Facultad 8

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Documentación imprescindible para los flujos de trabajo de diseño e implementación de software de gestión

AUTORAS: Karenia Donatien Goliath

Yudermis Rodriguez Martínez

TUTORA: Ing. Dainys Gainza Reyes

Ciudad de La Habana, Junio 2007

"Año 49 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____días del mes de Junio del año 2007.

Karenia Donatien Goliath	Yudermis Rodriguez Martínez	Dainys Gainza Reyes
_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor

PENSAMIENTO

"No existe problema alguno que el cerebro humano no pueda resolver. Todo lo que necesitamos es aprender a pensar"

Thomas Alva Edison

AGRADECIMIENTOS

A mi mamá por su amor, preocupación y apoyo.
A mi papá, que aunque la vida no le permitió estar aquí hoy, sé que estaría orgulloso de mí. Te quiero.
A mis hermanos por ser un ejemplo a seguir.
A mis compañeros y amigos, en especial a Arodys, Osdalme, Arcel, Nela, Rosanna, Liem, Virginia, Maelis, Zenoyda, Yenlys y Kirenia por los ratos compartidos y por haberme soportado a lo largo de mi carrera.
A mis vecinos que son como una familia para mí, especialmente a la familia Linares.
A mi familia por estar siempre presente.
A mi novio Arturo por su paciencia y cariño.
A Roberto por ser como un padre en los momentos buenos y malos.
A mi compañera de tesis por su dedicación al trabajo.
A mi tutora, por haber trasmitido sus conocimientos y dedicarnos tantas horas de su tiempo.
A todos, muchas gracias.

Karenia

A mi mamá por todo su sacrificio y dedicación, por escucharme siempre y saber comprenderme, gracias por contar conmigo y darme tanta confianza.
A mi papá por cuidarme siempre, por inculcarme tan buenos valores y siempre estar presente cuando te necesito, gracias por motivarme a ser cada vez mejor.
A mi hermanito querido por ser tan incondicional y compartirlo todo conmigo, por haber sido siempre mi ejemplo de estudiante, gracias por todo tu apoyo.
A toda mi familia, que se preocupan tanto por mí.
A mis vecinos de siempre que son como mi familia.
A Liusvani, por compartir cuatro años de la universidad, por su ayuda en los trabajos.
A Teudy por acompañarme en los últimos momentos de la carrera, que a pesar de ser poco tiempo para mí fue muy importante, gracias por soportarme.
A mis compañeros y amigos del preuniversitario y la universidad, por acompañarme en los buenos y malos momentos, por su apoyo y ayuda siempre que la necesité.
A mi compañera de tesis, por su responsabilidad y abnegación al trabajo.
A mi tutora por guiarnos y apoyarnos para que este trabajo fuera posible.
A todos los que de una manera u otra han contribuido a mi formación profesional.

Yudermis

DEDICATORIA

A mis padres y mis hermanos.

Karenia

A mi mamá, mi papá y mi hermano por hacerme sentir tan orgullosa de la familia que formamos, por estar siempre unidos, querernos y ayudarnos tanto. Muchas gracias.

Yudermis

RESUMEN

Un objetivo de décadas ha sido encontrar procesos o metodologías que mejoren la calidad del software, basándose en la necesidad de controlar, guiar y documentar los proyectos. En la actualidad la documentación relacionada con los sistemas de software, se ve como una pérdida de tiempo y no como un mapa que nos ayudará a alcanzar la meta. Una de las causas que provoca que no se le conceda a la documentación la importancia que se merece es la cantidad de artefactos que proponen las metodologías más usadas. Esto trae como consecuencia que la documentación se haga al finalizar el producto o simplemente no se haga. El objetivo principal del siguiente trabajo de investigación es: identificar la documentación imprescindible para los flujos de trabajo de diseño e implementación en el proceso de desarrollo de software de gestión en la Universidad de Ciencias Informáticas (UCI). Para lograr lo planteado anteriormente, se hizo un estudio de las metodologías de desarrollo más conocidas y de los artefactos relativos a las etapas de diseño e implementación, de estos se definieron los que conforman la propuesta y luego se efectúa la validación de la misma. Esta propuesta es aplicable a proyectos de software de gestión independientemente de la metodología que se emplee para conducir el proceso de desarrollo de software, apoyando la necesidad de documentar en paralelo a la construcción del producto. La práctica de documentar correctamente los sistemas de software permite que se trasmita el conocimiento a todo el equipo de desarrollo.

ÍNDICE

INTRODUCCIÓN

> *¿Qué pasaría, si el ingeniero civil o el arquitecto construyen una casa o un edificio sin hacer sus planos, proyectos o maquetas? ¿Crees que la obra pueda concluirse cubriendo las necesidades, con la calidad necesaria y a tiempo? Y todavía más allá, ¿Permitirías que tu propio cirujano te interviniera sin hacer los estudios respectivos para obtener las evidencias del problema de salud que te aqueja? O ¿permitirías a tu abogado que te defendiera sin conocer las pruebas y sin un plan para tu defensa? Entonces, ¿por qué los ingenieros Informáticos a veces cedemos al "chantaje de la falta de tiempo" y construimos software sin el análisis, diseño e implementación expresado en un proyecto, más allá de las ideas existentes "en nuestra cabeza"? ¿Por qué lo intentamos hacer sobre la marcha, pero nunca lo concluimos pues ya no hay tiempo? ¿Dónde quedó la ética profesional?... (ZABALA 2000)*

La industria del software es relativamente joven, por lo cual muchas veces se etiqueta como una industria no completamente madura. Sin embargo, es una de las que evoluciona más rápido a nivel mundial.(CANSECO 2004)

Debido al desarrollo alcanzado por esta industria, los proyectos se han tornado más complejos, son mucho más grandes y el tiempo que se exige para desarrollarlos es por lo general más corto, por lo que se requiere el uso de metodologías para poner orden y control al proceso de desarrollo del software. Sin embargo, muchas veces los realizadores de software no hacen uso de las metodologías existentes y menosprecian su valor en la elaboración del producto, por lo que no se tienen en cuenta ciertas prácticas como la documentación, la planeación y el análisis de riesgo que se deben hacer en el momento indicado, pues de esta forma se garantiza un eficiente desarrollo del proceso. (MEDINA 2005)

Es importante que el producto que se desarrolla sea de prestigio y confiable, y esto se logra siguiendo una metodología y documentando todo lo que pueda explicar cómo fue desarrollando el producto final. Esto garantizará una guía para futuros trabajos o para arreglar cualquier desperfecto en el producto entregado, además de tener la forma de socializar el conocimiento adquirido por el equipo de desarrollo.

Actualmente en el mundo, las empresas dedicadas al desarrollo de software, le confieren mayor importancia a obtener un producto funcional que satisfaga al cliente que a documentar el mismo. Tanto es así, que muchos desarrolladores abogan por la utilización de metodologías ágiles, que proponen una guía para desarrollar software de forma rápida, pero que dejan a un lado lo que se refiere a la documentación que debe ir acompañando el producto final.

Cuba no está al margen del desarrollo que ha alcanzado la informática en el mundo, y es por ello, que una de las principales tareas del gobierno cubano es: encaminarse resueltamente a la modernización informática mediante un programa integral que involucre a las organizaciones que deben proveer los recursos materiales, financieros e intelectuales y a las entidades económicas, políticas y sociales que deben traducirlos en más y mejores productos y servicios. La industria de los servicios informáticos deberá asegurar la modernidad de su base técnica y organizativa, y la elevación constante del nivel científico-técnico de sus especialistas con vistas a garantizar esos propósitos.(Resolución Económica V Congreso PCC 1997)

En el año 2002 se crea la Universidad de las Ciencias Informáticas (UCI), la cual tiene como misión:

1. Formar profesionales, comprometidos con su Patria, altamente calificados en la rama de la informática.
2. Producir software y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación.

La UCI fue creada sobre una fuerte base tecnológica y un amplio perfil productivo, ésta pretende convertir la industria cubana del software en un renglón fundamental de la economía, ser líder del desarrollo de las empresas de software en Cuba e insertarse en el mercado internacional.

En la universidad juega un papel fundamental la relación estudio-producción, o sea, el estudiante y el trabajador juegan papeles importantes en el desarrollo de productos de software, pues son precisamente ellos los encargados de desarrollar los sistemas y de lograr que se convierta en una potencia significativa en cuanto al desarrollo de software se refiere, no solo para la rama de la informática, sino para todas las ramas de la sociedad.

Actualmente en la UCI, como en el resto de las empresas productoras de software del mundo, la documentación de todo tipo (por ejemplo: los comentarios en el código fuente, el diseño de la arquitectura

y la documentación en línea) muchas veces se atrasa dejándola para escribirla al finalizar el proceso en cuestión, y no paralelamente como realmente debe hacerse, o en el peor de los casos, simplemente no se realiza. Esto trae consigo que se dejen de documentar flujos tan importantes como los de diseño e implementación, perdiendo de esta forma el conocimiento adquirido por las personas que trabajan directamente con estos flujos en el proceso de desarrollo, lo que provoca que no exista la manera de retroalimentarse cuando ocurra algún problema o se requiera realizar algún cambio y no esté presente la persona que trabajó en el diseño o la implementación.

Por otra parte la metodología *Rational Unified Process* (RUP), que es la más usada para organizar y guiar el proceso de desarrollo de software en la universidad, propone un gran número de artefactos y entregables, por consiguiente, si se cumple con todo lo recogido en la misma, se necesitará dedicar mucho tiempo a documentar el producto y se perdería en rapidez a la hora de entregar el software. Es posible adaptar esta metodología al software que se esté desarrollando, es decir, definir qué artefactos y entregables deberán ser entregados, pero de esta forma también se pierde tiempo, por lo que es importante especificar cuales serían los principales artefactos y entregables para los flujos de trabajo de diseño e implementación, que se adapten de manera general a todo software que se desee desarrollar, según las características y necesidades del entorno.

En el mundo, hoy, las principales investigaciones relacionadas con el proceso de desarrollo de software, están centradas en encontrar una metodología que se adapte a los requerimientos del cliente en el menor tiempo posible.

En Cuba no se han realizado investigaciones reconocidas sobre el tema de la documentación del software, aunque hay que destacar que el Grupo Nacional de Calidad realizó una investigación donde se definieron los principales entregables para la metodología RUP.

En la UCI la propuesta que existe para documentar los productos de software está basada en la metodología RUP, por lo que exige mucha documentación. Lo cual evidencia que no se han realizado investigaciones que recojan los artefactos necesarios que deben documentarse independientemente de la metodología a usar.

Por todo lo anterior, esta investigación centra su estudio en la documentación del software, principalmente la que debe ser entregada durante los flujos de trabajo de diseño e implementación, etapas muy importantes en el proceso de desarrollo de software, pues es en la fase de diseño donde se define la

arquitectura que se utilizará en la aplicación, los componentes, las interfaces, es decir, las características (planos) del sistema en general, en la etapa de implementación es donde se convierte el esquema lógico y demás especificaciones de diseño en un lenguaje de programación.

De esto se deriva el **problema de la investigación**: La falta de una guía que describa los artefactos imprescindibles para la documentación de software de gestión durante los flujos de trabajo de diseño e implementación en la UCI, provoca ineficiencias en el proceso de desarrollo de software.

El **objeto de estudio** de la investigación es: El proceso de desarrollo de software de gestión y el **campo de acción** en específico a analizar es: La documentación dentro de las metodologías de desarrollo de software.

Por lo que el **objetivo general** que se persigue es: Identificar la documentación imprescindible para los flujos de trabajo de diseño e implementación en el proceso de desarrollo de software de gestión en la UCI.

Los **objetivos específicos** serían:

1. Confeccionar el Marco teórico en relación con los requisitos de los flujos de trabajo de diseño e implementación de software y dejar definida la posición del investigador.
2. Estudiar los artefactos y entregables definidos en los proyectos de gestión que se ejecutan actualmente en la UCI para los flujos de trabajo de diseño e implementación.
3. Seleccionar en las metodologías, los componentes más eficientes para el desarrollo de software.
4. Detallar una guía que sirva para documentar los flujos de trabajo de diseño e implementación de software en proyectos productivos de gestión en la UCI.

Para el desarrollo de la investigación se parte de la **idea a defender** que: Si se especifica la documentación imprescindible que se debe entregar en los flujos de trabajo de diseño e implementación para proyectos que desarrollan software de gestión, se podrá tener una guía para futuros proyectos de este tipo, de forma que se trasmita el conocimiento a todo el equipo de desarrollo.

Para el cumplimiento de los objetivos trazados se han propuesto un conjunto de tareas que ayudarán a que la investigación se haga de forma eficiente, a continuación se relacionan:

1. Revisión bibliográfica sobre la temática y estudiarla para definir el estado del arte.
2. Análisis de las metodologías de desarrollo existentes para software de gestión.
3. Estudio de los proyectos de gestión que se ejecutan en la UCI para conocer cuales son los artefactos y entregables que definieron para los flujos de trabajo de diseño e implementación.
4. Aplicación de instrumentos para el diagnóstico y la recolección de la información.
5. Elaboración de una guía para la documentación de los flujos de trabajo de diseño e implementación para proyectos de gestión en la UCI.
6. Validación de la propuesta usando un instrumento.

Esta propuesta traerá consigo que el conocimiento que se adquiere en el desarrollo de productos informáticos se lleve a todos los integrantes del equipo de desarrollo, de manera que cada uno sepa en cada momento lo que está pasando en el proyecto, además de constituir una guía para futuros proyectos gestión implementados en la universidad.

La estrategia de investigación es descriptiva y se desarrollará en la UCI. Se realizó un estudio de la bibliografía referente al proceso de desarrollo de software. Se realizaron encuestas a los estudiantes y profesores de la universidad que pertenecen a los proyectos productivos que desarrollan software de gestión, los cuales fueron seleccionados por un método estratificado para tener personas de todas las facultades. Posteriormente se realizó el análisis de esta información recolectada para demostrar la importancia de la investigación. Se trabajó en la confección de un modelo que sirva de guía para los desarrolladores de software de la universidad, de manera que se documente sólo lo que es necesario y realmente importante para el proyecto y el equipo de desarrollo. Como se ha explicado, es una investigación que no se ha realizado anteriormente, de ahí la importancia y la novedad de la misma.

El presente trabajo estará dividido en tres capítulos. En el primer capítulo, Fundamento Teórico, se hace un análisis crítico de la bibliografía consultada y utilizada, realizando un estudio de la documentación que se establece en cada una de las metodologías de desarrollo de software que más se utilizan a nivel mundial.

En el segundo capítulo, Descripción de la Propuesta, se hace referencia a los procesos de diseño e implementación dentro del proceso de desarrollo de un software de gestión, los métodos, procedimientos y técnicas para llevar adelante la investigación. Además se describe la propuesta con la importancia de cada elemento que la compone.

En el tercer y último capítulo, Análisis de los Resultados, se presentan los resultados de la aplicación de los instrumentos.

CAPÍTULO
FUNDAMENTO TEÓRICO

1.1 Introducción

En este capítulo se exponen un grupo de conceptos relacionados con el proceso de desarrollo de software y las metodologías de desarrollo, se hace un estudio de los artefactos y entregables de cada una de estas metodologías y se plasma la opinión crítica del autor del trabajo con relación a la bibliografía consultada.

1.2 Proceso de desarrollo del software

El proceso de desarrollo de software (PDS) "es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo", es decir, "define quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo". (JACOBSON 1998)

En la ingeniería del software el objetivo es construir un producto software o mejorar uno existente. Un proceso efectivo proporciona normas para el desarrollo eficiente de software de calidad. Captura y presenta las mejores prácticas que el estado actual de la tecnología permite. En consecuencia, reduce el riesgo y hace el proyecto más predecible. Es necesario un proceso que sirva como guía para todos los participantes: clientes, usuarios, desarrolladores y directores ejecutivos. (IVAR JACOBSON *et al.* 2004)

No existe un proceso de software universal. Las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigen que el proceso sea configurable. (LETELIER 2004)

La formalización del proceso de desarrollo se define como un marco de referencia denominado ciclo de desarrollo del software o ciclo de vida del software. Se puede describir como, "el período de tiempo que

comienza con la decisión de desarrollar un producto software y finaliza cuando se ha entregado éste". (ASENSIO 2004)

El ciclo de vida de un proyecto especifica el enfoque general del desarrollo, indicando las fases, los flujos de trabajo, las actividades, el orden en que se van a realizar las tareas, los productos que se van a generar, los que se van a entregar al cliente y en qué orden hacerlo.

Para poder combinar todos estos elementos dentro del proceso de desarrollo de software se utilizan las metodologías de desarrollo de software.

La palabra metodología proviene de método, que quiere decir poner orden, organizar, ordenar, sistematizar. Se puede decir que una metodología es un conjunto de métodos utilizados en una investigación o proceso.

1.3 Metodología de desarrollo de software

Las metodologías de desarrollo de software surgieron a raíz de la necesidad de controlar, guiar y documentar proyectos cada vez más complejos, impulsadas principalmente por instituciones económicamente importantes y con requisitos de seguridad y fiabilidad en sus sistemas sumamente estrictos. (SERVETTO 1999)

La experiencia acumulada a lo largo de años de ejecutar proyectos, indica que los proyectos exitosos son aquellos que son administrados siguiendo una serie de procesos que permiten organizar y luego controlar al proyecto. Según esta visión, los proyectos que no sigan estos lineamientos corren un alto riesgo de fracasar. Para esto es necesario el uso de metodologías de desarrollo.

En la actualidad son muchas las metodologías de desarrollo de software que existen, el uso de una u otra dependen de las características de cada proyecto (equipo de desarrollo, recursos, alcance, etc.). Si se tiene un proyecto y un equipo de desarrollo pequeños y con cortos períodos de entrega, es recomendable usar la metodología XP. Si por el contrario se tiene un proyecto grande, y se quiere que se certifique el proceso de desarrollo es recomendable usar una metodología como RUP.(MOLPECERES 2002)

Las metodologías de desarrollo de software pueden ser clasificadas en dos tipos: las tradicionales o robustas y las ágiles. Las metodologías robustas se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir y las

herramientas que se usarán. Estas metodologías incluyen muchos artefactos para el desarrollo del software, aplicarlas cabalmente implicaría un atraso en el tiempo de entrega de los productos, pues son muchos los artefactos que se deben entregar. Por otra parte, las metodologías ágiles son las que se centran en otras dimensiones, como por ejemplo el factor humano o el producto software, dando mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo.

1.3.1 Las metodologías Ágiles

Según Jorge Fernández, las metodologías ágiles, tienen como común denominador un modelo de desarrollo incremental para producir tempranamente pequeñas entregas en ciclos rápidos, y predisposición para el cambio y la adaptación continua; según sea la conformidad o no de lo producido, y las modificaciones propuestas por los usuarios. Estas metodologías por lo general se centran en desarrollar productos funcionales más que en conseguir una buena documentación.

Lo que diferencia un proceso ágil de lo tradicional son sus características o principios, a continuación se relaciona los doce principios generales para las metodologías ágiles.

I. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.
II. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
III. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
IV. Los trabajadores del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
V. Construir el proyecto en torno a individuos motivados. Darles el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
VI. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
VII. El software que funciona es la medida principal de progreso.

VIII. Promover un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.

IX. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.

X. La simplicidad es esencial.

XI. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.

XII. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento. (FERNÁNDEZ 2006)

Aunque esto es general para todas las metodologías ágiles, cada una de ella tiene características propias y hace hincapié en algunos aspectos más específicos.

1.3.1.1 Adaptive Software Development (ASD)

Carlos Reynoso platea que esta metodología fue creada por James Highsmith hacia el año 2000 con la intención de ofrecer una alternativa a la idea, propia de CMM (Capability Maturity Model - Modelo de Madurez de Capacidad) Nivel 5, de que la optimización es la única solución para problemas de complejidad creciente. ASD es el modelo de implementación de patrones ágiles para desarrollo de software, que tiene como características básicas:

- Trabajo orientado y guiado por la misión del proyecto.
- Basado en la funcionalidad.
- Desarrollo iterativo.
- Desarrollo acotado temporalmente.
- Guiado por los riesgos.
- Trabajo tolerante al cambio.

ASD reconoce que las necesidades del cliente son siempre cambiantes. La iniciación de un proyecto involucra definir una misión para él, determinar las características y las fechas, así como descomponer el proyecto en una serie de pasos individuales, cada uno de los cuales puede abarcar entre cuatro y ocho semanas.

La idea subyacente a ASD (y de ahí su particularidad) radica en que no proporciona un método para el desarrollo de software sino que más bien suministra la forma de implementar una cultura adaptativa en la empresa, con capacidad para reconocer que la incertidumbre y el cambio son el estado natural. (REYNOSO 2004)

Al no ser ASD una metodología centrada en el proceso de desarrollo de software, no propone técnicas, ni prescribe tareas a la hora de llevar a cabo el diseño y la construcción del proyecto, simplemente se menciona que todas las prácticas que sirvan para reforzar la colaboración entre los usuarios y el equipo de desarrollo serán preferidas.

ASD está formada por tres fases básicas: especulación, colaboración y aprendizaje, se puede observar el detalle interno de cada fase, mostrándose con una flecha que trasciende las tres fases en sentido inverso, el bucle de aprendizaje *(Ver Anexo 1)*. En cada una de las fases anteriormente mencionadas se hace un énfasis grande a la calidad del producto, pues lo más importante es que el producto final cumpla con todas las reglamentaciones del usuario, para esto se realizan reuniones donde se determinan los aspectos positivos y negativos del proyecto.

En las metodologías tradicionales las diferencias respecto a lo planificado se ven como errores que se deben enmendar para cumplir lo pautado. ASD y las metodologías ágiles plantean la necesidad de que el bucle sea para aprender, dando la posibilidad de entender más respecto al dominio y construir la aplicación que mejor satisfaga las necesidades del cliente. (LUZ 2004)

Como ha sido analizado hasta aquí, ASD es un marco filosófico que permite encarar la construcción de software utilizando las prácticas que resulten convenientes, siempre centrando su importancia en la calidad del producto. Lo referido a los documentos que se deben emitir no es considerado importante en esta metodología.

1.3.1.2 Extreme Programing (XP)

La programación extrema es un modelo de programación, se basa en una serie de metodologías de desarrollo de software en la que se da prioridad a los trabajos que dan un resultado directo y que reducen la burocracia que hay alrededor de la programación. (GÓMEZ 2006)

Los objetivos de XP son muy simples. Esta metodología trata de lograr la satisfacción del cliente, dándole el software que él necesita y cuando lo necesita. Otro objetivo de importancia es potenciar al máximo el

trabajo en grupo, donde los jefes de proyecto, los clientes y desarrolladores son parte del equipo y están involucrados en el desarrollo del software.

Una de las características principales de este método de programación, es que sus ingredientes son conocidos desde el principio de la informática. Los autores de XP han seleccionado aquellos que han considerado mejores y han profundizado en sus relaciones y en como se refuerzan los unos con los otros. El resultado de esta selección ha sido esta metodología única y compacta. Por esto, aunque no está basada en principios nuevos, sí que el resultado es una nueva manera de ver el desarrollo de software.

XP se fundamenta en cuatro valores o principios que lo inspiran:

Comunicación: La comunicación entre los miembros del equipo y los clientes, se debe maximizar.

Simplicidad: Usar la solución más sencilla que pueda funcionar.

Retroalimentación: Siempre debe ser posible medir el producto que se está desarrollando y conocer qué le falta para satisfacer los requerimientos.

Coraje: Es necesario armarse de valor para incorporar cambios durante un proyecto. El coraje por sí solo es peligroso, pero sustentado con comunicación, simplicidad y retroalimentación, es una herramienta poderosa. (OCA 2006)

El ciclo de vida ideal de XP está compuesto por seis fases *(Ver Anexo 3):*

Fase I Exploración: En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo.

Fase II Planificación de la entrega: En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Esta fase dura unos pocos días.

Fase III Iteraciones: Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto

se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide las historias que se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción.

Fase IV Producción: La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase. Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento).

Fase V Mantenimiento: Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente.

Fase VI Muerte del Proyecto: Esta fase comienza cuando el cliente no tiene más historias para ser incluidas en el sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo. (ACUÑA 2006)

La programación extrema se basa en doce "prácticas básicas" que deben seguirse al pie de la letra, estas son:

1. **Equipo completo:** Forman parte del equipo todas las personas que tienen algo que ver con el proyecto, incluido el cliente y el responsable del proyecto.
2. **Planificación:** Se planifica el orden en que se van a hacer las historias de usuario y se elaboran las mini-versiones. La planificación se revisa continuamente.
3. **Pruebas:** Existen tanto pruebas internas (o pruebas de unidad), como pruebas de aceptación, para garantizar que el código hace lo que debe hacer. El cliente es el responsable de definir las pruebas de aceptación, no necesariamente de implementarlas.
4. **Versiones pequeñas:** Las mini-versiones deben ser lo suficientemente pequeñas como para poder hacer una cada pocas semanas. Deben ser versiones que ofrezcan algo útil al usuario final y no trozos de código que no pueda ver funcionando.
5. **Diseño simple:** Hacer siempre lo mínimo imprescindible de la forma más sencilla posible. Mantener siempre sencillo el código.
6. **Pareja de programadores:** Los programadores trabajan por parejas (dos delante del mismo ordenador) y se intercambian las parejas con frecuencia (un cambio diario).
7. **Mejora del diseño:** Mientras se codifica, debe mejorarse el código ya hecho con el que nos crucemos y que sea susceptible de ser mejorado. Extraer funcionalidades comunes, eliminar líneas de código innecesarias, etc.
8. **Integración continua (Refactorización):** Deben tenerse siempre un ejecutable del proyecto que funcione y en cuanto se tenga una nueva pequeña funcionalidad, debe recompilarse y probarse.
9. **El código es de todos:** Cualquiera puede y debe tocar y conocer cualquier parte del código. Para eso se hacen las pruebas automáticas.
10. **Normas de codificación:** Debe haber un estilo común de codificación (no importa cual), de forma que parezca que ha sido realizado por una única persona.

11. **Metáforas:** Se deben buscar frases o nombres que definan cómo funcionan las distintas partes del programa, de forma que sólo con los nombres se pueda tener una idea de qué es lo que hace cada parte del programa.
12. **Ritmo sostenible:** Se debe trabajar a un ritmo que se pueda mantener indefinidamente, para conseguir el objetivo cercano de terminar una historia de usuario o mini-versión. (RODRÍGUEZ 2006)

Los artefactos esenciales en XP son:

- Historias del Usuario.
- Tareas de Ingeniería.
- Pruebas de Aceptación.
- Pruebas Unitarias y de Integración.
- Plan de la Entrega.
- Código.

XP ignora la arquitectura, o sea, va hacia la codificación rápida confiando en que la refactorización resolverá todos los problemas de diseño, por tanto no dedica mucho tiempo a esta tarea. XP es un método ágil radical, menosprecia los diagramas en gran medida enfocándose en la obtención de un producto funcional. De forma general propone que la comunicación y la satisfacción del cliente es lo principal, por lo que la documentación en sentido general no es su plato fuerte, sólo es considerado importante definir los requerimientos y las pruebas de calidad.

1.3.1.3 Dynamic Solution Development Method (DSDM)

El DSDM empezó en Gran Bretaña en 1994 como un consorcio de compañías del Reino Unido que querían construir sobre Desarrollo Rápido de Aplicaciones (RAD) y desarrollo iterativo. Tiene una organización de tiempo completo que lo apoya con manuales, cursos de entrenamiento, programas de certificación y demás. (FOWLER 2006)

Según Andrés Delfino, el objetivo del DSDM es desarrollar soluciones que logren alcanzar los objetivos de negocio en el tiempo establecido. Esta metodología fue creada con el objetivo de ser utilizada en RAD, además de complementar otras metodologías como XP, RUP o combinaciones de éstas.

La idea dominante detrás de DSDM es explícitamente inversa a la que se encuentra en otras metodologías, y al principio resulta contraria a la intuición; en lugar de ajustar tiempo y recursos para lograr cada funcionalidad, el tiempo y los recursos se mantienen como constantes y se ajusta la funcionalidad de acuerdo con ello. Esto se expresa a través de reglas que se conocen como "reglas MoSCoW" por las iniciales de su estipulación en inglés. Las reglas se refieren a rasgos del requerimiento:

1. ***Must have:*** Debe tener. Son los requerimientos fundamentales del sistema. De estos, el subconjunto mínimo ha de ser satisfecho por completo.
2. ***Should have*:** Debería tener. Son requerimientos importantes para los que habrá una resolución en el corto plazo.
3. ***Could have:*** Podría tener. Podrían quedar fuera del sistema si no hay más remedio.
4. ***Want to have but won't have this time around:*** Se desea que tenga, pero no lo tendrá esta vuelta. Son requerimientos valorados, pero pueden esperar.

Esta metodología indica que debe haber varios equipos de 2 a 6 personas. Además, una persona puede cubrir dos o más roles y un rol puede estar dividido entre varias personas.

Las prácticas en esta metodología se llaman principios y son los siguientes:

1. Involucrar al usuario en forma activa.(DELFINO *et al.* 2005)
2. Los equipos de DSDM deben tener el poder de tomar decisiones.
3. Enfocarse en una entrega frecuente de productos (prototipos).
4. La conformidad con los propósitos del negocio es el criterio esencial para la aceptación de los entregables.
5. El desarrollo iterativo e incremental es necesario para converger hacia una correcta solución del negocio.
6. Todos los cambios durante el desarrollo son reversibles.

7. Los requerimientos están especificados a un alto nivel. (HERNÁN 2004)
8. Las pruebas no deben realizarse al final del proyecto, sino que se debe realizar a cada momento.
9. Un enfoque colaborativo y cooperativo entre todos los interesados es esencial.

El ciclo de desarrollo de DSDM está compuesto de 5 etapas o fases *(Ver Anexo 2):*

Etapa I: En la primera etapa se hace un **Estudio de Viabilidad** primero. Esto se realiza para saber si la tecnología o metodología a utilizar es útil para lo que se quiere obtener. La segunda parte de esta etapa se basa en un **Estudio del Negocio**, para saber de que trata el trabajo que se debe llevar a cabo. De este se obtiene el esqueleto de la arquitectura.

Etapa II: Iteración del Modelo Funcional. En esta etapa se hace un prototipo mínimo de lo que va a ser el requerimiento. Se construye la lista de requerimientos no funcionales.

Etapa III: Iteración de Diseño y Construcción. En esta etapa se refinan los requerimientos y se construye el sistema.

Etapa IV: Implementación. Se lleva a cabo la implementación del sistema.

Esta metodología no ofrece formatos de documentación detallado para sus productos de trabajo, pero ofrece guías tales como una breve descripción del problema, una lista de propósitos, y una media docena o más de preguntas de criterios de calidad para cada Producto de trabajo. Los elementos según las etapas del ciclo de vida se listan a continuación:

Estudio de Viabilidad:

- Informe de viabilidad.
- Plan de desarrollo.

Estudio del Negocio:

- Definición de los requerimientos funcionales.
- Diagrama de entidad-relación o modelo de objetos del negocio.
- Definición de la arquitectura del sistema.
- Plan de prototipo (la definición arquitectónica es un primer bosquejo)

Iteración del Modelo Funcional:

- Modelo funcional: Definición de los requerimientos, se puede adquirir la metodología de RUP.
- Funciones priorizadas: Es una lista de funciones entregadas al fin de cada iteración.
- Documentos de revisión funcional del prototipo: Reúnen los comentarios de los usuarios sobre el incremento actual para ser considerados en iteraciones posteriores.
- Requerimientos funcionales: Son listas que se construyen para ser tratadas en fases siguientes.
- Análisis de riesgo: Es un documento importante en la fase de iteración del modelo, porque desde la fase siguiente en adelante los problemas que se encuentren serán más difíciles de tratar.

Iteración de Diseño y Construcción:

- Pruebas del sistema: Se construye el sistema conforme a las reglas MoSCoW. De ahí surge un sistema que es testeado, donde el diseño y los prototipos funcionales son revisados por el usuario.

Implementación:

- Manual del usuario.
- Reporte de revisión del proyecto.(DELFINO *et al.* 2005)

DSDM incluye una interacción activa del usuario, entregas frecuentes y pruebas a lo largo del ciclo. Como otros métodos ágiles usa ciclos de plazos cortos. Hay un énfasis en la alta calidad y adaptabilidad hacia requisitos cambiantes. DSDM se apoya mucho en la fase de análisis y muy específicamente en la definición de requisitos, dejando las otras fases poco documentadas, no se definen entregables y artefactos específicos para los flujos de trabajo de diseño e implementación.

1.3.1.4 Feature Driven Development (FDD)

FDD es un método ágil, iterativo y adaptativo. A diferencia de otros métodos ágiles, no cubre todo el ciclo de vida sino sólo las fases de diseño y construcción y se considera adecuado para proyectos mayores y de misión crítica.

FDD no requiere un modelo específico de proceso y se complementa con otras metodologías. Enfatiza cuestiones de calidad y define claramente entregas tangibles y formas de evaluación del progreso.

FDD se estructura alrededor de la definición de rasgos o funcionalidades (*features*) que representan la funcionalidad que debe contener el sistema, y tienen un alcance lo suficientemente corto como para ser implementadas en un par de semanas. FDD posee también una jerarquía de *features*, siendo el eslabón superior el de *feature set* que agrupa un conjunto de *features* relacionadas con aspectos en común del negocio. Una de las ventajas de centrarse en las funcionalidades del software es el poder formar un vocabulario común que fomente que los desarrolladores tengan un diálogo fluido con los clientes, desarrollando entre ambos un modelo común del negocio.

Los principios de FDD son pocos y simples:

- Se requiere un sistema para construir sistemas si se pretende escalar a proyectos grandes.
- Un proceso simple y bien definido trabaja mejor.
- Los pasos de un proceso deben ser lógicos y su mérito inmediatamente obvio para cada miembro del equipo.
- Vanagloriarse del proceso puede impedir el trabajo real.
- Los buenos procesos van hasta el fondo del asunto, de modo que los miembros del equipo se puedan concentrar en los resultados.
- Los ciclos cortos, iterativos y orientados por *features,* son mejores.

FDD consiste en un conjunto de “mejores prácticas” que distan de ser nuevas pero se combinan de manera original. Las prácticas canónicas son:

1. Modelado de objetos del dominio, resultante en un *framework* cuando se agregan los rasgos. Esta forma de modelado descompone un problema mayor en otros menores; el diseño y la implementación de cada clase u objeto es un problema pequeño a resolver.
2. Desarrollo por rasgo. Hacer simplemente que las clases y objetos funcionen no refleja lo que el cliente pide. El seguimiento del progreso se realiza mediante examen de pequeñas funcionalidades descompuestas y funciones valoradas por el cliente. Un rasgo en FDD es una función pequeña expresada en la forma <acción> <resultado> <por | para | de | a> <objeto> con los operadores adecuados entre los términos.

3. Propiedad individual de clases (código). Cada clase tiene una sola persona nominada como responsable por su consistencia e integridad conceptual.
4. Equipos de Rasgos, pequeños y dinámicamente formados. La existencia de un equipo garantiza que un conjunto de mentes se apliquen a cada decisión y se tomen en cuenta múltiples alternativas.
5. Inspección. Se refiere al uso de los mejores mecanismos de detección conocidos.
6. *Builds* regulares. Siempre se tiene un sistema disponible. Los *builds* forman la base a partir de la cual se van agregando nuevos rasgos.
7. Administración de configuración. Permite realizar seguimiento histórico de las últimas versiones completas de código fuente.
8. Reporte de progreso. Se comunica a todos los niveles organizacionales necesarios. (REYNOSO 2004)

FDD consiste en cinco procesos secuenciales durante los cuales se diseña y construye el sistema. Cada fase del proceso tiene un criterio de entrada, tareas, pruebas y un criterio de salida.

Las fases que guían el proceso FDD son las siguientes *(Ver Anexo 4):*

Fase 1. Desarrollo de un modelo general: Esta fase sugiere un cierto paralelismo con la construcción de la arquitectura del software. En la creación de este modelo participan tanto los expertos de dominio que ya están al tanto de la visión, el contexto y los requerimientos del sistema a construir como los desarrolladores, a esta altura se espera que existan requerimientos tales como casos de uso o especificaciones funcionales. Mediante el esfuerzo de ambas partes se intenta lograr lo que el modelo en espiral proponía con sus primeras iteraciones: un conocimiento global de la aplicación a construir, el entendimiento del negocio en que esta embebida, un primer bosquejo de las funcionalidades del software, y la definición de restricciones y cuestiones no funcionales.

Para lograr todo esto, se desarrollarán: diagramas de paquetes, con las clases esenciales y las responsabilidades de las mismas; un documento similar al de Visión en donde se plasmen los objetivos del proyecto y como el mismo ayuda al negocio; un documento con los requerimientos no funcionales detectados; por último, el documento que podría llamarse arquitectura y en el que figuran las opciones de modelado surgidas durante esta actividad.

Entregables:

1. Diagrama de clases del sistema.
2. Lista informal de funcionalidades.
3. Registros de las alternativas más importantes sobre el modelo.

Fase 2. Construcción de la lista de rasgos: Los ensayos, modelos de objeto y documentación de requerimientos proporcionan la base para construir una amplia lista de rasgos. Los rasgos son pequeños *ítems* útiles a los ojos del cliente. Son similares a las tarjetas de historias de XP y se escriben en un lenguaje que todas las partes puedan entender. Las funciones se agrupan conforme a diversas actividades en áreas de dominio específicas. La lista de rasgos es revisada por los usuarios y patrocinadores para asegurar su validez y exhaustividad. Los rasgos que requieran más de diez días se descomponen en otros más pequeños.

Para el éxito de esta etapa el *Feature-List Team* debe tener completa cada *feature* del sistema, agrupada en *feature sets* y *mayor feature sets*.

Fase 3. Planeamiento por rasgo: Incluye la creación de un plan de alto nivel, en el que los conjuntos de rasgos se ponen en secuencia conforme a su prioridad y dependencia y luego es asignado a los programadores jefes. Las listas se priorizan en secciones que se llaman paquetes de diseño. Luego se asignan las clases definidas en la selección del modelo general a programadores individuales, o sea propietarios de clases. Se pone fecha para los conjuntos de rasgos.

El planificador del equipo debe producir un plan de desarrollo, sujeto a la revisión y aprobación del *Development Manager y del Chief Architect*. Este plan debe contener:

1. Una fecha global de terminación.
2. Para cada mayor *feature set, feature Set y feature* su *Chief Programmer* asigna fecha de comienzo y finalización.
3. Para cada clase, su responsable de desarrollo de la clase (*Class Owner*) correspondiente.

Fase 4. Diseño por rasgo: El desarrollador toma la próxima *feature* a ser diseñada, identifica las clases involucradas y contacta al *Class Owner* correspondiente. El *Class Owner* hace una descripción de la clase y sus métodos. Luego el equipo trabaja en la realización del diagrama de secuencia correspondiente.

Entregables:

1. Diagrama de secuencia detallado
2. Diagrama de clases actualizado
3. Descripción de clases y métodos
4. Notas del equipo que consideren significativas para el diseño.

Fase 5. Construcción por rasgos: En esta etapa cada *Class Owner* construye los métodos de clase para cada *feature* correspondiente y luego realiza las pruebas unitarias para cada una de las clases. A su vez, se realiza una inspección del código y luego que el código fue implementado e inspeccionado, el *Class Owner* realiza un *check-in* al CMS (*Configuration Management System*). Luego se realiza un *main build* en el cual se hace la integración con la funcionalidad antes realizada. También se realizan las pruebas de integración.

Entregables:

1. Diagrama de secuencia detallado.
2. Diagrama de clases actualizado.
3. Descripción de clases y métodos.
4. Notas del equipo que consideren significativas para el diseño. (MAUSQUES 2003)

En síntesis, FDD es un método de desarrollo de ciclos cortos que se concentra en la fase de diseño y construcción. En la primera fase, el modelo global de dominio es elaborado por expertos del dominio y desarrolladores; el modelo de dominio consiste en diagramas de clases, relaciones, métodos y atributos. Los métodos no reflejan conveniencias de programación sino rasgos funcionales.

Algunos agilistas sienten que FDD es demasiado jerárquico para ser un método ágil, porque demanda un programador jefe que dirige a los propietarios de clases y estos a su vez dirigen equipos de rasgos. Otros críticos sienten que la ausencia de procedimientos detallados de prueba en FDD es llamativa e impropia. Un rasgo llamativo de FDD es que no exige la presencia del cliente. (REYNOSO 2004)

FDD se centra en las fases de diseño e implementación del sistema, por lo que propone un conjunto de artefactos y entregables que pueden ser analizados en esta investigación.

1.3.1.5 SCRUM

Scrum es un complemento a las metodologías ágiles para el control, seguimiento y corrección de errores. Es una metodología de gestión del trabajo y no una metodología de análisis ni de diseño. Una de las características más importantes es que es fácil de explicar y de entender, lo que ayuda mucho a su implantación.

Por otra parte SCRUM puede ser aplicado a distintos modelos de calidad como podría ser CMMI (*Capability Maturity Model Integration*), puesto que estos dicen qué se tiene que hacer, es decir, que se tiene que gestionar del proyecto, pero no dicen cómo hacerlo. Ahí es donde entra SCRUM como modelo de gestión del proyecto. (ACUÑA 2006)

Como método, Scrum enfatiza valores y prácticas de gestión, sin pronunciarse sobre requerimientos, implementación y demás técnicas. Scrum se define como un proceso de *management* y control que implementa técnicas de control de procesos, se le puede considerar un conjunto de patrones organizacionales *(Ver Anexo 5).*

Como se ha visto hasta ahora esta metodología se centra mayoritariamente en la gestión de proyecto, la misma no propone entregables y artefactos para el desarrollo del software, es mas bien un complemento de otras metodologías, sean ágiles o no.

1.3.1.6 Evolutionary Project Management (Evo)

Tom Gilb fue el creador de esta metodología ágil, también conocida como: *Evolutionary Delivery, Evolutionary Management, Requirements Driven Project Management y Competitive Engineering*. Ofrece un planteamiento adaptativo orientado al cliente.(PALACIO 2006)

En las breves iteraciones de Evo, se efectúa un progreso hacia las máximas prioridades definidas por el cliente, liberando algunas piezas útiles para algunos participantes y solicitando su *feedback*. Esta es la práctica que se ha llamado Planeamiento Adaptativo Orientado al Cliente y Entrega Evolutiva. Otra idea distintiva de Evo es la clara definición, cuantificación, estimación y medida de los requerimientos de *performance* que necesitan mejoras. El *performance* incluye requisitos de calidad tales como robustez y tolerancia a fallas, al lado de estipulaciones cuantitativas de capacidad de carga y de ahorro de recursos.

En Evo se espera que cada iteración constituya una re-evaluación de las soluciones en procura de la más alta relación de valor contra costo, teniendo en cuenta tanto el *feedback* como un amplio conjunto de estimaciones métricas. Evo al igual que otros métodos ágiles requiere la participación activa de los clientes. Todo debe cuantificarse, se clasifican las apreciaciones cualitativas o subjetivas como “usable”, “mantenible” o “ergonómico”.

Los principios de ésta metodología son:

1. Se entregarán temprano y con frecuencia resultados verdaderos, de valor para los participantes reales.
2. El siguiente paso de entrega de Evo será el que proporcione el mayor valor para el participante en ese momento.
3. Los pasos de Evo entregan los requerimientos especificados de manera evolutiva.
4. No se puede saber cuáles son los requerimientos por anticipado, pero se pueden descubrir más rápidamente intentando proporcionar valor real a participantes reales.
5. Evo es ingeniería de sistemas holística (todos los aspectos necesarios del sistema deben ser completos y correctos) y con entrega a un ambiente de participantes reales (no es sólo sobre programación, también sobre satisfacción del cliente).
6. Los proyectos de Evo requieren una arquitectura abierta, porque se han de cambiar las ideas del proyecto tan a menudo como se necesite hacerlo, para entregar realmente valor a nuestros participantes.
7. El equipo de proyecto de Evo concentrará su energía como equipo hacia el éxito del paso actual. En este paso tendrán éxito o fracasarán todos juntos. No gastarán energías en pasos futuros hasta que hayan dominado los pasos actuales satisfactoriamente.
8. Evo tiene que ver con el aprendizaje a partir de la dura experiencia, tan rápido como se pueda: qué es lo que verdaderamente funciona, qué es lo que realmente entrega valor. Evo es una disciplina que permite confrontar los problemas tempranamente y progresar rápido cuando probadamente se ha trabajado bien.

9. Evo conduce a una entrega temprana, porque se lo ha priorizado así desde el inicio y porque se aprende desde el principio a hacer las cosas bien.
10. Evo debería permitir poner a prueba nuevos procesos de trabajo y deshacernos tempranamente de los que funcionan mal.

La metodología Evo posee cinco fases *(Ver Anexo 6):*

- **Metas, Valores y Costos:** Cuantificar los recursos y definir las Metas y Valores de los Participantes según la cultura, objetivos, metas estratégicas, requerimientos, propósitos, fines, ambiciones, cualidades e intenciones.
- **Soluciones:** Banco de ideas sobre la forma de alcanzar Metas y Valores dentro del rango de los Costos.
- **Estimación de Impacto:** Mapear las Soluciones a Metas y Costos para averiguar si se tienen ideas adecuadas para lograr las Metas dentro de los Costos.
- **Plan Evolutivo:** Es una idea general de la secuencia a desarrollar para evolucionar hacia las Metas. Los detalles necesarios evolucionan junto con el resto del plan a medida que se desarrolla el producto/servicio.
- **Funciones:** Describen qué hace el sistema. Son extremadamente secundarias y deben mantenerse al mínimo.

A diferencia de otras metodologías ágiles, donde todos son Participantes (*Stakeholders*), en Evo se llama Participante sólo al cliente. Cuando se inicia el ciclo, primero se definen los Valores y Metas del Participante, que no es más que una lista tradicional de recursos tales como dinero, tiempo y gente. Una vez que se comprende hacia dónde se quiere ir y cuándo se podría llegar ahí, se definen Soluciones para lograrlo. Utilizando una Tabla de Estimación de Impacto, se realiza la ingeniería de las Soluciones para satisfacer óptimamente las Metas y Valores de los Participantes. Se desarrolla un plan paso a paso llamado Entrega Evolutiva para entregar las mejoras a dichas Metas y Valores. Inicialmente las Soluciones y el Plan de Entrega Evolutiva se delinean a un alto nivel de abstracción. Tomando ideas de las Soluciones y del Plan se detallan, desarrollan, verifican y entregan a los Participantes reales o a quién se encuentre tan cerca de ellos como se pueda llegar. (REYNOSO 2004)

A medida que se desenvuelve el proyecto, se obtiene *feedback* en tiempo real sobre las mejoras que implica la Entrega Evolutiva, sobre las Metas y Valores del Participante y sobre el consumo de Recursos. Esta información se usa para establecer qué es lo que está bien y lo que no, cuáles son los desafíos y qué es lo que no se sabía desde un principio. También se aprende sobre las nuevas tecnologías y técnicas que no estaban disponibles cuando el proyecto empezó. Se ajusta luego todo según se necesite, pero sin detallar las Soluciones o las Entregas Evolutivas hasta que se esté próximo a la entrega. Por último vienen las Funciones y Sub-Funciones que son consideradas Soluciones a Metas.

En Evo, sin duda, se debe razonar de otro modo, todo tiene que ver con las Metas y Valores de los Participantes, expresadas en términos tales que una Solución (o como se la llama también Diseño, Estrategia, Táctica, Proceso, Funcionalidad, Arquitectura y Método) pueda definirse como un medio para un fin, a través del cual se lleve de la situación en la que se está a otra situación que se desea.

En proyectos evolutivos, las Metas se desarrollan tratando de comprender de quiénes vienen (Participantes), qué es lo que son (medios y fines) y cómo expresarlas (cuantificables, medibles y verificables). Se procura pasar el menor tiempo posible en tareas de documentación. (GARCÍA 2006)

Cómo se ha analizado hasta el momento, la metodología Evo se centra más en las metas que debe alcanzar la empresa y la forma de lograrlo que en la documentación. Por tal motivo no brinda los artefactos y entregables necesarios para obtener un software con calidad y que permita documentar adecuadamente los flujos de trabajo de diseño e implementación.

1.3.1.7 Crystal Method

Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. La familia Crystal dispone de un código de color para marcar la complejidad de una metodología: cuanto más oscuro un color, más “pesado” es el método. Por lo que existen cuatro variantes de metodologías en dependencia de dos factores: el número de personas en el equipo de desarrollo y la cantidad de riesgo.

Las cuatro variantes son: *Crystal Clear* (“Claro como el cristal”) para equipos de 8 o menos integrantes; Amarillo, de 8 a 20; Naranja, de 20 a 50; Rojo, de 50 a 100. Se promete seguir en un futuro con Marrón, Azul y Violeta.

Los principios fundamentales de esta metodología son:

1. Utilizar metodologías más grandes para equipos más grandes.
2. Utilizar metodologías más densas para proyectos más críticos.
3. La comunicación interactiva cara a cara, es la más eficaz.
4. El peso es costoso.

La más exhaustivamente documentada es *Crystal Clear* (CC), a continuación se describe el ciclo de desarrollo de esta metodología, la cual enfatiza el proceso como un conjunto de ciclos anidados *(Ver Anexo 7)*.

En la mayoría de los proyectos se perciben siete ciclos: (1) el proyecto, (2) el ciclo de entrega de una unidad, (3) la iteración (nótese que CC requiere múltiples entregas por proyecto pero no muchas iteraciones por entrega), (4) la semana laboral, (5) el período de integración, (6) el día de trabajo, (7) el episodio de desarrollo de una sección de código, de pocos minutos a pocas horas.

Los siete valores o propiedades de CC son:

- **Frecuencia en las entregas:** Entregar al usuario funcionalidad "usable" con una frecuencia de entre 2 semanas y no más de un mes.
- **Comunicación:** *Crystal Clear* toma como uno de sus pilares a la comunicación. Promueve prácticas como el uso de pizarrones, pizarras y espacios destinados a que todos (miembros del equipo y visitas) puedan ver claramente el progreso del trabajo.
- **Crecimiento reflexivo:** Es necesario que el equipo lleve a cabo reuniones periódicas de reflexión que permitan crecer y hacerse más eficientes.

Estas tres propiedades son "obligatorias" para *Crystal Clear*, las siguientes pueden agregarse en la medida de las necesidades de cada grupo y proyecto:

- **Seguridad personal:** Lograr que cada miembro del equipo pueda sentirse cómodo con el trabajo y el entorno.
- **Concentración:** Las entregas frecuentes permiten que cada desarrollador puede enfocar un problema por vez para evitar dispersiones.

- **Fácil acceso a usuarios clave:** Tratar de hacer que el usuario también sea parte del equipo es fundamental para ir depurando errores de manera temprana.
- **Entorno técnico con:** i) *Testing* automatizado (incorporación por ejemplo de *UnitTest*) ii) Integración frecuente (uso de herramientas específicas como *Cruise Control*).

Las principales técnicas son:

Las **reuniones diarias** (introducidas por la metodología Scrum) acompañan el seguimiento y mantienen el foco en el próximo paso a seguir, y también permiten la discusión productiva de líneas a seguir.

Las **reuniones de reflexión periódicas** son fundamentales para que los miembros del equipo se expresen abiertamente, para revisar el trabajo hecho y evaluar qué cosas dan resultado y cuáles no. (MORALES 2007)

A pesar que no contempla el desarrollo de software propiamente dicho, CC involucra una serie de productos de trabajo o artefactos. A continuación se mencionan los más importantes:

1. Declaración de la misión. Documento de un párrafo a una página, describiendo el propósito.
2. Estructura del equipo. Lista de equipos y miembros.
3. Metodología. Comprende roles, estructura, proceso, productos de trabajo y métodos de revisión.
4. Cronograma de visualización y entrega. Lista, planilla de hoja de cálculo o herramienta de gestión de proyectos.
5. Lista de riesgos. Descripción de riesgos por orden descendente de prioridad.
6. *Estatus* del proyecto. Lista hitos, fecha prevista, fecha efectiva y comentarios.
7. Lista de actores-objetivos. Lista de dos columnas, planilla de hoja de cálculo, diagrama de caso de uso o similar.
8. Archivo de requerimientos. Colección de información indicando qué se debe construir, quiénes han de utilizarlo, de qué manera proporciona valor y qué restricciones afectan al diseño.
9. Lista de Casos de Uso. Requerimientos funcionales.
10. Descripción de la Arquitectura.

11. Borradores de Pantallas.
12. Modelo Común de Dominio.
13. Notas.
14. Diagramas de Diseño.
15. Pruebas Sistema Empaquetado.
16. Mapa de Proyecto.
17. Plan o Secuencia de Entrega. Declaración o diagrama de dependencia que muestre el orden de las entregas y lo que hay en cada una.
18. Plan y Estado de Iteración.
19. Agenda o Cronograma de Visualización y entrega. Lista, planilla de hoja de cálculo o herramienta de gestión de proyectos.
20. Reporte de *Bugs*.
21. Manual de Usuario.
22. Estructura y Convenciones del Equipo.
23. Resultados del Taller de Reflexión.

Los métodos Crystal no prescriben las prácticas de desarrollo, las herramientas o los productos que pueden usarse, pudiendo combinarse con otros métodos como *Scrum, XP y Microsoft Solutions Framework*.(ANAISA 2004)

Esta familia de metodologías, al igual que el resto de las metodologías ágiles plantea que la comunicación con el cliente es vital y que es más importante el levantamiento de requisitos. Los artefactos que se dedican a las fases de diseño e implementación son muy reducidos y ya están contemplados en otras metodologías.

1.3.1.8 Lean Development (LD)

Es el método menos divulgado entre los reconocidamente importantes. LD, iniciado por Bob Charette, se inspira en el éxito del proceso industrial *Lean Manufacturing*, bien conocido en la producción automotriz y en manufactura desde la década de 1980. Este proceso tiene como precepto la eliminación de residuos a través de la mejora constante, haciendo que el producto fluya a instancias del cliente para hacerlo lo más perfecto posible.

En LD los cambios se consideran riesgos, pero si se manejan adecuadamente se pueden convertir en oportunidades que mejoren la productividad del cliente. Su principal característica es introducir un mecanismo para implementar dichos cambios.

LD se inspira en doce valores centrados en estrategias de gestión:

1. Satisfacer al cliente es la máxima prioridad.
2. Proporcionar siempre el mejor valor por la inversión.
3. El éxito depende de la activa participación del cliente.
4. Cada proyecto LD es un esfuerzo de equipo.
5. Todo se puede cambiar.
6. Soluciones de dominio, no puntos.
7. Completar, no construir.
8. Una solución al 80% hoy, en vez de una al 100% mañana.
9. El minimalismo es esencial.
10. La necesidad determina la tecnología.
11. El crecimiento del producto es el incremento de sus prestaciones, no de su tamaño.
12. Nunca empujar a LD más allá de sus límites.

Dado que LD es más una filosofía de *management* (administración) que un proceso de desarrollo no hay mucho que decir del tamaño del equipo, la duración de las iteraciones, los roles o la naturaleza de sus etapas. Últimamente LD ha evolucionado como *Lean Software Development* (LSD); su figura de referencia

es Mary Poppendieck, quien presenta una perspectiva ampliada del método, cuidadosamente decantada del *Lean Manufacturing* y de *Total Quality Management* (TQM). Los principios son los siguientes:

1. Eliminar basura: Entre la basura se cuentan diagramas y modelos que no agregan valor al producto.
2. Minimizar inventario: Igualmente, suprimir artefactos tales como documentos de requerimiento y diseño.
3. Maximizar el flujo: Utilizar desarrollo iterativo.
4. Solicitar demanda: Soportar requerimientos flexibles.
5. Otorgar poder a los trabajadores.
6. Satisfacer los requerimientos del cliente: Trabajar junto a él, permitiéndole cambiar de ideas.
7. Hacerlo bien la primera vez: Verificar temprano y refactorizar cuando sea preciso.
8. Abolir la optimización local: Alcance de gestión flexible.
9. Asociarse con quienes suministran: Evitar relaciones de adversidad.
10. Crear una cultura de mejora continua.

LD y LSD han sido pensados como complemento de otros métodos, y no como una metodología excluyente a implementar en la empresa. LD prefiere concentrarse en las premisas y modelos derivados de *Lean Production*. (LUZ 2004)

Esta metodología propone suprimir una serie de documentos importantes para el proyecto, como es el caso de los documentos de diseño, importantes para la buena comprensión del producto y su mantenimiento.

1.3.2 Las metodologías Tradicionales o Robustas

Estas metodologías también son consideradas como fuertes o establecidas, dentro de las cuales se destaca Rational Unified Process (RUP) y Microsoft Solutions Framework (MSF).

1.3.2.1 Rational Unified Process (RUP)

RUP es un proceso creado por Jacobson, Rumbaugh y Booch resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo y los mejores elementos de metodologías anteriores. Está preparado para desarrollar grandes y complejos proyectos y es orientado a objetos, el mismo define claramente *quién, cómo, cuándo y qué debe hacerse*. Como su enfoque esta basado en modelos utiliza un lenguaje de representación visual bien definido para tal fin, el UML.

RUP define como sus principales elementos:

- Trabajadores (“quién”): Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.
- Actividades (“cómo”): Es una tarea que tiene un propósito claro, los procesos que se determinan en cada iteración, es realizada por un trabajador y manipula elementos.
- Artefactos (”qué”): Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Puede ser un documento, un modelo, elementos dentro del modelo, código fuente y ejecutables.
- Flujo de trabajo (“cuándo”): Secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

Las características principales del proceso son:

- Guiado por los Casos de Uso.
- Centrado en la Arquitectura.
- Iterativo e Incremental.

RUP implementa las siguientes mejores prácticas asociadas al proceso de Ingeniería de Software:

- Desarrollo Iterativo.
- Manejo de los Requerimientos.
- Uso de una Arquitectura basada en componentes.
- Modelización Visual.
- Verificación Continua de la Calidad.
- Manejo de los Cambios. (PRESSMAN 2002)

La metodología RUP divide en 4 fases el desarrollo del software *(Ver Anexo 8)*. Cada Fase tiene definido un conjunto de objetivos y un punto de control específico.

Fase 1: Inicio

Es la primera fase del sistema y consiste en adquirir los requerimientos por parte de los distintos usuarios y consolidar una visión única de los objetivos y alcances del sistema.

Los objetivos particulares de esta fase son:

- Definición del producto, aceptada por todos los *Stakeholders* (partes interesadas) involucrados en el proyecto.
- Discriminar los casos de uso (funcionalidades) prioritarios de los posibles pero no imprescindibles.
- Proponer una arquitectura inicial.
- Estimar los recursos necesarios para el desarrollo del proyecto y la distribución de roles y responsabilidades.
- Definir las herramientas a utilizar en cada parte del proceso.

Fase 2: Elaboración

El objetivo de esta fase es definir la arquitectura del sistema proveyendo bases sólidas para el proceso de diseño e implementación. La definición de la arquitectura debe tener en cuenta los requerimientos obtenidos en la etapa de inicio y proveer las alternativas para el control de riesgos.

Los objetivos principales de esta fase incluyen:

- Asegurar que los requerimientos y definiciones obtenidos en la etapa de inicio sean sólidos y se hayan contemplado y mitigado todos los riesgos que podrían afectar el desarrollo del sistema.
- Definir los posibles escenarios de instalación y trabajo, verificando las necesidades de equipamiento de hardware e infraestructura.
- Analizar y profundizar en cada uno de los casos de uso obtenidos en la etapa de inicio para elaborar un prototipo funcional que permita verificar el alcance del desarrollo de software.
- Revisar que los requerimientos de software se correspondan con la estructura actual de trabajo y documentar las propuestas de reorganización.
- Afinar el diseño de las arquitecturas a fin de verificar que sea sólida y cumpla con los requerimientos del sistema. Analizar el posible re-uso de componentes dentro de la arquitectura seleccionada.
- Refinar el esquema de desarrollo seleccionando herramientas y metodologías particulares para la etapa siguiente (construcción).

Fase 3: Construcción

Es la etapa del desarrollo del sistema, en el cual se deben obtener finalmente las herramientas necesarias para resolver los requerimientos definidos en las etapas previas.

Objetivos a cumplir en esta etapa:

- Obtener un sistema de calidad en un tiempo acotado.
- Completar para cada módulo a desarrollar las etapas de análisis, diseño, desarrollo y prueba.
- Trabajar en paralelo en el desarrollo de subsistemas y módulos que pueden ser elaborados de forma independiente.
- Trabajar de forma iterativa e incremental en el desarrollo, documentando y completando las definiciones de los casos de uso, diseño y pruebas.
- Probar los ambientes de instalación y realizar instalaciones beta de los productos en entornos similares a los definitivos.

- Instalar y probar las redes y software de base necesarios para la futura instalación del sistema.

Fase 4: Transición

Es el momento en que el sistema debe ser entregado a sus usuarios finales. Esta fase puede contar con varias iteraciones pero involucra al usuario final y al equipo o empresa de desarrollo. Al finalizar esta etapa el sistema debe quedar en manos de los usuarios, para esto se debe lograr la confianza en el nuevo sistema.

Objetivos y tareas involucradas en esta fase:

- Instalación del software en el entorno final de trabajo, realizando instalaciones progresivas y pruebas.
- Capacitación de los usuarios con la nueva herramienta.
- Conversión e importación de datos anteriores al nuevo sistema.
- Ajuste del software y la organización.
- Medición de *performance* de la herramienta y del esquema organizacional.
- Pruebas de estrés sobre las redes y equipamiento, verificación de los planes de contingencia. (AIRES 2005)

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.

Vale mencionar que el ciclo de vida que se desarrolla por cada iteración, es llevada bajo dos disciplinas que contienen los flujos de trabajo *(Ver Anexo 8)*.

Disciplinas de Proceso:

Modelamiento del Negocio: Entender las necesidades del negocio, describiendo los procesos del mismo, identificando quiénes participan y las actividades que requieren automatización.

Artefactos:

- Documento Visión del negocio.

- Glosario de términos del negocio.
- Actor.
- Caso de uso.
- Diagramas de actividades.
- Reglas del negocio.
- Modelo de caso de uso del negocio.
- Modelo de Objeto.
- Modelo de análisis del negocio.
- Documento de arquitectura del negocio.
- Realización de cada CUN.

Requerimientos: Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.

Artefactos:

- Lista de requisitos.
- Modelo de casos de uso del sistema.
- Actor.
- Caso de uso.
- Descripción de la arquitectura (vista del modelo de casos de uso).
- Glosario de términos.
- Prototipo de interfaz usuario.

Análisis y Diseño: Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), trasladando estos últimos dentro de la arquitectura de software, por lo que indica con precisión lo que se debe programar.

Artefactos:

- Diagrama de clases del análisis.
- Diagrama de clases del diseño.
- Realización de casos de uso del análisis.
- Realización de casos de uso del diseño.
- Modelo de análisis.
- Modelo de diseño.
- Paquete de análisis.
- Descripción de la arquitectura (vista del modelo de análisis).
- Diagrama de colaboración.
- Diagrama de secuencia.
- Modelo de datos.
- Diagrama de Despliegue.

Implementación: Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación. Creando un software que se ajuste a la arquitectura y que tenga el comportamiento deseado.

Artefactos:

- El modelo de implementación.
- Componente.
- Interfaz.
- Subsistema de Implementación.
- Diagrama de Componentes.
- Documento de Arquitectura de Software (actualizado).

- Plan de Integración.

Pruebas (Testeo): Busca los defectos a lo largo del ciclo de vida. Asegurándose que el comportamiento requerido es el correcto y que todo lo solicitado está presente.

Artefactos:

- El plan de Pruebas.
- La estrategia de prueba.
- Los casos de prueba.
- Los Procedimientos de prueba.
- El listado de ideas de prueba.
- El listado de datos de prueba.
- El resumen de la evaluación de las pruebas.

Instalación o Despliegue: Produce *release* del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.

Artefactos:

- Producto.
- Material de Soporte.
- Manual de usuario.

Disciplinas de Soporte:

Administración de configuración y cambios: Hacer todo lo necesario para la salida del proyecto. Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.

Administración del Proyecto: Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes, administrando horarios y recursos.

Ambiente: Administrando el ambiente de desarrollo. Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto, así como el procedimiento para implementar el proceso en una organización.

Una particularidad de esta metodología es que en cada ciclo de iteración se hace exigente el uso de artefactos. Por este motivo es una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software. (REYNOX 2005)

A modo de resumen esta metodología plantea un gran número de artefactos y entregables bien definidos en cada uno de los flujos de trabajo, los cuales se consideran importantes para analizar en esta investigación por el aporte que da la misma a los flujos de trabajo de diseño e implementación.

1.3.2.2 Microsoft Solutions Framework (MSF)

MSF es una metodología flexible e interrelacionada con una serie de conceptos, modelos y mejores prácticas de uso que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos.

Concretamente MSF se compone de principios, modelos y disciplinas.

MSF contiene diez principios básicos:

1. Promover comunicaciones abiertas.
2. Trabajar para una visión compartida.
3. Fortalecer los miembros del equipo.
4. Establecer responsabilidades claras y compartidas.
5. Focalizarse en agregar valor al negocio.
6. Permanecer ágil y esperar los cambios.
7. Invertir en calidad.
8. Aprender de todas las experiencias.
9. *Partners* con clientes.
10. Siempre crear productos entregables.

Las disciplinas que ofrece MSF son:

Gestión de Proyectos: Esta disciplina proporciona una serie de estrategias para la gestión de proyecto. Se basa en planificar sobre entregas cortas, incorporar nuevas características sucesivamente e identificar cambios ajustando el cronograma.

Control de Riesgos: Diseñada para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.

Control de Cambios: Diseñada para que el equipo sea proactivo en lugar de reactivo. Los cambios deben considerarse riesgos inherentes y además deben registrarse y hacerse evidentes.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto. Se centra fundamentalmente en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

Equipo de Trabajo: Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles. (MENDOZA 2004)

Modelo de Proceso: Este modelo, a través de su estrategia iterativa en la construcción de productos del proyecto, suministra una imagen más clara del estado de los mismos en cada etapa sucesiva. El equipo puede identificar con mayor facilidad el impacto de cualquier cambio y administrarlo efectivamente, minimizando los efectos colaterales negativos mientras optimiza los beneficios. Ha sido diseñado por tanto, para mejorar el control del proyecto minimizando el riesgo y para aumentar la calidad acortando el tiempo de entrega *(Ver Anexo 9)*.

A continuación se relacionan los objetivos y entregables de cada una de las fases del proceso MSF.

1. Visión (Visión y Alcance Aprobados)

Objetivo:

Obtener una visión del proyecto compartida, comunicada y entendida, que se encuentre lineada con los objetivos del negocio.

Además, identificar los beneficios, requerimientos funcionales, sus alcances y restricciones; y los riesgos inherentes al proceso.

Entregables:

- Documento Visión.
    1. Antecedentes y Visión.
    2. Criterios de diseño.
- Documento Detalle de la Visión.
    1. Beneficios, metas, objetivos, restricciones.
    2. Perfiles de usuario.
    3. Casos de uso.
    4. Requerimientos funcionales, no funcionales.
    5. Requerimientos del sistema.
    6. Plan de instalación.
    7. Arquitectura lógica (Diagrama de componentes UML).
    8. Arquitectura física (Diagrama de despliegue UML).
    9. Documento Requerimientos Funcionales (incluye Script de Pruebas).
    10. Descripción detallada de los requerimientos y características que componen cada caso de uso descrito en el documento Detalle de la Visión, indicando perfiles asociados, recursos del equipo de proyecto, riesgos, observaciones y script de pruebas.
- Documento Matriz de Riesgos.
    1. Identifica posibles riesgos acerca de los requerimientos y las acciones a tomar en cada escenario.
- Acta de aprobación de Visión.

2. Planeación (Cronograma de Proyecto Aprobado)

Objetivo:

Obtener un cronograma de trabajo que cumpla con lo especificado en la fase de Visión dentro del presupuesto, tiempo y recursos acordados. Este cronograma debe identificar puntos de control específicos que permitan generar entregas funcionales y cortas en el tiempo.

Entregables:

- Documento de Cronograma.
- Acta de aprobación de Cronograma.

3. Desarrollo (Alcance Completo)

Objetivo:

Obtener iterativamente de las fases de Planeación y Estabilización, versiones del producto, entregables y medibles que permitan de cara al cliente probar características nuevas sucesivamente. Esto incluye los ajustes de cronograma necesarios.

Entregables:

- Fuentes y ejecutables (según lo acordado).
- Documentos Manuales técnicos, de usuario y de instalación si es necesario.
- Acta de finalización de Desarrollo.

4. Estabilización (Versión Aprobada)

Objetivo:

Obtener una versión final del producto probada, ajustada y aprobada en su totalidad.

Entregables:

- Documento Registro de pruebas.
- Acta de aprobación de Versión Aprobada.

5. Instalación (Entrega)

Objetivo:

Entregar (instalar) al cliente el producto finalizado en su totalidad, como garantía de que se han superado con éxito las etapas anteriores.

Entregables:

- Conjunto de archivos (ejecutables, directorios, archivos varios, bases de datos, scripts, instaladores, manuales, licencias, entre otros) propios del producto que permitan su instalación y correcto funcionamiento.
- Acta de Entrega y Finalización de Proyecto.

6. Soporte (Entrega Ajustada)

Objetivo:

Brindar soporte y garantía al producto durante el tiempo estipulado en el contrato; registrando los reportes de soporte y mantenimiento recibidos, así como los ajustes y versiones ajustadas obtenidas. Esto sólo será válido para ajustes que estén dentro de lo descrito en los documentos de la fase de Visión.

En esta fase es posible identificar características y requerimientos que no fueron tenidos en cuenta y que se salen del alcance de la fase de Visión. Por tanto es probable que se inicie otro proceso con los nuevos requerimientos, generando así un nuevo proyecto y un nuevo inicio de las fases de la metodología.

Entregables:

- Documento de registro de reportes de soporte y mantenimiento y ajustes hechos.

Dentro de las principales características de MSF se tienen:

- **Adaptable:** Es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- **Escalable:** Puede organizar equipos pequeños de 3 o 4 personas, así como también, proyectos que requieren 50 personas o más.
- **Flexible:** Es utilizada en el ambiente de desarrollo de cualquier cliente.

- **Tecnología Agnóstica:** Puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

La metodología MSF se adapta a proyectos de cualquier dimensión y de cualquier tecnología, propone un conjunto de artefactos y entregables que ya están presentes en otras metodologías.

1.4 Lenguaje de Modelado Unificado (UML)

Siempre que usemos una metodología de desarrollo para generar artefactos a través de las distintas etapas de su ciclo de vida, es bueno usar una manera legible para que cada integrante del equipo o cada cliente entienda lo que se quiere hacer llegar, es por esto que actualmente casi todas las empresas productoras de software utilizan un lenguaje reconocido mundialmente como UML.

UML (*Unified Modelling Language* por sus siglas en inglés), es el lenguaje de modelado de sistemas de software más conocido en la actualidad, está apoyado en gran medida por el OMG (*Object Management Group*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar artefactos de sistemas de software. Usar UML en el marco de un proceso de desarrollo como RUP, MSF o cualquier proceso Ágil (Agile Modeling, eXtreme Programming, SCRUM, Crystal), es una garantía definitiva para asegurar la calidad de los modelos y de todos los artefactos producidos con la notación.

El Lenguaje de Modelado Unificado es una notación estándar con carácter universal que utiliza una serie de diagramas y una semántica bien definida para representar y modelar la información con la que se trabaja en las fases de análisis y, especialmente, de diseño. (CARABALLO 2003)

Es importante destacar que UML es un "lenguaje" para especificar y no un método o un proceso, es decir, es una herramienta útil para representar los modelos del sistema en desarrollo, mas no ofrece ningún tipo de guía o criterios acerca de cómo obtener esos modelos. El UML se usa para definir un sistema de software, para detallar los artefactos en el sistema, para documentar y construir. El UML se puede usar en una gran variedad de formas para soportar una metodología de desarrollo de software pero no especifica en sí mismo qué metodología o proceso usar. (VERA 2006)

El UML es una parte muy importante para el desarrollo de Software Orientados a Objetos y en el Proceso de Desarrollo de Software. Utiliza, en su mayor parte, notaciones gráficas para expresar los proyectos de

diseño del Software. Utilizando el ayudante del UML puede comunicar el equipo de proyecto, explorar el potencial de diseños, y validar el diseño de la arquitectura del Software.

Las principales metas del UML son:

- Proveer usuarios con un "*ready-to-use*" (facilidad de uso), lenguaje de modelación visual expresivo donde ellos puedan desarrollar e intercambiar modelos significativos.
- Proveer extensamente y específicamente mecanismos para extender el núcleo de conceptos.
- Ser independientes en los lenguajes de programación particulares y procesos de desarrollo.
- Proveer una base formal para el entendimiento del lenguaje de modelación.
- Fomentar el crecimiento de las herramientas del mercado Orientado a Objetos.
- Soportar el concepto de desarrollo en alto nivel tal como colaboraciones, sistemas, modelos y componentes.
- Integrar mejores prácticas. (GIL 2006)

UML incluye los siguientes diagramas:

- Diagrama de casos de usos.
- Diagrama de clases.
- Diagrama de objetos.
- Diagrama de secuencia.
- Diagrama de colaboración.
- Diagrama de estado.
- Diagrama de actividades.
- Diagrama de componentes.
- Diagrama de despliegue.
- Diagrama de paquetes.

Los diagramas más interesantes y los más usados son los de casos de uso, clases y secuencia.

UML es además un método formal de modelado. Esto aporta las siguientes ventajas:

- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa (a partir del código fuente generar los modelos). Esto permite que el modelo y el código estén actualizados, con lo que siempre se puede mantener la visión de más alto nivel en el diseño de la estructura de un proyecto. (ORALLO 2005)

Ante los requisitos cambiantes y novedosos en materia de modelado de software que surgieron en los últimos años debido a la evolución de los sistemas software y las necesidades que se requieren y exigen los mismos, surge el UML 2.0 con novedades respecto a las versiones UML 1.x.

La nueva versión UML 2.0, tiene como principales objetivos:

1. Hacer el lenguaje de modelado mucho más extensible de lo que era.
2. Permitir la validación y ejecución de modelos creados mediante UML.

UML 2.0 propone tres diagramas nuevos: el diagrama de estructura compuesta, diagrama de tiempos y diagrama de estado de máquina. Al diagrama de colaboración se le llama en esta nueva versión diagrama de comunicación.

Desde la primera versión adoptada por OMG, UML ha evolucionado a través de diferentes versiones. Pero como ha señalado *Booch*, el crecimiento ha significado mejorar su expresividad y precisión, pero no ha sufrido cambios en sus aspectos esenciales.

UML ha ejercido un gran impacto en la comunidad software, tanto a nivel de desarrollo como de investigación. Su éxito ha sido enorme, como lo prueban por una parte, su utilización en todo el mundo para construir aplicaciones en todos los dominios y de todos los tamaños y por otra, que los entornos de desarrollo más extendidos, como son los de Borland, Microsoft e IBM, integran herramientas para el modelado con UML.(ROSSI 2004)

Existen muchas razones por las que UML ha sido ampliamente usado. En el desarrollo del mismo, han participado investigadores de reconocido prestigio, unido al apoyo por prácticamente todas las empresas importantes de informática y la aceptación como un estándar por la OMG. Además prácticamente todas las herramientas CASE (C*omputer Aided Software Engineering*, Ingeniería de Software Asistida por Ordenador) y de desarrollo la han adaptado como lenguaje de modelado. (ORALLO 2005)

Para lograr una aplicación software robusta, flexible y escalable, es necesario tanto un proceso como un lenguaje. Se propone el uso de UML, pues como bien se ha dicho hasta aquí es el lenguaje que la mayoría de los desarrolladores utilizan y entienden, además de todas las ventajas que nos ofrecen y que han sido mencionadas anteriormente, se puede considerar además el uso de las mejoras incorporadas por el UML 2.0 en caso de aplicaciones que lo requieran.

1.5 Conclusiones

Se ha llegado a la conclusión que el uso de una metodología u otra depende del equipo de desarrollo y del interés del cliente, es decir, si se quiere trabajar con proyectos de gran tamaño es recomendable usar metodologías tradicionales, si por el contrario se desean entregas rápidas y el proyecto es de tamaño pequeño, las metodologías ágiles proporcionan una solución a este problema, lo realmente importante es que independientemente de la metodología que se use, el software debe quedar bien documentado para conocimiento tanto del cliente como del equipo del trabajo.

Por todo esto es necesario proponer la documentación imprescindible para complementar estos problemas, es decir, definir un modelo que recoja cuales deben ser los principales entregables y artefactos a usar en los flujos de trabajo de diseño e implementación independientemente de la metodología que se use.

Hasta ahora se ha hecho un análisis detallado de un conjunto de metodologías de desarrollo y gestión de software, llegando a la conclusión que se debe centrar el estudio en las metodologías que propongan artefactos y entregables que brinden la información necesaria y realmente primordial para las fases de diseño e implementación.

Por tanto, para hacer dicha propuesta se tendrán en cuenta las metodologías RUP y FDD, pues definen artefactos y entregables que aportan información relevante para dicho trabajo.

CAPÍTULO
DESCRIPCIÓN DE LA PROPUESTA

2.1 Introducción

En el presente capítulo se describen los flujos de trabajo de diseño e implementación, cómo se desarrollan dentro de la Universidad de las Ciencias Informáticas. Se describe la encuesta aplicada a distintos proyectos productivos que desarrollan software de gestión y se analizan los resultados arrojados por la misma. Además se definen los artefactos y entregables imprescindibles para los flujos de trabajo de diseño e implementación, destacando la importancia de los mismos dentro del proceso de desarrollo de software, quedando conformada de esta forma la propuesta.

2.2 Caracterización del fenómeno

La UCI constituye un nuevo modelo de formación – investigación – producción que ofrece amplias posibilidades al desarrollo de la Industria Cubana del Software y los servicios informáticos.

Fue creada sobre una fuerte base tecnológica y un amplio perfil productivo, ésta pretende convertirse en la vanguardia del desarrollo de software en Cuba y de llevar la informatización a todos los sectores de la sociedad. Propiciando un avance tecnológico para convertir la industria en una importante base de desarrollo del país y una fuente de ingresos en divisas.

La tendencia actual en el software lleva a la construcción de sistemas más grandes y más complejos. Esto es debido en parte al hecho de que los computadores son más potentes cada año, y los usuarios, por tanto, esperan más de ellos. Esta tendencia también se ha visto afectada por el uso creciente de Internet para el intercambio de todo tipo de información. El apetito de software aún más sofisticado crece a medida

que se ve cómo pueden mejorarse los productos de una versión a otra. Se quiere un software que esté mejor adaptado a las necesidades, pero esto, a su vez, simplemente hace el software más complejo.

El proceso que se usa en la universidad para guiar el desarrollo de sistemas es el RUP, este brinda un conjunto de pasos y actividades para que todo el proceso sea satisfactorio. El proceso se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo consta de cuatro fases: inicio, elaboración, construcción y transición. Cada fase se subdivide a su vez en iteraciones, cada iteración pasa los cinco flujos de trabajo del proceso: requisitos, análisis, diseño, implementación y prueba.

El objeto de esta investigación lo constituyen los flujos de trabajo de diseño e implementación en el proceso de desarrollo de software de gestión en la UCI.

En flujo de trabajo de diseño es dónde se modela el sistema y se encuentra su forma para que soporte todos los requisitos. El diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción. Esto contribuye a una arquitectura estable y sólida y ayuda a crear un plano del modelo de implementación.

En el flujo de trabajo de implementación se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, *scripts*, ficheros de código binario, ejecutables, etc. Es el centro durante las iteraciones de construcción, aunque también se lleva a cabo trabajo de implementación durante la fase de elaboración para crear la línea base ejecutable de la arquitectura, y durante la transición, para tratar defectos tardíos cómo los encontrados con distribuciones beta del sistema.

Actualmente en la universidad estos flujos de trabajo se hacen de forma incompleta, pues no siempre se documenta lo que se hace o no se documenta en el momento requerido, esto trae consigo que se pierda información importante para todo el equipo de desarrollo, pues solamente las personas involucradas en estos flujos conocen lo que se hace.

Es importante que el quipo de desarrollo que participa en un proyecto tenga conocimiento de todo lo que pasa dentro del mismo y es necesario que cada flujo de trabajo deje explicado de alguna manera lo que se ha hecho para que el resto lo sepa, además de que se tenga un expediente del proyecto donde se pueda encontrar lo necesario para estos flujos.

2.3 Descripción de la Encuesta

Para hacer un diagnóstico del estado actual de los proyectos que desarrollan software de gestión en la UCI, fue necesario aplicar una encuesta para captar información cualitativa y cuantitativa del fenómeno. En el *Anexo 10* se puede ver el modelo de la encuesta aplicada.

Las preguntas de la encuesta fueron diseñadas para conocer el grado de conocimiento del personal involucrado en proyectos, en cuanto a la documentación de los flujos de trabajo de diseño e implementación. Así como el conocimiento de la metodología usada y los principales flujos de trabajo.

En la elaboración de la encuesta se combinaron los dos tipos de preguntas fundamentales, abierta y cerradas. Las preguntas cerradas se hicieron con el interés de conocer la información cuantitativa con respecto al tema y las preguntas abiertas con el objetivo de saber la opinión del encuestado.

2.4 Análisis de la encuesta

Las encuestas de realizaron en 83 proyectos productivos que se dedican al desarrollo de software de gestión, para un total de 171 encuestados, de estos, 5 vicedecanos de producción, 83 líderes de proyectos y 83 estudiantes.

Haciendo un análisis por preguntas se puede decir que:

Pregunta 1 y 2:

La pregunta arrojó que de las 171 personas entrevistadas el 65.06% plantean que en su proyecto si se establecen plantillas para los documentos, sin embargo de este porcentaje solamente el 2% sabía a que artefacto. El 9.64% del total de personas no saben si se establecen plantillas en su proyecto. Por lo que se puede concluir que en el equipo de trabajo no se tiene conocimiento de esta información.

Pregunta 3:

El 63.75% de las personas respondieron que tenían conocimiento de la metodología RUP, el 23.9% tiene conocimiento de XP y un porcentaje muy reducido conoce una de las otras metodologías que existen para guiar el proceso de desarrollo de software. Esto demuestra que las personas se casan con la metodología que se imparte en clases y no estudian el resto de las metodologías, ni siquiera tienen conocimiento de que existen.

Pregunta 4:

El 77.11% utiliza la metodología RUP, durante el proceso de desarrollo de su proyecto, el 9.64% usa la metodología XP y el 13.25% no sabe que metodología se usa en su proyecto, por lo que se puede ver que la metodología más usada es la que se imparte en clases y que las personas prefieren trabajar con algo que ya conocen y no dedicar tiempo a aprender algo nuevo.

Pregunta 5:

El 25.9% plantea que el flujo de trabajo más importante dentro del proceso de desarrollo lo primordial es el flujo de diseño, el 22.71% define que es el análisis, el 21.51% dice que el más importante es el flujo de captura de requisitos, el 13.15% aboga por el flujo de implementación, el 8.76% se inclina por el flujo de trabajo de modelado del negocio y el 3.19% define que las pruebas al sistema es lo fundamental. Esto demuestra que a pesar de que el mayor porcentaje de importancia conferida, corresponde al flujo de trabajo de diseño, la mayoría de los encuestados no consideran que los flujos de trabajo de diseño e implementación sean importantes durante el proceso de desarrollo de software, pues no se tiene una información detallada de las actividades que se realizan durante esos dos flujos cómo se verá más adelante.

Pregunta 6:

El 90% no supo responder el objetivo o la importancia del flujo de trabajo de diseño. Por otra parte, el 59.19% de las personas encuestadas plantean que dentro de las actividades que se realizan en este flujo, el análisis de la Arquitectura es la más importante, el 55.78% se inclinan por el diseño de la base de datos, el 52.19% considera que debe hacerse el diseño de las clases y otro porcentaje reducido se inclina por las demás actividades. Además el 90% no tiene conocimiento de los artefactos que se documentan en su proyecto, esto demuestra que la información no se socializa a todo el equipo de desarrollo.

Pregunta 7:

El 75% no supo responder el objetivo o la importancia del flujo de trabajo de implementación, el 22% no respondió nada, por lo que se demuestra que no existe un conocimiento por parte de los participantes en los proyectos sobre estas actividades.

Por otra parte, el 80% de las personas encuestadas plantea que dentro de las actividades que se realizan en el flujo de trabajo de implementación, lo más importante es entregar una versión del sistema, el 20% se

inclina por la descripción de los componentes, además el 70% no tenía conocimiento de los artefactos y entregables que se debían emitir durante este flujo de trabajo, lo que demuestra que no se documenta o que la información no se socializa dentro del equipo de trabajo.

De forma general se demostró que en la UCI hay poca cultura de documentar los sistemas de software y que no se cuenta con el conocimiento suficiente para decidir en cada momento que es lo importante documentar para estos flujos de trabajo durante el desarrollo del sistema, por tanto es muy importante que se establezcan cuáles son los principales entregables para que estos flujos queden bien documentados.

2.5 Documentación para los flujos de trabajo de diseño e implementación

Los artefactos que se definirán para conformar la propuesta, se consideran los necesarios para apoyar y ayudar a los desarrolladores en la construcción de software de gestión durante los flujos de trabajo de diseño e implementación, de manera que durante el ciclo de vida del software cada uno sepa lo que está pasando en el proyecto y se socialice el conocimiento adquirido en el mismo, y que además una vez terminado el producto se cuente con la documentación necesaria para que todo el equipo de trabajo entienda lo que se ha construido y sirva de sustento a los posibles cambios y mejoras del producto. A continuación se describirán los aspectos que se tuvieron en cuenta para la propuesta de los artefactos.

El estudio realizado en el Capítulo 1 acerca de las metodologías de desarrollo de software más reconocidas y la documentación que proponen las mismas, arrojó una serie de conclusiones que sirven de punto de partida para conformar la propuesta. Algunas de estas conclusiones son:

- La mayoría de las metodologías proponen que el diagrama de clases y el documento de la arquitectura son dos de los artefactos más importantes en el proceso de desarrollo de software.
- En algunas metodologías se proponen los diagramas de secuencia cómo artefactos que brindan información importante para el producto.
- Durante el ciclo de vida de una aplicación, muchos usuarios con distintos niveles de formación técnica necesitan interactuar y comunicar para lograr cumplir el objetivo común de crear una aplicación viable, o sea, que la comunicación con el cliente es vital como plantean las

metodologías ágiles, por tanto es importante proponer los artefactos más asequibles posibles para que los clientes puedan entenderlos con mayor facilidad.

Otro trabajo que sirve como punto de partida para la propuesta que se quiere realizar es el trabajo realizado por el grupo Nacional de Calidad, en el cual se determinan los principales artefactos y entregables para los flujos de trabajo de análisis, diseño e implementación, pero solamente para RUP.

También se tuvo en cuenta la encuesta aplicada, ya que se considera importante la opinión de los que están directamente vinculados con los proyectos productivos, el sentir de los mismos respecto a la documentación en los flujos de trabajo de diseño e implementación y lo que piensan que es realmente importante y necesario documentar, el resultado obtenido está por tanto reflejado de alguna manera en la propuesta.

Para hacer la propuesta se siguieron los siguientes principios:

1. Los artefactos y entregables deberán ser lo más cercano a la metodología RUP que es la más usada actualmente en la universidad.
2. Debe exponer los elementos fundamentales para trasmitir el conocimiento tácito a todos los integrantes del equipo de desarrollo y que no brinde información repetida.
3. Cumplir con la política de calidad de la universidad.
4. Haber desarrollado correctamente el flujo de trabajo de análisis, es decir, que se hayan entregado todos los artefactos requeridos en este flujo.

En la siguiente tabla se muestran los artefactos que conforman la propuesta.

Tabla 1. Propuesta de documentación para los flujos de trabajo de diseño e implementación.

Flujo de trabajo	Artefacto	Entregable si/no
Diseño	Diagrama de clases(paquetes)	si
	Especificación de las clases	si
	Diagrama de interacción (secuencia)	si
	Diseño de la base de datos (lógico)	si
	Documento de arquitectura (actualizado)	si
	Registro de alternativas importantes	si

Implementación	Diagrama de componentes	si
	Documento de arquitectura (actualizado)	si
	Registro de alternativas importantes (actualizado)	si

2.5.1 Artefacto Diagrama de clases

El diagrama de clases es un grafo en el que se relacionan todos los elementos modelados mediante la estructura de clases. Aparecen todas las características, estructura y comportamiento de cada clase, así como restricciones oportunas. Modela, de este modo, las diferentes funcionalidades de la aplicación que le corresponden a cada subsistema, así como las interrelaciones entre ellos. Además ayuda a descomponer la complejidad estructural de la aplicación. (NUÑO 2003)

Los diagramas de clase son el pilar básico del modelado con UML, un diagrama de clases esta compuesto por los siguientes elementos:

- Clase: atributos, métodos y visibilidad.
- Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

Estos elementos son los que nos indican que se debe programar y como hacerlo.(GRAU 2004)

Se dice que los diagramas de clases son diagramas “estáticos” porque muestran las clases, junto con sus métodos y atributos, así como las relaciones estáticas entre ellas: qué clases “conocen” a qué otras clases o qué clases “son parte” de otras clases, pero no muestran los métodos mediante los que se invocan entre ellas.*(SÁNCHEZ, ISABEL 2000)*

Se recomienda que en el estereotipo de las clases no se especifiquen los atributos y las operaciones, ya que se considera que la ventaja fundamental que nos brinda visualizar este diagrama es identificar las relaciones entre las clases *(Ver Figura 10)*. Se considera que la información referente a los atributos y las operaciones pueden observarse en el artefacto que se propone a continuación que es el de especificación de cada clase, el cual sirve de soporte a este diagrama y explica con un mayor detalle los componentes de cada clase. De esta manera se lograría un diagrama más simple, el cual se podrá entender mejor.

Cuando se relacionen varias clases será de mucha ayuda, pues se obtendrá un diagrama más pequeño, para contribuir a reducir el tamaño de los diagramas, se recomienda además el uso de paquetes.

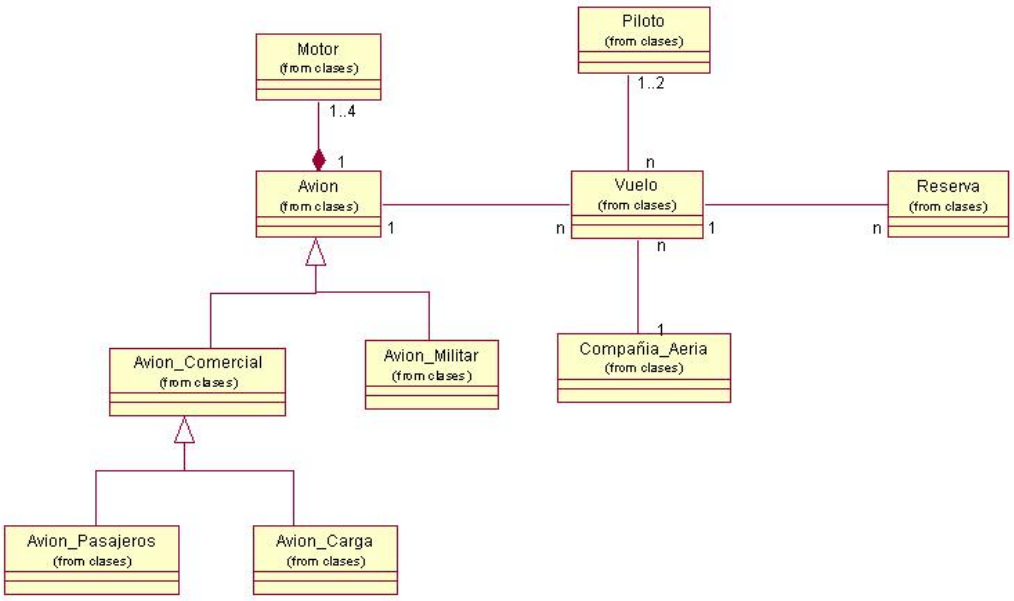

Figura 10. Diagrama de clases.

El diagrama de clases es el diagrama principal para el análisis y diseño de los sistemas informáticos. Durante el análisis del sistema, el diagrama se desarrolla buscando una solución ideal. Durante el diseño, se usa el mismo diagrama, y se modifica para satisfacer los detalles de las implementaciones.(WATSON 2007)

En el Diagrama de Clases de Diseño se añaden los detalles referentes al lenguaje de programación que se vaya a usar. Por ejemplo, los tipos de los atributos y parámetros se expresarán en el lenguaje de implementación escogido.(GRAU 2004)

Podemos concluir entonces con que los diagramas de clases del diseño son sumamente importantes en el proceso de desarrollo del software ya que muestran la estructura general del sistema, además constituyen otra vista más del código y están directamente asociados, o sea, que ayudan a entender y organizar lo que será programado. Tienen importancia además por los grandes beneficios que brinda al proyecto, ya que ayudan a entender la estructura de clases de los proyectos desarrollados por otras personas (o

desarrollados por el usuario hace mucho tiempo). Pueden ser utilizados para adaptar y presentar la información del proyecto, o compartirla con otros usuarios. Por lo que se considera uno de los artefactos necesarios a elaborar durante el flujo de trabajo de diseño.

2.5.2 Artefacto Especificación de cada clase

La especificación de la clase es usado para capturar toda la información importante acerca de una clase, es decir se describe la clase explicando todos sus elementos, el nombre, el tipo de clase, los atributos con el tipo de valor que guardan y las responsabilidades de cada clase, o sea, las operaciones, detallando su nombre así cómo una breve descripción de la misma, que permitirá entender claramente de que está encargada cada clase de forma sencilla y legible.

Se recomienda especificar además si la clase es persistente o no, por la importancia que reviste esta información, ya que una clase persistente equivale a una tabla en la base de datos. Esta información ayudará a la creación de otro artefacto que conforma la propuesta que es el diseño lógico de la base de datos, partiendo de aquí se minimiza el trabajo ya que el diseñador se pude centrar en esta información para la elaboración del diagrama. Las clases persistentes por lo general tienen como origen las clases clasificadas como entidad porque ellas modelan la información y el comportamiento asociado de algún fenómeno o concepto, como una persona, un objeto del mundo real o un suceso. Especificar la persistencia en este artefacto evita la elaboración del diagrama de clases persistentes.

La especificación de la clase es importante documentarla, partiendo de que el diagrama de clases propuesto no contiene todos los componentes que conforman a las clases, por tanto, si no se elabora este artefacto se pierden elementos fundamentales que le confieren la gran importancia que tiene la realización del diagrama de clases, por tanto es obligatoria su realización para completar el propósito de los diagramas de clases.

Se propone que para la descripción de las clases se use la siguiente tabla:

Tabla 2: Especificación de la clase.

Nombre: (1)	
Tipo de clase: (Interfaz, Controladora, Entidad, etc.)	
Persistencia: (Si o No)	
Atributo	**Tipo**

(2)	(3)
Para cada responsabilidad:	
Nombre:	(4)
Descripción:	(5)

Leyenda:

(1) Nombre de la clase

(2) Nombre de cada uno de los atributos

(3) Tipo de dato de cada uno de los atributos.

(4) Nombre de la responsabilidad (operación) con los parámetros que requiera

(5) Breve explicación de en qué consiste la responsabilidad.

Como se dijo anteriormente este artefacto servirá de apoyo al diagrama de clases, la posibilidad de especificar clases ayuda a entender mejor los elementos y responsabilidades de cada clase, pues se puede explicar en el nivel de detalle que se desee empleando un lenguaje sencillo, de ahí la importancia de elaborarlo correctamente, pues es un complemento del diagrama de clases propuesto, por tanto se considera un artefacto imprescindible para el flujo de trabajo de diseño.

2.5.3 Artefacto Diagrama de Interacción

Los diagramas de interacción se utilizan para modelar los aspectos dinámicos de un sistema, lo que conlleva modelar instancias concretas o prototípicas de clases interfaces, componentes y nodos, junto con los mensajes enviados entre ellos, todo en el contexto de un escenario que ilustra un comportamiento.(VILAS 2001)

Estos diagramas constituyen un medio para verificar la coherencia del sistema mediante la validación con el modelo de clases. Por tanto si no se elaboran, es muy difícil entender las distintas acciones del sistema y las clases que están involucradas en dichas acciones, lo que es relevante para comprender el sistema.

Existen dos tipos de diagramas de interacción:

- De **Secuencia**: muestran de forma explícita la secuencia de los mensajes intercambiados por los objetos.

- De **Colaboración**: muestran de forma más clara cómo colaboran los objetos, es decir, con qué otros objetos tiene vínculos o intercambia mensajes un determinado objeto. (SÁNCHEZ, DIANA MARCELA 2007)

Un diagrama de colaboración es una forma alternativa al diagrama de secuencia de mostrar un escenario. Este tipo de diagrama muestra las interacciones entre objetos organizadas entorno a los objetos y los enlaces entre ellos.

Los diagramas de secuencia proporcionan una forma de ver el escenario en un orden temporal - qué pasa primero, qué pasa después -. Los clientes entienden fácilmente este tipo de diagramas.(WATSON 2007)

Se considera necesario elaborar solamente uno de los dos diagramas de interacción, puesto que ambos brindan de forma similar el intercambio de información entre objetos en un escenario determinado, además existen herramientas que permiten generar a partir de uno de los diagramas el otro. Se sugiere la elaboración del diagrama de secuencia porque muestran de forma sencilla el orden en que ocurren las acciones en los diferentes casos de uso. Cuando se representan escenarios o casos de uso extensos, el diagrama de colaboración se vuelve un poco engorroso de elaborar y una vez elaborado difícil de entender y seguir la traza de sucesión de los eventos, por lo que representar todas las colaboraciones entre objetos en un solo diagrama puede confundir más que aclarar, en cambio con el diagrama de secuencia la representación se hace de forma cronológica, ya que en este diagrama se pueden organizar los objetos en el orden deseado de forma que sea más fácil y adecuada la transición de mensajes mientras que el de colaboración mantiene la misma estructura que el diagrama de clases y por tanto los mensajes no siguen una secuencia ordenada, el diagrama de secuencia cuenta además con otros elementos que ayudan a la comprensión de determinados sucesos, por ejemplo cuando se elimina un objeto, y apreciar detalles como el tiempo de vida de los mismos, en cambio en los diagramas de colaboración la creación y eliminación de objetos no tiene una representación gráfica. Por tanto aunque pueda existir preferencia o familiarización con uno u otro diagrama, se considera que el diagrama de secuencia es más atractivo para su elaboración y comprensión. (*Ver Figura 11*)

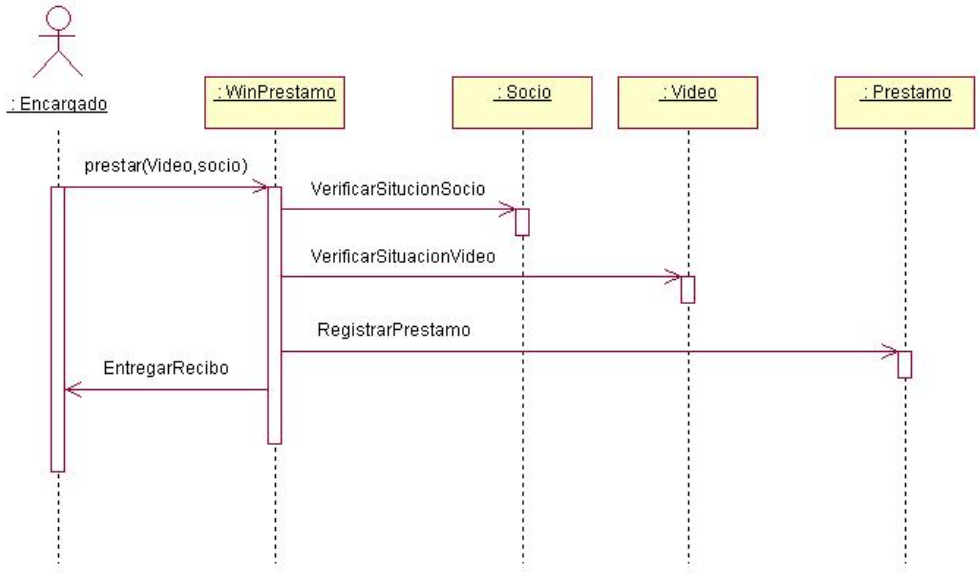

Figura 11. Diagrama de secuencia.

El diagrama de secuencia de un sistema es una representación que muestra, en determinado escenario de un caso de uso, los eventos generados por actores externos, su orden y los eventos internos del sistema. En esta fase del proyecto, el sistema mismo es una caja negra. Son diagramas de interacción cuyo objetivo es describir el comportamiento dinámico del sistema.

Este diagrama muestra gráficamente los eventos que originan los actores y que impactan al sistema. En él se representan las interacciones entre objetos ordenadas en una serie temporal que muestra su ciclo de vida. Muestra los objetos que se encuentran en el escenario y la secuencia de mensajes intercambiados entre los objetos para llevar a cabo la funcionalidad descrita por el escenario.

Este diagrama muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, el eje de tiempo también es vertical, incrementándose hacia abajo, de forma que los mensajes son enviados de un objeto a otro en forma de flechas con los nombres de la operación y los parámetros y los mensajes pasados entre los objetos se representan como vectores horizontales. Los mensajes se dibujan cronológicamente desde la parte superior del diagrama a la parte inferior; la distribución horizontal de los objetos es arbitraria.(HENSGEN 2003)

Los diagramas de secuencia fueron inicialmente pensados para refinar los requisitos de usuario, y profundizar en el conocimiento del sistema a construir, se centra a menudo, en intentar comprender un caso de uso describiendo ejemplos, proporcionando una vista detallada del mismo. Muestran una interacción organizada en una secuencia de tiempo y ayuda a documentar el flujo de lógica dentro de la aplicación. Los participantes se muestran en el contexto de los mensajes que se transfieren entre ellos. En un sistema de software amplio, el diagrama de secuencia puede incluir un mayor número de detalles y contener miles de mensajes.

El diagrama de secuencia es uno de los diagramas más efectivos para modelar interacción entre objetos en un sistema. Un diagrama de secuencia se modela para cada caso de uso. Mientras que el diagrama de caso de uso permite el modelado de una vista *"business"* del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes pasados entre los objetos.

Típicamente se examina la descripción de un caso de uso para determinar qué objetos son necesarios para la implementación del escenario. Si se tiene modelada la descripción de cada caso de uso como una secuencia de varios pasos, entonces se puede "caminar sobre" esos pasos para descubrir qué objetos son necesarios para que se puedan seguir los pasos.(WATSON 2002)

Los diagramas de secuencia proporcionados por UML constituyen una herramienta de modelado muy potente que no se limita a describir el paso de mensajes entre participantes en un sistema, sino que incluyen una abundante y variada información, que ayudan a entender mejor la funcionalidad del sistema.

La importancia de los diagramas de secuencia radica en que facilitan tanto la descripción, como la comprensión intuitiva de las especificaciones necesarias que el sistema debe realizar. Una de las ventajas de los diagramas de secuencia es su posibilidad de representar los mensajes en función del tiempo, lo que ayuda a comprender con mas precisión y cercano a la realidad las funcionalidades descritas. Por tanto son necesarios de elaborar ya que mediante estos diagramas se realiza una parte de la descripción del comportamiento del sistema, por tanto se considera un artefacto importante a realizar en el flujo de trabajo de diseño.

2.5.4 Diseño de la Base de Datos

La mayoría de las aplicaciones de software que se desarrollan en el mundo requieren del almacenamiento y gestión de grandes volúmenes de información. Con el auge del paradigma de la orientación a objetos, este proceso ha tomado nuevas dimensiones.(ANAISA 2004)

Cualquier aplicación que se desarrolle, con toda seguridad, necesitará estructuras de datos y con esto el empleo de una Base de Datos para almacenar la información. Se puede decir que los sistemas de gestión poseen al menos una base de datos relacionada ya que estos se diseñan para sustituir uno o varios procedimientos, tanto comerciales como administrativos, en los cuales se maneja gran cantidad de información que debe ser guardada, modificada y actualizada en un tiempo determinado, por lo que aunque al crear una aplicación la misma use alguna base de datos de otro sistema, el hecho de que exista la necesidad de elaborar el sistema en sí, implica la creación de una nueva base de datos propia de ese sistema que gestione la información.

Para la construcción de la Base de Datos es fundamental contar con un diagrama que sirva de soporte a la misma facilitando la representación geográfica de los datos, su importancia aumenta cuando se tiene que elaborar un sistema con un gran número de tablas relacionadas, de no tener un diagrama que represente visualmente la estructura y las relaciones entre tablas, para los ingenieros sería prácticamente imposible la construcción de la base de datos, pero además, un mal diseño de la base de datos muy probablemente traerá problemas al añadir, actualizar o eliminar los datos a las tablas, perjudicando de esta manera el correcto funcionamiento de la base de datos.

Es importante dedicar el tiempo preciso al buen diseño de la base de datos para obtener el mejor esquema que sea posible, de lo contrario, más adelante habrá que invertir mucho más tiempo en encontrar los errores y arreglar las inconsistencias, conllevando a retocar la misma siempre que dé problemas. Por tanto resulta esencial mantener organizados los datos para que se puedan utilizar con más eficacia y diseñar la base de datos para que almacene todos los datos relevantes y proporcione un acceso rápido a los mismos.

El diseño de bases de datos se descompone en tres etapas: diseño conceptual, diseño lógico y diseño físico.

Un diseño conceptual es un conjunto de conceptos que permiten describir la realidad mediante representaciones lingüísticas y gráficas. Es el proceso por el cual se construye un modelo de la información que se utiliza en una empresa u organización, dando una visión general del negocio, independientemente del Sistema de Gestión de Base de Datos (SGBD) que se vaya a utilizar para implementar el sistema y de los equipos informáticos o cualquier otra consideración física, se considera muy general y abstracto.(CHAVEZ 1996)

Cuando el diseño conceptual es transformado desde un modelo de datos de alto nivel a su implementación, es llamado modelo lógico. Este constituye una versión completa que incluye todos los detalles acerca de los datos.

El diseño lógico de una base de datos muestra cómo estructurar y ordenar los datos para cumplir con las necesidades de información de la organización. El diseño lógico de una base de datos incluye la identificación de las relaciones entre las diferentes sesiones de datos y su agrupamiento en una forma ordenada.(GONZÁLEZ 1997) (*Ver Figura 12*)

El diseño lógico es por tanto, el proceso de construir un esquema de la información que se utiliza en la aplicación, basándose en un modelo de base de datos específico, independiente del SGBD concreto que se vaya a utilizar y de cualquier otra consideración física.

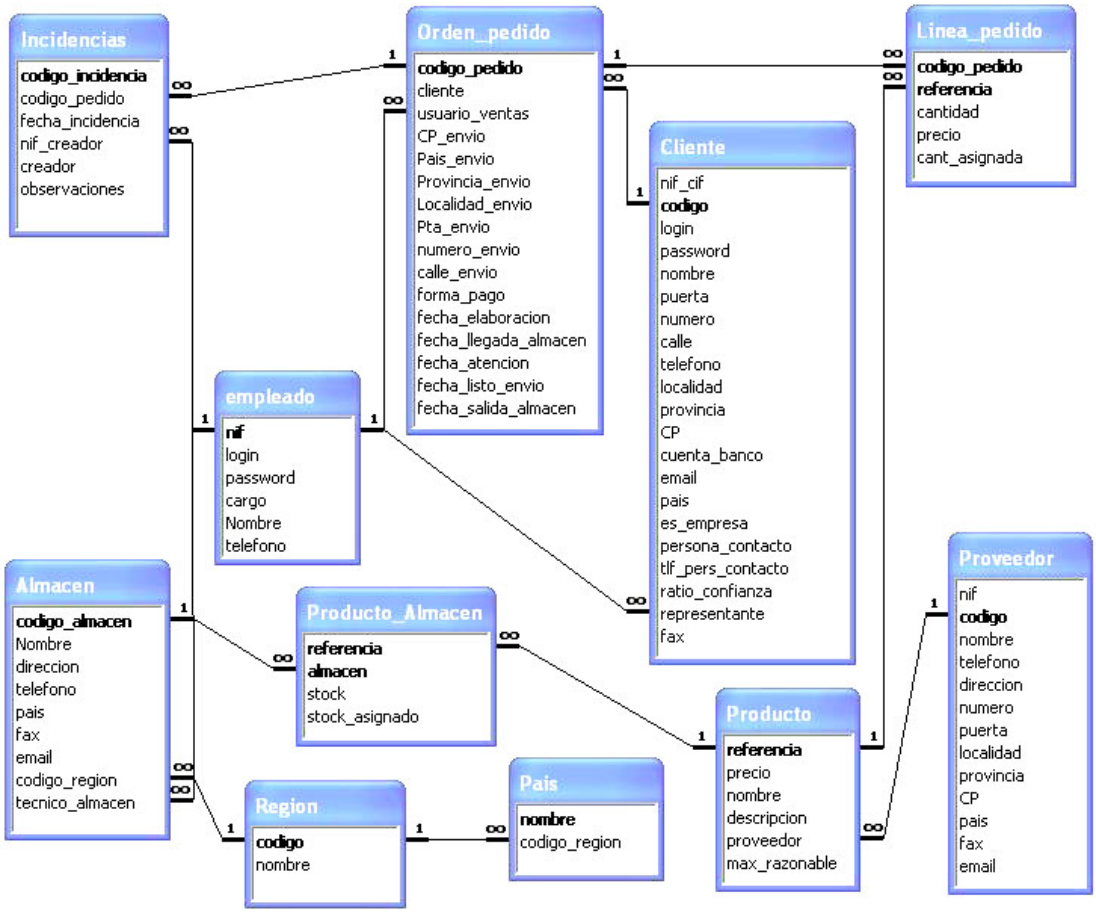

Figura 12. Modelo Lógico de la Base de Datos.

La principal ventaja que nos ofrece un diagrama resultante del diseño lógico, es la comodidad de poder realizar y estudiar las relaciones entre tablas de un modo visual, lo cual ofrece una información muy clara de cual es la estructura de las relaciones creadas. De este modo, podemos identificar rápidamente el tipo de relación existente entre dos tablas.(DELFINO *et al.* 2005)

El esquema lógico es una fuente de información para el diseño físico, o sea, partiendo del esquema lógico global obtenido durante el diseño lógico, se obtiene una descripción de la implementación de la base de datos.(QUIROZ 2003)

Mientras que en el diseño lógico se especifica qué se guarda, en el diseño físico se especifica cómo se guarda. Para ello, el diseñador debe conocer muy bien toda la funcionalidad del SGBD concreto que se vaya a utilizar y también el sistema informático sobre el que éste va a trabajar. (URALES 2004)

Se considera que se debe documentar sólo uno de los diagramas que se obtienen de cada una de las etapas del diseño descritas anteriormente. Considerando que los diseñadores de base de datos tienen el conocimiento y la habilidad de elaborar todos los diagramas, y que es posible además obtenerlos sin tener que seguir las etapas en la secuencia que está definida. Por tanto, se propone documentar el que representa el diseño lógico de la base de datos, teniendo en cuenta que se puede elaborar directamente sin necesidad de apoyarse en la etapa de diseño que lo precede, el diseño conceptual es un nivel mas abstracto y no muestra como tal las tablas que conformaran la base de datos, por tanto el diseño lógico es más completo, ya que recoge de una forma u otra lo mostrado en el diagrama que representa a la etapa anterior y se agregan otros elementos, esto se debe a que cada diseño antecesor sirve de fuente de información al siguiente hasta lograr un diseño lo más entendible y detallado posible.

Como se dijo anteriormente en el artefacto de especificación de la clase se incluye la clasificación de persistencia, lo cual ayuda a la elaboración de este diagrama al aportar las clases que deben persistir en la base de datos, pero no se considera suficiente esta información por si sola, ya que es necesario que se muestre de forma visual esta información, por la ventaja que poseen los esquemas de ser más representativos y fáciles de entender, el diseño lógico es una representación gráfica que permite mostrar los elementos que se ajusten más a lo que se necesita representar, además contiene todas las tablas que realmente conformarán la base de datos.

Para construir el sistema real e implementar la base de datos lógicamente el diseño que se utiliza es el físico ya que contiene todos los detalles necesarios. Sin embargo dar un paso atrás y comenzar con diseñar un modelo lógico es más adecuado ya que este describe el modelo de datos del sistema, por tanto, es fundamental partir del diseño lógico. Se considera entonces, que la elaboración del diseño lógico es idónea, teniendo en cuenta que a partir del mismo se puede generar un diseño físico eficiente.

El diseño propuesto es suficiente para que se entienda lo que es la información relacionada con la base de datos y la estructura de la misma, ya que este juega un papel importante durante la etapa de mantenimiento del sistema, permitiendo que los futuros cambios que se realicen sobre los programas de aplicación o sobre los datos, se representen correctamente en la base de datos.

El diseño lógico proporciona una visión de alto nivel del sistema que se va a construir, de forma simbólica, por tanto es de gran ayuda para su comprensión y asimilación por parte de los ingenieros y los usuarios, o

sea, de ser necesario en algún momento consultar la documentación para posibles mejoras o para entender la aplicación por parte de nuevos desarrolladores, es viable apoyarse en este diagrama,

Sin embargo, el diseño físico está atado a la tecnología predominante en un instante de tiempo; por lo que un cambio en la tecnología significa un cambio en el diseño físico, esto implicaría la realización de un diseño prácticamente nuevo, para lo cual sirve grandemente tener el diseño lógico relacionado. Por el contrario, el diseño lógico a partir del cual debería haber surgido el diseño físico se mantiene, en un caso ideal, prácticamente intacto. Además, un diseño físico es un modelo a un nivel demasiado bajo, en el sentido de que es mucho más complicado encontrar los errores conceptuales (o de alto nivel) que se producen inherentemente durante el proceso de diseño. Por tanto es más práctico entonces documentar como se dijo anteriormente el diagrama resultante del diseño lógico.

Basándose en los aspectos antes expuesto, se puede concluir que una de las tareas fundamentales que debe realizarse en el proceso de desarrollo del software, es la definición y el diseño de un esquema de la base de datos, específicamente se recomendó el diseño lógico por mostrar de forma apropiada la información necesaria para entender fácilmente la estructura de la base de datos, prescindir de este diagrama puede provocar la obtención de una base de datos inconsistente, por lo que se considera uno de los artefactos imprescindibles a elaborar como parte de la documentación a entregar en el flujo de trabajo de diseño.

2.5.5 Artefacto Documento de arquitectura

Uno de los principios de las metodologías modernas de desarrollo de software es priorizar la definición, el diseño, la implementación y la evaluación de la arquitectura del software, que es como se conoce al esqueleto de soporte del sistema.

La arquitectura implementa los requisitos más críticos a través de las estructuras de programa de mayor alcance en el sistema. Por ello la arquitectura encierra los mayores riesgos del desarrollo. La clave del éxito y, a su vez, la dificultad del principio de priorizar la arquitectura consiste en definir qué es y qué no es la arquitectura. Sus componentes deben ser los suficientes para garantizar la viabilidad del proyecto y, a su vez, los mínimos que permitan dar tales garantías con el mínimo gasto de tiempo, esfuerzo y recursos en general.(MOLLINEDA 2005)

Las buenas prácticas de la arquitectura de software aplicadas durante el ciclo de vida, tienen el potencial de incrementar la facilidad para entender un sistema de software y de desarrollar procesos para crearlo.(TAYLOR 2004)

Si una arquitectura de software se encuentra deficiente en su concepto o diseño, o en el peor de lo casos, no se cuenta con la del sistema que se desarrolla, se tendrá grandes posibilidades de construir un sistema que no alcanzará el total de los requerimientos establecidos. Esto, indudablemente, generará un re-trabajo complicado o, peor aún, conllevará al fracaso del sistema cuando se encuentre en operación.

Por tanto el desarrollo de la arquitectura de software es una de las etapas fundamentales y, en muchos casos, la más importante en el desarrollo de software, pues es aquí donde los profesionales aportan todos sus conocimientos, creatividad y experiencia para crear la mejor propuesta de solución que se dará al cliente que cumpla con los requerimientos funcionales y no funcionales establecidos para el sistema en desarrollo, así como sus preocupaciones principales de lo que esperan del sistema.(SUÁREZ 2006)

Si importante es el desarrollo de la arquitectura de software dentro del proceso de desarrollo, importante es por tanto el documento de arquitectura que se crea para describir la misma con todos sus elementos y especificaciones.

La importancia de representar de forma explícita la arquitectura de los sistemas de software es evidente. En primer lugar, estas representaciones elevan el nivel de abstracción, facilitando la comprensión de los sistemas de software complejos. En segundo lugar, hacen que aumenten las posibilidades de reutilizar tanto la arquitectura como los componentes que aparecen en ella. Por último, si la notación utilizada tiene una base formal adecuada, es posible analizar la arquitectura del sistema, determinando cuáles son sus propiedades aún antes de construirlo.(VELASCO 2000)

Toda documentación relativa a la arquitectura es de gran utilidad, e imprescindible en el desarrollo del software. Es necesario que un nuevo desarrollador o aquel que resucita el proyecto tras un tiempo, pueda comprender que decisiones de alto nivel guiaron el desarrollo. Esta documentación si que es valiosa puesto que sirve para comprender por qué se tomaron determinadas decisiones que no se cambian con facilidad a lo largo del proyecto y proporciona una primera aproximación que siempre es necesaria. La arquitectura de una aplicación no suele cambiar a menudo, lo que cambia es la implementación, el código. Documentar la arquitectura y que alguien la comprenda es algo muy importante. Esta documentación es simple de mantener porque es mucho más estática que la de diseño detallado. Además algo

especialmente valioso de la documentación de arquitectura es que establece el lenguaje común, la nomenclatura propia y el estilo que va a guiar del desarrollo.

El documento de arquitectura es una especificación de las ideas principales del diseño. Proporciona una descripción entendible de la arquitectura del sistema software y sirve como medio de comunicación entre el arquitecto de software y otros miembros del equipo del proyecto, acerca de las decisiones significativas que han sido tomadas para la arquitectura. Contiene varias vistas que muestran aspectos distintos del sistema y que brindan un resumen de la arquitectura. (MOLLINEDA 2005)

Es importante comprender que para la arquitectura existen varias perspectivas y que dentro de estas se pueden mostrar vistas dependiendo de los puntos de vista interesados en la misma y que al diseñar una solución se debe pensar en ellos y dejar plasmado en algún documento las decisiones arquitectónicas que se toman.(MERINDE 2007)

Las vistas arquitectónicas pueden representarse mediante lenguajes de modelado, como UML, aunque también existen lenguajes especializados de descripción arquitectónica (ADLs), para especificar de manera sintáctica y gráfica los componentes de una arquitectura de software.

La definición de las vistas de la arquitectura de software debe documentarse de manera completa, incluyendo toda la explicación de su diseño, es decir, lo que se ha representado gráficamente, así como las justificaciones de por qué se llegó, fue mejor o se omitieron partes de la propuesta de solución. De la misma manera que existe un documento de especificación de requerimientos de software, se debe crear un documento de la arquitectura de software del sistema deseado, que servirá para generar el diseño detallado de dicho sistema.(SUÁREZ 2006)

Desde los inicios de la construcción del software se comienza a describir su arquitectura, en cada flujo de trabajo se trata de mantener la integridad de la misma y agregar sólo los detalles que son nuevos o que han cambiado respecto a los otros flujos, por tanto el documento de arquitectura es un artefacto que se va actualizando. La propuesta se refiere a una actualización del documento de arquitectura en el flujo de diseño y a otra en el flujo de implementación, que contenga toda la información relevante relativa a estos flujos, que son los de interés y análisis en la investigación.

Este contexto de información se representa a través de las vistas correspondientes. Se comenzará por la vista de diseño y se continuará con la vista de implementación, estas vistas son importantes y su realización se considera obligatoria.

Dentro de la vista de diseño se considera que se debe de manera general:

- Detallar las partes del modelo de diseño que son significativas arquitectónicamente, como son su descomposición dentro de subsistemas y paquetes de servicios; y debe presentarse para cada paquete, su descomposición dentro de clases y ventajas de las clases.
- Incluir los paquetes de diseño significativos arquitectónicamente. Esta sección debe contener una sub-sección con el nombre del paquete, una breve descripción y un diagrama con todas las clases y paquetes significativos, contenidos dentro del paquete.
- Introducir las clases significativas arquitectónicamente, para cada clase significativa en el paquete se debe reflejar su nombre, una breve descripción y opcionalmente alguna descripción de sus principales responsabilidades, relaciones, funciones y característica.
- Repartir el procesamiento, para esto se debe describir la descomposición del modelo de diseño en términos de su rango de paquetes y capas. En este apartado incluir algunos diagramas reflejando los paquetes de nivel alto, así como su dependencia y sus capas.
- Describir los patrones y mecanismos de diseños empleados. Este documento debe reunir una explicación sobre los patrones de diseño empleados en el proyecto. También puede contener todos los mecanismos que han sido o podrán ser utilizados para garantizar los procedimientos de persistencia, seguridad, comunicación ínter proceso, etc.
- Realizar los casos de uso, debe ilustrar cómo normalmente el software opera, dando algunos ejemplos de como se llevarían a cabo en términos de clases, algunos casos de uso que se consideran lo más importantes

Dentro de la vista de implementación se considera que se debe de manera general:

- Describir la estructura general del modelo de implementación, se debe representar la descomposición del software en capas y subsistemas, y cualquier otro componente importante para la arquitectura

- Mostrar las dependencias y cómo se implementan los componentes físicos del sistema, agrupándolos en subsistemas organizados en capas y jerarquías.
- Dar una visión general, lo que implica nombrar y definir los contenidos de las distintas capas, las reglas que controlan la inserción dentro de una capa y las restricciones entre capas.
- Reflejar un diagrama de componentes que muestre las relaciones que existen entre capas. Para cada capa se debe incluir una sub-sección que contenga su nombre, su relación con elementos de la vista de diseño, una lista de los subsistemas ubicados en la capa, con nombre, abreviación y una breve descripción y un diagrama de componentes que muestre los subsistemas y sus dependencias importantes.
- Incluir los estándares de codificación que serán usados para mantener homogeneidad en el código y garantizar una mejor comprensión entre los desarrolladores, estos además son básicos para disminuir los esfuerzos en las actividades de mantenimiento, y de revisiones, así como la estimación y métricas de tamaño.

El documento de arquitectura recoge por tanto los aspectos más significativos de la aplicación, resuelve decisiones sobre qué desarrollar y qué reutilizar. Se especifican los casos de uso significativos arquitectónicamente que son los que cubren las principales tareas o funciones que el sistema ha de realizar, dando idea de por donde empezar a trabajar y no olvidar casos de uso que pueden tener algún impacto sobre la arquitectura o afectar a la misma. Si no se realiza este documento se puede correr el riesgo de no tener una arquitectura estable y es elemental tener todos los detalles importantes bien preparados antes de empezar con la construcción. Por lo que se considera este artefacto como uno de los primordiales para desarrollar un software que cumpla con los requisitos establecidos y satisfaga las expectativas del usuario.

Se puede concluir entonces que un aspecto crítico a la hora de desarrollar sistemas de software complejos es el diseño de su arquitectura, es imprescindible que se elabore correctamente, para ello, es necesario hacer explícita dicha arquitectura a través de un documento que contenga todo lo relacionado con la arquitectura del software con el fin de ayudar a todos los involucrados con el desarrollo. Esto se hace comparable con la construcción de un edificio para lo cual primero se crea la estructura o esqueleto del edificio, recogiendo las especificaciones del mismo en el plano de la construcción para luego ensamblar las distintas partes.

2.5.6 Artefacto Diagrama de componentes

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes software para representar la estructura del código. Desde el punto de vista del diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización, y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. Los elementos de modelado dentro de un diagrama de componentes serán componentes y paquetes. (FERNÁNDEZ, ANA 2002)

En este diagrama se pueden manejar paquetes *(Ver Figura 13),* que son contenedores de clases utilizados para mantener el espacio de nombres de clases dividido en compartimentos, de manera que se utilizan para representar subsistemas del sistema en el mundo físico, o sea, representan una división física del sistema. Cada paquete se relaciona con otros a través de dependencias, que se representan con flechas de líneas discontinuas que van del componente dependiente al componente del cual depende. Los paquetes se organizan en una jerarquía de capas donde cada capa tiene una interfaz bien definida. Cada paquete debe tener un diagrama de componentes para representar las clases que contiene internamente, similar a hacer un acercamiento o *"zoom"* al interior de cada uno de los paquetes *(Ver Figura 14)*.

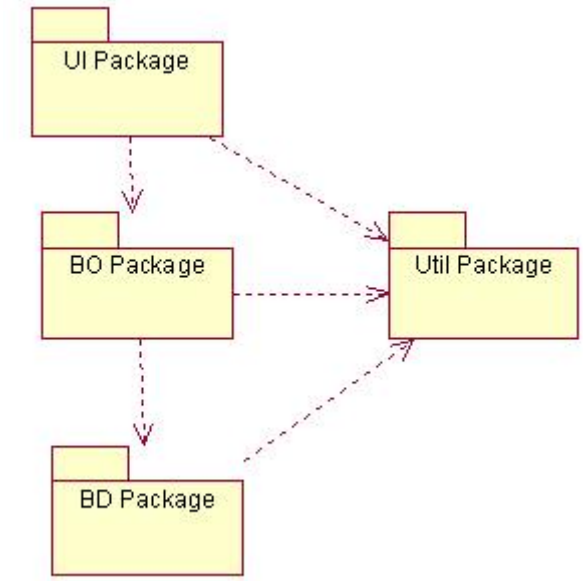

Figura 13. Diagrama de paquetes.

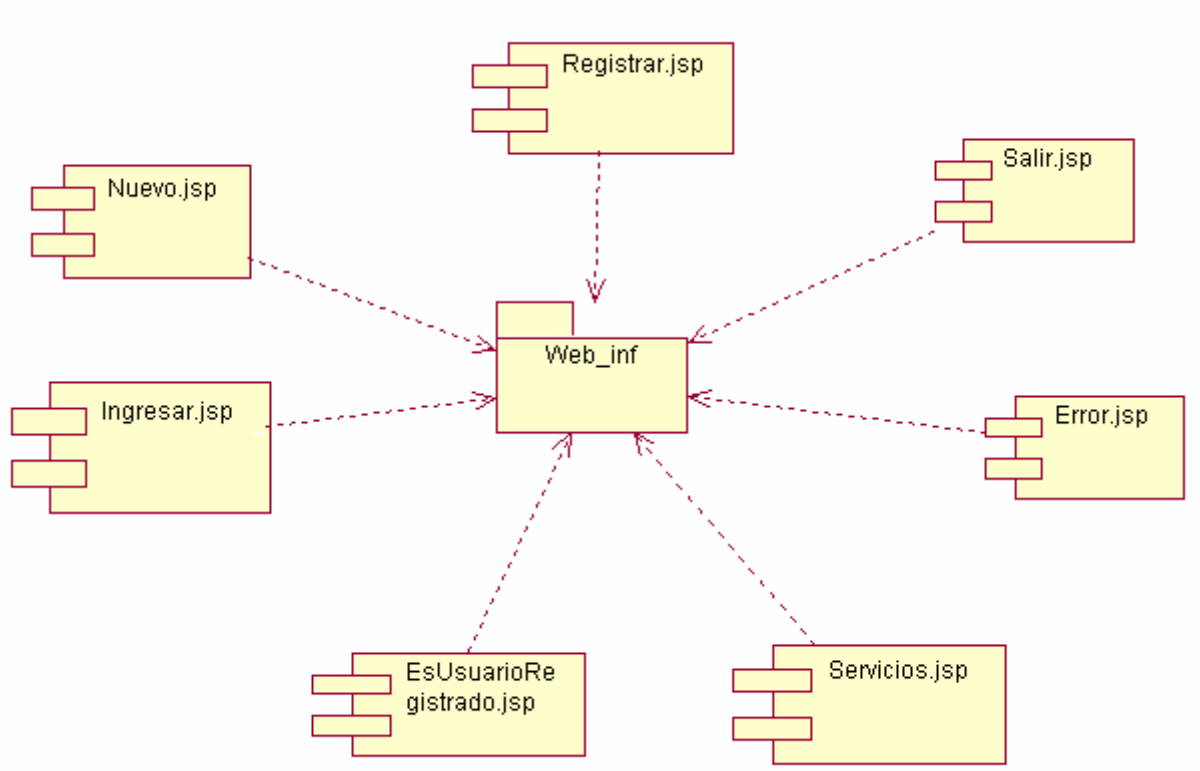

Figura 14. Diagrama de componentes.

Un diagrama de componentes se representa como un grafo de componentes software unidos por medio de relaciones de dependencia, se pueden utilizar estereotipos como <<link>> o <<compile>> para distinguir la distinta naturaleza de las dependencias. Igualmente se pueden definir estereotipos para las distintas clases de componentes. UML proporciona algunos estereotipos predefinidos como: <<file>>, <<library>>, <<executable>>, <<table>> y document>>. Normalmente los diagramas de componentes se utilizan para modelar código fuente, versiones ejecutables, bases de datos físicas, entre otros.

El diagrama de componente hace parte de la vista física de un sistema, la cual modela la estructura de implementación de la aplicación por sí misma y su organización en componentes. Esta vista proporciona la oportunidad de establecer correspondencias entre las clases y los componentes de implementación. La vista de implementación se representa con los diagramas de componentes. (FERNÁNDEZ, ANA 2002)

Se puede concluir entonces que los diagramas de componentes son necesarios de elaborar pues sirven para modelar los aspectos físicos de un sistema, ya que los componentes representan todos los tipos de

elementos software que se usan en la construcción de aplicaciones informáticas, por tanto su importancia radica en que nos muestra la estructura del software de forma organizada con sus especificaciones y requisitos como son la del lenguaje de programación y las herramientas a usar, lo cual ayuda grandemente a la fabricación del sistema en sí, o sea, a su implementación. Se considera entonces por su gran utilidad un artefacto importante a realizar en el proceso de desarrollo de software.

2.5.7 Registro de alternativas importantes

Este es un artefacto nuevo que no se ha propuesto en ninguna metodología estudiada pero que reviste gran importancia durante el desarrollo de un proyecto, pues recogerá las principales decisiones que se tomen en el mismo cuando ante una situación determinada surgen varias alternativas o vías de solución.

Debe documentarse la situación dada, que puede ser un problema o cualquier condición para la cual sea necesaria escoger un camino determinado , las posibles vías de solución encontradas para dicho problema y la vía que se escoge para darle respuesta, incluyendo una breve explicación donde de puntualice el por qué se eligió esa vía y no otra.

Se propone que para el registro de alternativas importantes se use la siguiente tabla:

Tabla 3: Registro de alternativas importantes.

Registro de alternativas	
Problema encontrado	
Posibles soluciones	
Solución escogida y justificación	

2.6 Algunas buenas prácticas a aplicar durante estas etapas

Para un mejor trabajo durante los flujos de trabajo de diseño e implementación, que ayude al equipo de desarrollo y aporte buenos resultados, se considera importante tener en cuenta las siguientes recomendaciones:

1. El uso de las herramientas CASE existentes para elaborar los artefactos recomendados, según las características requeridas por el sistema y la valoración de las mismas.
2. Hacer pruebas al código fuente usando las pruebas de unidad para verificar la interacción de componentes, la integración adecuada de los mismos y que se han implementado correctamente todos los requisitos, para asegurar que los defectos encontrados se corrijan antes de entregar el software al cliente.
3. Recomendable la programación por pares, pues ahorraría tiempo y se contaría con una ayuda para un mejor trabajo.
4. Definir responsabilidades individuales dentro de los desarrolladores y control sistemático de las mismas.
5. Reuniones periódicas de reflexión que permitan crecer, hacerse más eficientes y debatir acerca de los problemas del producto y sus posibles soluciones. Recoger en acta y un resumen en el documento de registros de alternativas.
6. La documentación referente al cliente debe ser entregada conforme se haya acordado al inicio del proyecto (manual de usuario, prototipos, código fuente).
7. El análisis y uso de patrones de diseño.
8. El uso de estándares de codificación.
9. La reutilización de componentes.
10. Hacer revisión del código antes de la primera compilación. Utilizar una lista de comprobación para la revisión de código con ajuste de la misma de forma periódica.
11. Llevar el registro de tiempo y de defectos.
12. Corregir y registrar todos los defectos encontrados.

2.7 Conclusiones

Se describió la situación existente en los proyectos productivos de la Universidad de las Ciencias Informáticas, indicando que existen serias deficiencias. Sin una adecuada y rápida solución no podrán resolverse todos los problemas existentes en el proceso de desarrollo de software, para lograr el avance al que se aspira en la industria de software. Se describió la encuesta como uno de los métodos utilizados para llevar acabo la investigación y se analizaron los resultados obtenidos en la misma. Luego se desarrolló la propuesta de solución, la cual se obtuvo fundamentalmente a partir del análisis de los principales artefactos y entregables que proponen las metodologías de desarrollo de software más usadas a nivel mundial. Se tuvieron en cuenta además, los lineamientos propuestos por el Grupo Nacional de Calidad y los resultados arrojados por la encuesta. Gracias a todo este trabajo se puede contar con una guía que contiene los artefactos imprescindibles para los flujos de trabajo de diseño e implementación.

CAPÍTULO
EVALUACIÓN TÉCNICA DE LA PROPUESTA

3.1 Introducción

En este capítulo se realizará la evaluación técnica de la propuesta descrita en el capítulo anterior. Se usará el método multicriterio para dicha evaluación, el cuál se basa en la evaluación cuantitativa de criterios previamente definidos por parte de expertos en el tema. Por lo que se describirá la forma de aplicar este método y los elementos necesarios para el mismo, posteriormente se presentarán los resultados obtenidos de la evaluación.

3.2 Método para la validación de la propuesta

Para validar técnicamente la propuesta se utilizó el método de experto, que permite tomar decisiones para aceptar o no la propuesta de acuerdo con los criterios definidos.(LEÓN 2002)

Para llevar a cabo el desarrollo del mismo se efectuaron un conjunto de pasos:

1. Se elabora los criterios de evaluación de acuerdo a las características de la propuesta y se organizan por grupos.

   Grupo No. 1: Criterios de mérito científico

   - Valor científico de la propuesta.
   - Calidad de la investigación.
   - Aporte científico.
   - Novedad científica

Grupo No. 2: Criterios de implantación

- Satisfacción de las necesidades de los ingenieros de software.
- Necesidad del empleo de la propuesta.
- Uso del UML.
- Uso de los principios básicos de la ingeniería de software

Grupo No.3: Criterios de flexibilidad

- Adaptabilidad a proyectos productivos independientemente de la metodología a usar.
- Uso de las herramientas necesarias para la elaboración de los artefactos que el documentador considere adecuado o atractivo, o con el cual este familiarizado.

Grupo No.4: Criterios de impacto

- Repercusión en los proyectos productivos.
- Organización en el proceso de documentación del software.
- Posibilidades de aplicación.
- Impacto en el área para la cual está destinada.

2. Se le asigna un peso relativo a cada grupo de criterios de acuerdo al porcentaje que representa cada grupo del total y los intereses a evaluar.

   Grupo No. 1……. 25

   Grupo No. 2……. 30

   Grupo No.3……. 10

   Grupo No.4……...35

3. Se organiza un comité de expertos con una cantidad mínima de 7 teniendo en cuenta su especialidad, grado científico y currículo.
4. Se les entrega a los expertos la propuesta para que estudien el tema a evaluar y dos modelos, uno para que valore el peso relativo de cada criterio *(Ver Anexo 11)* y otro para realizar una evaluación

cuantitativa de cada criterio con una escala de 1-5 y la apreciación cualitativa con una clasificación final del proyecto en excelente, bueno, aceptable, cuestionable y malo. También se da la posibilidad de dar su opinión haciendo una valoración final del proyecto, emitiendo todas aquellas consideraciones que estimaron convenientes *(Ver Anexo 12)*.

5. Después de recibir los valores del peso relativo de cada criterio se construye la Tabla No.1

    - Sea C el número de criterios que van a evaluarse y E el número de expertos que realizan la evaluación.

Tabla No.1Resultado del trabajo de expertos

G	C/E	E1	E2	E3	E4	E5	E6	E7	Ep
25	C1								
	C2								
	C3								
	C4								
30	C5								
	C6								
	C7								
	C8								
10	C9								
	C10								
35	C11								
	C12								
	C13								
	C14								

T									

6. Se verifica la consistencia en el trabajo de los expertos, para lo que se utiliza el coeficiente de concordancia de Kendall y el estadígrafo Chi cuadrado (X^2). Se sigue el procedimiento siguiente:

   - Para cada criterio se determina:

     ∑E: Sumatoria del peso dado por cada experto

     Ep: Puntuación promedio del peso dado por cada experto

     M∑E: media de los ∑E

     ΔC: Diferencia entre ∑E y M∑E

   - Se determina la desviación de la media, que posteriormente se eleva al cuadrado para obtener la dispersión (S) por la expresión

     $S = \sum (\sum E - \sum \sum E / C)^2$

   - Conociendo la dispersión se puede calcular el coeficiente de concordancia de Kendall (W)

     $W = S / E^2 (C^3 – C)/ 12$

   - El coeficiente de concordancia de Kendall permite calcular el Chi cuadrado real

     $X^2 = E (C-1) W$

   - Los valores obtenidos se muestran en la Tabla No.2.

Tabla No.2 Tabla para el cálculo de concordancia de Kendall

Expertos/Criterios	**E_1**	**E_2**	**E_3**	**E_4**	**E_5**	**E_6**	**E_7**	**∑E**	**Ep**	**ΔC**	**ΔC^2**
C_1								0	0	0	0
C_2								0	0	0	0
C_3								0	0	0	0
C_4								0	0	0	0

C_5								0	0	0	0
C_6								0	0	0	0
C_7								0	0	0	0
C_8								0	0	0	0
C_9								0	0	0	0
C_{10}								0	0	0	0
C_{11}								0	0	0	0
C_{12}								0	0	0	0
C_{13}								0	0	0	0
C_{14}								0	0	0	0
DC	0	0	0	0	0	0	0	0	0	0	0
M ∑E	0										
W	0										
X^2	0										

- El Chi cuadrado calculado se compara con el obtenido del las tablas estadísticas
- Si se cumple:

  $X^2_{real} < X^2_{(\alpha, c-1)}$

  Existe concordancia en el trabajo de expertos

7. Si no existe concordancia se hace necesario repetir el trabajo de expertos
8. Después de comprobar la consistencia del trabajo de expertos se puede definir el peso relativo de cada criterio (P).

9. Conociendo el peso de cada criterio y la calificación dada por los evaluadores en una escala de 1-5 se puede construir la Tabla No.3, para obtener el valor de de P × c., donde (c), es el criterio promedio concebido por los expertos.

Tabla No.3 Tabla de calificación de cada criterio

Criterios	**Calificación (c)**					**P**	**P × c**
	1	**2**	**3**	**4**	**5**		
C_1							
C_2							
C_3							
C_4							
C_5							
C_6							
C_7							
C_8							
C_9							
C_{10}							
C_{11}							
C_{12}							
C_{13}							
C_{14}							

10. Se calcula el Índice de aceptación del proyecto (IA).

$$IA = \sum (P \times c) / 5$$

11. Por último se determina la probabilidad de éxito de la propuesta

Rangos predefinidos de Índice de Aceptación.

IA > 0,7 Existe alta probabilidad de éxito

0,7 > IA > 0,5 Existe probabilidad media de éxito

0,5 > IA > 0,3 Probabilidad de éxito baja

0,3 > IA Fracaso seguro

Por lo que la probabilidad de éxito es:

3.3 Análisis de la evaluación técnica de la propuesta

Se utilizaron 7 expertos para que dieran su opinión y valoraran la propuesta. Primeramente los expertos emitieron su juicio para darle peso a cada criterio con la cual se elaboró la tabla de los valores de peso relativo de cada criterio *(Ver anexo # 13)*.

Luego se llevaron los valores de la tabla para el cálculo de concordancia entre los expertos *(Ver anexo # 14)*.

El resultado de los cálculos fueron los siguientes:

X^2 real es 11,1475, para seleccionar el X^2 de la tabla se toma $1-\alpha = 0.99$, dónde α es el error permisible, entonces α= 0.01. Debe cumplirse que $X^2 < X^2(\alpha, c-1)$

De esta forma quedaría:

$11{,}1475 < 27.6882$ por lo que se puede afirmar que existe concordancia entre los expertos, por lo que se puede pasar a construcción de la tabla de clasificación de cada criterio para saber el índice de aceptación de la propuesta *(Ver anexo # 15)*.

Después de tener todos los datos en la tabla se calcula el valor del Índice de Aceptación (IA) que sería:

0,767414, se compara el valor con los valores que aparecen a continuación para saber la valoración de la propuesta.

IA > 0,7 Existe alta probabilidad de éxito

0,7 > IA > 0,5 Existe probabilidad media de éxito

0,5 > IA > 0,3 Probabilidad de éxito baja

0,3 > IA Fracaso seguro

Por lo que la probabilidad de éxito es alta.

3.4 Conclusiones

Se usó el método multicriterio para determinar si la propuesta es viable. Se analizó el resultado de aplicar dicho método, en el cual se obtuvo una probabilidad de éxito alta, indicando que la aplicación de la propuesta proporcionará resultados favorables y que lo planteado hasta el momento brinda un aporte significativo capaz de resolver los problemas existentes por los que se inició la investigación.

CONCLUSIONES

En este trabajo se logró proponer una guía con la documentación imprescindible para los flujos de trabajo de diseño e implementación en el proceso de desarrollo de software de gestión de la UCI.

Se hizo un estudio de los artefactos y entregables definidos en los proyectos de gestión en la universidad para los flujos de interés en la investigación.

Se analizaron las metodologías de desarrollo existentes para software de gestión y la documentación que proponen las mismas, seleccionando los elementos fundamentales de los flujos antes mencionados.

A partir de todo el trabajo realizado, se obtuvo la propuesta, la cual recoge los artefactos y entregables necesarios para la construcción del software y las revisiones futuras durante los flujos en cuestión.

Finalmente se hizo la evaluación técnica de la propuesta, arrojando que la probabilidad de éxito es alta, lo que implica desde el punto de vista teórico, el cumplimiento de la idea a defender planteada.

La definición de la documentación imprescindible que se debe entregar en los flujos de trabajo de diseño e implementación para proyectos que desarrollan software de gestión, permite contar con una guía para futuros proyectos de este tipo, de forma que se trasmita el conocimiento a todo el equipo de desarrollo.

La propuesta es adaptable a todo desarrollo de software, independientemente de la metodología a usar y para la elaboración de los artefactos sugeridos se puede hacer uso de las herramientas que se consideren adecuadas, o con la cual se esté familiarizado. Apoya la necesidad de documentar en paralelo a la construcción del software.

RECOMENDACIONES

Los objetivos planteados en este trabajo se alcanzaron de manera general, pero se considera que para solucionar los problemas con la documentación del software, se necesita de un proceso investigativo mucho más amplio y este trabajo es sólo una parte del mismo, teniendo en cuenta esto se pueden hacer las siguientes recomendaciones:

- Hacer talleres como parte de las actividades de producción de la facultad donde se deje clara la importancia de la documentación durante el desarrollo de software.
- La profundización del estudio sobre los distintos aspectos, características propias y necesidades de los proyectos productivos en nuestra universidad.
- Generalizar la propuesta a los demás flujos de trabajo, con el fin de obtener un modelo completo por el que se rijan todos los proyectos productivos dedicados al desarrollo de software de gestión en la UCI.
- Que la guía propuesta sea tomada en cuenta por parte de las instancias superiores y que se use en los proyectos que desarrollan software de gestión actualmente en la universidad.

REFERENCIAS BIBLIOGRÁFICAS

ACUÑA, G. *CMM & Agile Methods*, 2006. [Disponible en: http://www.exa.unicen.edu.ar/catedras/ingsoft/docs/presentaciones/AGL-Informe.doc

AIRES, U. D. B. *metodologías de desarrollo*, 2005. [Disponible en: http://www.votoelectronico.es/Archivos/PruebaArgentina/Software-metodologia-de-desarrollo.pdf

ANAISA, H. G. *Un Método para el Diseño de la Base de Datos a partir del Modelo Orientado a Objetos*, 2004. [Disponible en: http://www.ejournal.unam.mx/compuysistemas/vol07-04/CYS07402.pdf

ASENSIO, B. *Ingeniería de Software*, 2004. [Disponible en: http://www.wikilearning.com/ciclo_de_desarrollo-wkccp-3616-3.htm

CANSECO, E. *Metodología MSF*, 2004. [Disponible en: http://www.scoutech.com/nota.asp?id=28

CARABALLO, L. A. S. *Prolegómenos Sobre el Lenguaje de Modelado Unificado (UML)*, 2003. [Disponible en: http://www.willydev.net/descargas/prev/Prolego.pdf

CHAVEZ, C. A. G. *Diseño de base de datos relacionales*, 1996. [Disponible en: http://www.mailxmail.com/curso/informatica/disenobasesdatosrelacionales/capitulo7.htm

DELFINO; KOLOCSAR GASTÓN*, et al. Métodos Ágiles: Dynamic System Development Method (DSDM)*, 2005. [Disponible en: http://www.exa.unicen.edu.ar/catedras/agilem/grupo4.pdf

FERNÁNDEZ. *Las metodologías ágiles* 2006. [Disponible en: http://sistemasdecisionales.blogspot.com/2006/06/que-son-eso-de-las-metodologas-giles.html

FERNÁNDEZ, A. *Diagrama de iteracción*, 2002. [Disponible en: http://www-gris.det.uvigo.es/~avilas/UML/node41.html

FOWLER, M. *La Nueva Metodología* 2006. [Disponible en: http://www.programacion.com/tutorial/nuevametodologia/5/

GARCÍA, J. *Gestión de proyectos con Scrum*, 2006. [Disponible en: http://www.ingenierosoftware.com/equipos/scrum.php.

GIL, M. D. P. R. *Ingeniería del Software*, 2006. [Disponible en: http://www.monografias.com/trabajos34/ingenieria-software/ingenieria-software.shtml

GÓMEZ, D. *XP Programación Extrema.*, 2006. [Disponible en: http://www.clubdevelopers.com/index.php?p=38.

GONZÁLEZ, M. *Base de Datos* 1997. [Disponible en: http://www.monografias.com/trabajos27/bases-datos/bases-datos.shtml

GRAU, X. F. *Fase de Construcción Diseño de bajo Nivel*, 2004. [Disponible en: http://www.clikear.com/manuales/uml/faseconstruccionbajonivel.asp

HENSGEN, P. *Elementos de UML*, 2003. [Disponible en: http://docs.kde.org/stable/es/kdesdk/umbrello/uml-elements.html

HERNÁN, S. M. *Diseño de una metodología ágil de desarrollo de software*. facultad de ingeniería. Argentina, Universidad de Buenos Aires, 2004. p.

IVAR JACOBSON; GRADY BOOCH*, et al. El Proceso Unificado de Desarrollo de Software*. 2004. p.

JACOBSON, I. *Applying UML in The Unified Process*, Presentación, 1998. [Disponible en: http://www.rational.com/uml

LEÓN, R. A. H. *El paradigma cuantitativo de la investigación científica*. 2002. p. 959-16-0343-6

LETELIER, P. *Introducción a Rational Unified Process*, 2004. [Disponible en: http://www.dsic.upv.es/~letelier/pub/p16.ppt

LUZ, G. D. *Métodos Ágeis*, 2004. [Disponible en: http://www.ime.usp.br/~gdaltonl/ageis/ageis_6pp.pdf.

MAUSQUES, G. *Metodología ASD*. Facultad de Ingeniería. Uruguay, University ORT, 2003. p.

MEDINA, E. *La necesidad de un sistema de la calidad para prevenir y controlar los problemas del software.*, 2005. [Disponible en: www.procuno.com/users/taller/presentaciones/art%C3%ADculo-INGENIA-taller-universidad.doc

MENDOZA, M. A. *Metodologías De Desarrollo De Software*, 2004. [Disponible en: http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html

MOLLINEDA, R. *Arquitectura del Software*, 2005. [Disponible en: http://www.iti.upv.es/services/reviewtic/public/2005/02/2005-02-AS

MOLPECERES, A. *Procesos de desarrollo: RUP, XP y FDD*, 2002. [Disponible en: http://www.javahispano.org/articles.article.action?id=76

MORALES, A. *Crystal Clear, una forma de trabajo*, 2007. [Disponible en: http://ideas3p.blogspot.com/search/label/Crystal%20Clear

NUÑO, I. C. *El análisis según OMT.*, 2003. [Disponible en: http://pisuerga.inf.ubu.es/icruzado/tfc/OMT_AnaUML_Coinfiti.pdf

OCA, C. M. D. *eXtreme Programming (XP).* , 2006. [Disponible en: http://www.softwareguru.com.mx/downloads/SG-200603-Agil.pdf

ORALLO, E. H. *El Lenguaje Unificado de Modelado (UML)*, 2005. [Disponible en: http://www.disca.upv.es/enheror/pdf/ActaUML.PDF

PALACIO, J. *Compendio de Ingeniería del Software Glosario*, 2006. [Disponible en: http://www.navegapolis.net/files/cis/CIS%20Glosario%20004.pdf

PRESSMAN, R. S. *Ingeniería del software. Un enfoque práctico*. 2002. p.

QUIROZ, J. P. T. *Sistema para el control de inventarios de los laboratorios de ingenierías*, 2003. [Disponible en: http://www.iec.uia.mx/proy/titulacion/proy07/creacion.html

Resolución Económica V Congreso PCC, 1997.

REYNOSO, C. *Métodos Heterodoxos en Desarrollo de Software*, 2004. [Disponible en: http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/heterodox.asp

REYNOX, S. I. *Metodología de Desarrollo de Software (MDS)* 2005. [Disponible en: http://www.reynox.com/sistemas/metodologia.php

RODRÍGUEZ, A. *Programación Extrema - Metodología XP.* , 2006. [Disponible en: http://www.wikilearning.com/programacion_extrema_metodologia_xp-wkccp-20349-1.htm

ROSSI, G. *UML: el lenguaje estándar para el modelado de software*, 2004. [Disponible en: http://www.ati.es/novatica/2004/168/168-4.pdf

SÁNCHEZ, D. M. *Métrica V3-Técnicas*, 2007. [Disponible en: http://www.csi.map.es/csi/metrica3/index.html

SÁNCHEZ, I. *Diagrama de clase*, 2000. [Disponible en: www.mcc.unam.mx/~cursos/Objetos/Cap8/cap8.html

SERVETTO, L. A. C. *Ambiente Integrado de Ingeniería Automática de Sistemas*

1999. [Disponible en: http://www.fi.uba.ar/laboratorios/lsi/p-servetto-proyectodetesis.htm

SUÁREZ, J. D. J. H. *Arquitectura de software: importancia de su ciclo de vida*, 2006. [Disponible en: http://www.enterate.unam.mx/Articulos/2006/febrero/arquitec.htm

TAYLOR, R. N. *Desarrollo Centrado en la Arquitectura: Un acercamiento diferente a la Ingeniería de Software*, 2004. [Disponible en: http://www.acm.org/crossroads/espanol/xrds12-4/arqcentric.html

URALES, J. C. *Diseño de base de datos relacionales*, 2004. [Disponible en: http://www.mailxmail.com/curso/informatica/disenobasesdatosrelacionales/capitulo9.htm

VELASCO, C. C. *Un Lenguaje para la Especificación y Validación de Arquitecturas de Software*, 2000. [Disponible en: http://www.lcc.uma.es/~canal/papers/tesis/index.html

VERA, K. L. *Ingeniería de Software – RUP - UML*, 2006. [Disponible en: http://www.mmug.cl/articulos.php?id=287&tod=1

VILAS, A. F. *Diagramas de Interacción*, 2001. [Disponible en: http://www-gris.det.uvigo.es/~avilas/UML/node41.html

WATSON, M. *Análisis y Diseño con el Diagrama de Clase*, 2007. [Disponible en: http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/multiple-html/x219.html

---. *Modelado de sistemas UML*, 2002. [Disponible en: http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/multiple-html/x194.html

ZABALA, R. *Diseño de un Sistema de Información Geográfica sobre internet*, 2000. [Disponible en: www.LaIngenieríadeSoftware.htm

ANEXOS

Anexo 1. Fases del ciclo de vida de ASD

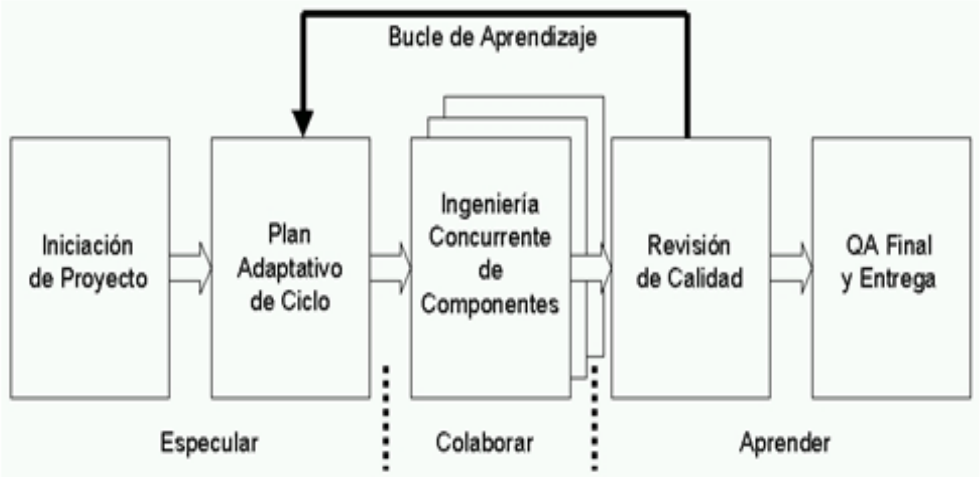

Figura. 1 Fases del ciclo de vida de ASD.

Anexo 2. Ciclo de desarrollo de DSDM

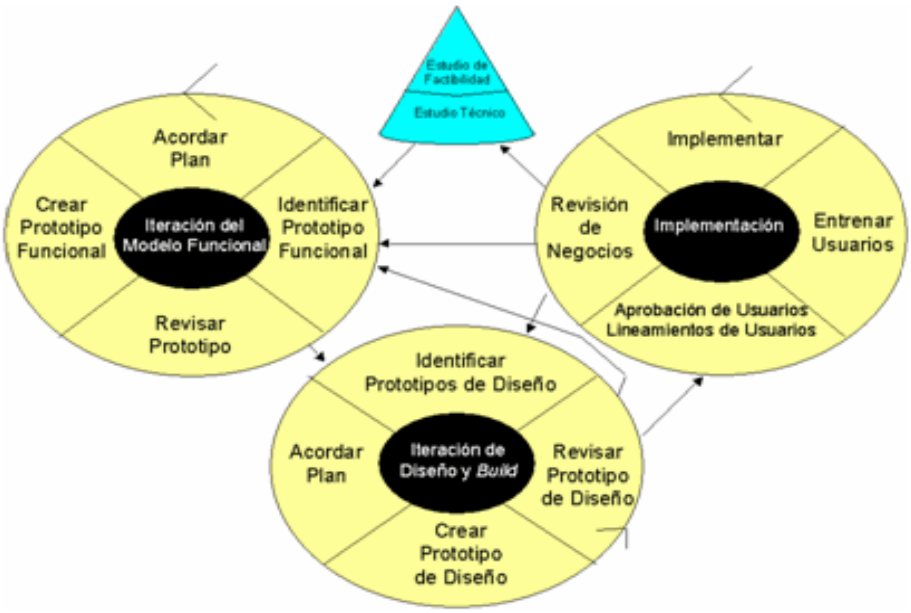

Figura 2. Ciclo de desarrollo de DSDM.

Anexo 3. Ciclo de vida de XP

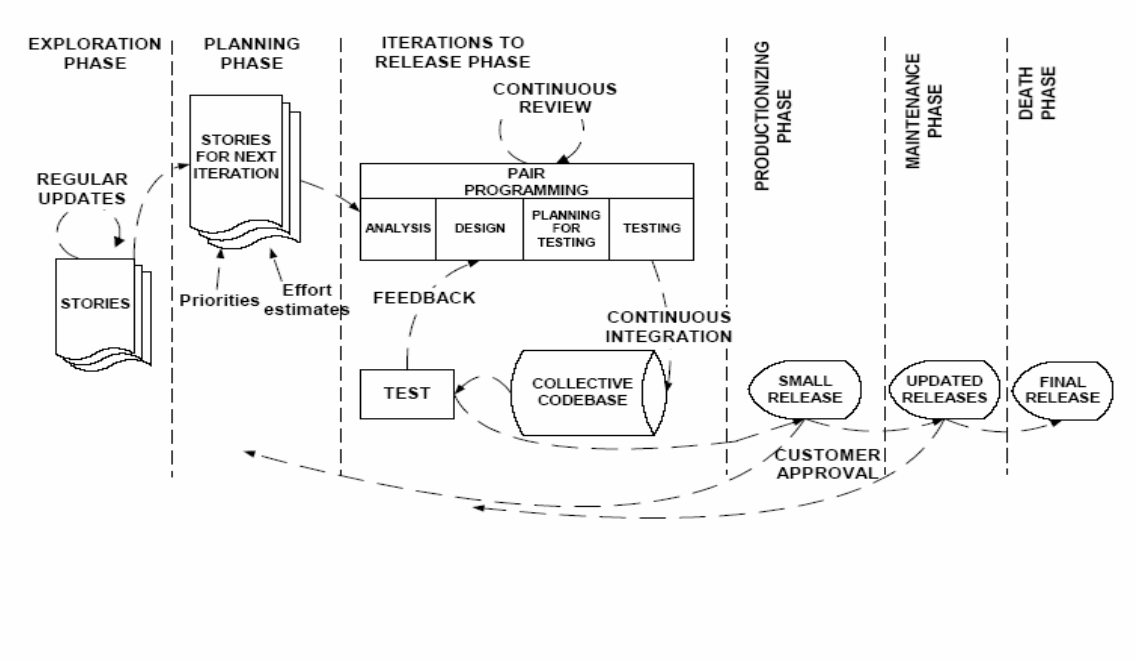

Figura 3. Ciclo de vida de XP.

Anexo 4. Fases del Proceso FDD

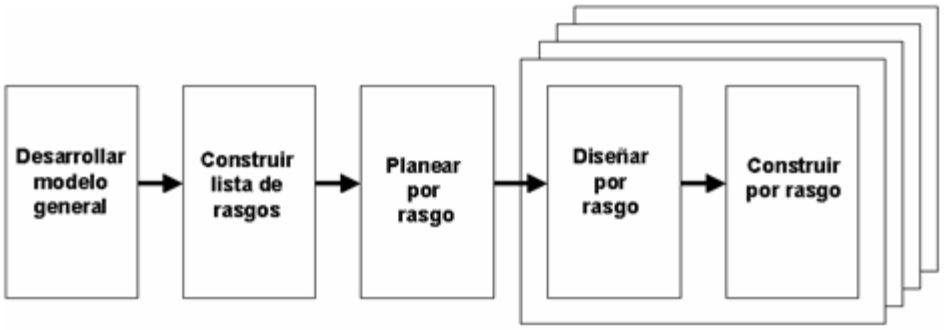

Figura 4. Fases del proceso FDD.

Anexo 5. Ciclo de Scrum

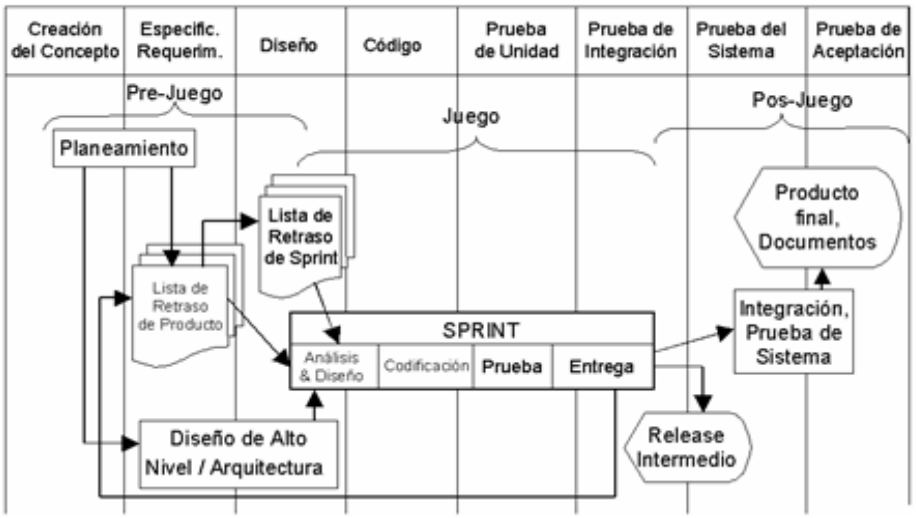

Figura 5. Ciclo de Scrum.

Anexo 6. Fases de Evo

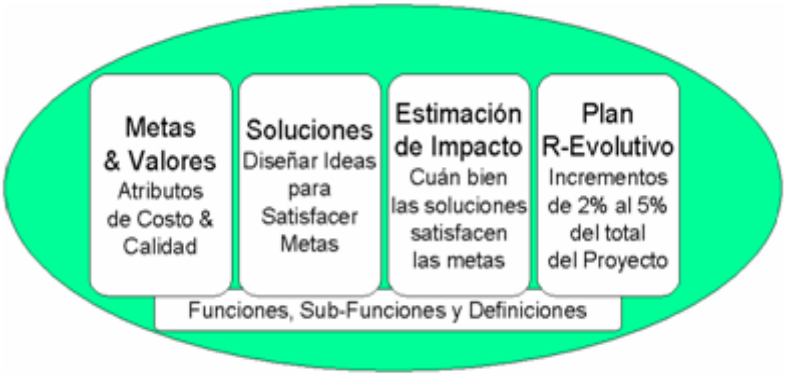

Figura 6. Fases de Evo.

Anexo 7. Ciclos anidados de Crystal Clear

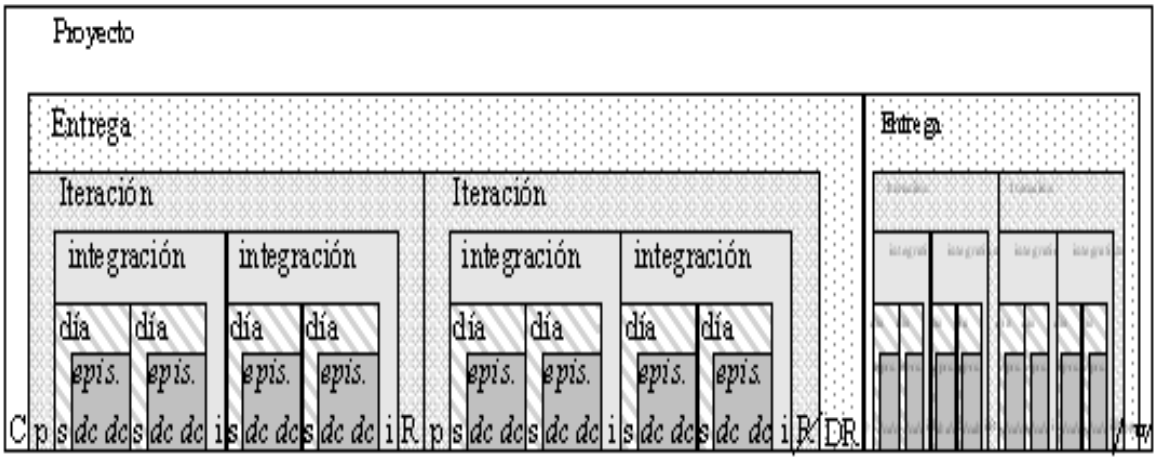

Figura 7. Ciclos anidados de Crystal Clear.

Anexo 8. Fases y flujos de trabajo de RUP

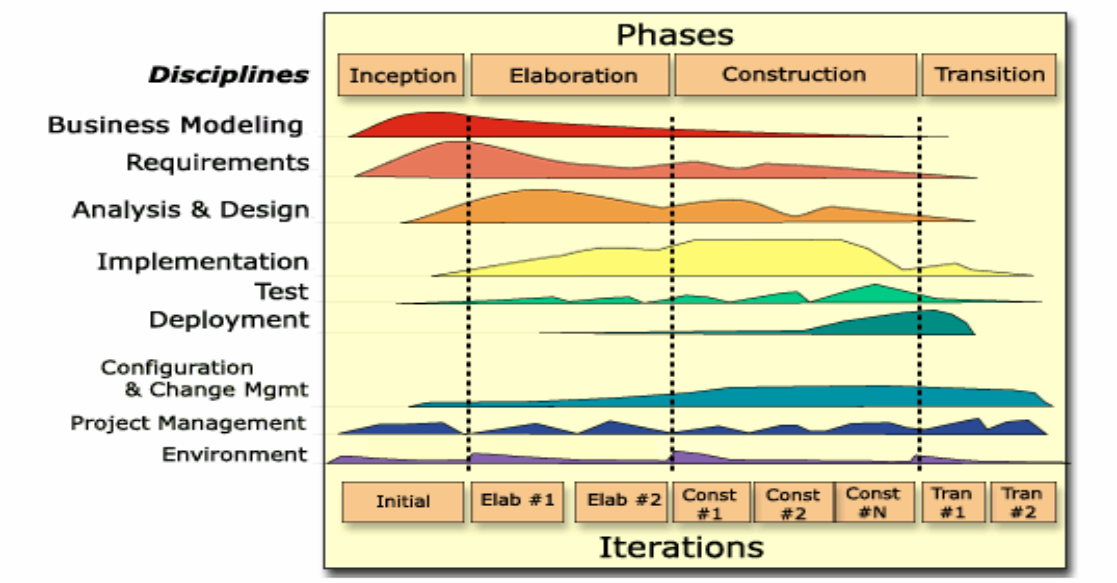

Figura 8. Fases y flujos de trabajo de RUP.

Anexo 9. Fases del modelo de proceso de MSF

Figura 9. Fases de modelo de proceso de MSF.

Anexo 10. Modelo de la encuesta

Compañero(a):

La Universidad de las Ciencias Informáticas se encuentra haciendo la presente encuesta con el propósito de conocer como se comporta la documentación dentro de los proyectos productivos. No es necesario que ponga su nombre, su identidad nunca será revelada. Esperamos que colabore y para ello le pedimos que a la hora de responder las preguntas sea lo más sincero posible, ya que de ello dependerá en gran medida el resultado de nuestro trabajo, muchas gracias de antemano por su ayuda.

1. ¿Tiene conocimientos si en su proyecto se establecen plantillas para los documentos?
   Si __ No __
2. En caso de establecer plantillas, seleccione para que artefactos. (Marque con una cruz (X) su respuesta).

   __ Modelo de Análisis

   __ Modelo de Diseño

   __ Modelo de despliegue

   __ Documento de Arquitectura de Software

   __ Mapa de navegación

   __ Realización de CU

   __ Modelo de datos

   __ Paquete

   __ Clase

   __ Subsistema de diseño

   __ Sistemas externos con los que interactúa

   __ Capa

   __ Interfaz de usuario

__ Interfaz de Aplicación

__ Componente

__ Patrón de Diseño

__ Mecanismo de Diseño

__ Componentes reutilizados

__ Diagrama de Clases

__ Diagrama de Secuencia

__ Diagrama de Colaboración

__ Diagrama de Estados

__ Capas

__ Componentes

__ Subsistemas

__ Ficheros Fuentes

__ Ejecutables

__ Sistemas Externos empleados

__ Componentes Externos empleados

__ Componentes de Prueba

__ Plan de Integración

__ *Releases* del sistema

__ Versión del sistema

3. De las siguientes metodologías de desarrollo de software seleccione las que mejor conozca. (Marque con una cruz (X) como máximo tres).

__ RUP

__ XP

__ Otra(s) ¿Cuál(es)?_____

4. ¿Cuál se usa en su proyecto?

_____

5. De la metodología mencionada, ¿a que flujos de trabajo usted le confiere más importancia?_____

   ¿Porqué?_____
   _____

6. Para el flujo de trabajo de diseño, responda:

   a) ¿Cuál es su objetivo fundamental?

      _____

   b) ¿Qué actividades usted considera que deberán efectuarse? (Marque con una cruz (X) su respuesta).

      __ Análisis de la arquitectura

      __ Análisis de Casos de Uso

      __ Diseño de Casos de Uso

      __ Diseño de Subsistema

      __ Diseño de Capas

      __ Diseño de clases

__ Diseño de Interfaz de usuario

__ Diseño de BD

__ Diseño de Pruebas

__ Otras

c) ¿Cuáles artefactos usted considera necesario documentar en su proyecto? (Marque con una cruz (X) su respuesta).

__ Modelo de Análisis

__ Modelo de Diseño

__ Modelo de Despliegue.

__ Documento de Arquitectura de Software

__ Mapa de navegación

__ Realización de CU

__ Modelo de datos

__ Paquete

__ Clase

__ Subsistema de diseño

__ Sistemas externos con los que interactúa

__ Capa

__ Interfaz de usuario

__ Interfaz de Aplicación

__ Componente

__ Patrón de Diseño

__ Mecanismo de Diseño

__ Componentes reutilizados

__ Diagrama de Clases

__ Diagrama de Secuencia

__ Diagrama de Colaboración

__ Diagrama de Estados

__ Otros

d) ¿Cuáles artefactos usted emplea en su proyecto? (Marque con una cruz (X) su respuesta).

__ Modelo de Análisis

__ Modelo de Diseño

__ Modelo de despliegue

__ Documento de Arquitectura de Software

__ Mapa de navegación

__ Realización de CU

__ Modelo de datos

__ Paquete

__ Clase

__ Subsistema de diseño

__ Sistemas externos con los que interactúa

__ Capa

__ Interfaz de usuario

__ Interfaz de Aplicación

__ Componente

__ Patrón de Diseño

__ Mecanismo de Diseño

__ Componentes reutilizados

__ Diagrama de Clases

__ Diagrama de Secuencia

__ Diagrama de Colaboración

__ Diagrama de Estados

__ Otros

7. Para el flujo de trabajo de implementación, responda:
   a) ¿Cuál es su objetivo fundamental?

   _____

   b) ¿Qué actividades usted considera que deberán efectuarse? (Marque con una cruz (X) su respuesta).

   __ Estructurar el modelo de Implementación.

   __ Planificar la integración

   __ Implementar (codificar) componentes.

   __ Integrar Subsistemas y sistema

   __ Implementar elementos de pruebas.

   __ Ejecutar pruebas de Unidad.

   __ Otras

   c) ¿Cuáles artefactos usted considera necesario documentar en su proyecto? (Marque con una cruz (X) su respuesta).

   __ Capas

   __ Componentes

   __ Subsistemas

__ Ficheros Fuentes

__ Ejecutables

__ Sistemas Externos empleados

__ Componentes Externos empleados

__ Componentes de Prueba

__ Plan de Integración

__ *Releases* del sistema

__ Versión del sistema

__ Otros

¿Cómo lo documenta?

__ Diagramas

__ Descripciones textuales

__ Otra ¿Cuál?_____

Anexo 11. Guía para informar el peso de los criterios

Modelo No. 1

Guía para informar el peso de los criterios.

Fecha de recepción _____

Fecha de entrega _____

Nombre y Apellidos del evaluador _____

Le otorgará un peso a cada criterio de acuerdo a su opinión y el peso total de cada grupo debe sumar:

- Grupo No.1…… 25
- Grupo No.2…… 30

- Grupo no.3……. 15
- Grupo No.4……..30

Para que el peso total asignado sea 100.

Grupo No. 1: Criterios de mérito científico

1. Valor científico de la propuesta.

   Peso.....

2. Calidad de la investigación.

   Peso.....

3. Aporte científico.

   Peso.....

4. Novedad científica.

   Peso.....

Grupo No. 2: Criterios implantación

5. Satisfacción de las necesidades de los ingenieros de software.

   Peso.....

6. Necesidad del empleo de la propuesta.

   Peso.....

7. Uso del UML.

   Peso.....

8. Uso de los principios básicos de la ingeniería de software.

   Peso.....

Grupo No.3: Criterios de flexibilidad

9. Adaptabilidad a proyectos productivos independientemente de la metodología a usar.

Peso.....

10. Uso de las herramientas necesarias para la elaboración de los artefactos que el documentador considere adecuado o atractivo, o con el cual este familiarizado.

    Peso.....

Grupo No.4: Criterios de impacto

11. Repercusión en entidades en los proyectos productivos.

    Peso.....

12. Organización en el proceso de documentación del software.

    Peso.....

13. Posibilidades de aplicación.

    Peso.....

14. Impacto en el área para la cual está destinada.

    Peso.....

Anexo 12. Guía para la evaluación

Modelo No. 2

Guía para la evaluación.

Fecha de recepción _____

Fecha de entrega _____

Nombre y Apellidos del evaluador_____

- Criterios de medida que se evalúan en una escala de 1 - 5

Grupo No. 1: Criterios de mérito científico

1) Valor científico de la propuesta.

    Peso.....

2) Calidad de la investigación.

   Peso.....

3) Aporte científico.

   Peso.....

4) Novedad científica.

   Peso.....

Grupo No. 2: Criterios implantación

5) Satisfacción de las necesidades de los ingenieros de software.

   Peso.....

6) Necesidad del empleo de la propuesta.

   Peso.....

7) Uso del UML.

   Peso.....

8) Uso de los principios básicos de la ingeniería de software.

   Peso.....

Grupo No.3: Criterios de flexibilidad

9) Adaptabilidad a proyectos productivos independientemente de la metodología a usar.

   Peso.....

10) Uso de las herramientas necesarias para la elaboración de los artefactos que el documentador considere adecuado o atractivo, o con el cual este familiarizado.

    Peso.....

Grupo No.4: Criterios de impacto

11) Repercusión en entidades en los proyectos productivos.

Peso.....

12) Organización en el proceso de documentación del software.

Peso.....

13) Posibilidades de aplicación.

Peso.....

14) Impacto en el área para la cual está destinada.

Peso.....

- Categoría final del proyecto

___ Excelente: Alta novedad científica, con aplicabilidad y resultados relevantes.

___ Bueno: Novedad científica, resultados destacados.

___ Aceptable: Suficientemente bueno con reservas.

___ Cuestionable: No tiene relevancia científica y los resultados son malos.

___ Malo: No aplicable.

- Valoración final

Sugerencias del evaluador para mejorar la calidad del proyecto

Elementos críticos que deben mejorarse.

Anexo 13. Tabla de los valores del peso relativos a cada criterio

G	C / E	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_p
25	C_1	6	5	6	6	7	8	8	**6,571**
	C_2	7	6	7	7	7	6	7	**6,714**
	C_3	6	7	6	6	5	6	5	**5,857**
	C_4	6	7	6	6	6	5	5	**5,857**
30	C_5	6	7	6	8	7	7	6	**6,714**

	C_6	8	7	8	8	8	7	7	**7,571**
	C_7	8	8	8	7	7	8	8	**7,714**
	C_8	8	8	8	7	8	8	9	**8,000**
10	**C_9**	5	4	4	4	5	5	5	**4,571**
	C_{10}	5	6	6	6	5	5	5	**5,429**
35	**C_{11}**	8	8	9	8	8	8	9	**8,286**
	C_{12}	9	9	8	8	9	9	8	**8,571**
	C_{13}	8	8	9	7	8	8	8	**8,000**
	C_{14}	10	10	9	12	10	10	10	**10,143**
T		**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**

Anexo 14. Tabla para el cálculo de Concordancia.

E/C	E_1	E_2	E_3	E_4	E_5	E_6	E_7	∑E	Ep	ΔC	ΔC^2
C_1	6	5	6	6	7	8	8	46	6,571	4	16
C_2	7	6	7	7	7	6	7	47	6,714	3	9
C_3	6	7	6	6	5	6	5	41	5,857	9	81
C_4	6	7	6	6	6	5	5	41	5,857	9	81
C_5	6	7	6	8	7	7	6	47	6,714	3	9
C_6	8	7	8	8	8	7	7	53	7,571	3	9
C_7	8	8	8	7	7	8	8	54	7,714	4	16
C_8	8	8	8	7	8	8	9	56	8,000	6	36
C_9	5	4	4	4	5	5	5	32	4,571	18	324
C_{10}	5	6	6	6	5	5	5	38	5,429	12	144
C_{11}	8	8	9	8	8	8	9	58	8,286	8	64
C_{12}	9	9	8	8	9	9	8	60	8,571	10	100
C_{13}	8	8	9	7	8	8	8	56	8,000	6	36
C_{14}	10	10	9	12	10	10	10	71	10,143	21	441
DC	100	100	100	100	100	100	100	700	99,998	116	1366
M ∑E	50										
W	0,1225										
X2	11,1475										

Anexo 15. Tablas para la calificación de cada criterio

Criterios	Calificación (c)					P	P × c
	1	2	3	4	5		
C_1				X		0,06571	0,26284
C_2				X		0,06714	0,26856
C_3			X			0,05857	0,17571
C_4			X			0,05857	0,17571
C_5				X		0,06714	0,26856
C_6				X		0,07571	0,30284
C_7				X		0,07714	0,30856
C_8				X		0,08000	0,32000
C_9			X			0,04571	0,13713
C_{10}				X		0,05429	0,21716
C_{11}				X		0,08286	0,33144
C_{12}				X		0,08571	034284
C_{13}				X		0,08000	0,32000
C_{14}				X		0,10143	0,40572
Total							**3,83707**
IA	0,767414						

GLOSARIO DE TÉRMINOS

ADLs: Los Lenguajes de Descripción de Arquitectura (ADL por sus siglas en inglés) son notaciones diseñadas específicamente para lograr las especificaciones de rigor y precisión de las arquitecturas. Muchos ADLs han sido desarrollados para documentar conceptos importantes para ciertos dominios o áreas de interés.

Artefactos: Son los elementos de entrada y salida de las actividades. Son productos tangibles del proyecto. Las cosas que el proyecto produce o usa para componer el producto final (modelos, documentos, código, ejecutables…)

Atributo: Una unidad con nombre de un clasificador que describe el rango de valores que las instancias de una propiedad pueden tomar.

Builds: Es uno de los tipos de distribución que tiene un producto. Si bien no son productos en etapa 'Beta' (representa generalmente la primera versión completa del programa informático o de otro producto para su inspección previa), están diseñados para ser puestos en producción de manera temprana. Son una especie de versiones o 'pruebas en el mundo real'. Es extremadamente importante que el producto sea estable.

Clase: Una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.

CMM: Es un estándar en ingeniería de software que hace hincapié en la mejora del proceso de software en base a los procedimientos internos y sin descuidar a las personas. Establece cinco niveles progresivos para los procesos relacionados con la construcción de aplicaciones informáticas, hasta alcanzar la madurez total: 1-Inicial, 2-Repetible, 3-Definido, 4- Gestionado y 5-En Mejora Continua.

CMMI (Capability Maturity Model Integration en inglés): Modelo para la mejora o evaluación de los procesos de desarrollo y mantenimiento de sistemas y productos de software. Fue desarrollado por el Instituto de Ingeniería del Software de la Universidad Carnegie Mellon (SEI), y publicado en su primera versión en enero de 2002.

CMS: Es un repositorio en el cual se almacena toda la información del proyecto como puede ser documentación, código fuente, etc.

Código fuente: Es un conjunto de líneas que conforman un bloque de texto, escrito según las reglas sintácticas de algún lenguaje de programación destinado a ser legible por humanos.

Componente: Una parte física y reemplazable de un sistema que se ajusta a, y proporciona la realización de un conjunto de interfaces.

Diseño (flujo de trabajo): Flujo de trabajo fundamental cuyo propósito principal es el de formular modelos que se centran en los requisitos no funcionales y el dominio de la solución, y que prepara para la implementación y pruebas del sistema.

Entregable: Lo que se quiere que una persona o un equipo entregue a otra persona o equipo: caso de uso, especificaciones de diseño, documentación, de *framework,* diagramas de secuencia.

Estereotipo: Una extensión del vocabulario de UML, que permite la creación de nuevos tipos de bloques de construcción que se derivan de otros existentes pero que son específicos a un problema particular.

Fase: Periodo de tiempo entre dos hitos principales de un proceso de desarrollo.

Feature: Son pequeñas funcionalidades que el cliente quiere.

Feature List: Lista que agrupa toda la funcionalidad del sistema

Feature Sets: Es una lista de features agrupadas por la misma funcionalidad.

Mayor Feature Sets: Son una lista de feature agrupada por cada área del dominio. Puede estar compuesta por varias feature Sets.

Feedback: Realimentación o información de retorno.

Flujo de trabajo (workflow en inglés): Es el estudio de los aspectos operacionales de una actividad de trabajo: cómo se estructuran las tareas, cómo se realizan, cuál es su orden correlativo, cómo se sincronizan, cómo fluye la información que soporta las tareas y cómo se le hace seguimiento al cumplimiento de las tareas.

Framework: Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, librerías y un lenguaje de scripting entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Herramientas CASE: Son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, calculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Implementación (flujo de trabajo): Flujo de trabajo fundamental cuyo propósito esencial es implementar el sistema en términos de componente, es decir, código fuente, guiones, ficheros binarios, ejecutables, etc.

Ítems: Un ítem o registro es una unidad de almacenamiento, un objeto de información. Consiste en uno o más archivos. Una copia o ítem puede tener más de una nota.

Método: La implementación de una operación.

Metodologías de desarrollo de software: Son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software.

OMG (Object Management Group en inlgés): El Grupo de Gestión de Objetos es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos como por ejemplo UML. Es una organización no lucrativa que promueve el uso de tecnología orientada a objetos mediante guías y especificaciones para tecnologías orientadas a objetos. El grupo está formado por compañías y organizaciones de software como lo son: IBM, *Apple Computer*, entre otras.

Paquetes: Son usados para organizar y manipular la complejidad de los modelos largos. Un grupo de paquetes modelan elementos y los diagramas semejantes como el uso de casos, clases, actividades, procesos, estados, etc., y sus diagramas asociados; en tal camino que eso puede ser remitido como uno entero. Los paquetes pueden ser representados en un diagrama, remitido como Diagrama de Paquete.

Performance: Representación, actuación. Referente a un trabajo: desempeño, ejecución, realización, rendimiento, resultados, cumplimiento. Combinar esfuerzos entre todos los miembros del equipo, con el fin de fomentar la efectividad, por medio de las sugerencias de los demás acerca de los deberes asignados.

Producto: Resultado de cada etapa.

Prototipo: Primera versión de un nuevo tipo de producto, en el que se han incorporado sólo algunas características del sistema final, o no se han realizado completamente.

Pruebas de caja blanca: Permiten examinar la estructura interna del programa. Se diseñan casos de prueba para examinar la lógica del programa. Es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar casos de prueba que garanticen la ejercitación de todos los caminos independientes de cada módulo y todas las decisiones lógicas, así como que se ejecuten todos los bucles y las estructuras de datos internas.

Pruebas de unidad: Es un tipo de prueba que se centra en el módulo. Usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo. La prueba de unidad hace uso intensivo de las técnicas de prueba de caja blanca.

RAD (Rapid Application Development en inglés): Es un proceso de desarrollo de software, desarrollado inicialmente por James Martin en 1980. El método comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE. Tradicionalmente, el desarrollo rápido de aplicaciones tiende a englobar también la usabilidad, utilidad y la rapidez de ejecución.

Requisitos: Son las funciones, servicios y restricciones operativas del sistema.

Requisitos funcionales: Son aquellos que describen lo que debe hacer el sistema.

Requisitos no funcionales: Son aquellos que describen las facilidades que debe proporcionar el sistema.

Software: Se refiere a los programas y datos almacenados en un ordenador.