1	3
E06	Estabilidad de los requerimientos	2	3
E07	Personal de medio tiempo	-1	5
E08	Dificultad del lenguaje de programación	-1	0

$$EF = 1.4 - 0.03 \times \Sigma [Peso(i) \times Valor\ asignado(i)] = 0,95 \qquad (8.5)$$

Por lo tanto, el resultado final de Puntos de Casos de Uso será de,

$$UCP = UUCP \times TCF \times EF = 114 \times 0{,}83 \times 0{,}95 = 89{,}889 \quad (8.6)$$

8.1.3 Resultado Final

Una vez que se ha obtenido el valor de UCP, se realiza la conversión a la métrica de Horas Hombre, mediante el conteo de EF para poder hallar cuál es la relación entre UCP y Horas Hombre (Sección 5.3.3).

Tabla 8.5. Resumen de EF para el cálculo de Horas Hombre.

ECF	***PESOS***
Valores menores a 3 (EF1 - EF6)	1
Valores mayores a 3 (EF7 - EF8)	1
Total	2

Por lo tanto, el ajuste será 1 UCP = 20 Horas Hombre.

Con el resultado de la Tabla 8.5, los resultados de esfuerzo serán los siguientes,

$$E\ (Horas\ Hombre) = UCP \times 20 = 1797{,}78\ Horas\ Hombre \quad (8.7)$$

$$E\ (Meses\ Hombre) = E\ (Horas\ Hombre) / 160 = 11{,}24\ Horas\ Hombre \quad (8.8)$$

Así, se tiene que con el equipo de desarrolladores de dos personas, se necesitará un tiempo aproximado de cinco a seis meses de desarrollo.

8.2 Tecnologías y Herramientas Utilizadas

El sistema fue desarrollado para un entorno de ventanas sobre una plataforma Microsoft Windows XP o superior, con el lenguaje de programación C# y se utilizó como Sistema Gestor de Bases de Datos a PostgreSQL.

Las herramientas que fueron utilizadas para aplicar los requisitos impuestos por el modelo son las siguientes.

8.2.1 Subversion

El sistema de gestión de versiones utilizado fue Subversion que es gratuito (*open source*). El mismo fue ideado como el reemplazante “obligado” para el CVS (utilizado como

ejemplo de este tipo de herramientas en la sección 6.2.1) en la comunidad de desarrollo de software de código libre [Tigris s.a.a].

Justamente, se realizó la elección de subversión (SVN) porque además de ofrecer características similares a las del CVS, ofrece nuevas características como el seguimiento de directorios, copias y cambios de nombres, confirmaciones (*commit*) atómicas, soporte para diversos protocolos de comunicación y mejoras de rendimiento [Tigris s.a.a].

El servidor de versiones se encontraba disponible en una dirección pública, por lo que el acceso al código fuente era posible simplemente con un acceso a Internet y una cuenta de usuario en el mismo.

El cliente utilizado es TortoiseSVN (gratuito), que permite ejecutar los comandos SVN desde la interfaz del explorador de Microsoft Windows, permitiendo así no depender de ningún entorno de desarrollo específico, ya que es un programa totalmente independiente y no solamente una extensión [Tigris s.a.b].

8.2.2 Sharp Develop (#develop)

El entorno de desarrollo elegido para el Caso de Estudio fue #develop, que es también es gratuito y soporta los lenguajes C#, VB.Net y Boo [Ic#code s.a.].

Entre las funcionalidades que ofrece, se destacan el soporte para pruebas de unidad, la integración con TortoiseSVN y la disponibilidad de estándares de codificación dentro de la ayuda disponible, y la generación automática de documentación HTML a partir de los comentarios (siempre que se utilicen los comentarios XML del lenguaje).

Las mencionadas características lo hacen ideal para su uso en el Caso de Estudio del modelo propuesto, ya que facilita la aplicación de prácticas requeridas como pruebas de unidad, gestión de versiones y seguimiento de estándares de codificación.

Como nota adicional, se utilizó la herramienta NCover [Ncover s.a.] para obtener estadísticas sobre el porcentaje del código fuente que es ejercitado por las pruebas automatizadas. Sharp Develop permite integrar el NCover dentro del entorno de desarrollo, facilitando la obtención de los resultados luego de correr las pruebas desde el mismo entorno de desarrollo.

8.2.3 Seguimiento de Errores

En este caso, por la poca familiaridad del representante del cliente con el uso de Internet, se optó por usar la documentación mediante procesadores de texto, y gestionar sus cambios mediante el sistema de gestión de versiones utilizado con el código fuente. No se utilizó ninguna herramienta en este caso.

En el Anexo C, se mostrarán los documentos generados durante los dos primeros ciclos del desarrollo ordenados cronológicamente.

8.3 Descripción de los Ciclos de Desarrollo

A continuación, se describirán las actividades de los dos ciclos de desarrollo del Caso de Estudio.

8.3.1 Primer ciclo

La **fecha de inicio del proyecto** es el 14 de agosto del 2007, donde se redacta el Documento de Especificación de Requerimientos (Anexo A). Para el desarrollo del sistema ya se contaba con módulos que proveían el acceso a bases de datos, y funcionalidades básicas como inicio de sesión. Esto agilizó un poco el trabajo del primer ciclo ya que existía una base escrita.

En cuanto al **equipo de desarrollo**, para el primer ciclo solo trabajó una persona, ya que la otra persona que integraría el equipo de desarrollo recién podría acoplarse para el segundo ciclo.

En la primera semana, mientras comenzaba la codificación, se realizó la configuración del servidor SVN para que a partir del código existente antes del inicio del proyecto, se mantenga un control completo sobre las versiones.

Así transcurrió el ciclo de desarrollo hasta la primera reunión con el representante de los clientes, que se realizó conjuntamente con la primera entrega el día 11 de agosto del mismo año.

A partir de allí se cerró el primer ciclo, explicando al cliente que luego esta primera reunión, deberían haber siempre reuniones semanales para informar avances, recibir inquietudes, sugerencias y/o pedidos de cambios.

Como se puede ver en las Especificaciones de Casos de Uso (Anexo B), los casos de uso entregados en este primer ciclo fueron:

- CU1: Crear Cuenta de Administrador (ya estaba escrito previamente).
- CU2: Iniciar Sesión (ya estaba escrito previamente).
- CU3: Cambiar Contraseña (ya estaba escrito previamente).
- CU4: Asignar Permisos sobre Ítems del Menú.
- CU5: Creación de Usuarios de Login.
- CU6: Creación de Personas que utilizarán el sistema.
- CU7: Asignar Perfiles a Usuarios.
- CU8: Carga de Gastos.
- CU10: Carga de Servicios.
- CU11: Carga de Pacientes.
- CU12: Editar Datos Personales.

Las **pruebas automatizadas** fueron implementadas para los casos de usos cinco al doce. Pero es importante destacar que solo se realizaron dos casos de prueba: uno para el caso de uso doce y uno para el resto, ya que estos últimos utilizaban un mismo generador de ABM (Alta-Baja-Modificación).

Al finalizar la reunión, se redactó en el instante con el representante del cliente una **solicitud de mejora**, que es la primera visualizada en el Anexo C, relativa a la posibilidad de tener un bloc de notas en la pantalla principal a modo de recordatorios.

8.3.2 Segundo Ciclo

Al día siguiente de la primera entrega (12 de agosto), comienza el segundo ciclo en el cual se eligieron los casos de uso de carga de datos por parte de los doctores, dejando para ciclos posteriores lo relacionado con los horarios y con los listados.

Se debía terminar la implementación de lo relacionado con las atenciones a pacientes, pagos realizados por los mismos, carga de ocurrencias de gastos y control de stock de insumos.

También, en base a conversaciones con el representante del cliente, se decidió realizar un nuevo documento de solicitud de cambio (segundo documento del Anexo C) para registrar la posibilidad de distintos tipos de precio en base a los seguros médicos existentes con los cuales pueda trabajar un doctor.

De las dos solicitudes de cambio existentes, se dio prioridad baja a la primera y alta a la segunda, ya que esta última tiene relación directa con la carga de atención a pacientes que es parte de los objetivos del segundo ciclo.

Transcurrió una semana para realizar una nueva reunión (18 de agosto) donde en base a conversaciones, se decidió incluir un nuevo caso de uso sencillo (tercer documento del Anexo C) en el cual se ingresan las distintas unidades de medida a ser utilizadas en el control de stock. La prioridad fue alta también porque está relacionado con un objetivo del ciclo actual.

En la reunión del 25 de agosto, simplemente se informó al representante del cliente los cambios que hasta ese momento se implementaron, sin generarse ninguna solicitud de cambio.

La lista de casos de uso implementados en el segundo ciclo, y que fue presentada en la reunión del 2 de octubre, es la siguiente:

- CU9: Carga de Insumos.
- CU13: Carga de Seguros Médicos.
- CU14: Búsqueda de Pacientes.

- CU15: Carga de Atención al Paciente.
- CU16: Carga de Ocurrencias de Gastos.
- CU17: Carga de Unidades de Medida.
- CU18: Pagos de Pacientes.
- CU19: Carga de Proveedores.
- CU20: Carga de Compras de Insumos.
- CU21: Carga de Utilización de Insumos.

En este caso, las **pruebas automatizadas** abarcaron a los casos de uso 13, 16, 17 y 19 como parte del generador de ABM, y se escribieron casos de pruebas específicos para caso de uso 15.

En este ciclo, se utilizó **una semana de programación en pares** para nivelar el conocimiento con la segunda persona que integra el equipo de desarrollo, y así permitir que la misma desarrolle los casos de uso relacionados a insumos.

La **reunión** en la que se llevó a cabo la **segunda entrega** se realizó el martes 2 de octubre de 2007, en la cual se generaron dos **solicitudes de cambios**. Una para reportar un error menor y otra para agregar una extensión a una funcionalidad ya ofrecida por el sistema (como se puede ver en los últimos dos documentos del Anexo C).

Capítulo 9 Conclusiones y Trabajos Futuros

Luego de la implementación de los dos primeros ciclos de desarrollo del Caso de Estudio explicado en el capítulo anterior, se presentarán los resultados obtenidos, las conclusiones resultantes y los trabajos futuros.

9.1 Resultados

A continuación se presentan los resultados del Caso de Estudio en base a dos criterios. El primero relaciona el tiempo de desarrollo con el tiempo estimado por los Puntos de Casos de Uso; mientras que el segundo criterio evalúa las ventajas reales obtenidas de la utilización del modelo propuesto.

9.1.1 Evaluación de Tiempos

Utilizando la misma técnica de estimación de esfuerzo del proyecto, en base a los requerimientos funcionales cubiertos en cada ciclo se realizó la estimación para obtener el tiempo aproximado de desarrollo de cada ciclo, cuyos resultados se ven en la Tabla 9.1.

Tabla 9.1. Estimación de Esfuerzo de cada ciclo.

CICLO	*Semanas / Hombre*
Primer Ciclo	14
Segundo Ciclo	9

Como se puede ver, para dos personas los tiempos son de aproximadamente siete y cuatro semanas respectivamente. Claro está, que no son estimaciones exactas pero sirven de referencia para la planificación de proyectos.

Así, se puede ver que el primer ciclo se realizó en cuatro semanas, aprovechando la disponibilidad de código reutilizable como se mencionó en el principio de la descripción del primer ciclo. Teniendo eso en cuenta, se pudo mejorar el tiempo estimado por la métrica de Puntos de Casos de Uso utilizada.

En el segundo ciclo también se cumplió con la estimación ya que se terminaron los objetivos propuestos en tres semanas. Sumando ambos ciclos, se obtuvo una mejora de cuatro semanas de avance en relación a la estimación de esfuerzo inicial.

9.1.2 Ventajas Obtenidas

Independientemente de los tiempos obtenidos como resultado, las ventajas obtenidas de la aplicación del modelo propuesto son las siguientes:

1. El establecimiento de una metodología de trabajo bien definida que incluye fechas límite para entrega de incrementos de software, asignación de responsables sobre las tareas existentes y documentaciones resultantes.

2. El uso de una herramienta de gestión de versiones facilita el trabajo en equipo, cumpliendo con todas las ventajas que se señalan en el Capítulo 6.

3. La utilización de programación en pares demostró ser útil para nivelar el conocimiento sobre el sistema en los integrantes del equipo.

4. Utilización de estándares de codificación para ayudar a la legibilidad del código por cualquiera de los integrantes del equipo.

5. El uso de pruebas de unidad con los criterios establecidos por el modelo propuesto permitieron una cobertura de código promedio de 40%. La cobertura de código recomendada para modelos que utilizan pruebas de unidad sobre todos los casos de uso no debe ser menor a 80% [Cornett2007].

6. La definición de un mecanismo mediante el cual el cliente puede realizar pedidos de modificación y mejoras sobre el sistema, y a su vez, estar enterado del progreso de la implementación de los mismos.

7. El establecimiento de reuniones semanales con el representante de los clientes involucra más a los clientes en el desarrollo del sistema.

Un Caso de Estudio no es suficiente para realizar afirmaciones contundentes, pero los resultados son más que alentadores en comparación a la situación de nuestros entornos, en base a lo expuesto en la encuesta del Capítulo 4.

9.2 Conclusiones

Luego de exponer los resultados del caso de estudio realizado, se presentarán los aportes principales del presente trabajo, para luego describir la conclusión del mismo.

9.2.1 Aportes

Los dos aportes principales de este trabajo son:

- Análisis del Estado del Desarrollo de Software en el país: Se realizó, mediante una encuesta, una evaluación de rasgos básicos relativos al desarrollo de software en nuestro país, donde se identificaron los problemas más importantes que deben enfrentarse a la hora de proponer un modelo de procesos.

- Modelo Inicial de Desarrollo de Software (MIDS): En base al resultado de la encuesta realizada, se realizó una propuesta de modelo de procesos que se adecue a los entornos de desarrollo locales, y mediante un caso de estudio, se pudieron ver las ventajas obtenidas de la aplicación del modelo.

Otro aporte relacionado con los anteriores es la medición de la aplicabilidad de los distintos paradigmas de desarrollo de software a los entornos locales, dando como resultado que un enfoque "ágil" podría ofrecer mejores resultados que un enfoque "tradicional".

9.2.2 Conclusión

En base a la hipótesis realizada (Sección 1.4), al estudio de las características de los modelos existentes (Capítulos 2 y 3), a la encuesta realizada para determinar el estado de los entornos locales (Sección 4.2), y a los resultados obtenidos en el Caso de Estudio (Sección 9.1); se llegó a la propuesta de un Modelo de Procesos, llamado "Modelo Inicial de Desarrollo de Software MIDS" que se ajusta a las necesidades de los Entornos Locales de Desarrollo, enfrentando sus principales problemas como:

- La falta de disciplina en el desarrollo del software: Resultado 1 (Sección 9.1.2).

- Falta de utilización de Herramientas de Versiones: Resultado 2 (Sección 9.1.2).

- Falta de utilización de prácticas de programación: Resultados 3 y 4 (Sección 9.1.2).

- Carencia de métodos de evaluación de la calidad del software desarrollado: Resultados 5 y 7 (Sección 9.1.2).

- Definición de mecanismos de interacción con los clientes: Resultados 6 y 7 (Sección 9.1.2).

- Cumplimiento de plazos establecidos por la herramienta de estimación de esfuerzos utilizada: Evaluación del Tiempo (Sección 9.1.1).

9.3 Trabajos Futuros

A continuación, se describirán los distintos trabajos que pueden realizarse como continuación del presente trabajo.

9.3.1 Utilización a Nivel Comercial

Uno de los principales trabajos posteriores es el de impulsar la utilización del modelo propuesto en las empresas de desarrollo de software locales, ya que el objetivo del trabajo es el de proponer un modelo de procesos que pueda ser utilizado por nuestras empresas.

Como revelaron los resultados de la encuesta realizada, existe una buena cantidad de empresas que necesitan definir un modelo de procesos, y el MIDS podría ser su paso inicial en la utilización de los mismos.

9.3.2 Documentación de Experiencias del MIDS

Esto se podría realizar en conjunción con el trabajo futuro presentado anteriormente. La documentación de más casos de estudio podrá ratificar los resultados obtenidos en este trabajo. En caso de que los nuevos resultados no estén acordes a los presentes, se podrían proponer las modificaciones al modelo que sean necesarias.

Independientemente de los resultados obtenidos en los nuevos casos de estudio, mientras más utilizaciones se reporten, más se favorecerá a su adopción por parte de otras empresas.

9.3.3 Extensión del Modelo

Como se enunció en uno de los objetivos del trabajo, el modelo propuesto debe servir como paso inicial hacia la utilización de otros modelos más complejos y reconocidos a nivel mundial.

Entonces, se propone como trabajo futuro, tomar como base un entorno de desarrollo que utilice el MIDS y realizar los pasos (agregar prácticas y/o documentos) de manera tal que se logre evolucionar naturalmente desde el MIDS hacia un modelo objetivo deseado.

Se recomienda como modelo objetivo Extreme Programming (XP), por ser del mismo paradigma (ágil) utilizado por MIDS y por la popularidad y aceptación de XP. Sin embargo, el modelo propuesto podría ser extendido hacia cualquier otro enfoque evolutivo.

9.3.4 Integración con Modelos de Calidad Regionales

Por último, se propone la integración del MIDS con estándares de calidad regionales. Esto obligará a la introducción de más prácticas al modelo propuesto para adecuarse a los distintos requisitos impuestos por el modelo de calidad seleccionado. Ejemplos de estándares regionales de calidad son el MPS-BR (Brasil) [Softex2006] y el MoProSoft (México) [Amcis2005].

De esta manera se logrará que el modelo propuesto (o el resultante de su ampliación), ya no sea visto solamente como un paso inicial hacia el uso de modelos de procesos, sino que ya pueda ser tenido en cuenta como el modelo de procesos a utilizar para lograr certificaciones de calidad.

9.3.5 Estimación de Esfuerzos Evolutiva

En el MIDS se utilizó la técnica de estimación basada en Puntos de Casos de Uso. La misma se basa en un conjunto de parámetros cuyos valores se establecen en base a criterios sobre todo el proyecto.

Como mejor complemento al MIDS, se propone el desarrollo de un método de Estimación de Esfuerzos Evolutivo, que dinámicamente ajuste los parámetros iniciales a medida que avanza el proyecto, para obtener resultados más precisos.

Referencias

[Abrahamsson2002] P. Abrahamsson, O. Salo, J. Ronkainen, J. Warsta: Agile Software Development Methods, Review and Analisis. VTT Publications, 2002.

[AgileAlliance s.a.] Agile Alliance: <http://www.agilealliance.org/>.

[AgileManifesto2001] Agile Manifesto (Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunnimgham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas): <http://www.agilemanifesto.org/>. 2001.

[Amcis2005] Asociación Mexicana para la Calidad en Ingeniería del Software (AMCIS): Modelo de Procesos para la Industria de Software MoProSoft. Universidad Nacional Autónoma de México (UNAM), 2005.

[Bar2003] M. Bar, K. Fogel: Open Source Development With CVS. Tercera Edición. Paraglyph Press, 2003.

[Beck1999a] K. Beck: Embracing Change with Extreme Programming. IEEE Computer 32(10): 70-77. 1999.

[Beck1999b] K. Beck: Extreme Programming Explained. Addison-Wesley, 1999.

[Bugzilla2007] The Bugzilla Team: Bugzilla Guide – 3.0 Release. 2007. <www.bugzilla.org>.

[Bugzilla s.a.] Bugzilla.org: Download Bugzilla. <http://www.bugzilla.org/download/>.

[Cornett2007] S. Cornett: Code Coverage Analysis. Bullseye Testing Technologies. 2007. <http://www.bullseye.com/coverage.html>.

[Dahl1972] O. J. Dahl, E. W. Dijkstra, C. A. R. Hoare: Structured Programming. Academic Press, New York, 1972.

[Fernández1996] P. Fernández: Determinación del Tamaño Muestral. Unidad de Epidemiología Clínica y Bioestadística. Complejo Hospitalario Juan Canalejo. 1996. <www.fisterra.com/mbe/investiga/9muestras/9muestras.htm>.

[George2002] Boby George: Analysis and Quantification of Test Driven Development Approach. North Carolina State University. 2002.

[Ghezzi1991] Carlo Ghezzi. Fundamentals of Software Engineering. Prentice-Hall. 1991.

[Herrera1999] R. Herrera, R. Caldera, M. Martínez: Análisis y Diseño de Sistemas con el Lenguaje de Modelado Universal (UML). Universidad Católica "Redemptoris Mater" de Managua, Nicaragua. 1999.

[Highsmith1997] Jim Highsmith: Messy, Exciting and Anxiety-Ridden: Adaptive Software Development. American Programmer, 1997.

[Highsmith2000] J. Highsmith: Agile Software Development. A Collaborative Approach to Managing Complex Systems. Dorset House Publishing, 2000.

[Hunt2006] J. Hunt. Agile Software Construction. Springer, 2006.

[Ic#code s.a.] Ic#code: #develop. <http://www.icsharpcode.net/OpenSource/SD/>.

[IEEE1990] "IEEE Standard Glossary of Software Engineering Terminology," IEEE std. 610.12-1990, 1990.

[IEEE2004] Guide to Software Engineering Body of Knowledge. IEEE. 2004.

[Iso2006] International Organization for Standarization: ISO/IEC 15504-5:2006.

<http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38934>.

[Kruchten2000] P. Kruchten. The Rational Unified Process: An Introduction. Addison-Wesley. 2000.

[Langr2005] J. Langr: Pair Programming Observations. Langr Software Solutions, 2005.

[Letelier2006] P. Letelier, Mª C. Penadés: Metodologías ágiles para el desarrollo de software, eXtreme Programming (XP). 2006.

[Martínez2007a] M. Martínez Mígueles. Revista Information Technologies. Agosto, 2007.

[Martínez2007b] S. Martínez. Modelo de Desarrollo de Sistemas de Información Municipal. Universidad Mayor de San Andrés, Bolivia. 2007.

[Ncover s.a.] NCover Code Coverage: <http://ncover.org/>.

[Oncins s.a.] M. Oncins: Encuestas: Metodología para su Utilización. Ministerio de Trabajo y Asuntos Sociales. España. <http://www.mtas.es/insht/ntp/ntp_283.htm>.

[Paulk1993] M. Paulk, B. Curtis, M. Beth, C. Weber: Capability Maturity Model for Software, Version 1.1. Software Engineering Institute - CMU, 1993.

[Peralta s.a.] M. Peralta: Estimación del Esfuerzo Basada en Casos de Uso. Instituto Tecnológico de Buenos Aires.

[Poole2006] D. Poole. Can agile development make your team more productive? ACM Queue. Octubre, 2006.

[Schwaber2002] K. Schwaber, M. Beedle: Agile Software Development with Scrum. Prentice Hall, 2002.

[Sei s.a.] Software Engineering Institute: CMMI Website. Carnegie Mellon University. <http://www.sei.cmu.edu/cmmi/>.

[Shaw2003] M. Shaw: Writing Good Software Engineering Research Papers. Carnegie Mellon University. IEEE Computer Society, 2003.

[Sierra2003] Restituto Sierra Bravo: Tesis Doctorales y Trabajos de Investigación Científica. Quinta Edición. Thomson, 2003.

[Softex2006] Softex: Melhoria de Processos do Software Brasileiro. 2006. <http://www.softex.br/mpsbr/>.

[Sommerville2005] Ian Sommerville. Ingeniería del Software, 7ª edición. Pearson. 2005.

[Steindl s.a.] Christoph Steindl: Test Driven Development. Agile Alliance.

[Tigris s.a.a] Tigris.org: Subversion Project. <http://subversion.tigris.org/>.

[Tigris s.a.b] Tigris.org : TortoiseSVN Project. <http://tortoisesvn.tigris.org/>.

[Takeuchi1986] H. Takeuchi, I. Nokata: The New Product Development Game. Hardvard Bussines Review, 1986.

[TDD s.a.] Introduction to Test Driven Design: <http://www.agiledata.org/essays/tdd.html>.

[Rediex2007] Red de Inversiones y Exportaciones: Más de 800 desarrolladores independientes y unas 130 empresas de software. 2007. <http://www.rediex.gov.py/index.php?option=com_content&task=view&id=376&Itemid=451>.

[Reynoso2004] C. Reynoso. Métodos Heterodóxos en Desarrollo de Software. Universidad de Buenos Aires. MSDN, 2004.

[Wikipedia s.a.a] Wikipedia.org: Revision Control. <http://en.wikipedia.org/wiki/Revision_control>.

[Wikipedia s.a.b] Wikipedia.org: Puntos de Caso de Uso. <http://es.wikipedia.org/wiki/Puntos_de_caso_de_uso>.

[Wikipedia s.a.c] Wikipedia.org: Sistema de Seguimiento de Errores. <http://es.wikipedia.org/wiki/Sistema_de_seguimiento_de_errores>.

[Wikipedia s.a.d] Wikipedia.org: Comparison of Issue Tracking Systems. <http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems>.

Anexo A. Especificación de Requerimientos

A continuación, se desplegará el contenido del documento de especificación de requerimientos (DER) del Sistema GCO, desarrollado como Caso de Estudio del presente trabajo. No se incluirán la portada e índices, sino solamente el contenido.

1 Introducción

El desarrollo de sistemas informáticos debe estar documentado inicialmente, para poder guiar el proceso a lo largo del tiempo. Justamente, con este documento, se pretende establecer dicha guía.

1.1 Objetivos del documento

Como se señaló en el párrafo inicial, el objetivo del documento es servir de guía para el proceso de desarrollo del Sistema de Gestión de Consultorios Odontológicos GCO. Esto se logra mediante la especificación de los distintos requerimientos que el sistema deberá cumplir. Éstos se pueden clasificar en Requerimientos Funcionales y No Funcionales.

Los Requerimientos Funcionales (RF) son las funcionalidades que el sistema de-be ofrecer al usuario una vez que esté terminado. Por otro lado, los Requerimientos No Funcionales (RNF) son aquellas cualidades que el sistema debería exhibir pero que no están relacionadas directamente con las funcionalidades. Ejemplos de RNF son la robustez, seguridad, calidad de las interfases gráficas, portabilidad, entre otros.

Por lo tanto, este documento describirá cada uno de los distintos RF y RNF que el sistema GCO deberá cumplir al final del proceso.

1.2 Ámbito de la Aplicación

Por las características de un consultorio odontológico, el sistema GCO podrá ser utilizado por distintos Doctores para la gestión de sus Fichas de Atención a Pacientes, y a su vez por un usuario que actúe como secretario de los profesionales que se encuentren en el consultorio, para gestionar la asignación de turnos a los pacientes.

Por lo tanto, el sistema deberá soportar una arquitectura de red local (LAN) para comunicar a todos los usuarios del sistema, sin la necesidad de acceso a Internet, para cumplir con sus funcionalidades.

1.3 Definiciones, Siglas y Abreviaturas

- LAN: Local Area Network. Red de Área Local.

- GCO: Gestión de Consultorios Odontológicos.

- GUI: Grafic User Interface. Interfaz gráfica de usuario.

- Login: Inicio de Sesión.

- Mbps: Mega Bits por Segundo. Unidad de medida de tasa de transferencia de datos.

- RF: Requisitos Funcionales.

- RNF: Requisitos No Funcionales.

- SGBD: Sistema Gestor de Bases de Datos.

2 Descripción General

A continuación, se describirá a grandes rasgos las características que el Sistema GCO ofrecerá.

2.1 Función y Propósito

El propósito del Sistema GCO es el de permitir la gestión de consultorios odontológico por parte de los profesionales, más la opción de la gestión de los horarios por parte de un usuario que actúe de secretario.

La gestión por parte de los profesionales estará abocada a la carga y consulta sobre las Fichas de Atención a Pacientes, más la gestión de las cuentas de los mismos. Además, permitirá la carga de los gastos incurridos para poder facilitar un control de ingresos y egresos. Por último, se permitirá el control de stock de sus insumos y consultar su agenda.

El usuario de secretaría se encargará de carga y actualización de los horarios de atención de cada Doctor, y de la asignación de turnos a los pacientes.

Como aclaración, el Sistema GCO proveerá un usuario administrador para poder asignar uno de estos dos roles (Doctor o Secretario) a los demás usuarios, y para asignar permisos a estos roles sobre pantallas disponibles desde ítems del menú principal.

2.2 Alcance de la Aplicación

En base a lo mencionado anteriormente, el Sistema GCO ofrecerá funcionalidades para la automatización de los procesos de gestión de cuentas y cobros a los clientes, almacén de datos de Fichas de Atención a Pacientes, control de stock de insumos, registro de gastos y consulta de agenda laboral.

Sin embargo, es importante aclarar que por más que en pantalla se muestren los importes de los servicios de atención con el Impuesto al Valor Agregado, el Sis-tema GCO no incluye la emisión de facturas ni la gestión contable del consultorio.

También proveerá las interfaces necesarias para el manejo de los horarios de atención de los doctores y asignación de turnos a pacientes.

Por último, permitirá la creación de usuarios y la asignación de roles a los mismos.

2.3 Restricciones Generales

El sistema será desarrollado con tecnologías basadas en *Windows Forms* (formularios para Windows), así que el primer requerimiento para su implementación será la utilización de sistemas operativos Windows XP o superior. Esto será suficiente en caso de ser utilizado por un solo usuario.

En caso de querer utilizar el sistema en un entorno multiusuario, con varios profesionales siendo asistidos por una secretaría, se requerirá una LAN que interconecte a todas las computadoras que utilizarán el sistema. Como ya se mencionó anteriormente, la conexión a Internet no es necesaria.

Más adelante se darán especificaciones técnicas en cuanto a requerimientos de Hardware, Software y Comunicaciones.

2.4 Descripción del Modelo

A continuación se describirán los modelos del sistema (arquitectura del mismo) y del proceso de desarrollo (metodología de trabajo).

2.4.1 Sistema

El sistema se basará en una arquitectura Cliente – Servidor. Este esquema consiste en una máquina que se encargue de almacenar todos los datos en un SGBD, y atiende los pedidos de las demás computadoras, que tendrán instalado el software cliente, que es la interfaz de ventanas del Sistema GCO.

Por lo tanto, la computadora que actúa de Servidor, debe estar encendida siempre que algún cliente desee utilizar el sistema. Es más, es recomendable que el servidor esté prendido y conectado a la red local las 24 horas, todos los días; y que dicha máquina no sea accesible a ningún usuario en particular.

2.4.2 Proceso

La metodología de trabajo a utilizarse utiliza un enfoque iterativo para la construcción de software. Esto quiere decir que mensualmente se entregarán incrementos de software a ser probados por los clientes, y así ir dando la oportunidad a los mismos de dar sugerencias acerca de lo que se esté probando, y obtener retroalimentación lo antes posible.

Para solicitar cambios sobre el sistema, se entregará como adjunto (Anexo A) una guía de documento de pedido y seguimiento de cambios, para realizar las solicitudes pertinentes.

También se establece como parte de la metodología reuniones semanales con los clientes para realizar informes de lo desarrollado hasta ese momento, y recibir pedidos de cambios.

Este documento servirá como base de acuerdo entre las partes sobre las funcionalidades que deberá exhibir el sistema al final del proyecto. Se podrán realizar cambios sobre este documento, previo acuerdo entre ambas partes, redactando una nueva versión del mismo. Este nuevo número de versión será indicado en el encabezado.

3 Especificación de Requisitos

Esta sección está definida en dos partes: La especificación de Requisitos Funcionales y la de Requisitos No Funcionales.

3.1 Requisitos Funcionales

A continuación, se enumerarán los roles iniciales predefinidos, y las funcionalidades que exhibirá el sistema.

3.1.1 Roles

El sistema define por defecto tres roles:

RL1: Administrador. Es el rol que tendrá permisos para utilizar por completo la aplicación. Sin embargo, es recomendable que luego de la instalación del sistema, se reduzcan sus permisos a las funcionalidades exclusivas de administración (que serán citadas más adelante), sin que se solapen con las del Doctor y el Secretario.

RL2: Doctor. Este es el rol principal de la aplicación. Tendrá a su cargo la car-ga de las Fichas de Atención al Paciente (con la creación de los mismos cuan-do sea necesario), el manejo de las cuentas de sus pacientes, y la consulta y/o edición de los turnos que le correspondan. También le permitirá la carga de gastos y el manejo de stock de insumos.

RL3: Secretario. Se encargará de la manutención de la asignación de turnos de los distintos doctores del consultorio.

3.1.2 Funcionalidades

Funcionalidades para el rol de Administrador:

RF1: Podrá crear usuarios del sistema cargando el nombre de login y la contraseña.

RF2: Podrá crear datos de una persona, y asociarlos con un usuario de login.

RF3: Podrá asociar uno de los roles a los usuarios de login creados.

RF4: Podrá asociar permisos sobre ítems del menú a cada uno de los roles del sistema.

Funcionalidades de rol de Doctor:

RF5: Podrá crear pacientes cargando los datos personales del mismo.

RF6: Podrá cargar los datos del trabajo realizado sobre el paciente en Fichas de Atención al Cliente. A su vez, podrá cargar el costo del tratamiento, y el pa-go realizado por el paciente. También, podrá especificar el responsable del pago de la cuenta, en caso de ser otra persona que no sea el paciente en cuestión.

RF7: Podrá consultar el estado de cuentas de un paciente en cualquier momento.

RF8: Podrá crear distintos tipos de gastos.

RF9: Podrá cargar la ocurrencia de los gastos, con la fecha y el monto.

RF10: Podrá crear distintos tipos de insumos que serán controlados por el sistema.

RF11: Podrá cargar la adquisición o consumo de los insumos registrados en el sistema.

RF12: Podrá consultar los saldos de insumo que posee.

RF13: Podrá consultar un resumen de ingresos y egresos en cualquier momento.

RF14: Podrá consultar su agenda de turnos, permitiéndole modificar los turnos.

RF15: Visualizará en la pantalla los tres días siguientes de su agenda a modo de recordatorio.

Funcionalidades del Rol de Secretario:

RF16: Podrá consultar las agendas de todos los Doctores registrados en el consultorio.

RF17: Podrá asignar turnos y marcar si los pacientes asistieron o no.

Funcionalidades Comunes:

RF18: El usuario deberá poder iniciar sesión mediante el uso de su nombre de login y su contraseña.

RF19: El usuario podrá cambiar su contraseña.

RF20: El usuario podrá editar sus datos personales.

3.2 Requisitos No Funcionales

Estos requisitos se verán agrupados en tres categorías relacionadas con la Seguridad del sistema y de los datos, la Mantenibilidad referente a las posibles modificaciones y/o extensiones del sistema, y los requisitos de Hardware, Software y Comunicaciones.

3.2.1 Seguridad

Cada rol tiene un distinto nivel de acceso a datos. Por ejemplo, un secretario no podrá ver los datos de las fichas de los doctores, ni acceder al menú de carga de gastos. Esto está delimitado en los requisitos funcionales. Sin embargo, un requisito de seguridad que no quedó delimitado es el de acceso a datos entre distintos doctores.

El sistema no permitirá que un doctor acceda a ninguno de los datos de otro doctor. La tipificación de gastos e insumos será independiente entre cada uno de los doctores del consultorio, garantizando así que en ningún momento se solaparán los datos.

El usuario administrador tampoco podrá acceder a datos de atención a clientes, ni a los de las agendas de ninguno de los doctores. Es más, se recomienda que luego de la creación del usuario administrador, se eliminen de su rol todos los ítems del menú que acceden a funcionalidades no administrativas.

Por otro lado, además de la seguridad de datos entre usuarios del sistema, es importante que el servidor utilice dispositivos eléctricos estabilizadores y genera-dores para no perder datos. Además, se configurará la realización de copias de resguardo de la base de datos constantemente. El equipo de desarrollo comunicará a los usuarios la carpeta en donde se encontrarán dichas copias, de manera que puedan copiarlas en medios externos para aumentar la seguridad.

3.2.2 Mantenibilidad

El mantenimiento en un sistema informático puede definirse como todos los cambios que se realicen sobre el mismo una vez que se realice la entrega final. Estos cambios pueden ser para corregir errores, para adaptar el sistema a nuevos requerimientos, o para perfeccionarlo.

Por la naturaleza de la metodología de trabajo, mantenimientos se podrán realizar sobre el sistema a partir del segundo incremento de software entregado a los clientes. Por lo tanto el mantenimiento será parte natural del sistema en sí. Cuan-do se llegue a un acuerdo, se realizará una entrega formal, y a partir de ahí, cualquier pedido de cambio tendrá un trato distinto y un presupuesto y plan de implementación.

3.2.3 Hardware, Software y Comunicaciones

La computadora que actuará como "servidor", independientemente de si también actúa de cliente, deberá poseer las siguientes características mínimas:

- Procesador: AMD Athlon XP 2000+ o superior; o Intel Pentium 4 de 2000 MHz.

- Memoria RAM de 512 MB.

- Disco Duro Serial ATA de 80 GB.

El servidor podrá utilizar cualquier sistema operativo soportado por el SGBD Postgres SQL, como por ejemplo Microsoft Windows XP o cualquier distribución de Linux.

Sin embargo, para correr el programa cliente (GUI) se requiere el uso del sistema operativo Microsoft Windows XP, más las librerías del .NET Framework, que serán instaladas con el sistema.

En cuanto a Comunicaciones, en caso de usar el sistema para varios usuarios, todas las computadoras deberán estar conectadas en una LAN, con velocidades recomendadas de 100 Mbps.

4 Especificación de la Entrega

Cuando estén implementadas todas las funcionalidades especificadas en los **RF**s, se procederá a la entrega final del sistema, con un disco de instalación y los manuales correspondientes.

El posible cronograma de desarrollo e implementación del sistema tendrá una duración estimada de seis meses (en base a estimaciones de puntos de casos de uso), y cuyas actividades son las siguientes:

- Diseño de la Arquitectura: Una a dos semanas.

- Desarrollo: Veinte semanas. Es importante resaltar que cada cuatro semanas se deberá entregar un incremento como se había establecido en la metodología de trabajo. También, en este periodo se realizarán las reuniones semanales.

- Escritura del manual de usuario y entrega final: Una semana.

Es importante mencionar que no se toma un tiempo aparte para la capacitación e instalación porque eso se realizará continuamente con cada entrega de software que se realice. Por lo tanto, la adquisición de las máquinas y de la infraestructura de red ya se debe realizar desde un principio del proyecto.

Anexo A. Documento de Solicitud y Seguimiento de Cambios.

Consta de las siguientes partes:

1. Autor: Es el nombre, organización y cargo de la persona que origina el documento. Es decir, la persona que encuentra la falla, o que solicita una nueva funcionalidad.

2. Producto y Módulo: Especifica en qué sistema o módulo del mismo, se encuentra el error. De igual manera se procede para los casos de nuevas funcionalidades. La lista de módulos puede encontrarse en el documento de especificación de requerimientos.

3. Versión: La versión del sistema que está siendo utilizada por el autor. El sistema de numeración para este campo será el número de entrega realizada.

4. Tipo de Cambio: Se dará una clasificación a los cambios de acuerdo a los siguientes valores:

    4.1. Bloqueador: inhibe la continuidad de desarrollo o pruebas del programa.

    4.2. Crítico: Caída de la aplicación, pérdida de datos o fuga de memoria severa.

    4.3. Mayor: pérdida mayor de funcionalidad, como menús inoperantes, datos de salida extremadamente incorrectos, o dificultades que inhiben parcial o totalmente el uso del programa.

    4.4. Normal: Una parte menor del componente no es funcional.

    4.5. Menor: Una pérdida menor de funcionalidad, o un problema al cual se le puede dar la vuelta.

    4.6. Trivial: Un problema cosmético, como puede ser una falta de ortografía o un texto desalineado.

    4.7. Mejora: Solicitud de una nueva característica o funcionalidad.

5. Resumen: En caso de un error, se debe describir brevemente la ocurrencia del mismo, y luego citar los pasos que llevaron a dicha situación. En caso de una nueva funcionalidad, se deberá describir el nuevo Requisito Funcional.

6. Responsable: Dejar en blanco. Será completado por el Equipo de Desarrollo.

7. Prioridad: Dejar en blanco. Será completado por el Equipo de Desarrollo.

8. Estado: En este campo irá la lista de los distintos estados por los que pasará la solicitud. Se deberá completar colocar la fecha y el estado Nuevo. A partir de allí, se podrá pasar a los siguientes estados:

8.1. Nuevo: Estado inicial. El pedido de cambio podría volver a este estado inicial en el caso de que se deba reasignar el pedido a otra persona.

8.2. Asignado: Cuando uno de los desarrolladores se hace responsable de la implementación del pedido.

8.3. Resuelto: Cuando el desarrollador termina la implementación de los cambios.

8.4. Verificado: Cuando se realizan las validaciones para sobre los cambios, para asegurar su calidad. En caso de no aprobar las pruebas requeridas, de vuelve a trabajar hasta llegar de nuevo al estado Resuelto.

8.5. Reabierto: Cuando se han aprobado las pruebas, pero los cambios no satisfacen totalmente los requerimientos solicitados. De igual manera que cuando no se aprueba la etapa Verificado, se vuelve a trabajar hasta llegar al estado de Resuelto.

8.6. Cerrado: Cuando la verificación es aprobada y los requerimientos son satisfechos completamente.

9. Observaciones: Cualquier comentario que el responsable quiera hacer constar por escrito, a lo largo de los diversos estados por los que vaya pasando el cambio.

Anexo B. Especificación de Casos de Uso

De igual manera que en la especificación de requerimientos, se adjunta el contenido del Documento de Especificación de Casos de Uso (ECU) de los casos de uso desarrollados en las dos primeras iteraciones del modelo propuesto.

1 Introducción

En el presente documento se enumerarán primero todos los Actores que se relacionarán con el Sistema de Gestión de Consultorios Odontológicos (GCO). Actores es la denominación que se da a los distintos tipos de usuarios que utilizarán el sistema, y que se extraen a partir de la definición de roles del Documento de Especificación de Requerimientos (DER). Por cada actor se realizará una referencia al rol (RL) especificado en el DER y el número de la última entrega del sistema en la fue modificado.

Luego se listarán los Casos de Uso (CU) del sistema, agrupado por actores.

Más adelante, se describirán todos los CU que posea el sistema. En el título del CU se verá entre paréntesis la referencia al requisito funcional (RF) del DER y el número de la última entrega en la que sufrió modificaciones.

2 Lista de Actores

En base a las especificaciones del DER, el sistema contará con los siguientes actores:

AC1: Administrador (RL1, 1).

AC2: Doctor (RL2, 1).

AC3: Secretario (RL3, 1).

3 Enumeración de Casos de Uso por Actores

AC1: Administrador.

CU1: Crear Cuenta Administrador.

CU2: Iniciar Sesión.

CU3: Cambiar Contraseña.

CU4: Asignar permisos sobre ítems del menú.

CU5: Creación de Usuarios de Login.

CU6: Creación de Personas que utilizarán el sistema.

CU7: Asignar Perfiles a Usuarios.

AC2: Doctor.

CU2: Iniciar Sesión.

CU3: Cambiar Contraseña.

CU8: Carga de Gastos.

CU9: Carga de Insumos.

CU10: Carga de Servicios.

CU11: Carga de Pacientes.

CU12: Editar Datos Personales.

CU13: Carga de Seguros Médicos.

CU14: Búsqueda de Pacientes.

CU15: Carga de Atención al Paciente.

CU16: Carga de Ocurrencias de Gastos.

CU17: Carga de Unidades de Medida.

CU18: Pagos de Pacientes.

CU19: Carga de Proveedores.

CU20: Carga de Compras de Insumos.

CU21: Carga de Utilización de Insumos.

AC3: Secretario.

CU2: Iniciar Sesión.

CU3: Cambiar Contraseña.

CU12: Editar Datos Personales.

Observación: Entre los CU del Administrador se tiene la potestad de asignar permisos distintos a los que serán ofrecidos por defecto por el sistema. Sin embargo, se recomienda que para el correcto funcionamiento del mismo, no se realicen cambios, excepto los ajustes iniciales para que los perfiles y permisos coincidan con esta especificación.

4 Descripción de Casos de Usos

CU1: Crear Cuenta de Administrador (1)

Este CU se ejecutará una sola vez cuando en el sistema no exista ningún usuario. Permitirá la creación de la cuenta que tendrá el rol de administrador.

- Precondiciones: Ninguna.
- Flujo Principal: El sistema primero desplegará un mensaje en el cuál notifica al usuario de que no existen usuarios y que se creará la cuenta de administrador. El usuario ingresará la cuenta de usuario y la contraseña, con su respectiva

confirmación. Por último, el usuario podrá utilizar el botón “Guardar” para confirmar la creación de la cuenta, y se procederá al CU2. El usuario administrador poseerá inicialmente permisos sobre todos los ítems del menú principal del sistema.

- Subflujos: Ninguno.

- Flujos Alternos:

  E1: Ante cualquier carga de datos errónea, el sistema desplegará un mensaje de error y no permitirá la creación de la cuenta.

  E2: Si se cierra el formulario sin haber creado la cuenta, no se podrá acceder al sistema.

CU2: Iniciar Sesión. (RF18, 1)

Este es el caso de uso inicial de la aplicación, salvo en el caso de no existir usuarios donde se iniciará automáticamente el CU1.

- Precondiciones: Creación del usuario administrador (CU1).

- Flujo Principal: El usuario verá un formulario en el que simplemente cargará su nombre de login y su contraseña. Si los datos son correctos, el formulario se cerrará y se desplegará el menú inicial de la aplicación, donde podrán darse las siguientes alternativas.

- Subflujos:

  S1: El sistema permite la asignación de permisos a roles (actores en es-te documento) sobre ítems del menú principal. Luego del inicio de sesión exitoso, se desplegará el menú principal con los ítems que le corresponden al actor que inicia sesión.

  S2: Si el actor no tiene asociado ningún rol, se desplegará un mensaje al usuario, y se le sugerirá contactar con el administrador.

- Flujos alternos:

  E1: Ante cualquier dato dejado en blanco, el sistema notificará al usuario.

  E2: Si los datos son incorrectos, no se permitirá el acceso al sistema.

  E3: Si se cierra la ventana sin realizar el inicio de sesión, el sistema se cerrará.

CU3: Cambiar Contraseña (RF19, 1).

Este CU permitirá al usuario actual el cambio de su contraseña. El mismo reque-rirá nuevamente su contraseña actual por cuestiones de seguridad.

- Precondiciones: Inicio de Sesión (CU1).

- Flujo Principal: El usuario ingresará a la pantalla de cambio de contraseña a través del Menú Principal, Usuario Actual, Cambiar Contraseña. El nombre de login se completará automáticamente y se pedirá al usuario que ingrese la contraseña actual, y que ingrese dos veces la nueva contraseña. Para confirmar los cambios, el usuario deberá hacer clic en el botón "Guardar" y ver el mensaje de confirmación.

- Subflujos: Ninguno.

- Flujos Alternos:

  E1: Ante cualquier campo vacío, el sistema notificará al usuario.

  E2: Si la contraseña actual no es correcta, el sistema desplegará un mensaje de error y dejará que el usuario vuelva a intentarlo.

  E3: Si las dos cargas de la nueva contraseña no coinciden, se realizará también una notificación para que el usuario corrija los datos.

CU4: Asignar permisos sobre ítems del menú (RF4, 1).

En este CU, el usuario administrador podrá elegir cualquiera de los roles existente, y asignarles permisos sobre ítems del menú.

- Flujo Principal: El usuario ingresará a la pantalla de Asignación de Permisos a Perfiles a través del Menú Principal, Administración, Asignar Permisos a Perfiles. Luego, seleccionará el perfil que desea modificar de una lista que se desplegará en la pantalla. Al elegir el perfil, se cargará en una vista de árbol (TreeView) los ítems del menú de la aplicación, con un indicador que señalará si se posee actualmente permiso. Con un clic sobre la casilla de verificación se podrá cambiar el permiso sobre cada ítem, y haciendo clic en el botón "Guardar" se actualizan los datos.

- Subflujos: Ninguno.

- Flujos Alternos:

  E1: Como la lista permite la carga mediante el teclado del nombre de perfil, puede darse el caso en el que se introduzca un nombre equivocado. En este caso, el sistema vaciará la lista y los ítems del menú; y además inhabilitará el botón "Guardar" hasta que se seleccione un perfil.

  E2: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón "Guardar".

  E3: Si se cambia de perfil seleccionado sin hacer clic en el botón "Guardar", se perderán los cambios realizados desde que se abrió el mismo, o desde el último clic en el botón "Guardar".

CU5: Creación de Usuarios de Login (RF1, 1).

Este CU desplegará una pantalla en la que se permitirán las funciones de inserción, modificación y eliminación de usuarios del sistema.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Administración, Usuarios de Login. En la pantalla se desplegará en una tabla la lista de usuarios del sistema, y un conjunto de campos para ingresar los valores para un nuevo usuario.

- Subflujos:

  S1: El usuario podrá completar los datos en los campos de entrada, y una vez finalizada la carga, podrá hacer clic en el botón "Insertar" para crear el usuario.

  S2: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Se podrá cambiar el nombre y la contraseña, y con el botón "Actualizar" se guardarán los cambios realizados.

  S3: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Con el botón "Eliminar" se podrá borrar al usuario, siempre y cuando no esté relacionado con ningún otro dato dentro del sistema.

  S4: En cualquier momento, haciendo clic en el botón "Nuevo", la pantalla volverá al estado inicial.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón "Insertar" o "Actualizar".

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

  E3: En el caso de la inserción, las contraseñas no pueden quedar vacías y se realiza la validación de que coincidan las dos cargas de contraseñas. En caso de no coincidir, se alertará al usuario.

  E4: En el caso de la actualización, si las contraseñas quedan vacías, no se actualizan los valores. En caso de cargar la contraseña, se realiza la misma validación de E3.

CU6: Creación de Personas que utilizarán el sistema (RF2, 1).

Este CU permitirá la carga, modificación y eliminado de los datos personales de las personas que utilizarán el sistema.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Administración, Personas. Se desplegarán en una tabla todas las personas carga-das, y la lista de campos de entrada para cargar los datos, que son Documento de Identidad,

Apellidos, Nombres, Género, Fecha de Nacimiento, Dirección, Teléfonos, Correo Electrónico y Usuario de Login.

- Subflujos:

  S1: El usuario podrá completar los datos en los campos de entrada, y una vez finalizada la carga, podrá hacer clic en el botón "Insertar" para crear la persona.

  S2: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Se podrán cambiar todos los datos, y con el botón "Actualizar" se guardarán los cambios realizados.

  S3: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Con el botón "Eliminar" se podrá borrar a la persona.

  S4: En cualquier momento, haciendo clic en el botón "Nuevo", la pantalla volverá al estado inicial.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón "Insertar" o "Actualizar".

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

  E3: El sistema desplegará un error si se trata de asignar el mismo Usuario de Login a más de una persona, no dejando guardar los datos actuales. Se notificará al usuario para que realice el cambio correspondiente.

CU7: Asignar Perfiles a Usuarios (RF3, 1)

En este CU, el usuario podrá elegir un Usuario de Login y uno de los perfiles existentes para realizar una asignación.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Administración, Asignar Perfiles a Usuarios. Se desplegarán en una tabla todas las asignaciones cargadas, y los campos de entrada para cargar los datos, que son la lista de usuarios y la lista de perfiles.

- Subflujos:

  S1: El usuario podrá completar los datos en los campos de entrada, y una vez finalizada la carga, podrá hacer clic en el botón "Insertar" para crear la asignación.

  S2: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Se

podrá cambiar tanto el usuario como el perfil, y con el botón “Actualizar” se guardarán los cambios realizados.

S3: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Con el botón “Eliminar” se podrá borrar la asignación.

S4: En cualquier momento, haciendo clic en el botón “Nuevo”, la pantalla volverá al estado inicial.

- Flujos Alternos:

E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón “Insertar” o “Actualizar”.

E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

E3: Si se intenta ingresar una asignación repetida, el sistema desplegará una notificación al usuario.

CU8: Carga de Gastos (RF8, 1)

En este CU, el usuario podrá crear distintos tipos de gastos.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Mantenimiento, Gastos. Se desplegarán en una tabla todos los gastos cargados por el usuario (la lista de gastos es independiente para cada usuario), y la lista de campos de entrada para cargar los datos.

- Subflujos:

S1: El usuario podrá completar los datos en los campos de entrada, y una vez finalizada la carga, podrá hacer clic en el botón “Insertar” para crear el gasto.

S2: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Se podrá cambiar la descripción, y con el botón “Actualizar” se guardarán los cambios realizados.

S3: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Con el botón “Eliminar” se podrá borrar al gasto, siempre y cuando no esté relacionado con ningún otro dato dentro del sistema.

S4: En cualquier momento, haciendo clic en el botón “Nuevo”, la pantalla volverá al estado inicial.

- Flujos Alternos:

E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón “Insertar” o “Actualizar”.

E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

CU9: Carga de Insumos (RF10, 2)

En este CU, el usuario podrá crear distintos tipos de insumos.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Mantenimiento, Insumos. Se desplegarán en una tabla todos los insumos cargados por el usuario (la lista de insumos es independiente para cada usuario), y la lista de campos de entrada para cargar los datos.

- Subflujos:

  S1: El usuario podrá completar los datos en los campos de entrada, y una vez finalizada la carga, podrá hacer clic en el botón “Insertar” para crear el insumo.

  S2: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Se podrá cambiar la descripción, y con el botón “Actualizar” se guardarán los cambios realizados.

  S3: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Con el botón “Eliminar” se podrá borrar al insumo, siempre y cuando no esté relacionado con ningún otro dato dentro del sistema.

  S4: En cualquier momento, haciendo clic en el botón “Nuevo”, la pantalla volverá al estado inicial.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón “Insertar” o “Actualizar”.

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

CU10: Carga de Servicios (1)

En este CU, el usuario podrá crear distintos tipos de servicios que prestará.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Mantenimiento, Servicios. Se desplegarán en una tabla todos los servicios cargados por el usuario

(la lista de servicios es independiente para cada usuario), y la lista de campos de entrada para cargar los datos.

- Subflujos:

  S1: El usuario podrá completar los datos en los campos de entrada, y una vez finalizada la carga, podrá hacer clic en el botón “Insertar” para crear el servicio.

  S2: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Se podrá cambiar la descripción, y con el botón “Actualizar” se guardarán los cambios realizados.

  S3: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Con el botón “Eliminar” se podrá borrar al servicio, siempre y cuando no esté relacionado con ningún otro dato dentro del sistema.

  S4: En cualquier momento, haciendo clic en el botón “Nuevo”, la pantalla volverá al estado inicial.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón “Insertar” o “Actualizar”.

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

CU11: Carga de Pacientes (RF5, 1).

Este CU permitirá la carga, modificación y eliminado de los datos personales de los pacientes de un doctor.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Mantenimiento, Pacientes. Se desplegarán en una tabla todos los pacientes cargados por el usuario (la lista de pacientes es independiente por cada usuario), y la lista de campos de entrada para cargar los datos, que son Documento de Identidad, Apellidos, Nombres, Género, Fecha de Nacimiento, Dirección, Teléfonos y Correo Electrónico.

- Subflujos:

  S1: El usuario podrá completar los datos en los campos de entrada, y una vez finalizada la carga, podrá hacer clic en el botón “Insertar” para crear al paciente.

  S2: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Se

podrán cambiar todos los datos, y con el botón “Actualizar” se guardarán los cambios realizados.

S3: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Con el botón “Eliminar” se podrá borrar al paciente, siempre que no tenga datos relacionados en el sistema.

S4: En cualquier momento, haciendo clic en el botón “Nuevo”, la pantalla volverá al estado inicial.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón “Insertar” o “Actualizar”.

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

  E3: El sistema desplegará un error si se trata de asignar el mismo número de Documento de Identidad a un paciente del mismo usuario.

CU12: Editar Datos Personales (RF20, 1).

Este CU permite al usuario editar sus datos personales registrados en el sistema.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Usuario Actual, Editar Datos Personales. Se desplegarán todos los datos relacionados con la persona permitiéndole editar cualquiera de ellos. Para confirmar los cambios, el usuario debe hacer clic en el botón guardar.

- Subflujos: Ninguno.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón “Guardar”.

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

CU13: Carga de Seguros Médicos (2)

En este CU, el usuario podrá crear los distintos seguros médicos con los cuales trabajará.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Mantenimiento, Seguros Médicos. Se desplegarán en una tabla todos los seguros cargados por el usuario (la lista de seguros es independiente para cada usuario), y la lista de campos de entrada para cargar los datos.

- Subflujos:

  S1: El usuario podrá completar los datos en los campos de entrada, y una vez finalizada la carga, podrá hacer clic en el botón “Insertar” para crear el seguro.

  S2: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Se podrá cambiar la descripción, y con el botón “Actualizar” se guardarán los cambios realizados.

  S3: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Con el botón “Eliminar” se podrá borrar al seguro, siempre y cuando no esté relacionado con ningún otro dato dentro del sistema.

  S4: En cualquier momento, haciendo clic en el botón “Nuevo”, la pantalla volverá al estado inicial.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón “Insertar” o “Actualizar”.

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

  E3: No se dejará guardar cambios sobre un seguro médico si el valor de porcentaje de descuento no está entre 0 % y 100 %.

CU14: Búsqueda de Pacientes (2)

Este CU permitirá la búsqueda de pacientes y luego seleccionar uno, devolviendo el identificador del mismo. Es usado por otros casos de uso.

- Flujo Principal: El usuario ingresará mediante cualquier botón “Buscar” cuando se refiera a Pacientes. Se desplegarán tres posibles criterios de búsqueda, que son el documento de identidad, apellidos y nombres. Para desplegar la lista de quienes cumplan con esos criterios, se deberá hacer clic en el botón “Buscar”. Para seleccionar un paciente, se deberá hacer doble clic sobre la fila correspondiente. El resultado se visualizará en la pantalla que invocó a la búsqueda.

- Subflujos: Ninguno

- Flujos Alternos:

  E1: Si el formulario se cierra, no se seleccionará ningún paciente y el resultado devuelto será vacío (cadena vacía).

E2: Si no existen pacientes que cumplan con los criterios introducidos, no se desplegarán pacientes y por ende no se podrá seleccionar ninguno.

CU15: Carga de Atención al Paciente (RF6, 2)

Este es uno de los CU principales de la aplicación. Aquí, un doctor podrá cargar los detalles del trabajo realizado sobre un paciente por cada atención.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Fichas, Atención a Pacientes. Seleccionará un paciente haciendo clic sobre el botón de búsqueda de pacientes y se procede al CU14. Al seleccionar el paciente se lo selecciona también como responsable del pago, y recupera su estado de cuentas. Una vez seleccionados los datos de paciente, con un clic sobre el botón "Pasar a servicios" se comienza la carga de los detalles de la atención. El flujo de esta sección es similar al de las demás car-gas del sistema, cuyos datos de entrada son el servicio, el costo, el cuadrante (número del 1 al 8), el diente (número del 1 al 8) y la cara del diente. Para agregar, modificar o borrar un detalle, se utilizan los botones "Insertar", "Actualizar" y "Eliminar", como en otros CU de carga de datos. Por último, se debe cargar la cantidad que paga el paciente, y seleccionar opcionalmente la cobertura de seguro médico, y se hace clic en "Confirmar Atención" para guardar los cambios, y generar los datos de cuentas a pagar del paciente y/o seguro médico.

- Subflujos:

  S1: Si el usuario prefiere, se permite mediante un botón, la búsqueda del responsable de pago mediante el CU14. Al cambiar el responsable, se actualiza el estado de cuenta con el saldo del responsable.

  S2: Posee subflujos similares a los de inserción, modificación y eliminación del CU5.

- Flujos alternos:

  E1: Si el formulario se cierra sin haber confirmado la atención, todos los datos cargados se perderán.

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

  E3: En la carga de detalles, el sistema avisará al usuario si los valores relativos a la especificación de dientes son incorrectos, y no guardará los datos.

CU16: Carga de Ocurrencias de Gastos (RF9, 2)

En este CU, el doctor podrá cargar cada gasto en el que incurra ingresando el tipo de gasto, la fecha, un número de documento (recibo, comprobante, factura, etc) una descripción y el monto.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Movimientos, Gastos, Carga de Gastos. Se desplegarán en una tabla todos los gastos cargados por

el usuario (la lista de seguros es independiente para cada usuario), y la lista de campos de entrada para cargar los datos.

- Subflujos:

  S1: El usuario podrá completar los datos en los campos de entrada, y una vez finalizada la carga, podrá hacer clic en el botón “Insertar” para crear la nueva ocurrencia de un gasto.

  S2: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Se podrá cambiar la descripción, y con el botón “Actualizar” se guardarán los cambios realizados.

  S3: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Con el botón “Eliminar” se podrá borrar a la ocurrencia del gasto.

  S4: En cualquier momento, haciendo clic en el botón “Nuevo”, la pantalla volverá al estado inicial.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón “Insertar” o “Actualizar”.

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

CU17: Carga de Unidades de Medida (2)

En este CU, el usuario podrá crear distintos tipos de unidades de medida, que utilizará en la carga de insumos.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Mantenimiento, Unidades de Medida. Se desplegarán en una tabla todos los datos cargados por el usuario (la lista de unidades de medida es independiente para cada usuario), y la lista de campos de entrada para cargar los datos.

- Subflujos:

  S1: El usuario podrá completar los datos en los campos de entrada, y una vez finalizada la carga, podrá hacer clic en el botón “Insertar” para crear la unidad de medida.

  S2: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Se podrá cambiar la descripción, y con el botón “Actualizar” se guardarán los cambios realizados.

S3: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Con el botón "Eliminar" se podrá borrar a la unidad de medida, siempre y cuando no esté relacionado con ningún otro dato dentro del sistema.

S4: En cualquier momento, haciendo clic en el botón "Nuevo", la pantalla volverá al estado inicial.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón "Insertar" o "Actualizar".

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

CU18: Pagos de Pacientes (RF6, 2)

En este CU, el usuario podrá cargar los pagos realizados por un paciente, en el caso que se realicen en otro momento que no sea el de la carga de la atención al paciente (CU15).

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Fichas, Pagos. De manera similar al CU15, se buscará el paciente que efectuará el pago. Luego, se desplegará en una tabla el detalle del estado de cuenta del paciente seleccionado, se seleccionará opcionalmente a qué atención corresponde el pago y se introducirá el monto. Con un clic en el botón Guardar, se confirmarán los datos ingresados.

- Subflujos:

  S1: En cualquier momento se podrá cambiar el paciente.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón "Guardar".

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

  E3: Si se ha seleccionado asociar el pago a una atención, no se permitirá que el monto del pago sea mayor al saldo de la atención.

CU19: Carga de Proveedores (2)

En este CU, el usuario podrá cargar los distintos proveedores de quienes compran sus insumos.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Mantenimiento, Proveedores. Se desplegarán en una tabla todos los datos carga-dos por el usuario

(la lista de proveedores es independiente para cada usuario), y la lista de campos de entrada para cargar los datos.

- Subflujos:

  S1: El usuario podrá completar los datos en los campos de entrada, y una vez finalizada la carga, podrá hacer clic en el botón “Insertar” para crear el proveedor.

  S2: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Se podrá cambiar la descripción, y con el botón “Actualizar” se guardarán los cambios realizados.

  S3: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Con el botón “Eliminar” se podrá borrar al proveedor, siempre y cuando no esté relacionado con ningún otro dato dentro del sistema.

  S4: En cualquier momento, haciendo clic en el botón “Nuevo”, la pantalla volverá al estado inicial.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón “Insertar” o “Actualizar”.

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

CU20: Carga de Compras de Insumos (RF11, 2)

Este CU permite al usuario la carga de compras de insumos, ingresando datos correspondientes a la compra como el proveedor, las cantidades y el insumo.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Movimientos, Insumos, Compras. Se desplegarán en una tabla todas las compras cargadas previamente por el usuario (el stock de insumos es independiente para cada usuario), y la lista de campos de entrada para cargar los datos.

- Subflujos:

  S1: El usuario podrá completar los datos en los campos de entrada, y una vez finalizada la carga, podrá hacer clic en el botón “Guardar” para confirmar la compra.

  S2: El usuario podrá hacer doble clic sobre una de las filas de la tabla. Los campos de entrada se llenarán automáticamente con los datos de la fila seleccionada. Con el botón “Eliminar” se podrá borrar la compra.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón “Guardar”.

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

CU21: Carga de Utilización de Insumos (RF11, 2)

Este CU permite al usuario la carga de utilización de insumos seleccionando uno de los insumos existentes, y cargando la cantidad utilizada.

- Flujo Principal: El usuario ingresará mediante el Menú Principal, Movimientos, Insumos, Utilización. Se desplegarán en una tabla todos los insumos disponibles del usuario. El usuario podrá seleccionar un insumo mediante doble clic sobre su fila en la tabla. Allí podrá cargar la cantidad de insumos utilizados.

- Subflujos: Ninguno.

- Flujos Alternos:

  E1: Si el formulario se cierra, no se guardarán ninguno de los cambios hechos desde que se abrió el mismo, o desde el último clic en el botón “Guardar”.

  E2: Ante cualquier campo requerido que se haya dejado vacío, el sistema desplegará un mensaje alertando al usuario.

  E3: El sistema no dejará que se cargue la utilización de más insumos de los que existen. Se desplegará un mensaje notificando al usuario.

Anexo C. Solicitudes de Cambio

Por último, se adjuntan las solicitudes y seguimientos de cambios generados durante los dos primeros ciclos del desarrollo del Caso de Estudio, en orden cronológico.

DOCUMENTO DE SOLICITUD Y SEGUIMIENTO DE CAMBIOS

Autor: Dra. Estela Poggi.

Producto/Módulo: Pantalla Principal.

Versión (Nº de Entrega): 1.

Tipo de Cambio: Marcar con una "x" dentro del paréntesis correspondiente.

( ) Bloqueador.

( ) Crítico.

( ) Mayor.

( ) Normal.

( ) Menor.

( ) Trivial.

(x) Mejora.

Resumen:

Agregar un bloc de notas en la pantalla principal de la aplicación que pueda servir como recordatorio.

Responsable: Enrique Bañuelos.

Prioridad: Baja.

Estados:

Estado	Fecha
Nuevo	11/09/2007

Observaciones:

DOCUMENTO DE SOLICITUD Y SEGUIMIENTO DE CAMBIOS

Autor: Enrique Bañuelos.

Producto/Módulo: Mantenimiento.

Versión (Nº de Entrega): 1.

Tipo de Cambio: Marcar con una "x" dentro del paréntesis correspondiente.

( ) Bloqueador.

( ) Crítico.

( ) Mayor.

( ) Normal.

( ) Menor.

( ) Trivial.

(x) Mejora.

Resumen:

El sistema deberá permitir la carga de distintos tipos de seguros médicos, con su respectivo porcentaje de descuento.

Responsable: Enrique Bañuelos.

Prioridad: Alta.

Estados:

Estado	Fecha
Nuevo	12/09/2007
Resuelto	12/09/2007

Observaciones:

DOCUMENTO DE SOLICITUD Y SEGUIMIENTO DE CAMBIOS

Autor: Dra. Estela Poggi.

Producto/Módulo: Mantenimiento.

Versión (Nº de Entrega): 1.

Tipo de Cambio: Marcar con una "x" dentro del paréntesis correspondiente.

( ) Bloqueador.

( ) Crítico.

( ) Mayor.

( ) Normal.

( ) Menor.

( ) Trivial.

(x) Mejora.

Resumen:

El sistema debe permitir especificar, en la carga de los tipos de insumo, la unidad de medida que se utilizará para el control de existencia del mismo.

Responsable: Enrique Bañuelos.

Prioridad: Alta.

Estados:

Estado	Fecha
Nuevo	18/09/2007
Resuelto	19/09/2007

Observaciones:

Como resultado de este pedido, se agrega un nuevo caso de uso donde el doctor debe cargar las distintas unidades de medida con las que debe trabajar.

DOCUMENTO DE SOLICITUD Y SEGUIMIENTO DE CAMBIOS

Autor: Dra. Estela Poggi.

Producto/Módulo: Fichas.

Versión (Nº de Entrega): 2.

Tipo de Cambio: Marcar con una "x" dentro del paréntesis correspondiente.

( ) Bloqueador.

( ) Crítico.

( ) Mayor.

( ) Normal.

(x) Menor.

( ) Trivial.

( ) Mejora.

Resumen:

Cuando se quiere cargar un pago por adelantado del paciente, el sistema emite un mensaje y no deja guardar.

Responsable: Enrique Bañuelos.

Prioridad: Alta.

Estados:

Estado	Fecha
Nuevo	02/10/2007

Observaciones:

DOCUMENTO DE SOLICITUD Y SEGUIMIENTO DE CAMBIOS

Autor: Dra. Estela Poggi.

Producto/Módulo: Movimientos.

Versión (Nº de Entrega): 2.

Tipo de Cambio: Marcar con una "x" dentro del paréntesis correspondiente.

( ) Bloqueador.

( ) Crítico.

( ) Mayor.

( ) Normal.

( ) Menor.

( ) Trivial.

(x) Mejora.

Resumen:

Al cargar la compra de insumos, que se pueda cargar el monto total y el monto pagado al proveedor. Así, se podrá mantener el estado de cuenta de los proveedores.

Responsable: Ellen Méndez.

Prioridad: Media.

Estados:

Estado	Fecha
Nuevo	02/10/2007

Observaciones: