DETERMINACIÓN DE LOS FACTORES QUE CONLLEVAN A LA FORMULACIÓN DE UNA NUEVA METODOLOGIA ÁGIL DE DESARROLLO DE SOFTWARE

JOHAN FELIPE OSPINA PUERTAS

JOHAN ANDRES CANAVAL NUÑEZ

UNIDAD CENTRAL DEL VALLE DEL CAUCA

FACULTAD DE INGENIERIA

PROGRAMA DE INGENIERIA DE SISTEMAS

TULUA 2013

DETERMINACIÓN DE LOS FACTORES QUE CONLLEVAN A LA FORMULACIÓN DE UNA NUEVA METODOLOGIA ÁGIL DE DESARROLLO DE SOFTWARE

JOHAN FELIPE OSPINA PUERTAS

JOHAN ANDRES CANAVAL NUÑEZ

TRABAJO DE GRADO

DIRECTOR:

ING. JOSE GABRIEL PEREZ CANENCIO

UNIDAD CENTRAL DEL VALLE DEL CAUCA

FACULTAD DE INGENIERIA

PROGRAMA DE INGENIERIA DE SISTEMAS

TULUA 2013

Nota de aceptación:

Aprobado por el Comité de Grado
En cumplimiento con los requisitos
Exigidos por la Universidad Central
Del Valle del Cauca para optar por
El título de Ingeniero de Sistemas.

_____

Jurado

_____

Jurado

_____

Jurado

_____

Director

Tuluá, Julio 28 de 2013

LISTA DE CONTENIDO

LISTA DE IMAGENES

LISTA DE TABLAS

TITULO

DETERMINACIÓN DE LOS FACTORES QUE CONLLEVAN A LA FORMULACIÓN DE UNA NUEVA METODOLOGIA ÁGIL DE DESARROLLO DE SOFTWARE

1. IDENTIFICACION DEL PROYECTO

1.1 EL PROBLEMA

El concepto de metodología, dentro de la *Ingeniería del Software* es, sin duda, uno de los más oscuros y que más confusión produce tanto en estudiantes como en profesionales involucrados en procesos de desarrollo de software.

Tanto es así, que en muchas empresas de desarrollo (no todas, por supuesto), la aplicación de una metodología brilla por su ausencia.

Además, la constante innovación tecnológica, cambios en los procesos y economías de los países hace que cada vez sea necesaria la aplicación de nuevas metodologías adaptadas a los nuevos tiempos y necesidades de cada país, sin embargo, las empresas siguen aplicando dichas metodologías las cuales fueron creadas hace bastante tiempo y pensadas para viejos problemas, por lo cual se puede decir que se carece de formas y estrategias modernas para atender los nuevos problemas.

Algunos profesionales de las tecnologías de la información como algunos de sus clientes se van dando cuenta de que se hace necesario seguir unas ciertas pautas predefinidas en el desarrollo del software de calidad: es decir, llevar un comportamiento metódico, que no es otra cosa que seguir una metodología.

Esto lleva a pensar que la ausencia de una metodología en el desarrollo de un proyecto de software garantiza con seguridad la ausencia de calidad.

Las actuales metodologías son hechas en otros países y se caracterizan por ser muy extensas y con un alto costo para su implementación ya que van acorde a la economía del país que la realizo y a los tipos de empresa que en él se encuentran, empresas con un poder monetario muy superior al nuestro, por lo cual no son fácilmente adaptables en Colombia.

En Colombia la situación económica es bastante difícil para muchas de las pequeñas y medianas empresas, por lo tanto se ve la necesidad del desarrollo de una metodología ágil, eficaz y de buena calidad y lo más importante a un costo moderado el cual sea asequible y se ajuste a las necesidades y economía de nuestras organizaciones.

1.2 FORMULACIÓN DEL PROBLEMA

¿Comparar las metodologías ágiles más comunes para el desarrollo de software, nos conducirá a la determinación de los factores que inciden en el proceso para la formulación de una nueva metodología?

2. OBJETIVOS

2.1 OBJETIVO GENERAL

Determinar los factores diferenciadores que lleven a proponer una forma innovadora de construir software, mediante el estudio de las principales metodologías de desarrollo agiles.

2.2 OBJETIVOS ESPECÍFICOS

- Realizar el estudio del estado del arte de las metodologías de desarrollo ágil.
- Identificar las fases de cada metodología estableciendo los aspectos comunes entre ellas.
- Determinar los beneficios de cada metodología de acuerdo con sus características individuales.
- Definir las etapas que debería tener una nueva metodología ágil.
- Crear instrumentos que se conviertan en herramientas claves para el proceso de levantamiento de información.

3. JUSTIFICACIÓN

Las empresas en Colombia al momento de desarrollar software trabajan con metodologías extranjeras, las cuales están enfocadas en dichos mercados y condicionan mucho su trabajo debido al alto costo que estas presentan. En Colombia empresas como PSL utilizan las metodologías Scrum y RUP, para desarrollo de software ágil y desarrollo de software más complejo respectivamente, también encontramos casos como IT Group el cual utiliza una metodología propia, la cual se adapta más a la economía y personal de su empresa, o casos como la empresa ArquitecSoft la cual deja elegir a sus desarrolladores con que metodología trabajar como se puede observar en la Tabla 6.

Este proyecto permitirá comparar varias metodologías, identificar sus diferencias rescatando aquellos elementos que representen beneficios en cada una de ellas.

Para los procesos de desarrollo de software, este trabajo generará una alternativa que puede llegar a convertirse en valor agregado para maximizar los recursos de la organización y reducir tiempos de proceso en las fases de construcción de aplicaciones.

4. MARCOS DE REFERENCIA

4.1 MARCO TEORICO

Este trabajo comprenderá teorías, métodos y herramientas que serán convenientes para la formulación de una nueva metodología ágil basada en el desarrollo de software en Colombia.

A continuación se obtendrá información más detallada sobre los temas que rigen el proyecto y que gracias a estos aportes se desarrollará adecuadamente la propuesta presentada.

4.1.1 Definiciones de Metodología de desarrollo de software:

1. Son un conjunto de procedimientos, técnicas y ayudas para el desarrollo de productos software. En sí pasos naturales o lógicos para la construcción de software[1].

2. Se definen como un conjunto de pasos como análisis, diseño, desarrollo, implementación y pruebas llamados ciclo de vida como una definición general[1].

3. Conjunto ordenado de pasos a seguir para llegar a la solución de un problema u obtención de un producto de software, son también los pasos generales que sigue el proceso de desarrollo de un producto software. Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software[1].

4.1.2 **Metodología ágil.**

Las metodologías ágiles (como por ejemplo XP, SCRUM, DSDM, Crystal, etc.) forman parte del movimiento de desarrollo ágil de software, que se basan en la adaptabilidad de cualquier cambio como medio para aumentar las posibilidades de éxito de un proyecto[2].

De forma que una metodología ágil es la que tiene como principios que:

- Los individuos y sus interacciones son más importantes que los procesos y las herramientas.
- El software que funciona es más importante que la documentación exhaustiva.
- La colaboración con el cliente en lugar de la negociación de contratos.
- La respuesta delante del cambio en lugar de seguir un plan cerrado.

---

[1] METODOLOGIAS DE DESARROLLO DE SOFTWARE. 2003, Disponible en http://es.scribd.com/doc/12983329/Metodologia-de-Desarrollo-de-Software.

[2] GÓMEZ DANIEL, ARANDA ELISABET, FRABREGA JORDI, Programación Extrema, Disponible en: http://eisc.univalle.edu.co/materias/WWW/material/lecturas/xp.pdf

Se puede decir que, este movimiento empezó a existir a partir de febrero de 2001, cuando se reunieron los representantes de cada una de estas metodologías y terminaron poniendo en común sus ideas en una declaración conjunta.

4.1.3 Definición de las principales metodologías de desarrollo

4.1.3.1 Metodologías de desarrollo ágil.

4.1.3.1.1 **Programación Extrema**. "La programación extrema es una metodología de desarrollo ligera (o ágil) basada en una serie de valores y de prácticas de buenas maneras que persigue el objetivo de aumentar la productividad a la hora de desarrollar programas.

Este 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.

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 cómo 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, el resultado es una nueva manera de ver el desarrollo de software.

El objetivo que se perseguía en el momento de crear esta metodología era la búsqueda de un método que hiciera que los desarrollos fueran más sencillos, aplicando el sentido común[3].

4.1.3.1.2 **Scrum.** Es una metodología ágil de desarrollo de proyectos que toma su nombre y principios de los estudios realizados sobre nuevas prácticas de producción por Hirotaka Takeuchi e Ikujijo Nonaka a mediados de los 80.

Aunque surgió como modelo para el desarrollo de productos tecnológicos, también se emplea en entornos que trabajan con requisitos inestables y que requieren rapidez y flexibilidad; situaciones frecuentes en el desarrollo de determinados sistemas de software.

[3] GÓMEZ DANIEL, ARANDA ELISABET, FRABREGA JORDI, Programación Extrema, Disponible en: http://eisc.univalle.edu.co/materias/WWW/material/lecturas/xp.pdf

Jeff Sutherland aplicó el modelo Scrum al desarrollo de software en 1993 en Easel Corporation (Empresa que en los macro-juegos de compras y fusiones se integraría en VMARK, luego en Informix y finalmente en Ascential Software Corporation). En 1996 lo presentó junto con Ken Schwaber como proceso formal, también para gestión del desarrollo de software en OOPSLA 96. Más tarde, en 2001 serían dos de los promulgadores del Manifiesto ágil. En el desarrollo de software Scrum está considerado como modelo ágil por la Agile Alliance[4].

4.1.3.2 **Otras metodologías de desarrollo**

4.1.3.2.1 **RUP**. El Rational Unified Process o Proceso Unificado Rational. Es un proceso de ingeniería de software que suministra un enfoque para asignar tareas y responsabilidades dentro de una organización de desarrollo.

Su objetivo es asegurar la producción de software de alta calidad que satisfaga la necesidad del usuario final dentro de un tiempo y presupuesto previsible. Es una metodología de desarrollo iterativo enfocada hacia "los casos de uso, manejo de riesgos y el manejo de la arquitectura".

El RUP mejora la productividad del equipo ya que permite que cada miembro del grupo sin importar su responsabilidad específica acceda a la misma base de datos de conocimiento. Esto hace que todos compartan el mismo lenguaje, la misma visión y el mismo proceso acerca de cómo desarrollar software[5].

4.1.3.2.2 **Iconix**. El proceso ICONIX (Rosenberg & Scott, 1999) se define como un "proceso" de desarrollo de software práctico. ICONIX está entre la complejidad del RUP (Rational Unified Processes) y la simplicidad y pragmatismo del XP (Extreme Programming), sin eliminar las tareas de análisis y de diseño que XP no contempla.

ICONIX es un proceso simplificado en comparación con otros procesos más tradicionales, que unifica un conjunto de métodos de orientación a objetos con el fin de abarcar todo el ciclo de vida de un proyecto. Fue elaborado por Doug Rosenberg y Kendall Scott a partir de una síntesis del proceso unificado de los "tres amigos" Booch, Rumbaugh y Jacobson y que ha dado soporte y conocimiento a la metodología ICONIX desde 1993. Presenta claramente las actividades de cada fase y exhibe una secuencia de pasos que deben ser seguidos. Además ICONIX está adaptado a los patrones y ofrece el soporte de UML, dirigido por casos de uso y es un proceso iterativo e incremental[6].

---

[4] PALACIO JUAN, El Modelo Scrum, 2006. Disponible en: http://www.navegapolis.net/files/s/NST-010_01.pdf

[5] GALLEGO GOMEZ Juan Pablo, Septiembre 16 de 2007, Universidad tecnológica de Pereira, Fundamentos de la Metodología RUP, Disponible en http://es.scribd.com/doc/297224/RUP.

[6] DE SAN MARTIN OLIVA CARLA REBECA PATRICIA, Metodología ICONIX, disponible en: http://www.portalhuarpe.com.ar/Seminario09/archivos/MetodologiaICONIX.pdf

4.2 MARCO CONCEPTUAL

Eficiencia. Es la capacidad del software para hacer buen uso de los recursos que manipula.

Una práctica muy común en los desarrolladores es la optimización excesiva, lo importante es mantener un balance adecuado entre eficiencia y corrección.

Casos de Uso. Notación utilizada para representar los requerimientos funcionales de un sistema basada en el esquema propuesto por Ivar Jacobson a principios de la década del '90.

IS/IT (Information Systems/Information Technology). Se refiere a los sectores de las organizaciones relacionados con el manejo de la información y con las tecnologías involucradas en este manejo.

Metáfora. Equivalente de la arquitectura en el universo de XP. La metáfora guía al desarrollo proveyendo de integridad conceptual al diseño y comunicando a todos los involucrados los elementos básicos de la solución y sus relaciones.

Product Backlog. En el universo de Scrum este es el conjunto de requerimientos a ser implementados para el sistema en construcción siendo el mismo priorizado continuamente por el Cliente para armar el Backlog de cada iteración (denominado Sprint Backlog).

Programación de a Pares (Pair Programming). Una de las doce prácticas de XP, esa que dos personas programan frente a una computadora. Una de ellas escribe el código mientras la otra inspecciona continuamente lo que se desarrolla realizando un Control de Calidad en el momento.

Refactoring. Técnica que mantiene intacto el funcionamiento del software mejorando la estructura interna del mismo.

Sprint. En el universo de Scrum este es el nombre que se le da a una iteración.

Stakeholder. Toda aquella persona u organización siendo influenciada o ejerciendo influencia sobre el software que está siendo construido.

Unit Testing. Técnica utilizada por muchas de las metodologías ágiles descritas que consiste en realizar pruebas automatizadas que verifiquen el correcto funcionamiento de las clases que son desarrolladas. Para esto se utilizan frameworks como el JUnit, Cactus, HttpUnit, etc.

Iteración. En programación, cuando el bloque de instrucciones de un bucle se ejecuta, se dice que se ha producido una iteración.

Operacionalización de la Hipótesis. Operacionalizar las hipótesis equivale a descender el nivel de abstracción de las variables y en esta forma hacer referencia empírica de las mismas; implica desglosar la variable en indicadores por medio de un proceso de deducción lógica, los cuales se refieren a situaciones específicas de las variables. Los indicadores pueden medirse mediante índices o investigarse por ítems o preguntas que se incluyen en los instrumentos que se diseñan para la recopilación de la información; así una vez que el investigador desglosa la variable en indicadores, estos le permiten definir la información básica para verificar las hipótesis.

5. HIPOTESIS

Las metodologías tradicionales empleadas en los procesos de desarrollo de software deben ser revisadas y reformuladas dadas las características tecnológicas actuales que han marcado tendencias diferentes a las que históricamente motivaron a los creadores de éstas y además es importante involucrar el factor económico que conlleva su aplicación completa en un proceso de factoría.

Para el caso de nuestro país, es importante que se tenga en cuenta que para las empresas de desarrollo de software es vital contar con una única metodología que produzca desarrollo ágil en tiempos cortos.

5.1 OPERACIONALIZACION DE LA HIPOTESIS

Variables	Indicadores
Fases del proceso de desarrollo	Cantidad de fases Actividades de cada fase
Tiempo de desarrollo	Disponibilidad del cliente que suministra la información Requisito de tiempo suministrado por el usuario Herramientas de desarrollo empleadas por el equipo de trabajo Instrumentos para la recolección de la información en la fase inicial
Vigencia de la metodología	Fecha de creación País de origen
Qué tipo de problemas resuelven	Clase de producto a crear Aplicación de escritorio Aplicación en red Aplicación web
Metodología empleada en las soluciones	Empresa vs metodología

6. ASPECTOS METODOLOGICOS

6.1 TIPO DE ESTUDIO

El tipo de estudio a aplicar es descriptivo porque se estudiarán comportamientos concretos de un objeto en particular.

El objetivo de este estudio descriptivo consiste en llegar a conocer las situaciones predominantes a través de la descripción exacta de las actividades, objetos, procesos y personas en las empresas de desarrollo de software, que inciden en la determinación de tiempos y costos del proceso de factoría.

6.2 METODO DE INVESTIGACION

Mediante el método de análisis a aplicar se identificarán las partes que caracterizan el tema a tratar para determinar la relación causa efecto entre los elementos que componen el objeto a investigar en las empresas de desarrollo de software.

6.3 FUENTES Y TECNICAS DE RECOLECCIÓN DE LA INFORMACION

6.3.1 Fuentes primarias

- **Artículos**: estos son tomados de fuentes fiables y verificables, como IEEE en la cual encontramos archivos de científicos, doctores, ingenieros, entre otros; la mayoría de esta información la encontramos en inglés.
- **Portales**: son las páginas web o de internet de los sitios oficiales confiables donde extraemos la información, como el portal de IBM donde encontramos la información de la metodología RUP.
- **Libros**: estos textos consultados son de grandes escritores en los cuales encontramos sus investigaciones, pensamientos y/o teorías acerca del trabajo que estamos realizando, un ejemplo es Extreme Programming Explained: Embrace Change de Kent Beck.
- **Personas**: grandes conocedores del tema, con experiencia y credibilidad, los cuales sirven como fuente de consulta fiable para nuestro trabajo, como el Ingeniero Manuel Dávila Sguerra (Ver Anexo A)

6.3.2 Fuentes secundarias

- **Revistas**: Especializadas en temas de ingeniería de software, donde encontramos la información más actualizadas del mundo del software, artículos de desarrolladores y/o empresarios del mismo.

7. FASE TECNICA

En este trabajo son tomadas las metodologías Scrum, XP, Iconix y RUP, para las cuales se estudiaran más a detalle sus fases, artefactos y tiempos con el fin de identificar cuáles son los ítems principales y/o más utilizados que nos llevaran a la formulación de una nueva metodología.

7.2 METODOLOGÍA XP

7.2.1 Historia. "La Programación Extrema, como proceso de creación de software diferente al convencional, nace de la mano de Kent Beck (gurú de la XP y autor de los libros más influyentes sobre el tema).

Chrysler Corporation hacía tiempo que estaba desarrollando una aplicación de nóminas, pero sin demasiado éxito por parte de la gente que tenía en el proyecto. El verano de 1996, Beck entró en nómina en la compañía y se le pidió de hacer esta aplicación como trabajo. Es en esta aplicación cuando nace la Programación Extrema como tal.

Beck reconoció que el proceso (o metodología) de creación de software o la carencia de este era la causa de todos los problemas y llegó a la conclusión que para proporcionar un proceso que fuera flexible era necesario realizar ciertos cambios en la estructura o manera de hacer de los programadores, los cuales se tenían que acomodar al cambio a realizar. Él tenía varias ideas de metodologías para la realización de programas que eran cruciales para el buen desarrollo de cualquier sistema.

Las ideas primordiales de su sistema las comunicó en la revista C++ Magazine en una entrevista que ésta le hizo el año 1999. En ésta decía que él estaba convencido que la mejor metodología era un proceso que enfatizase la comunicación dentro del equipo, que la implementación fuera sencilla, que el usuario tenía que estar muy informado e implicado y que la toma de decisiones tenía que ser muy rápida y efectiva.

Los autores (o mejor dicho, los propulsores como el propio Kent Beck, Ward Cunningham o Ron Jeffries entre otros) de la Programación Extrema, fueron a la web Portland Pattern Repository y empezaron a hablar de ella y promocionarla, de lo que era y cómo realizarla. Estos propulsores de la XP hablaban de ella en cada ocasión que tenían y en cada página que, poco o mucho hablara de temas de programación.

Este hecho, llegó a molestar a buena parte de la comunidad que intentaba discutir sobre temas de programación. Fue tanta esta molestia que nació el fenómeno XP Free Zone (zona libre de XP) en determinadas webs como petición de no hablar de Programación Extrema en ella.

La discusión sobre temas de diseño de modelos de programación sobre los cambios recientes se hizo tema difícil porque la mayoría de la actividad fue relacionada con la Programación Extrema.

Eventualmente, y debido a esto, la mayoría de la gente que solía discutir sobre los temas de diseño de modelos de programación fue apartándose de este ambiente para discutir sus ideas en otros ambientes más "reservados". La comunidad empezó a referirse a estos sitios como a Salas Wiki (Wards Wiki).

7.2.2 Principios básicos. La Programación Extrema se basa en 12 principios básicos agrupados en cuatro categorías:

- **Retroalimentación a escala fina.** El principio de pruebas: se tiene que establecer un período de pruebas de aceptación del programa (llamado también período de caja negra) donde se definirán las entradas al sistema y los resultados esperados de estas entradas. Es muy recomendable automatizar estas pruebas para poder hacer varias simulaciones del sistema en funcionamiento. Para hacer estas simulaciones automatizadas, se pueden utilizar Ambientes de Prueba (Unittesting frameworks). Un buen ejemplo de un ambiente de prueba es el JUnit para Java (www.junit.org/index.htm). Otros ambientes de pruebas para otros lenguajes como C, C++, Java Script, XML y servicios Web.

Proceso de planificación: en esta fase, el usuario tendrá que escribir sus necesidades, definiendo las actividades que realizará el sistema. Se creará un documento llamado Historias del usuario (User Stories). Entre 20 y 80 historias (todo dependiendo de la complejidad del problema) se consideran suficientes para formar el llamado Plan de Liberación, el cual define de forma específica los tiempos de entrega de la aplicación para recibir retroalimentación por parte del usuario. Por regla general, cada una de les Historias del usuario suelen necesitar de una a tres semanas de desarrollo.

Las reuniones periódicas son muy importantes y tienen que ser una constante durante esta fase de planificación. Estas pueden ser a diario, con todo el equipo de desarrollo para identificar problemas, proponer soluciones y señalar aquellos puntos a los que se les ha de dar más importancia por su dificultad o por su punto crítico.

El cliente en el sitio: se le dará poder para determinar los requerimientos, definir la funcionalidad, señalar las prioridades y responder las preguntas de los programadores. Esta fuerte interacción cara a cara con el programador disminuye el tiempo de comunicación y la cantidad de documentación, junto con los altos costes de su creación y mantenimiento. Este representante del cliente estará con el equipo de trabajo durante toda la realización del proyecto.

Programación en parejas: uno de los principios más radicales y en el que la mayoría de gerentes de desarrollo pone sus dudas. Requiere que todos los programadores XP escriban su código en parejas, compartiendo una sola

máquina. De acuerdo con los experimentos, este principio puede producir aplicaciones más buenas, de manera consistente, a iguales o menores costes.

Aunque el pair-programming puede no ser para todo el mundo, la evidencia anecdótica en la lista de correo de XP demuestra un gran éxito"[7].

- **Proceso continuo en lugar de por lotes.** Integración continua: permite al equipo hacer un rápido progreso implementando las nuevas características del software. En lugar de crear builds (o versiones) estables de acuerdo a un cronograma establecido, los equipos de programadores XP pueden reunir su código y reconstruir el sistema varias veces al día. Esto reduce los problemas de integración comunes en proyectos largos y estilo cascada.

Refactorización: permite a los equipos de programadores XP mejorar el diseño del sistema a través de todo el proceso de desarrollo. Los programadores evalúan continuamente el diseño y re codifican lo necesario. La finalidad es mantener un sistema enfocado a proveer el valor de negocio mediante la minimización del código duplicado y/o ineficiente.

Entregas pequeñas: colocan un sistema sencillo en producción rápidamente que se actualiza de forma rápida y constante permitiendo que el verdadero valor de negocio del producto sea evaluado en un ambiente real. Estas entregas no pueden pasar las 2 o 3 semanas como máximo.

- **Entendimiento compartido.** Diseño simple: se basa en la filosofía de que el mayor valor de negocio es entregado por el programa más sencillo que cumpla los requerimientos. Simple Design se enfoca en proporcionar un sistema que cubra las necesidades inmediatas del cliente.

Este proceso permite eliminar redundancias y rejuvenecer los diseños obsoletos de forma sencilla.

Metáfora: desarrollada por los programadores al inicio del proyecto, define una historia de cómo funciona el sistema completo. XP estimula historias, que son breves descripciones de un trabajo de un sistema en lugar de los tradicionales diagramas y modelos UML (Unified Modeling Language). La metáfora expresa la visión evolutiva del proyecto que define el alcance y propósito del sistema.

Las tarjetas CRC (Clase, Responsabilidad y Colaboración) también ayudarán al equipo a definir actividades durante el diseño del sistema. Cada tarjeta representa una clase en la programación orientada a objetos y define sus responsabilidades (lo que ha de hacer) y las colaboraciones con las otras clases (cómo se comunica con ellas).
Propiedad colectiva del código: un código con propiedad compartida. Nadie es el propietario de nada, todos son el propietario de todo. Este método difiere en

[7] GÓMEZ DANIEL, ARANDA ELISABET, FRABREGA JORDI, Programación Extrema, Disponible en: http://eisc.univalle.edu.co/materias/WWW/material/lecturas/xp.pdf

mucho a los métodos tradicionales en los que un simple programador posee un conjunto de código. Los defensores de XP argumentan que mientras haya más gente trabajando en una pieza, menos errores aparecerán.

Estándar de codificación: define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona.

➢ **Bienestar del programador.** La semana de 40 horas: la programación extrema sostiene que los programadores cansados escriben código de menor cualidad. Minimizar las horas extras y mantener los programadores frescos, generará código de mayor calidad. Como dice Beck, está bien trabajar tiempos extra cuando es necesario, pero no se ha de hacer durante dos semanas seguidas.

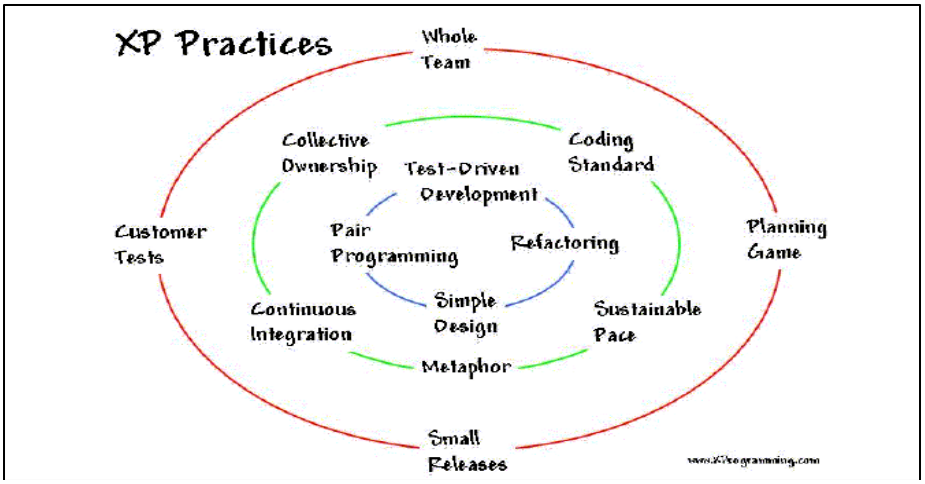

Imagen 1. Proceso XP
FUENTE: xprogramming.com

➢ **Proceso de desarrollo.** La programación extrema parte del caso habitual de una compañía que desarrolla software, normalmente a medida, en la que hay diferentes roles: un equipo de gestión (o diseño), uno de desarrollo y los clientes finales. La relación entre el equipo de diseño, los que desarrollan el software y clientes es totalmente diferente al que se ha producido en las metodologías tradicionales, que se basaba en una fase de captura de los requisitos previa al desarrollo, y de una fase de validación posterior al mismo.

➢ **Interacción con el cliente.** En este tipo de programación el cliente pasa a ser parte implicada en el equipo de desarrollo. Su importancia es máxima en el momento de tratar con los usuarios y en efectuar las reuniones de planificación.

➢ Tiene un papel importante de interacción con el equipo de programadores, sobre todo después de cada cambio, y de cada posible problema localizado, mostrando las prioridades, expresando sus sensaciones. En este tipo de

programación existirán pruebas de aceptación de la programación que ayudarán a que su labor sea lo más provechosa posible.

El cliente se encuentra mucho más cerca del proceso de desarrollo. Se elimina la fase inicial de recopilación de requerimientos y se permite que éstos se vayan cogiendo a lo largo del proyecto, de una manera ordenada. De esta forma se posibilita que el cliente pueda ir cambiando de opinión sobre la marcha, pero a cambio han de estar siempre disponibles para solucionar las dudas del equipo de desarrollo.

En XP aparece un nuevo concepto llamado "*Historia de usuario*". Se trata de una lista de características que el cliente necesita que existan en el producto final. Estas constan de dos fases.

1. En la primera fase, el cliente describe con sus propias palabras las características y es el responsable del equipo, el encargado de informarlo de las dificultades técnicas de cada una de ellas y de su costo. A consecuencia de este diálogo, el cliente deja por escrito un conjunto de historias y las ordena en función de la prioridad que tiene para él. De esta manera ya es posible definir unas fechas aproximadas para ellos.

2. En la segunda fase, el cliente cogerá las primeras historias a implementar y las dividirá en trabajos a realizar. El cliente también participa, pero hay más peso por parte del equipo de desarrolladores, esto dará como resultado una planificación más exacta. Este método se repetirá para cada historia.

   A diferencia de otras técnicas, como puede ser UML, en el caso de XP, se exige que sea el cliente el encargado de escribir los documentos con las especificaciones de lo que realmente quiere, como un documento de requisitos de usuario.

3. En esta fase, el equipo técnico será el encargado de catalogar las historias del cliente y asignarles una duración. La norma es que cada historia de usuario tiene que poder ser realizable en un espacio entre una y tres semanas de programación. Las que requieran menos tiempo serán agrupadas y las que necesiten más serán modificadas o divididas.

4. Finalmente decir que las historias de los usuarios serán escritas en tarjetas, lo que facilitará que el cliente pueda especificar la importancia relativa entre las diferentes historias de usuario, así como el trabajo de los programadores que podrán catalogarlas correctamente. Este formato también es muy útil en el momento de las pruebas de aceptación.

   ➢ **Planificación del proyecto.** En este punto se tendrá que elaborar la planificación por etapas, donde se aplicarán diferentes iteraciones. Para hacerlo será necesaria la existencia de reglas que se han de seguir por las partes implicadas en el proyecto para que todas las partes tengan voz y se sientan realmente partícipes de la decisión tomada.

Las entregas se tienen que hacer cuanto antes y con cada iteración, el cliente ha de recibir una nueva versión. Cuanto más tiempo se tarde en introducir una parte esencial, menos tiempo se tendrá para trabajar con ella después. Se aconseja muchas entregas y muy frecuentes. De esta manera un error en la parte inicial del sistema tiene más posibilidades de detectarse rápidamente.

Una de las máximas a aplicar es, los cambios, no han de suponer más horas de programación para el programador, ya que el que no se termina en un día, se deja para el día siguiente.

Se ha de tener asumido que en el proceso de planificación habrán errores, es más, serán comunes, y por esto esta metodología ya los tiene previstos, por lo tanto se establecerán mecanismos de revisión. Cada tres o cinco iteraciones es normal revisar las historias de los usuarios y renegociar la planificación.

Cada iteración necesita también ser planificada, es lo que se llama *planificación iterativa*, en la que se anotarán las historias de usuarios que se consideren esenciales y las que no han pasado las pruebas de aceptación. Estas planificaciones también se harán en tarjetas, en las que se escribirán los trabajos que durarán entre uno y tres días.

Es por esto que el diseño se puede clasificar como continuo. Añade agilidad al proceso de desarrollo y evita que se mire demasiado hacia delante, desarrollando trabajos que aún no han estado programados.

Este tipo de planificación en iteraciones y el diseño iterativo, hace que aparezca una práctica que no existía en la programación tradicional. Se trata de las discusiones diarias informales, para fomentarla comunicación y para hacer que los desarrolladores tengan tiempo de hablar de los problemas a los que se enfrentan y de ver cómo van con sus trabajos.

- **Diseño, desarrollo y pruebas.** El desarrollo es la parte más importante en el proceso de la programación extrema. Todos los trabajos tienen como objetivo que se programen lo más rápidamente posible, sin interrupciones y en dirección correcta.

También es muy importante el diseño y se establecen los mecanismos, para que éste sea revisado y mejorado de manera continuada a lo largo del proyecto, según se van añadiendo funcionalidades al mismo.

La clave del proceso de desarrollar XP es la comunicación. La mayoría de los problemas en los proyectos son por falta de comunicación en el equipo.

En XP, aparece un nuevo concepto llamado Metáfora. Su principal objetivo es mejorar la comunicación entre todos los integrantes del equipo, al crear una visión global y común de lo que se quiere desarrollar. La metáfora tiene que ser

expresada en términos conocidos por los integrantes del equipo, por ejemplo comparando el sistema que se desarrollará con alguna cosa de la vida real.

Antes de empezar a codificar se tienen que hacer pruebas unitarias, es decir: cada vez que se quiere implementar una parte de código, en XP, se tiene que escribir una prueba sencilla, y después escribir el código para que la pase. Una vez pasada se amplía y se continúa. En XP hay una máxima que dice "Todo el código que puede fallar tiene que tener una prueba".

Con estas normas se obtiene un código simple y funcional de manera bastante rápida. Por esto es importante pasar las pruebas al 100%.

Respecto a la integración del software, en XP se ha de hacer una integración continua, es decir, cada vez se tienen que ir integrando pequeños fragmentos de código, para evitar que al finalizar el proyecto se tenga que invertir grandes esfuerzos en la integración final. En todo buen proyecto de XP, tendría que existir una versión al día integrada, de manera que los cambios siempre se realicen en esta última versión.

Otra peculiaridad de XP es que cada programador puede trabajar en cualquier parte del programa.

De esta manera se evita que haya partes "propietarias de cada programador". Por esto es tan importante la integración diaria.

Para terminar, otra peculiaridad que tiene la XP. La de fomentar la programación en parejas, es decir, hacer que los programadores no trabajen en solitario, sino que siempre estarán con otra persona. Una pareja de programadores ha de compartir el teclado, el monitor y el ratón. El principio fundamental de este hecho es realizar de manera continua y sin parar el desarrollo de código. Las parejas tienen que ir cambiando de manera periódica, para hacer que el conocimiento se difunda en el grupo. Está demostrado que de esta manera el trabajo es más eficaz y también se consigue más y mejor código."[8]

[8] XP EXTREME PROGRAMMING EXPLAINED PDF. Disponible en la página web :http//www.extremeprogramming.org, www.xprogramming.com

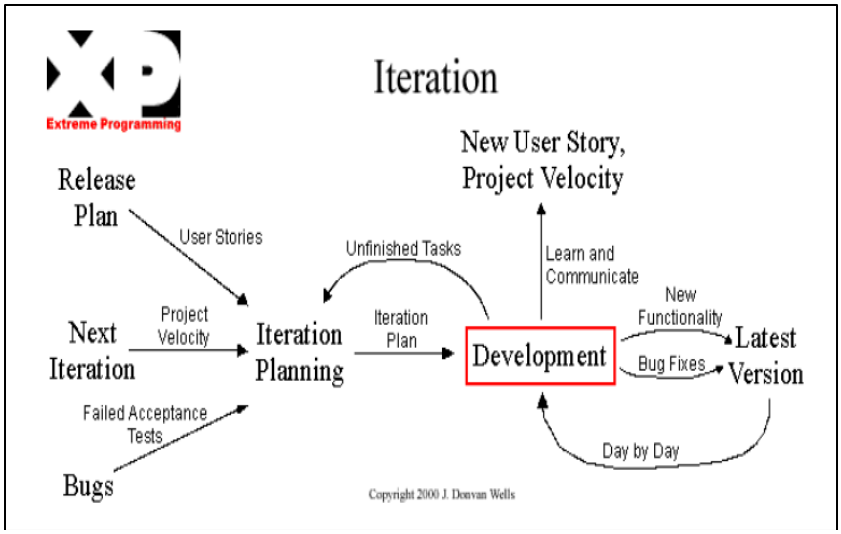

Imagen 2. Iteraciones XP
FUENTE: xprogramming.com

7.3 METODOLOGÍA RUP

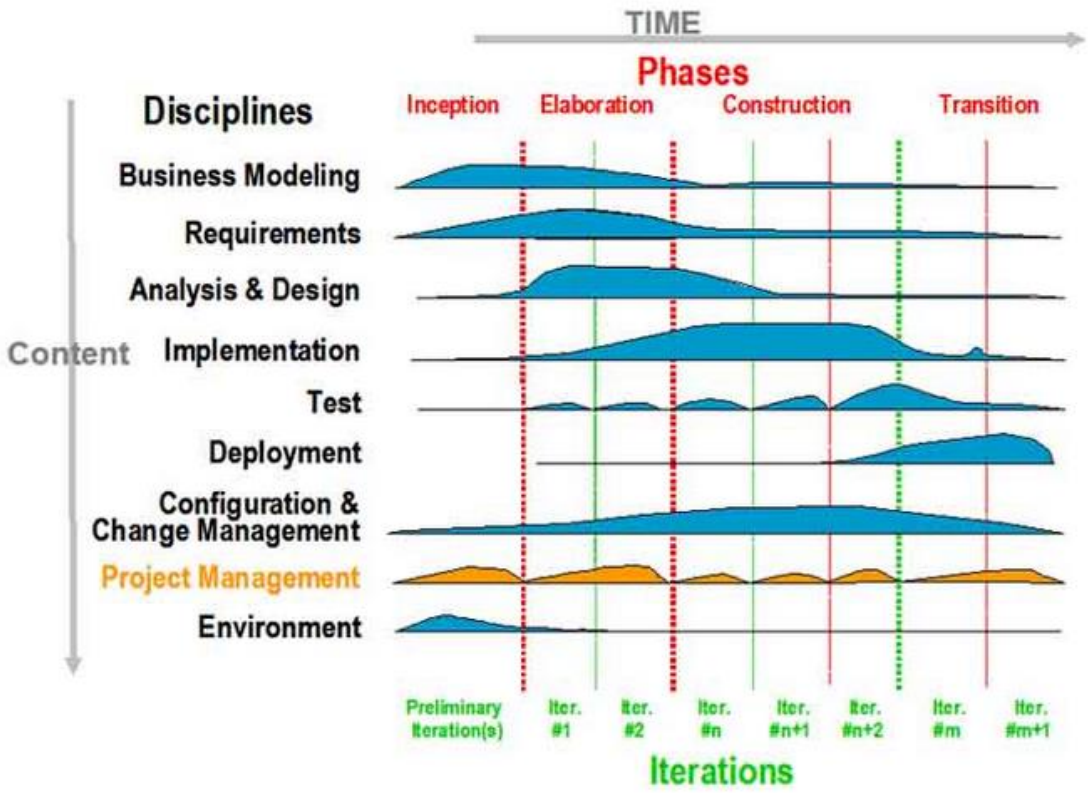

Imagen 3. Los elementos clave de IBM Rational Unified Process
FUENTE: http://www.ibm.com/developerworks/rational/library/4763.html

7.3.1 Descripción de las actividades. “Dependiendo de la iteración del proceso el equipo de desarrollo puede realizar 7 tipos de actividades en este:

➢ **Fase de inicio.** Durante la fase de inicio las iteraciones hacen ponen mayor énfasis en actividades modelado del negocio y de requisitos.

• **Modelado del negocio**. En esta fase el equipo se familiarizará más al funcionamiento de la empresa, sobre conocer sus procesos.

1. Entender la estructura y la dinámica de la organización para la cual el sistema va ser desarrollado.

2. Entender el problema actual en la organización objetivo e identificar potenciales mejoras.

3. Asegurar que clientes, usuarios finales y desarrolladores tengan un entendimiento común de la organización objetivo.

• **Requisitos**. En esta línea los requisitos son el contrato que se debe cumplir, de modo que los usuarios finales tienen que comprender y aceptar los requisitos que especifiquemos.

1. Establecer y mantener un acuerdo entre clientes y otros *stakeholder* sobre lo que el sistema podría hacer.

2. Proveer a los desarrolladores un mejor entendimiento de los requisitos del sistema.

3. Definir el ámbito del sistema.

4. Proveer una base para estimar costos y tiempo de desarrollo del sistema.

5. Definir una interfaz de usuarios para el sistema, enfocada a las necesidades y metas del usuario.

➢ **Fase de construcción.**

• **Implementación**. Se implementan las clases y objetos en ficheros fuente, binarios, ejecutables y demás. El resultado final es un sistema ejecutable.

1. Planificar qué subsistemas deben ser implementados y en qué orden deben ser integrados, formando el Plan de Integración.

2. Cada implementador decide en qué orden implementa los elementos del subsistema.

3. Si encuentra errores de diseño, los notifica.

4. Se integra el sistema siguiendo el plan.

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

1. Encontrar y documentar defectos en la calidad del software.

2. Generalmente asesora sobre la calidad del software percibida.

3. Provee la validación de los supuestos realizados en el diseño y especificación de requisitos por medio de demostraciones concretas.

4. Verificar las funciones del producto de software según lo diseñado.

5. Verificar que los requisitos tengan su apropiada implementación.

• **Despliegue**. Esta actividad tiene como objetivo producir con éxito distribuciones del producto y distribuirlo a los usuarios. Las actividades implicadas incluyen:

1. Probar el producto en su entorno de ejecución final.

2. Empaquetar el software para su distribución.

3. Distribuir el software.

4. Instalar el software.

5. Proveer asistencia y ayuda a los usuarios.

6. Formar a los usuarios y al cuerpo de ventas.

7. Migrar el software existente o convertir bases de datos."[9]

[9]GALLEGO GOMEZ Juan Pablo, Septiembre 16 de 2007, Universidad tecnológica de Pereira, Fundamentos de la Metodología RUP, Disponible en http://es.scribd.com/doc/297224/RUP.

7.4 METODOLOGÍA ICONIX

7.4.1 Historia. Los desarrollos de aplicaciones van cambiando por innovaciones tecnológicas, estrategias de mercado y otros avatares de la industria de la informática, esto lleva a los desarrolladores de aplicaciones a evolucionar para obtener aplicaciones en menor tiempo, más vistosas y de menor costo.

Los usuarios exigen calidad frente a los requisitos y los desarrollos de aplicaciones deben contar con técnicas y herramientas logrando satisfacer las necesidades de los usuarios y obteniendo sistemas fáciles de mantener, extender y modificar.

Claro está, que es indispensable, el uso de una metodología para el desarrollo de sistemas, logrando un sistema sano, que cumpla con los requerimientos de los usuarios.

Una metodología consiste en un lenguaje de modelamiento y un proceso. El lenguaje de modelamiento es la notación gráfica (incluye diferentes tipos de diagramas) en este caso UML. El proceso define quien debe hacer qué, cuándo y cómo alcanzar un objetivo.

La realidad de la industria del software de gestión impone la adopción de procesos ágiles de desarrollo para lograr competitividad, ya que el proceso de desarrollo de software trae aparejado: altos costos, alta complejidad, dificultades de mantenimiento y una disparidad entre las necesidades de los usuarios y los productos desarrollados.

Reflejo de ello, en el ámbito internacional, es la creciente consolidación de la filosofía AGILE. El objetivo principal de un método ágil es minimizar la documentación de desarrollo empleándola fundamentalmente como vehículo de comprensión de problemas dentro del grupo de trabajo y de comunicación con los usuarios.

Esta herramienta importa una contribución para la comunidad informática dedicada al desarrollo de sistemas de gestión, dado que implica la adopción de una metodología simple y precisa que favorece la participación de los usuarios finales y mantiene a todo desarrollo permanentemente documentado.

La participación y el compromiso de los usuarios finales en desarrollos basados en esta herramienta se presumen garantizados debido a que los modelos empleados para las especificaciones son de un alto nivel de abstracción y comprensibles para personas no especializadas; además el modelo dinámico tal como el de casos de uso en el Proceso Unificado de Desarrollo permite verificar la completitud y rastrear el cumplimiento de sistemas a partir de la especificación

del diseño de interfaces, optimiza las relaciones contractuales facilitando la aprobación de fases y ciclos de evolución.

- Las tres características fundamentales de ICONIX son:

A. Iterativo e incremental: varias iteraciones ocurren entre el desarrollo del modelo del dominio y la identificación de los casos de uso. El modelo estático es incrementalmente refinado por los modelos dinámicos.

B. Trazabilidad: cada paso está referenciado por algún requisito. Se define trazabilidad como la capacidad de seguir una relación entre los diferentes artefactos producidos.

C. Dinámica del UML: La metodología ofrece un uso "dinámico del UML" como los diagramas del caso de uso, diagramas de secuencia y de colaboración.

7.4.2 Las Tareas de ICONIX. Rosenberg y Scoot destacan un análisis de requisitos, un análisis y diseño preliminar, un diseño y una implementación como las principales tareas.

1. Análisis de Requisitos. Identificar en el "mundo real" los objetos y todas las relaciones de agregación y generalización entre ellos. Utilizar un diagrama de clases de alto nivel definido como modelo de dominio.

El trabajo es iniciado con un relevamiento informal de todos los requisitos que en principio deberían ser parte del sistema. Luego con los requisitos se construye el diagrama de clases, que representa las agrupaciones funcionales con que se estructura el sistema que se desarrolla.

De generarse el sistema a este nivel de especificación, se obtendría el menú principal del sistema con la interfaces iníciales de los casos o actividades de cada división funcional. Los diagramas del segundo nivel o superior, accesibles a partir de cada escenario o estado del nivel anterior, representan los casos, actividades y secuencias de interacción de cada división funcional.

En estos se pueden reutilizar interfaces ya definidas en otros diagramas, representándose con bordes tenues.

Presentar, si es posible, una prototipación rápida de las interfaces del sistema, los diagramas de navegación, etc., de forma que los clientes puedan comprender mejor el sistema propuesto.

Con el prototipo se espera que las especificaciones iníciales estén incompletas. En general se necesita entre 2 y 3 reuniones para establecer las especificaciones iníciales. La rapidez con la que se genera el sistema es esencial para que no se pierda el estado de ánimo sobre el proyecto y que los usuarios puedan comenzar a evaluar la aplicación en la mayor brevedad posible.

Durante la evaluación se debe capturar información sobre lo que les gusta y lo que les desagrada a los usuarios, al mismo tiempo poner atención al porque reaccionan los usuarios en la forma en que lo hacen.

Los cambios al prototipo son planificados con los usuarios antes de llevarlos a cabo. El proceso se repite varias veces y finaliza cuando los usuarios y analistas están de acuerdo en que el sistema ha evolucionado lo suficiente como para incluir todas las características necesarias o cuando es evidente que no se obtendrá mayor beneficio con una iteración adicional.

El diseño de prototipos es una técnica popular de ingeniería para desarrollar modelos a escala (o simulados) de un producto o sus componentes. Cuando se aplica al desarrollo de sistemas de información el diseño de prototipos implica la creación de un modelo o modelos operativos de trabajo de un sistema o subsistema.

Existen cuatro tipos de prototipos:

1. Prototipo de viabilidad: para probar la viabilidad de una tecnología específica aplicable a un sistema de información.

2. Prototipo de Necesidades: utilizado para “descubrir” las necesidades de contenido de los usuarios con respecto a la empresa.

3. Prototipo de Diseño: es el que usa Iconix. Se usa para simular el diseño del sistema de información final. Se centra en la forma y funcionamiento del sistema deseado. Cuando un analista crea un prototipo de diseño, espera que los usuarios evalúen este prototipo, como si formara parte del sistema final. Los usuarios deberían evaluar la facilidad de aprendizaje y manejo del sistema, así como el aspecto de las pantallas y los informes y los procedimientos requeridos para utilizar el sistema. Estos prototipos pueden servir como especificaciones parciales de diseño o evolucionar hacia prototipos de información.

4. Prototipo de Implantación: es una extensión de los prototipos de diseño donde el prototipo evoluciona directamente hacia el sistema de producción.

Los prototipos de pantallas también proporcionan una manera de obtener las reacciones de los usuarios hacia la cantidad de información presentada sobre la pantalla de visualización. Tal vez el usuario decida que un diseño en particular es muy denso ya que existen demasiados detalles sobre la pantalla.

En otros casos la información sobre la pantalla aunque no es excesiva en el sentido de causar que la pantalla se vuelva densa, tal vez sea mucho mayor que la que un individuo necesita durante todo el tiempo.

Identificar los casos de uso del sistema mostrando los actores involucrados.

Utilizar para representarlo el modelo de casos de uso.

Los casos de uso describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista de un usuario; permiten definir los límites del sistema y las relaciones entre el sistema y el entorno.

Un caso de uso es una manera específica de utilizar un sistema. Es la imagen de una funcionalidad del sistema, desencadenada en respuesta a la estimulación de un actor externo.

El modelo de los casos de uso comprende los actores, el sistema y los propios casos de uso. El conjunto de funcionalidades de un sistema se determina examinando las necesidades funcionales de cada actor.

Los casos de usos reubican la expresión de las necesidades sobre los usuarios partiendo del punto de vista muy simple que dice que un sistema se construye ante todo para sus usuarios. La estructuración del método se efectúa respecto a las interacciones de una sola categoría de usuarios a la vez; esta partición del conjunto de necesidades reduce considerablemente la complejidad de la determinación de las necesidades.

Los casos de uso permiten a los usuarios estructurar y articular sus deseos; les obligan a definir la manera como querrían interactuar con el sistema, a precisar que informaciones quieren intercambiar y a describir lo que debe hacerse para obtener el resultado esperado. Los casos de uso concretan el futuro sistema en una formalización próxima al usuario, incluso en ausencia de un sistema a criticar.

Organizar los casos de uso en grupos, o sea, utilizar los diagramas de paquetes.

Asociar los requisitos funcionales con los casos de uso y con los objetos del dominio (trazabilidad).

Un importante aspecto de ICONIX es que un requisito se distingue explícitamente de un caso de uso. En este sentido, un caso de uso describe un comportamiento; un requisito describe una regla para el comportamiento.

Además, un caso de uso satisface uno o más requisitos funcionales; un requisito funcional puede ser satisfecho por uno o más casos de uso.

2. Análisis y Diseño Preliminar. Debería usarse un estilo consistente que sea adecuado al contexto del proyecto.

a) Describir los casos de uso, como un flujo principal de acciones, pudiendo contener los flujos alternativos y los flujos de excepción. La principal sugerencia de ICONIX, en esta actividad, es que no se debe perder mucho tiempo con la descripción textual.

b) Realizar un diagrama de robustez. Se debe ilustrar gráficamente las interacciones entre los objetos participantes de un caso de uso. Este diagrama permite analizar el texto narrativo de cada caso de uso e identificar un conjunto inicial de objetos participantes de cada caso de uso.

c) Estos objetos que forman parte de los diagramas de robustez se clasifican dentro de los tres tipos siguientes:

- Objetos de interfaz: usados por los actores para comunicarse con el sistema. Son con los que los actores interactúan con el sistema, generalmente como ventanas, pantalla, diálogos y menús.

- Objetos entidad: son objetos del modelo del dominio. Son a menudo tablas y archivos que contiene archivos para la ejecución de dicho caso de uso.

- Objetos de control: es la unión entre la interfaz y los objetos entidad. Sirven como conexión entre los usuarios y los datos. Los controles son "objetos reales" en un diseño, pero usualmente sirven como una especie de oficinista para asegurar que no se olvide ninguna funcionalidad del sistema la cual puede ser requerida por algún caso de uso.

Esta técnica tan simple pero poderosa sirve como interfaz entre el "que" y el "como" de un análisis. Además el análisis de robustez provee de una gran ayuda a saber si las especificaciones del sistema son razonables.

El análisis de robustez facilita el reconocimiento de objetos. Esto es un paso crucial ya que es casi seguro que se olvida algunos objetos durante el modelado del dominio; y de esta manera se podrán identificar antes de que esto cause problemas serios, además sirve para identificar más y mejores clases, antes del desarrollo del diagrama de secuencias.

Las reglas básicas que se deben aplicar al realizar los diagramas de análisis de robustez:

1. Actores solo pueden comunicarse con objetos interfaz.

2. Las interfaces solo pueden comunicarse con controles y actores.

3. Los objetos entidad solo pueden comunicarse con controles.

4. Los controles se comunican con interfaces, objetos identidad y con otros controles pero nunca con actores.

5. Tomando en cuenta que los objetos entidad y las interfaces son sustantivos y los controles son verbos. Se pueden enunciar de manera sencilla que los sustantivos nunca se comunican con otros sustantivos, pero los verbos, si pueden comunicarse con otros verbos y a su vez con sustantivos.

c) Actualizar el diagrama de clases ya definido en el modelo de dominio con las nuevas clases y atributos descubiertas en los diagramas de robustez.

3. Diseño. Especificar el comportamiento a través del diagrama de secuencia.

Para cada caso de uso identificar los mensajes entre los diferentes objetos. Es necesario utilizar los diagramas de colaboración para representar la interacción entre los objetos.

El diagrama de secuencia muestra interacciones entre objetos según un punto de vista temporal. El contexto de los objetos no se representa de manera explícita como en los diagramas de colaboración. La representación se concentra sobre la expresión de las interacciones.

A pesar de que a partir de los diagramas de casos de uso y de los diagramas de robustez ya tenemos entre un 75 y 80 por ciento de atributos de nuestras clases identificados, es hasta el diagrama de secuencia donde se empiezan a ver qué métodos llevaran las clases de nuestro sistema.

Esto se debe a que hasta que vemos interactuando a los objetos de nuestras clases con los actores y con otros objetos de manera dinámica, hasta ese momento tenemos suficiente información como para poder empezar a especificar los métodos de nuestras respectivas clases.

El diagrama de secuencia es el núcleo de nuestro modelo dinámico y muestra todos los cursos alternos que pueden tomar todos nuestros casos de uso. Los diagramas de secuencia se componen de 4 elementos que son: el curso de acción, los objetos, los mensajes y los métodos (operaciones).

Terminar el modelo estático, adicionando los detalles del diseño en el diagrama de clases.

Verificar si el diseño satisface todos los requisitos identificados.

4. Implementación.

1. Utilizar el diagrama de componentes, si fuera necesario para apoyar el desarrollo. Es decir, mostrar la distribución física de los elementos que componen la estructura interna del sistema.

El diagrama de componentes describe los elementos físicos y sus relaciones en el entorno de realización. El diagrama muestra las opciones de realización.

2. Escribir/ Generar el código.

La importancia de la interactividad, interactividad, accesibilidad y navegación en el software harán que el usuario se sienta seguro y cómodo al poder hacer uso de la aplicación sin inconvenientes tales como son los problemas de

comunicación. Este y otros problemas como la realización de cambios, son factores que deben ser tenidos en cuenta.

Pero además debemos tener en cuenta factores como:

1. La Reusabilidad: que es la posibilidad de hacer uso de los componente en diferentes aplicaciones.

2. La Extensibilidad: que consiste en modificar con facilidad el software.

3. La Confiabilidad: realización de sistemas descartando las posibilidades de error.

a. Realizar pruebas. Test de unidades, de casos, datos y resultados. Test de integración con los usuarios para verificar la aceptación de los resultados.[10]

7.5 METODOLOGÍA SCRUM

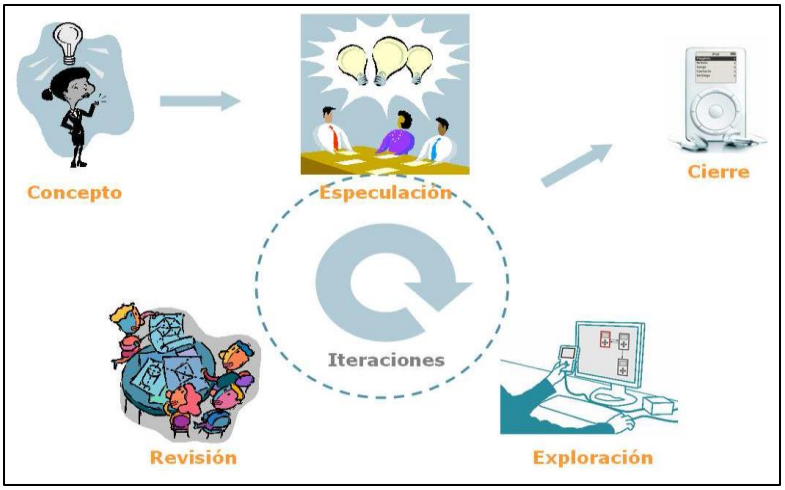

Imagen 4. Estructura del desarrollo ágil
FUENTE: http://www.navegapolis.net/files/s/NST-010_01.pdf

Se comienza con la visión general del producto, especificando y dando detalle a las funcionalidades o partes que tienen mayor prioridad de desarrollo y que pueden llevarse a cabo en un periodo de tiempo breve (normalmente de 30 días).

Cada uno de estos periodos de desarrollo es una iteración que finaliza con la producción de un incremento operativo del producto.

Estas iteraciones son la base del desarrollo ágil, y Scrum gestiona su evolución a través de reuniones breves diarias en las que todo el equipo revisa el trabajo realizado el día anterior y el previsto para el día siguiente.

[10] DE SAN MARTIN OLIVA CARLA REBECA PATRICIA, Metodología ICONIX, disponible en: http://www.portalhuarpe.com.ar/Seminario09/archivos/MetodologiaICONIX.pdf

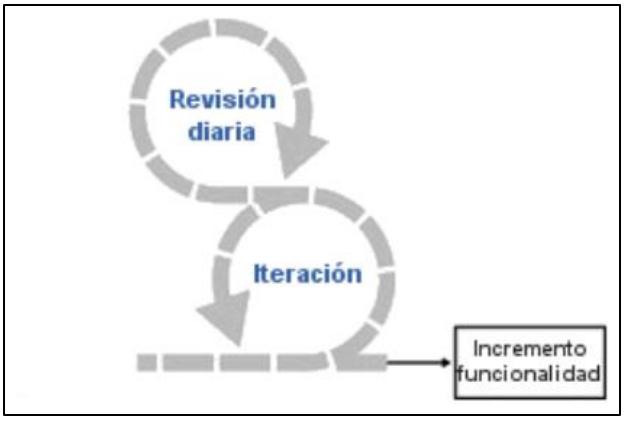

Imagen 5. Estructura central de Scrum
FUENTE: http://www.navegapolis.net/files/s/NST-010_01.pdf

- **Control de la evolución del proyecto**. Scrum controla de forma empírica y adaptable la evolución del proyecto, empleando las siguientes prácticas de la gestión ágil:

- **Revisión de las Iteraciones.** Al finalizar cada iteración (normalmente 30 días) se lleva a cabo una revisión con todas las personas implicadas en el proyecto. Este es el periodo máximo que se tarda en reconducir una desviación en el proyecto o en las circunstancias del producto.

- **Desarrollo incremental.** Durante el proyecto, las personas implicadas no trabajan con diseños o abstracciones. El desarrollo incremental implica que al final de cada iteración se dispone de una parte del producto operativa que se puede inspeccionar y evaluar.

- **Desarrollo evolutivo.** Los modelos de gestión ágil se emplean para trabajar en entornos de incertidumbre e inestabilidad de requisitos.

Intentar predecir en las fases iníciales cómo será el producto final, y sobre dicha predicción desarrollar el diseño y la arquitectura del producto no es realista, porque las circunstancias obligarán a remodelarlo muchas veces. Para qué predecir los estados finales de la arquitectura o del diseño si van a estar cambiando.

En Scrum se toma a la inestabilidad como una premisa, y se adoptan técnicas de trabajo para permitir esa evolución sin degradar la calidad de la arquitectura que se irá generando durante el desarrollo.

El desarrollo Scrum va generando el diseño y la arquitectura final de forma evolutiva durante todo el proyecto. No los considera como productos que deban realizarse en la primera “fase” del proyecto.

- **Auto-organización.** Durante el desarrollo de un proyecto son muchos los factores impredecibles que surgen en todas las áreas y niveles. La gestión predictiva confía la responsabilidad de su resolución al gestor de proyectos.

En Scrum los equipos son auto-organizados (no auto-dirigidos), con margen de decisión suficiente para tomar las decisiones que consideren oportunas.

Colaboración. Las prácticas y el entorno de trabajo ágiles facilitan la colaboración del equipo. Ésta es necesaria, porque para que funcione la auto organización como un control eficaz cada miembro del equipo debe colaborar de forma abierta con los demás, según sus capacidades y no según su rol o su puesto.

Visión general del proceso. Scrum denomina "sprint" a cada iteración de desarrollo y recomienda realizarlas con duraciones de 30 días.

El sprint es por tanto el núcleo central que proporciona la base de desarrollo iterativo e incremental.

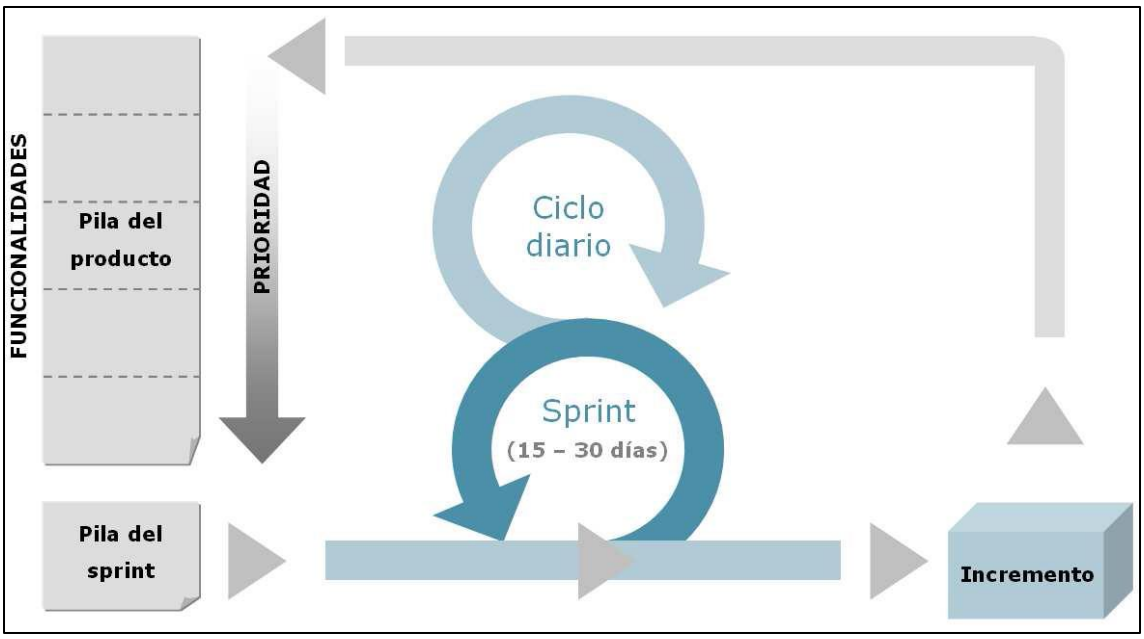

Imagen 6. Los elementos que conforman el desarrollo
FUENTE: http://www.navegapolis.net/files/s/NST-010_01.pdf

- **Las reuniones**

Planificación de sprint: Jornada de trabajo previa al inicio de cada sprint en la que se determina cuál va a ser el trabajo y los objetivos que se deben cumplir en esa iteración.

Reunión diaria: Breve revisión del equipo del trabajo realizado hasta la fecha y la previsión para el día siguiente.

Revisión de sprint: Análisis y revisión del incremento generado.

- **Los elementos**

Pila del producto: lista de requisitos de usuario que se origina con la visión inicial del producto y va creciendo y evolucionando durante el desarrollo.

Pila del sprint: Lista de los trabajos que debe realizar el equipo durante el sprint para generar el incremento previsto.

Los roles. Scrum clasifica a todas las personas que intervienen o tienen interés en el desarrollo del proyecto en: propietario del producto, equipo, gestor de Scrum (también Scrum Manager o Scrum Master) y "otros interesados".

Propietario del producto: El responsable de obtener el mayor valor de producto para los clientes, usuarios y resto de implicados.

Equipo de desarrollo: grupo o grupos de trabajo que desarrollan el producto.

Scrum Manager: gestor de los equipos que es responsable del funcionamiento de la metodología Scrum y de la productividad del equipo de desarrollo.

- **Valores.** Scrum es una "carrocería" para dar forma a los principios ágiles. Es una ayuda para organizar a las personas y el flujo de trabajo; como lo pueden ser otras propuestas de formas de trabajo ágil: Cristal, DSDM, etc.

Delegación de atribuciones (*empowerment*) al equipo para que pueda auto-organizarse y tomar las decisiones sobre el desarrollo.

Respeto entre las personas. Los miembros del equipo deben confiar entre ellos y respetar sus conocimientos y capacidades.

Responsabilidad y auto-disciplina (no disciplina impuesta).

Trabajo centrado en el desarrollo de lo comprometido.

Información, transparencia y visibilidad del desarrollo del proyecto.[11]

[11] PALACIO JUAN, El Modelo Scrum, 2006. Disponible en: http://www.navegapolis.net/files/s/NST-010_01.pdf

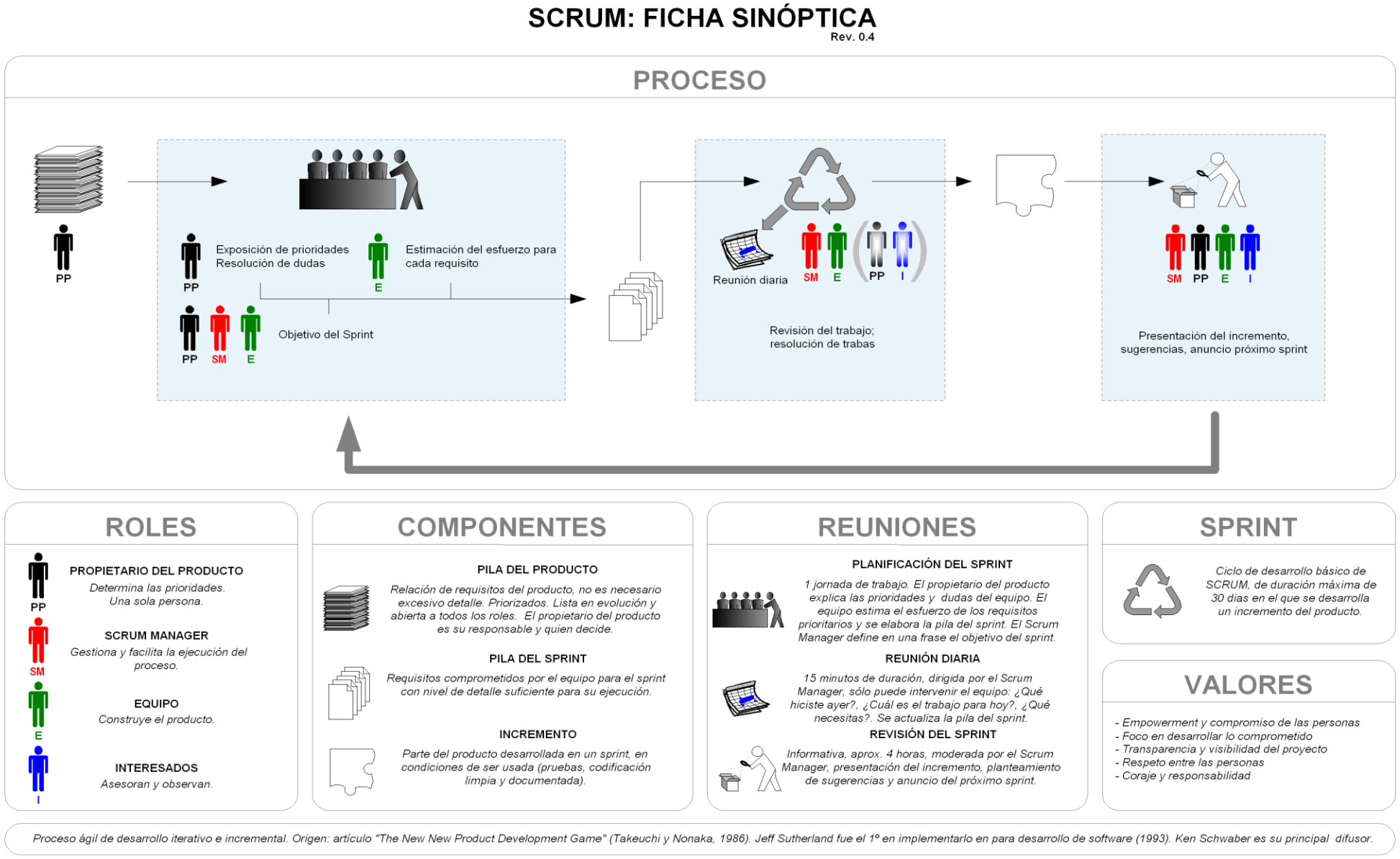

Imagen 7. Scrum ficha sinóptica
FUENTE: http://www.navegapolis.net/files/s/NST-010_01.pdf

A continuación se mostraran las ventajas y desventajas en un modo de resumen para las metodologías que se han tenido en cuenta para este proyecto.

VENTAJAS			
XP	**RUP**	**ICONIX**	**SCRUM**
Permite realizar el desarrollo del sistema en parejas para complementar los conocimientos;	Mitigación temprana de posibles riesgos altos	Es un modelo pequeño y firme que no desecha el análisis y el diseño.	Se obtiene software lo más rápido posible y este cumple con los requerimientos más importantes.
Implementa una forma de trabajo donde se adapte fácilmente a las circunstancias	Progreso visible en las primeras etapas	Usa un análisis de robustez que reduce la ambigüedad al describir los casos.	Se trabaja en iteraciones cortas, de alto enfoque y total transparencia.
Apropiado para entornos volátiles	Temprana retroalimentación que se ajuste a las necesidades reales	Es usado en proyectos más ligeros que los usados en RUP, por lo que tiene un mayor campo de aplicabilidad.	Se acepta que el cambio es una constante universal y se adapta el desarrollo para integrar los cambios que son importantes.
Estar preparados para el cambio, significa reducir su coste.	Gestión de la complejidad	Proporciona suficientes requisitos y documentación de diseño, pero sin parar el análisis.	Se incentiva la creatividad de los desarrolladores haciendo que el equipo sea auto administrado.
Planificación más transparente para nuestros clientes, conocen las fechas de entrega de funcionalidades. Vital para su negocio	Conocimiento adquirido en una iteración puede aplicarse de iteración a iteración	Es refinado y actualizado a lo largo del proyecto, por lo que siempre refleja la actual comprensión del problema de espacio.	Se mantiene la efectividad del equipo habilitando y protegiendo un entorno libre de interrupciones e interferencias.
Permitirá definir en cada iteración cuales son los objetivos de la siguiente		Proceso ágil para obtener un sistema informático.	Permite producir software de una forma consistente, sostenida y competitiva.
Permite tener realimentación de los usuarios muy útil.		Dedicada a la construcción de sistemas de gestión de pequeña y mediana complejidad con la participación de los usuarios finales.	Las reuniones se dedican a inconvenientes recientes, evitando el estancamiento
La presión está a lo largo de todo el proyecto y no en una entrega final		Los usuarios se hacen participantes más activos en los desarrollos del sistema. Suelen mostrarse más interesados en los prototipos de trabajo que en las especificaciones de diseño.	
El código es sencillo y entendible		Las necesidades se simplifican por el hecho de que los usuarios no son capaces de enumerar detalladamente sus necesidades hasta que ven un prototipo	

Tabla 1. Ventajas metodologías de desarrollo de software
FUENTE: Los autores

DESVENTAJAS			
XP	**RUP**	**ICONIX**	**SCRUM**
No se tiene la definición del costo y el tiempo de desarrollo	Método Pesado	No puede ser usado para proyectos grandes.	Requiere delegar responsabilidades al equipo, incluso permite fallar si es necesario.
El sistema va creciendo después de cada entrega al cliente y nadie puede decir que el cliente no querrá una función más	Por el grado de complejidad puede ser no muy adecuado	Necesita información rápida y puntual de los requisitos, el diseño y las estimaciones.	Es una metodología que difiere del resto, y esto causa cierta resistencia en su aplicación para algunas personas
Se necesita de la presencia constante del usuario, lo cual en la realidad es muy difícil de lograr.		Se debe conocer los diagramas UML.	
		Los prototipos suelen pasar a las fases de análisis y diseño con demasiada rapidez. Esto empuja al analista a pasar demasiado rápido a la codificación, sin haber comprendido las necesidades y los problemas.	

Tabla 2. Desventajas metodologías de desarrollo de software
FUENTE: Los autores

Metodologías de Desarrollo de Software				
	Metodologías Agiles		Otras Metodologías	
Fases	**Scrum**	**XP**	**Iconix**	**RUP**
1	Relevamiento Procesos Negocio	Historia de Usuario	Relevamiento Informal de Requisitos	Documento Visión Proyecto
	Definición y Secuenciamiento Actividades	Plan de Estrategias	Diagrama de Clases	Casos de Uso
	Definición de Alcance	Velocidad de Proyecto	Casos de Uso	Glosario Inicial Proyecto
	Estimación de Tiempos	Iteraciones	Menú Principal del Sistema con Interfaces Iniciales	Caso de Uso Inicial de Negocio (Contexto del Negocio, Criterios de Éxito y Planificación Financiera)
	Definición de Recursos	Rotaciones	Prototipación Rápida (Prototipo de Diseño)	Estudio Inicial Riesgos
	Análisis de Riesgos	Reuniones Seguimiento Diarias	2 a 3 Reuniones para Especificaciones Iniciales	Plan Proyecto (Fases e Iteraciones)
	Estimación de Costos	40 Horas por Semana (Sin Horas Extras)	Evaluación Usuarios (Que le Gusta y que No Gusta)	Requisitos, Modelar Interfaz de Usuario (Diseño Lógico) y Prototipo Interfaz Usuario (Diseño Físico)
2	Definiciones Funcionales	Metáfora del Sistema	Describir Flujos Alternativos	Modificación Requisitos y Casos de Uso
	Definición de Requisitos (Casos de Uso)	Tarjeta CRC (Clase, Responsabilidad y Colaboración)	Describir Flujos de Excepción	Descripción Arquitectura del Software
	Planificación Etapas Posteriores	Soluciones Puntuales	Diagrama de Robustez	Prototipo Ejecutable de Arquitectura
	Ajuste Tiempos Prestablecidos	Funcionalidad Mínima	Actualizar Diagramas de Clases	Lista Riesgos
		Reciclaje (Código Limpio, Fácil Comprender, Modificar y Ampliar)		Manual Preliminar Usuario
3	Diagrama Entidad Relación	Disponibilidad del Cliente	Diagrama de Secuencia	Producto Software Integrado sobre Plataforma Adecuada
	Diseño Interfaces de Usuario	Unidad de Prueba	Diagramas de Colaboración	Manual Usuario
	Diseño Integraciones	Programación por Pareja	Verificar si el Diseño satisface todos los Requisitos	Descripción Versión Actual

	Pruebas Puntos Críticos Proyecto	Integración Código		Planificar Subsistemas (Cuales son Implementados y Orden de Integración, Según Plan Integración)
				Verificación Errores Diseño
				Probar Producto Entorno Final
				Empaquetar y Distribuir Software
				Instalación, Asistencia, Capacitación y Migración del Software
4	Programación y Desarrollo de Todos los Componentes y Funcionalidades	Implantación (Después de Superar Unidades Test)	Diagrama de Componentes	Marco Trabajo para Gestión de Proyectos de Software
	Implementación Estructura de Datos y Procedimientos	Protección Contra Fallos	Escribir/Generar Código (Reusabilidad, Extensibilidad, Confiabilidad)	Guías Prácticas para Planeación, Contratar Personal, Ejecutar y Monitorear Proyecto
	Elaboración Documentación Técnica y Ajustes Funcionales	Pruebas de Aceptación	Realizar Pruebas (Test de Unidades, Casos, Datos, Resultados e Integración)	Marco Trabajo Gestión de Riesgos
	Implementación Integraciones			Selección, Adquisición y Configuración de Herramientas
	Pruebas Usabilidad, Funcionalidad y Carga Datos			Configuración y Mejora del Proceso y Servicio Técnico
5	Aplicación Ambiente Producción			
	Elaboración Manuales Operativos			
	Integración Ambiente Producción con Terceras Partes			

Tabla 3. Comparación metodologías de desarrollo de software
FUENTE: Los autores

	1	2	3	4	5
Scrum	Planificacion	Analisis	Diseño	Construccion y Pruebas	Implementacion
XP	Planificacion	Diseño	Desarrollo	Pruebas	
Iconix	Analisis de Requisitos	Analisis y Diseño Preliminar	Diseño	Implementacion	
RUP	Inicio	Elaboracion	Construccion	Transicion	

ASPECTOS COMUNES METODOLOGIAS DE DESARROLLO									
	FASE 1				FASE 2		FASE 3	FASE 4	
Metodologias \ Aspectos Comunes	**Relevamiento Informal de Requisitos**	**Plan de Estrategias**	**Estudio de Riesgos**	**Interfaces Iniciales**	**Definicion de Requisitos**	**Funcionalidad**	**Pruebas de Integracion**	**Programacion y desarrollo de los componentes**	**Pruebas**
SCRUM	Relevamiento Proceso de Negocio	Definicion y Secuenciamiento de Actividades Definicion de Alcance Estimacion de Tiempos Definicion de Recursos Estimacion de costos	Analisis de Riesgos		Definicion de Requisitos (casos de uso)	Definiciones Funcionales	Diseño Integraciones Pruebas Puntos Criticos Proyecto	Programacion y desarrollo de todos los componentes y funcionalidades	Pruebas Usabilidad, Funcionalidad y Carga Datos
XP	Historia de Usuario	Plan de Estrategias Reuniones de seguimiento diarias				Funcionalidad Minima	Unidad de Prueba	Escribir / Generar codigo	Pruebas de Aceptacion
ICONIX	Relevamiento Informal de Requisitos	2 a 3 Reuniones para especificaciones iniciales		Menu Principal del Sistema con Interfaces Iniciales Prototipacion Rapida (Prototipo de Diseño)	Definir Flujos Alternativos Describir Flujos de Excepcion		Verificar si el diseño satisface todas los requisitos		Realizar Pruebas (Test de unidades, Casos, Datos, Resultados e Integracion)
RUP	Documento Vision Proyecto Glosario Inicial Proyecto	Plan Proyecto (fases e iteraciones)	Estudio Inicial de Riesgos	Requisitos Modelar Interfaz de Usuario (Diseño Logico) Prototipo Interfaz Usuario (Diseño Fisico)	Modificacion de Requisitos y Casos de Uso		Producto software integrado sobre plataforma adecuada Planificar subsistemas (cuales son implementados y orden de integracion, según plan integracion) Verificacion errores de diseño Probar producto entorno final		

Tabla 4. Cuadro comparativo aspectos comunes metodologías de desarrollo
FUENTE: Los autores.

Las metodologías que se estudiaron en este trabajo son las más comunes en el entorno de desarrollo de software a nivel mundial, sin embargo, se encontró que en el país, no son las más utilizadas y aceptadas por las empresas debido a las diferentes magnitudes de los proyectos, los costos que de su implementación, el personal dedicado a aplicarla, entre otros factores, los cuales serán abordados en los capítulos siguientes.

En la Tabla 3 se comparan las diferentes metodologías, donde se encuentra que en las fases de Implementación o fases donde se empieza a escribir el código, se propone un código limpio y la reusabilidad del código para lo cual recomendamos el uso de Frameworks (Ver Anexo B) para cumplir con estas indicaciones.

En la Tabla 4 se compara cada fase de las metodologías de desarrollo de software, estableciendo los aspectos comunes entre ellas, lo cual permite acercarse más a la determinación de los factores que inciden en el proceso para el desarrollo de una metodología de software ágil.

7.6 UTILIZACIÓN DE LOS ELEMENTOS DE MODELADO EN LAS METODOLOGÍAS DE DESARROLLO

La siguiente Tabla 5 se realizó para identificar los diagramas UML que se usan en las diferentes metodologías como lo son ICONIX, XP, RUP, SCRUM las cuales permiten tener una base para el modelado de sistemas de software.

	Comparacion Diagramas										
Metodologias	Casos de Uso	Clases	Secuencia	Componentes	Robustez	Implementacion	Entidad Relacion	Estado	Colaboracion	Actividad	Despliegue
ICONIX	X	X	X	X	X		X		X		
XP	X	X				X	X				
RUP	X	X	X	X			X	X	X	X	X
SCRUM	X	X	X				X			X	

Tabla 5. Elementos de modelado en las metodologías agiles
FUENTE: Los autores.

7.7 METODOLOGÍAS UTILIZADAS POR LAS EMPRESAS DESARROLLADORES DE SOFTWARE

EMPRESA\METODOLOGIA	RUP	ICONIX	XP	SCRUM	OTRA	PROPIA	TIPO
PSL	x			x			
IT GROUP						x	Ágil
NEXOS SOFTWARE					MSF(Microsoft)	x	Tradicional
ASESOFTWARE	x			x	CASE METHOD(Oracle)		
PALMERA SOFT						x	Ágil
BIOSALC			x		Elección del Desarrollador	x	Tradicional
ISE & E				x			
SOFTENG				x			
MVM INGENIERIA DE SOFTWARE					UP (Unified Process)	x	Tradicional
ARQUITESOFT						x	Ágil
SAP	x		x				

Tabla 6. Comparación metodología-empresa
FUENTE: Los autores.

7.7.1 PSL

PSL es una compañía del sector informático, que se destaca mundialmente por la adopción de las mejores prácticas existentes en Ingeniería de software e Ingeniería de Sistemas en el mundo.

En esta compañía presenta dos grandes líneas de negocio: el desarrollo de aplicaciones por encargo (aplicaciones web, aplicaciones móviles, aplicaciones de misión crítica) y el desarrollo de productos pre-programados de software (sistemas ERP, sistemas de gestión de garantías, plataformas web-banking, entre otras).

La compañía PSL existe hace más de 28 años**,** tiene grandes clientes como Ecopetrol, el Canal de Panamá y Bancolombia, hasta cientos de exitosas y pujantes empresas PYME[12].

7.7.1.1 Metodología utilizada por PSL[13]

- RUP utilizado para desarrollos largos.
- SCRUM utilizado para desarrollos cortos.

[12] PSL, Breve Reseña de PSL, Disponible en: http://www.psl.com.co/psl/sobre-nuestra-compania.html
[13] PSL, Disponible en: http://www.psl.com.co/servicios/desarrollo-a-la-medida/metodologias-desarrollo-de-software.html

7.7.2 IT GROUP

Es una empresa Colombiana que se dedica a brindar soluciones a las organizaciones mediante el desarrollo de software a la medida.

En la empresa IT Group se piensa que los sistemas deben responder a las necesidades propias de cada organización, de tal forma que la solución se adapte al usuario y no el usuario a la solución**.** Los productos son desarrollados con el fin de que los clientes puedan, al final del proceso, evidenciar una mejora dentro de su empresa, bien sea porque ahorran tiempo, dinero o porque simplifican sus procesos. Uno de los objetivos principales es que los clientes no sólo estén satisfechos con el producto que reciben, sino que además éste supere sus expectativas[14].

7.7.2.1 Metodología utilizada por IT GROUP[15]

La empresa utiliza una metodología propia la cual está adecuada a las necesidades de la misma, la cual sigue estos pasos:

1. Se escucha las necesidades del cliente.
2. Propuesta del alcance y presupuesto del proyecto.
3. Proceso de contratación de acuerdo a los requerimientos de cada cliente.
4. Reunión de apertura del proyecto con los responsables de las partes interesadas.
5. Levantamiento de información.
6. Definición y prototipo de la herramienta.
7. Definición del cronograma ajustado del proyecto con reuniones de seguimiento.
8. Diseño de la base de datos y su diccionario de datos.
9. Implementación del sistema por módulos.
10. Pruebas preliminares.
11. Implementación de cambios y correcciones.
12. Pruebas finales.
13. Manual de instalación, usuario y documentos de soporte.
14. Instalación de la herramienta de acuerdo a los requisitos del cliente.
15. Capacitaciones a los usuarios e interesados.
16. Garantía y servicio post venta.

[14] IT Group, Quienes Somos, Disponible en: http://www.itgroup.com.co/desarrollo-de-software/itgroup.html

[15] IT GROUP , Internet Technology Group, Disponible en http://www.itgroup.com.co/desarrollo-de-software/metodologia.html

7.7.3 NEXOS SOFTWARE

La empresa Nexos Software S.A.S. es una casa desarrolladora de software que tiene más de 12 años en el mercado nacional e internacional prestando servicios de desarrollo de software a la medida en la modalidad outsourcing, por ello su evolución ha sido paralela con el avance tecnológico y el incremento en diversidad de plataformas y tecnologías.

Para el desarrollo de software utiliza la metodología MSF (Microsoft Solutions Framework) adaptada por Nexos Software como metodología de desarrollo y certificada bajo la norma **ISO 9001-2008** por el Icontec.

Desde el año 2010, se cuenta con el certificado **ITmark**, y otorgado por Certificado por Fundación Esicenter Sinertic Andino, el cual se compone de 3 niveles, Corporativo (Modelo **10Square**), Producto (**CMMi**) y Seguridad en la información (**ISO/IEC 27002:2005,** circular 051 de la Súper Financiera).

Se cuenta con importantes aliados tecnológicos como **IBM** y **Microsoft**, con el fin de estar al día en su tecnología y brindar un adecuado soporte en estas plataformas, además de poner en práctica algunas metodologías desarrolladas y propuestas por ambas compañías, exitosas a nivel mundial.

Nexos Software actualmente pertenece a **Fedesoft**, Federación Colombiana de la Industria del Software y Tecnologías Relacionadas con las que se trabajan temas relacionados con la industria de TI a nivel nacional e internacional[16].

7.7.3.1 Metodología utilizada por NEXOS SOFTWARE

Como se nombra anteriormente la empresa utiliza la metodología MSF de Microsoft (Microsoft Solutions Framework), tomando algunas bases para diseñar una metodología propia que se utilizará en todos los proyectos y que se encuentra certificada con la norma ISO 9001-2008 con la entidad internacional ICONTEC y el reconocimiento de este certificado por IQNET, organismo de acreditación internacional.[17]

7.7.4 ASESOFTWARE

Asesoftware es una compañía colombiana que tiene más de 20 años como proveedora de outsourcing de tecnologías de la información en el desarrollo de software (Web-based y aplicaciones de negocios).

---

[16] NEXOS SOFTWARE, Compañía Historia, Disponible en: http://www.nexos-software.com.co/compania.aspx
[17] NEXOS SOFTWARE S.A.S 2000, Disponible en http://www.nexos-software.com.co/metodologia.aspx

Sus servicios están destinados a empresas, donde los procesos de IT son críticos para su funcionamiento, lo cual permite enfocarse en el CORE de su negocio, sin perder la calidad y mejorando sus procesos[18].

7.7.4.1 Metodología utilizada por ASESOFTWARE

Para el desarrollo de sus trabajos aplica diferentes metodologías de desarrollo de software como RUP, SCRUM y CASE METHOD de ORACLE".[19]

7.7.5 PALMERA SOFT

La empresa Palmera Soft es colombiana y se especializa en mejorar las estrategias de negocio de sus clientes a través de la prestación de servicios de consultoría en tecnologías de información, Ingeniería Web, Ingeniería de Software y Marketing por Internet[20].

7.7.5.1 Metodología utilizada por PALMERA SOFT

Esta empresa utiliza una metodología de desarrollo propia la cual incluye desde el análisis integral de software a desarrollar hasta la puesta en producción de la aplicación, en donde los procesos están enmarcados, bajo un mejoramiento continuo o Círculo de Deming (PHVA).[21]

Las principales disciplinas del proceso de desarrollo bajo el Ciclo PHVA son:

---

[18] ASESOFTWARE, Nosotros, Disponible en: http://www.asesoftware.com/asw/index.php?option=com_content&view=article&id=141&Itemid=57&lang=es

[19] ASESOFTWARE, Disponible en http://www.asesoftware.com/asw/index.php?option=com_content&view=article&id=124&Itemid=158&lang=es

[20] PALMERA SOFT, Disponible en: http://www.palmerasoft.com.co/nuestra-compania

[21] PALMERASOFT, Tecnología adaptable, Disponible en http://www.palmerasoft.com.co/desarrollo-de-software-a-la-medida

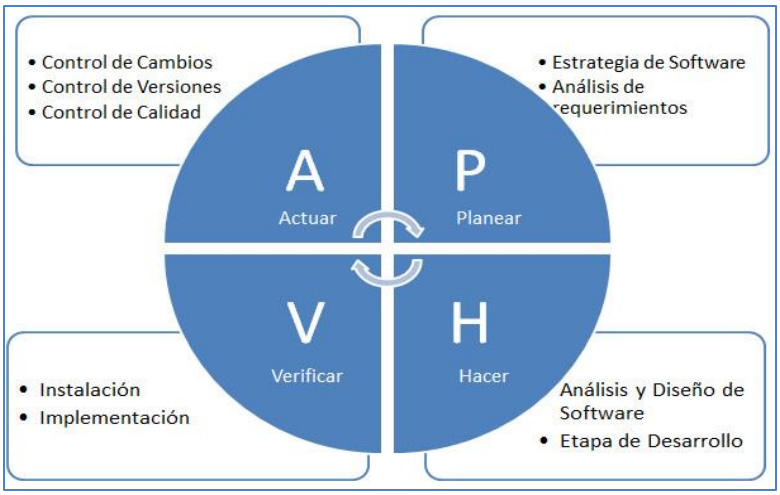

Imagen 8. Ciclo PHVA
FUENTE:http://www.palmerasoft.com.co/desarrollo-de-software-a-la-medida

7.7.6 BIOSALC

Esta empresa se originó en los inicios de la década del 90 y hoy acumula más de 15 años de experiencia y especialización en sistemas para la agroindustria.

Actualmente tiene más de 150 clientes, destacándose entre ellos, los mayores grupos sucro alcoholeros del Brasil y también grandes productores de semillas, frutas, hortalizas, palma africana, eucalipto, entre otros.

Son el mayor exportador de soluciones de TI, con foco en el agro negocio, para América del Sur.

Cuenta con 60 colaboradores en la casa matriz en Brasil y 24 Colaboradores en la filial de Colombia en Cali, Valle del Cauca. Entre ellos analistas, programadores y consultores de negocio.[22]

En Biosalc al momento de realizar un proyecto de software es de elección propia del desarrollador elegir la metodología con la cual se sienta más a gusto en su trabajo para llevar a cabo dicho proyecto.

[22] BIOSALC, que es Biosalc, Disponible en: http://www.biosalc.com.co/biosalc-colombia.html

7.7.7 ISE & E (Ingeniería Servicios Eléctricos y Electrónicos)

Es una empresa boliviana la cual brinda a sus clientes soluciones tecnológicas eficientes y de calidad. Es una empresa de servicios: amplió su área de actividades a la provisión de equipos, enfocando soluciones tecnológicas completas.

Cuenta con sedes en las ciudades de: Cochabamba, La Paz, Santa Cruz y Oruro.

Tienen quince años de existencia. Cuentan con un equipo profesional técnico de más de 40 personas a nivel nacional. Una empresa dedicada a la creatividad, innovación y espíritu de servicio.

Gracias a este esfuerzo conjunto ISE&E ha obtenido un lugar de privilegio en el escenario de la tecnología de la Información.[23]

7.7.7.1 Metodología utilizada por ISE & E

La metodología utilizada por la empresa ISE & E es Scrum la cual se aplica de manera regular en un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.[24]

Producto de ISE & E: Orion.dev

7.7.8 SOFTENG

Esta empresa es especialista en consultoría y desarrollo de software. Desde 1997 ayuda a las organizaciones a ser más competitivas, optimizando sus sistemas, mejorando su productividad y ayudándoles a materializar la innovación.

La empresa es especialista en "Ingeniería de sistemas y soluciones en la nube", "Consultoría sobre Microsoft SharePoint e "Integración y desarrollo de software a medida". También son fabricantes e implantadores de SOFTENG PORTAL BUILDER, una plataforma de última generación en modelo SaaS dirigida

[23] ISE & E, Disponible en: http://www.iseye.com.bo/indexp.php?ide=1
[24] ISE & E, Orion.dev, Disponible en: http://www.iseye.com.bo/ADMINDEV/index.php

a facilitar la administración de portales web públicos e impulsarlos para hacer más negocio a través de Internet.[25]

Debido a que no es viable analizar y determinar con precisión todas las funcionalidades que se requieren y que además con seguridad cambiarán durante el desarrollo a medida que el cliente tiene más información o el negocio tiene otras prioridades, utilizan la metodología SCRUM. Esta metodología, permite descubrir, priorizar e incorporar de forma iterativa las funcionalidades de mayor valor para el cliente, reduciendo riesgos y maximizando el retorno de la inversión para su empresa.[26]

7.7.9 MVM Ingeniería de Software

La empresa MVM ayuda a sus clientes a agregar valor y reducir costos en sus Organizaciones, mediante la construcción y gestión de soluciones informáticas.

Establece fuertes relaciones con sus clientes, gracias a sus valores y al apoyo de un equipo de trabajo altamente calificado y comprometido, se adapta a las mejores prácticas de la industria, lo que le permite ofrecer un servicio de clase mundial e integrar sus procesos con los de sus clientes.

Gerencia activamente las relaciones con sus socios tecnológicos para el mejoramiento de los servicios que presta.[27]

7.7.9.1 Metodología utilizada por MVM Ingeniería de Software

UP (UNIFIED PROCESS)[28]

7.7.10 ARQUITESOFT

Es una empresa Colombiana que nace en el año 2007 con el objetivo de atender las necesidades del mercado de servicios públicos domiciliarios, en lo referente a la gestión de procesos de la cadena de valor utilizando tecnología avanzada. La

[25] SOFTENG, Acerca de Softeng, Disponible en: http://www.softeng.es/es-es/empresa/acerca-de-softeng.html
[26] SOFTENG, Scrum Metodología de Trabajo, Disponible en: http://www.softeng.es/es-es/empresa/metodologias-de-trabajo.html
[27] MVM ingeniería de Software, Enfoque, Disponible en: http://www.mvm.com.co/Enfoqu.html
[28] MVM ingeniería de Software, Desarrollo a la medida, Disponible en: http://www.mvm.com.co/DesaMedi.html

experiencia de estos años ha permitido la consolidación en este sector y ampliar los servicios a otros segmentos de la economía. En la actualidad se centra en proporcionar a los clientes soluciones tecnológicas y servicios de consultoría de negocio y tecnologías de la información, que contribuyan al éxito de sus empresas[29].

En la empresa Arquitesoft para el desarrollo de un nuevo software se utilizan una serie de pasos o procedimientos, también al momento de desarrollo son dependientes del cliente, ya que dichos clientes manejan sus propias metodologías y le empresa se adapta a ellas.

En entrevista con uno de los funcionarios de la empresa el cual labora en el área de desarrollo de software, comentaba que al momento de aplicar una metodología existente para uno de sus proyectos, utilizaban una metodología que fue desarrollada internamente la cual cumplía con todas las necesidades de la empresa.

7.7.11 SAP

SAP América Latina y Caribe es una empresa que inicio en 1994 con el objetivo de ofrecer a sus clientes eficiencia y flexibilidad a través de aplicaciones enfocadas en el crecimiento rentable del negocio. Cuenta con más de 12,000 clientes y un ecosistema de 450 socios de negocios en la región, permite a las empresas de todos los tamaños y de cualquier sector de industria ser más competitivas y rentables, hacer más eficientes sus procesos, potenciar el resultado de sus negocios y reducir costos a través de la innovación tecnológica[30].

SAP al momento de realizar sus módulos o soluciones, utiliza dos metodologías de desarrollo, Rational Unified Process (RUP) para proyectos medianos y grandes los cuales demandan más tiempo y Extreme Programing (XP) para el caso de proyectos pequeños y de corta duración[31].

Según el estudio realizado en algunas empresas desarrolladoras de software, las metodologías más utilizadas son las que se muestran en la Imagen 9.

---

[29] Arquitesoft, Perfil, Disponible en: http://arquitecsoft.com.cws10.my-hosting-panel.com/index.php/perfil

[30] SAP, Nuestra Empresa, Disponible en: http://www.sap.com/latinamerica/about-sap/index.epx

[31] Benavides Klever, Campaña Mauricio y Caizaguano Carlos, Análisis, Diseño e Implementación interface para control y gestión de materias primas, desarrollado en ABAP/4 lenguaje de SAP, Metodología, pág. 3. Disponible en: http://repositorio.espe.edu.ec/bitstream/21000/5905/1/AC-SIS-ESPE-034404.pdf

Imagen 9. Metodologías de desarrollo de software utilizadas por las empresas
FUENTE: Los autores

Como se puede observar en la imagen anterior, un gran porcentaje de las empresas encuestadas no se decantan por ninguna de las metodologías incluidas, prefieren adaptarse a una serie de pasos o procedimientos para llevar a cabo el desarrollo de software y esto se debe a que las metodologías existentes no se acoplan a sus necesidades, ya sean económicas, de personal o de tiempos.

COMPARACION DE PROCESOS Y ACTIVIDADES DE LAS METODOLOGIAS DE DESARROLLO						
Procesos	¿Cómo están caracterizados los procesos del método?	Las actividades cubiertas por el método				
			XP	SCRUM	RUP	ICONIX
		Puesta en Marcha del Proyecto	FALSO	FALSO	VERDADERO	VERDADERO
		Definición de Requisitos	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Modelado	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Código Fuente	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas Unitarias	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas de Integración	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas del Sistema	VERDADERO	VERDADERO	VERDADERO	VERDADERO
		Pruebas de Aceptación	FALSO	FALSO	VERDADERO	VERDADERO

Tabla 7. Comparación de procesos
FUENTE: Los autores

7.8 PROYECTOS

Con el fin de dar más soporte a esté trabajo de grado, se mostrarán algunos proyectos que fueron propuestos para una empresa del sector azucarero del país, en los cuales se mostrarán cifras que indican los costos de producción de software en algunas empresas de desarrollo, con el fin de contrastar estos datos con la realidad económica del país, también se pretende mostrar las diferencias económicas (tiempos, recursos, talento humano, etc.) que representa trabajar un proyecto empleando diferentes metodologías, de tal manera que se sustente tema de los costos productivos al emplear una u otra metodología.

7.8.1 TOTVS COLOMBIA – ADA

La empresa TOTVS ofrece un desarrollo de software para cubrir todos los procesos de un Ingenio Azucarero del país, esta empresa utiliza una metodología propia de desarrollo (MIT – Metodología de Implantación TOTVS) la cual fue basada en el PMBOK Guide y PMI[32].

Este trabajo se divide en 2 fases, la primera es de Implementación, en esta se llevan a cabo la etapa 1 de Análisis y Definición, etapa 2 de Preparación de la Implementación, etapa 3 de Validación y Entrenamiento, etapa 4 de Simulación y la

[32] Metodología de Implementación TOTVS (MIT), Julio 21 de 2010, Bogotá, Colombia, Propuesta TOTVS Colombia-ADA, pág. 6.

etapa 5 de Sistema disponible oficialmente y la segunda fase hace referencia al Acompañamiento y Estabilización del sistema.

La duración de este proyecto está estimada en 18 semanas y un precio aproximado de USD 1.315.000 + IVA[33].

7.8.2 ARQUITESOFT

La empresa ARQUITESOFT ofrece un desarrollo de software para la integración de los diferentes departamentos o áreas de una empresa del sector azucarero de Colombia, en la cual se llevara el inventario, facturación, entre otras; esta empresa al momento del desarrollo de software es de elección propia del desarrollador elegir la metodología con la cual se sienta más a gusto en su trabajo para llevar a cabo dicho proyecto.

Este trabajo se divide en 3 módulos, el primero es el módulo de Facturación, el cual está compuesto por 6 casos, el segundo módulo hace referencia a la Gestión de procesos de un área, el cual está compuesto por 5 casos y por último el módulo de Almacén de Artículos Deportivos, el cual está compuesto por 6 casos[34].

El tiempo presupuestado para el desarrollo del proyecto es de 1263 horas y con un costo aproximado a los $21.000.000 mct[35].

A modo de prueba y realizando un aproximado en el cual se trabaja en jornadas de 8 horas diarias y de lunes a viernes, en el cual tendríamos un total de 40 horas, se logra determinar que este proyecto se realizaría en 32 semanas, lo cual nos da un total de 8 meses.

7.8.3 COMPUNET S.A - SAP

La empresa CompuNet S.A. ofrece un desarrollo de software de SAP para soluciones de negocio a una empresa del sector azucarero de Colombia, en el cual se realizara una solución integrada que en línea permita gestionar todos los procesos del negocio bajo una sola plataforma; SAP utiliza RUP para proyectos medianos y grandes y XP para proyectos de corta duración[36].

---

[33] Propuesta TOTVS Colombia-ADA, Julio 21 de 2010, Bogotá, Colombia, TOTVS Colombia-ADA

[34] Alcance, Proyecto FEP, Enero 13 de 2010, Cali, Colombia, Propuesta Arquitesoft, pág. 6.

[35] Propuesta Arquitesoft, Proyecto FEP, Enero 13 de 2010, Cali, Colombia, Arquitesoft

[36] Benavides Klever, Campaña Mauricio y Caizaguano Carlos, Análisis, Diseño e Implementación interface para control y gestión de materias primas, desarrollado en ABAP/4 lenguaje de SAP, Metodología, pág. 3. Disponible en: http://repositorio.espe.edu.ec/bitstream/21000/5905/1/AC-SIS-ESPE-034404.pdf

Este trabajo se divide en 5 fases, la primera es la fase de Preparación de Datos, la segunda fase hace referencia a la Preparación del Proyecto, la tercera fase corresponde al Business Blueprint, la cuarta fase es acerca de la Realización y por último la fase de Preparación Final[37].
El tiempo presupuestado para el desarrollo del proyecto es de 5 meses y con un costo aproximado a los USD 486.310 y sin tener en cuenta 2 propuestas de licenciamiento más Runtime, la cual ofrece 60 usuarios SAP por USD 174.960 y otra con 115 usuarios SAP por USD 266.800[38].

En conclusión, este proyecto tiene un costo de USD 661.270 teniendo en cuenta la propuesta de licenciamiento más baja.

7.8.4 WEBMASTER – Smart Access

La empresa WEBMASTER ofrece un desarrollo de software llamado Smart Access, el cual permite el registro y control de acceso a empleados, contratistas, visitantes y vehículos[39].

Smart Access es un producto terminado, el cual se mostrará para hacer una comparación en los precios de los diferentes productos de software.

El costo aproximado es de USD 12000 más USD 1440 de la implementación del sistema, para un total de USD 13440[40].

A continuación el resumen de las diferentes propuestas:

PROPUESTA	METODOLOGIA	TIEMPO (MESES)	PRECIO USD	PRECIO HORA USD
TOTVS – ADA	MIT	5	1315000	1643,75
ARQUITESOFT – FEP	ELECCION DEL DESARROLLADOR	8	11150	8,7109375
COMPUNET S.A. - SAP	RUP Y XP	5	661270	826,5875
WEBMASTER - SMART ACCESS	NO APLICA	NO APLICA	13440	NO APLICA

Tabla 8. Comparación propuesta
FUENTE: Los autores

[37] Metodología, Proyecto CompuNet S.A. - SAP, Agosto 30 de 2007, Propuesta SAP, pág. 21.
[38] Valor de la Inversión, Proyecto CompuNet S.A. - SAP, Agosto 30 de 2007, Propuesta SAP, pág. 27.
[39] Proyecto WEBMASTER – Smart Access, Noviembre de 2009, Propuesta Smart Access
[40] Propuesta Comercial, Proyecto WEBMASTER – Smart Access, Noviembre de 2009, Propuesta Smart Access, pág. 18.

La anterior comparación se realizó con el fin de tabular el tiempo requerido por las diferentes empresas, el valor de sus proyectos y se ha calculado el valor cobrado por hora de trabajo de estas empresas, tomando como variables que se trabaja 5 días a la semana en jornadas de 8 horas, todo con el fin de comparar con la empresa Isocron System de México, de la cual es la única que se tiene referencia de costo por hora como simulación para la realización de sus proyectos.

7.8.5 ISOCRON SYSTEMS

Isocron System Computacionales S.A. de C.V., con su sede en Monterrey, Nuevo León, México, es una pequeña empresa proveedora de servicios de consultoría en computación para empresas y negocios en crecimiento; creada por diseñadores y desarrolladores de software profesionales, cuya vocación es el desarrollo de software de alta calidad.[41]

7.8.5.1 Precios. La mayoría de los proyectos se cobran por cada hora que toma el desarrollo del mismo, y el precio varía de acuerdo a la complejidad de los requerimientos.

Precio por hora: Desde $1,000 pesos Mexicanos ($80USD) / Hora

Este precio incluye: administración del proyecto, consultoría en sistemas, análisis de requerimientos, diseño y desarrollo de software, aseguramiento de la calidad, configuración, instalación, y demás actividades que sean requeridas para cumplir con los requerimientos y expectativas del cliente.

Para algunos proyectos se basan en experiencias en proyectos similares para estimar el tiempo y presupuesto para su desarrollo, en este caso añaden un cierto porcentaje al precio estimado para ayudar a cubrir cambios inesperados que puedan surgir durante el proyecto.

La mayoría de los proyectos de desarrollo de software empresarial han requerido de 2 a 4 semanas para su desarrollo, con un costo aproximado de $40,000 a $80,000 pesos mexicanos ($3,000 - $7,000 USD).

El tiempo de entrega por lo regular es de 4 a 8 semanas adicionales al tiempo de desarrollo, dependiendo de la carga de trabajo que se tenga en el momento de confirmar el proyecto y la disponibilidad del cliente para estar evaluando el proyecto durante su desarrollo.[42]

---

[41] http://www.isocron.net/acerca

[42] http://www.isocron.net/servicios/desarrollo-de-software

En conclusión, tomando en cuenta los valores de la Tabla 8, podemos deducir que no se pueden comparar las diferentes propuestas, ya que hay factores que influyen para estas como la magnitud de cada uno de los proyectos, la metodología utilizada, el recurso humano requerido, entre otros.

Teniendo en cuenta la propuesta de Arquitesoft, que es la empresa más “nacional” entre las analizadas, ya que las otras comercializan un producto terminado como SAP, el cual es internacional (Alemán), obtenemos que el precio por hora en su desarrollo de software es mínimo, USD 8.8, siendo acá donde este proyecto tiene más énfasis y fuerza afirmando que las empresas internacionales cobran por sus servicios cantidades de dinero que son poco accesibles para las compañías nacionales. También comparando el costo de la empresa caleña por hora vemos una gran diferencia con la empresa mexicana que cobra casi 10 veces más.

8. ETAPAS QUE DEBERÍA TENER UNA NUEVA METODOLOGÍA ÁGIL

En este trabajo se incluyen los instrumentos Plan General de Desarrollo del Software, Formato de Especificación de Requisitos y Documento Arquitectura de Software los cuales son parte fundamental al momento de formular una nueva metodología.

8.1 INSTRUMENTO. PLAN GENERAL DE DESARROLLO DEL SOFTWARE

Este Plan General de Desarrollo del Software es una versión preparada para ser incluida en la propuesta elaborada como trabajo de grado denominado **"**DETERMINACIÓN DE LOS FACTORES QUE CONLLEVAN A LA FORMULACIÓN DE UNA NUEVA METODOLOGIA ÁGIL DE DESARROLLO DE SOFTWARE".

Este documento provee una visión global del enfoque de desarrollo propuesto.

El proyecto destaca las acciones que se deben seguir en cada una de las fases propuestas para el desarrollo de un producto software enmarcado dentro de los parámetros de calidad requeridos para tal efecto.

8.1.1 Propósito

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

Los usuarios que lideran el Plan General de Desarrollo del Software son:

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

8.1.2 Alcance

El Plan General de Desarrollo del Software describe el plan global usado para el desarrollo del "Sistema para Gestión del proyecto". El detalle de las iteraciones individuales se describe en los planes de cada iteración, documentos que se aportan en forma separada.

Durante el proceso de desarrollo en el documento Plan General de Desarrollo del software se definen las características del producto a desarrollar, lo cual constituye la base para la planificación de las iteraciones.

Para la versión inicial del Plan General de Desarrollo del Software, nos hemos basado en la captura de requisitos por medio del stakeholder representante de la empresa para hacer una estimación aproximada, una vez comenzado el proyecto y durante la fase de Inicio se generará la primera versión del documento Plan General de Desarrollo de Software, el cual se utilizará para refinarlo.

Posteriormente, el avance del proyecto y el seguimiento en cada una de las iteraciones ocasionará el ajuste de este documento produciendo nuevas versiones actualizadas.

8.1.3 Resumen

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

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

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

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

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

8.1.4 Oportunidad de Negocio

Aquí se incluyen las oportunidades del negocio. Por ejemplo: Este sistema permitirá a la empresa automatizar el control de sus actividades (gestión de stock en cada almacén, gestión de pedidos, etc.), lo cual supondrá un acceso rápido y sencillo a los datos, gracias a interfaces gráficas sencillas y amigables. Además, los datos accedidos estarán siempre actualizados, lo cual es un factor muy importante para poder llevar un control centralizado de los distintos almacenes

El sistema también permite a los clientes acceder a los servicios de la empresa a través de web, de forma rápida y sencilla y sin necesidad de intermediarios.

8.1.5 Sentencia que define el problema

El problema de	*Escriba el problema a resolver*. Por ejemplo: Controlar el stock existente en los distintos almacenes, de forma que se puedan servir los pedidos que reciben dichos almacenes. Gestionar las órdenes de compra realizadas por los clientes. Gestionar los pedidos realizados a los proveedores. Gestionar la facturación de la empresa.
Afecta a	*Incluya las dependencias que se ven afectadas con el problema a resolver*. Por ejemplo: Departamento de logística, Jefes de almacenes, Técnicos de almacenes, Encargados de transporte, Usuarios de ventas de cada región, Departamento de contabilidad / facturación, Departamento de recursos humanos, Departamento de marketing.
El impacto asociado es	*Describa el impacto que tendrá una solución al problema*. Por ejemplo: Almacenar toda la información referente a los almacenes, pedidos y órdenes de compra recibidas, y que esta información esté al instante accesible y actualizado en lugares físicamente muy distantes es un proceso prácticamente imposible de realizar en el caso de que no esté informatizado.
Una solución adecuada sería	*Mencione el beneficio de una solución*. Por ejemplo: Informatizar el proceso, usando una red local con una base de datos accesible desde los distintos nodos de la red y generar interfaces amigables y sencillas con las cuales se pueda acceder a la base de datos.

8.1.6 Sentencia que define la posición del Producto

Para	*¿A quiénes estará dirigido el producto?* Por ejemplo: Departamento de logística, Jefes de almacenes, Técnicos de almacenes, Encargados de transporte, Usuarios de ventas de cada región, Departamento de contabilidad / facturación, Departamento de recursos humanos, Departamento de marketing.
Quienes	*¿Qué hacen estos usuarios?* Por ejemplo: Controlan los pedidos, los almacenes (stock), las órdenes de pedido y la facturación.
El nombre del producto	Es una herramienta software.
Que	*¿Qué hará la solución?* Por Ejemplo: Almacena la información necesaria para gestionar una empresa de distribución.
Nuestro producto	*Beneficio del producto*. Por ejemplo: Permite gestionar las distintas actividades de la empresa mediante una interfaz gráfica sencilla y amigable. Además proporciona un acceso rápido y actualizado a la información desde cualquier punto que tenga acceso a la base de datos.

8.2 VISTA GENERAL DEL PROYECTO

8.2.1 Propósito, Alcance y Objetivos

Incluya una breve descripción de la empresa. Como por ejemplo:

El proyecto debe proporcionar una propuesta para el desarrollo de todos los subsistemas implicados en la gestión de artículos deportivos y bases de datos departamentales". Estos subsistemas se pueden diferenciar en siete grandes bloques:

a) Gestión de Ventas, incluyendo:
   - Procedimiento de venta de productos vía operadoras de teléfono.
   - Procedimiento de venta mediante la atención de comerciales a domicilio del cliente.
   - Procedimiento de venta mediante el sistema online, vía web.

b) Gestión de Almacenes, incluyendo:
   - Gestión de nuevos pedidos.
   - Reserva de stock para la preparación de pedidos.
   - Gestión de incidencias de stock.
   - Gestión de pedidos para envío.
   - Gestión de consultas de estado de pedidos
   - Cancelación de pedidos solicitado por el cliente.

c) Gestión de Envíos, incluyendo:
   - Gestión de Pedidos para envío.
   - Gestión de recibos.

d) Departamento de Recursos Humanos.

e) Departamento de Marketing.

f) Departamento de Logística.

g) Contabilidad y Facturación.

8.2.2 Suposiciones y Restricciones

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

a) Debe contemplarse las implicaciones de los siguientes puntos críticos:
    - Compatibilidad de la solución con protocolos IPv6
    - Caracteres multilingües
    - Sistemas seguros: protección de información, seguridad en las trasmisiones de datos, etc.
    - Gestión de flujos de trabajo, seguridad de transacciones e intercambio de información
    - Adaptación a la normativa de Protección de Datos
b) La automatización de la gestión interna del registro debe ajustarse a la legislación vigente y considerar la previsión de la nueva legislación referente a los dominios de tercer nivel.
c) El subsistema “Gestión de Almacenes” debe diseñarse como módulo independiente para ser utilizado posteriormente en otras regiones de los distintos almacenes no centralizados encargados de proveer a cada región de clientes de la empresa *(Nombre de la Empresa)*.

Como es natural, la lista de suposiciones y restricciones se incrementará durante el desarrollo del proyecto, particularmente una vez establecido el artefacto Plan General de Desarrollo del Software.

8.2.3 Entregables del proyecto

A continuación se indican y describen cada uno de los artefactos que serán generados y utilizados por el proyecto y que constituyen los entregables. Esta lista constituye la configuración de la metodología aplicada desde la perspectiva de artefactos, y que se proponen para este proyecto.

Es preciso destacar que de acuerdo a la filosofía aplicada, todos los artefactos son objeto de modificaciones a lo largo del proceso de desarrollo, con lo cual, sólo al término del proceso se podría tener una versión definitiva y completa de cada uno de ellos. Sin embargo, el resultado de cada iteración y los hitos del proyecto están enfocados a conseguir un cierto grado de completitud y estabilidad de los artefactos. Esto será indicado más adelante cuando se presenten los objetivos de cada iteración.

Mediante un estudio comparativo de las metodologías de desarrollo de software como ICONIX, RUP, XP y SCRUM se logró asociar cuales eran los artefactos utilizados por estas, consiguiendo así identificar los modelos en común de dichas metodologías, estos son utilizados en gran porcentaje al momento del desarrollo

del proyecto con cualesquiera de las metodologías anteriormente estudiadas, por lo tanto da a entender que son de vital importancia y serán tomados en cuenta para la proposición o formulación de una nueva metodología de desarrollo de software ágil.

Modelo de Casos de Uso

El modelo de Casos de Uso presenta las funciones del sistema y los actores que hacen uso de ellas. Se representa mediante Diagramas de Casos de Uso.

Especificaciones de Casos de Uso

Para los casos de uso que lo requieran (cuya funcionalidad no sea evidente o que no baste con una simple descripción narrativa) se realiza una descripción detallada utilizando una plantilla de documento, donde se incluyen: precondiciones, post-condiciones, flujo de eventos, requisitos no-funcionales asociados. También, para casos de uso cuyo flujo de eventos sea complejo podrá adjuntarse una representación gráfica mediante un Diagrama de Actividad.

Diagramas de Clases

Los diagramas de clase forman parte de la vista estática y muestran los bloques de construcción de cualquier sistema orientado a objetos, y es donde definiremos las características de cada una de las clases, interfaces, colaboraciones y relaciones de dependencia y generalización.

Diagrama se Secuencia

Un diagrama de secuencia es una forma de diagrama de interacción que muestra los objetos como líneas de vida a lo largo de la página y con sus interacciones en el tiempo representadas como mensajes dibujados como flechas desde la línea de vida origen hasta la línea de vida destino. Los diagramas de secuencia son buenos para mostrar qué objetos se comunican con qué otros objetos y qué mensajes disparan esas comunicaciones. Los diagramas de secuencia no están pensados para mostrar lógicas de procedimientos complejos.

Diagrama de Entidad Relación

Los diagramas de entidad relación (E-R o DER) son diagramas empleados para representar los modelados de datos, logrando que el diseñador se concentre en

los elementos esenciales para el contexto o problema que se está tratando[43]. Este modelo representa a la realidad a través de un esquema gráfico empleando la terminología de Entidades, que son objetos que existen y son los elementos principales que se identifican en el problema a resolver con el diagramado y se distinguen de otros por sus características particulares denominadas Atributos, el enlace que rige la unión de las entidades está representada por la relación del modelo[44].

8.2.4 Entregables Opcionales

Los entregables opcionales son diagramas que según el estudio comparativo no son comunes entre las metodologías estudiadas, por lo cual no son tomados en cuenta en el momento para la proposición o formulación de una nueva metodología de software ágil, pero que en algún momento pueden llegar a ser importantes y de gran ayuda para continuar con el proceso de desarrollo del proyecto.

Diagrama de Componentes

Los Diagramas de Componentes muestran las relaciones entre los componentes de software, sus dependencias, comunicación, ubicación, controladores embebidos, etc. que conformarán un sistema. Un diagrama de Componentes tiene un nivel más alto de abstracción que un diagrama de clase – usualmente un componente se implementa por una o más clases (u objetos) en tiempo de ejecución[45].

Diagrama de Colaboración

Un diagrama de colaboración es un diagrama de interacción que resalta la organización estructural de los objetos que envían y reciben los mensajes. Este tipo de diagrama muestra un conjunto de objetos, enlaces entre ellos y los mensajes que intercambian. Un diagrama de colaboración es un grafo, donde los nodos del grafo son los objetos y los arcos son los enlaces. Un enlace es una instancia de una asociación o una dependencia entre clases. Se representa con una línea continua que une los dos objetos[46].

---

[43] http://augusta.uao.edu.co/moodle/file.php/3167/Modelo_Entidad_Relacion.pdf
[44] http://www.ecured.cu/index.php/Diagrama_Entidad_Relaci%C3%B3n
[45] http://www.sparxsystems.com.ar/resources/tutorial/uml2_componentdiagram.html
[46] http://www.arqhys.com/general/diagrama-de-colaboracion.html

Diagrama de Actividades

Los diagramas de actividades permiten describir como un sistema implementa su funcionalidad, modelan el comportamiento dinámico de un procedimiento, transacción o caso de uso haciendo énfasis en el proceso que se lleva a cabo, es uno de los elementos de modelado que son mejor comprendidos por todos, ya que son herederos directos de los diagramas de flujo[47].

Diagrama de Despliegue

Los diagramas de Despliegue son utilizados para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes, este muestra las relaciones físicas entre los componentes hardware y software en el sistema final, es decir, la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes software (procesos y objetos que se ejecutan en ellos)[48].

1) **Plan General de Desarrollo del Software**
   Es el presente documento.

2) **Modelo de Casos de Uso del Negocio**
   Es un modelo de las funciones de negocio vistas desde la perspectiva de los actores externos (Agentes de registro, solicitantes finales, otros sistemas etc.). Permite situar al sistema en el contexto organizacional haciendo énfasis en los objetivos en este ámbito. Este modelo se representa con un Diagrama de Casos de Uso usando estereotipos específicos para este modelo.

3) **Modelo de Objetos del Negocio**
   Es un modelo que describe la realización de cada caso de uso del negocio, estableciendo los actores internos, la información que en términos generales manipulan y los flujos de trabajo (workflows) asociados al caso de uso del negocio. Para la representación de este modelo se utilizan Diagramas de Colaboración (para mostrar actores externos, internos y las entidades (información) que manipulan, un Diagrama de Clases para mostrar gráficamente las entidades del sistema y sus relaciones y Diagramas de Actividad para mostrar los flujos de trabajo.

[47] http://www.ctr.unican.es/asignaturas/procodis_3_II/Doc/stateDiagram.pdf
[48] http://umldaniel.blogspot.com/2009/05/diagrama-de-despliegue.html

4) **Glosario**
   Es un documento que define los principales términos usados en el proyecto. Permite establecer una terminología consensuada.

5) **Especificaciones Adicionales**
   Este documento capturará todos los requisitos que no han sido incluidos como parte de los casos de uso y se refieren requisitos no-funcionales globales. Dichos requisitos incluyen: requisitos legales o normas, aplicación de estándares, requisitos de calidad del producto, tales como: confiabilidad, desempeño, etc., u otros requisitos de ambiente, tales como: sistema operativo, requisitos de compatibilidad, etc.

6) **Prototipos de Interfaces de Usuario**
   Se trata de prototipos que permiten al usuario hacerse una idea más o menos precisa de las interfaces que proveerá el sistema y así, conseguir retroalimentación de su parte respecto a los requisitos del sistema. Estos prototipos se realizarán como: dibujos a mano en papel, dibujos con alguna herramienta gráfica o prototipos ejecutables interactivos, siguiendo ese orden de acuerdo al avance del proyecto. Sólo los de este último tipo serán entregados al final de la fase de Elaboración, los otros serán desechados. Asimismo, este artefacto, será desechado en la fase de Construcción en la medida que el resultado de las iteraciones vayan desarrollando el producto final.

7) **Modelo de Análisis y Diseño**
   Este modelo establece la realización de los casos de uso en clases y pasando desde una representación en términos de análisis (sin incluir aspectos de implementación) hacia una de diseño (incluyendo una orientación hacia el entorno de implementación), de acuerdo al avance del proyecto.

8) **Modelo de Datos**
   Previendo que la persistencia de la información del sistema será soportada por una base de datos relacional, este modelo describe la representación lógica de los datos persistentes, de acuerdo con el enfoque para modelado relacional de datos. Para expresar este modelo se utiliza un Diagrama de Clases (donde se utiliza un profile UML para Modelado de Datos, para conseguir la representación de tablas, claves, etc.).

9) **Modelo de Implementación**

Este modelo es una colección de componentes y los subsistemas que los contienen. Estos componentes incluyen: ficheros ejecutables, ficheros de código fuente, y todo otro tipo de ficheros necesarios para la implantación y despliegue del sistema. (Este modelo es sólo una versión preliminar al final de la fase de Elaboración, posteriormente tiene bastante refinamiento).

10) **Modelo de Despliegue**

Este modelo muestra el despliegue la configuración de tipos de nodos del sistema, en los cuales se hará el despliegue de los componentes.

11) **Casos de Prueba**

Cada prueba es especificada mediante un documento que establece las condiciones de ejecución, las entradas de la prueba y los resultados esperados. Estos casos de prueba son aplicados como pruebas de regresión en cada iteración. Cada caso de prueba llevará asociado un procedimiento de prueba con las instrucciones para realizar la prueba y dependiendo del tipo de prueba dicho procedimiento podrá ser automatizable mediante un script de prueba.

12) **Solicitud de Cambio**

Los cambios propuestos para los artefactos se formalizan mediante este documento. Mediante este documento se hace un seguimiento de los defectos detectados, solicitud de mejoras o cambios en los requisitos del producto. Así se provee un registro de decisiones de cambios, de su evaluación e impacto y se asegura que éstos sean conocidos por el equipo de desarrollo. Los cambios se establecen respecto de la última baseline (el estado del conjunto de los artefactos en un momento determinado del proyecto) establecida. En nuestro caso al final de cada iteración se establecerá una baseline.

13) **Plan de Iteración**

Es un conjunto de actividades y tareas ordenadas temporalmente, con recursos asignados, dependencias entre ellas. Se realiza para cada iteración y para todas las fases.

14) **Evaluación de Iteración**

Este documento incluye le evaluación de los resultados de cada iteración, el grado en el cual se han conseguido los objetivos de la iteración, las lecciones aprendidas y los cambios a ser realizados.

15) Lista de Riesgos

Este documento incluye una lista de los riesgos conocidos y vigentes en el proyecto, ordenados en orden decreciente de importancia y con acciones específicas de contingencia o para su mitigación.

16) Manual de Instalación

Este documento incluye las instrucciones para realizar la instalación del producto.

17) Material de Apoyo al Usuario Final

Corresponde a un conjunto de documentos y facilidades de uso del sistema, incluyendo: Guías del Usuario, Guías de Operación, Guías de Mantenimiento y Sistema de Ayuda en Línea

18) Producto

Los ficheros del producto empaquetados y almacenadas en un CD con los mecanismos apropiados para facilitar su instalación. El producto, a partir de la primera iteración de la fase de Construcción es desarrollado incremental e iterativamente, obteniéndose una nueva reléase al final de cada iteración.

Los artefactos 16, 17 y 18 se generarán a partir de la fase de Construcción, con lo cual se han incluido aquí sólo para dar una visión global de todos los artefactos que se generarán en el proceso de desarrollo.

8.2.5 Evolución del Plan General de Desarrollo del Software

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

8.3 Organización del Proyecto

8.3.1 Participantes en el Proyecto

Describa los participantes en el proyecto, por ejemplo:

De momento no se incluye el personal que designará la empresa *(Nombre de la Empresa)* como Responsable del Proyecto, Comité de Control y Seguimiento, otros participantes que se estimen convenientes para proporcionar los requisitos y validar el sistema.

El resto del personal del proyecto (por la parte de la empresa adjudicataria), considerando las fases de Inicio, Elaboración y dos iteraciones de la fase de Construcción, estará formado por los siguientes puestos de trabajo y personal asociado:

Jefe de Proyecto. *Nombre del funcionario encargado de la labor*, estudios y experiencia en los métodos nombrados.

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

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

Ingeniero de Software. El perfil establecido es: Ingeniero en Informática recién titulado que participará como becario en el convenio universidad-empresa, realizando labores de gestión de requisitos, gestión de configuración, documentación y diseño de datos. Encargado de las pruebas funcionales del sistema, *Nombre del funcionario encargado de la labor.*

Los Currículos Vitae del personal del proyecto que ya ha comprometido su participación se adjuntan por separado.

8.3.2 Entorno de usuario

Describir el entorno en el que estará el usuario, por ejemplo:

Los usuarios entrarán al sistema identificándose sobre un ordenador con un sistema operativo Windows 7 y tras este paso entrarán a la parte de aplicación diseñada para cada uno según su papel en la empresa. Este sistema es similar a cualquier aplicación Windows y por tanto los usuarios estarán familiarizados con su entorno.

Los informes serán generados con Microsoft Word versión 2010, lo cual también resultará familiar.

8.3.3 Interfaces Externas

La empresa *(Nombre de la Empresa)* definirá los participantes del proyecto que proporcionarán los requisitos del sistema, y entre ellos quiénes serán los encargados de evaluar los artefactos de acuerdo a cada subsistema y según el plan establecido.

El equipo de desarrollo interactuará activamente con los participantes de la empresa *(Nombre de la Empresa)* para especificación y validación de los artefactos generados.

8.3.4 Roles y Responsabilidades

Describa los roles y las responsabilidades del proyecto, por ejemplo:

A continuación se describen las principales responsabilidades de cada uno de los puestos en el equipo de desarrollo durante las fases de la metodología aplicada.

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

Ingeniero de Software	Gestión de requisitos, gestión de configuración y cambios, elaboración del modelo de datos, preparación de las pruebas funcionales, elaboración de la documentación. Elaborar modelos de implementación y despliegue.

8.4 Gestión del Proceso

8.4.1 Estimaciones del Proyecto

El presupuesto del proyecto y los recursos involucrados se adjuntan en un documento separado.

8.4.2 Perspectiva del producto

El producto a desarrollar es un sistema global para la empresa *(Nombre de la Empresa),* con la intención de agilizar su funcionamiento. Las áreas a tratar por el sistema son: logística, gestión de recursos humanos, contabilidad y marketing.

8.4.3 Plan del Proyecto

En esta sección se presenta la organización en fases e iteraciones y el calendario del proyecto.

8.4.3.1 Plan de las Fases

El desarrollo se llevará a cabo en base a fases con una o más iteraciones en cada una de ellas. La siguiente tabla muestra una la distribución de tiempos y el número de iteraciones de cada fase.

	Fase	Nro. Iteraciones	Duración
Pruebas	Fase de Análisis	#	# semanas
	Fase de Diseño	#	# semanas
	Fase de Implementación	#	# semanas

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

Descripción	Hito
Fase de Análisis	En esta fase se realizara la entrevista con el cliente para el levantamiento de los requisitos del producto desde la perspectiva del usuario, los cuales serán establecidos en el artefacto Plan General de Desarrollo del Software. De igual manera ser realizara la planificación del proyecto y el modelo de dominio. La aceptación del cliente / usuario y validación de los requisitos del artefacto Plan General de Desarrollo del Software marcan el final de esta fase.
Fase de Diseño	Se desarrolla un prototipo de arquitectura (incluyendo las partes más relevantes y / o críticas del sistema). Al final de esta fase, todos los casos de uso correspondientes a requisitos que serán implementados en la primera reléase de la fase de Construcción deben estar analizados y diseñados (en el Modelo de Análisis / Diseño). También se desarrollaran los modelos de: Clases, Secuencia,

	Entidad Relación, Planificación de las Pruebas y Ejecución Pruebas y de ser necesario otro grupo de hitos que son: modelo de despliegue, modelo de componentes, modelo de actividad y modelo de colaboración. La revisión y aceptación del prototipo de la arquitectura del sistema marca el final de esta fase.
Fase de Implementación	Durante la fase de implementación se realiza los prototipos de interfaces de usuario. Se comienza la elaboración de material de apoyo al usuario. El hito que marca el fin de esta fase son las pruebas funcionales.

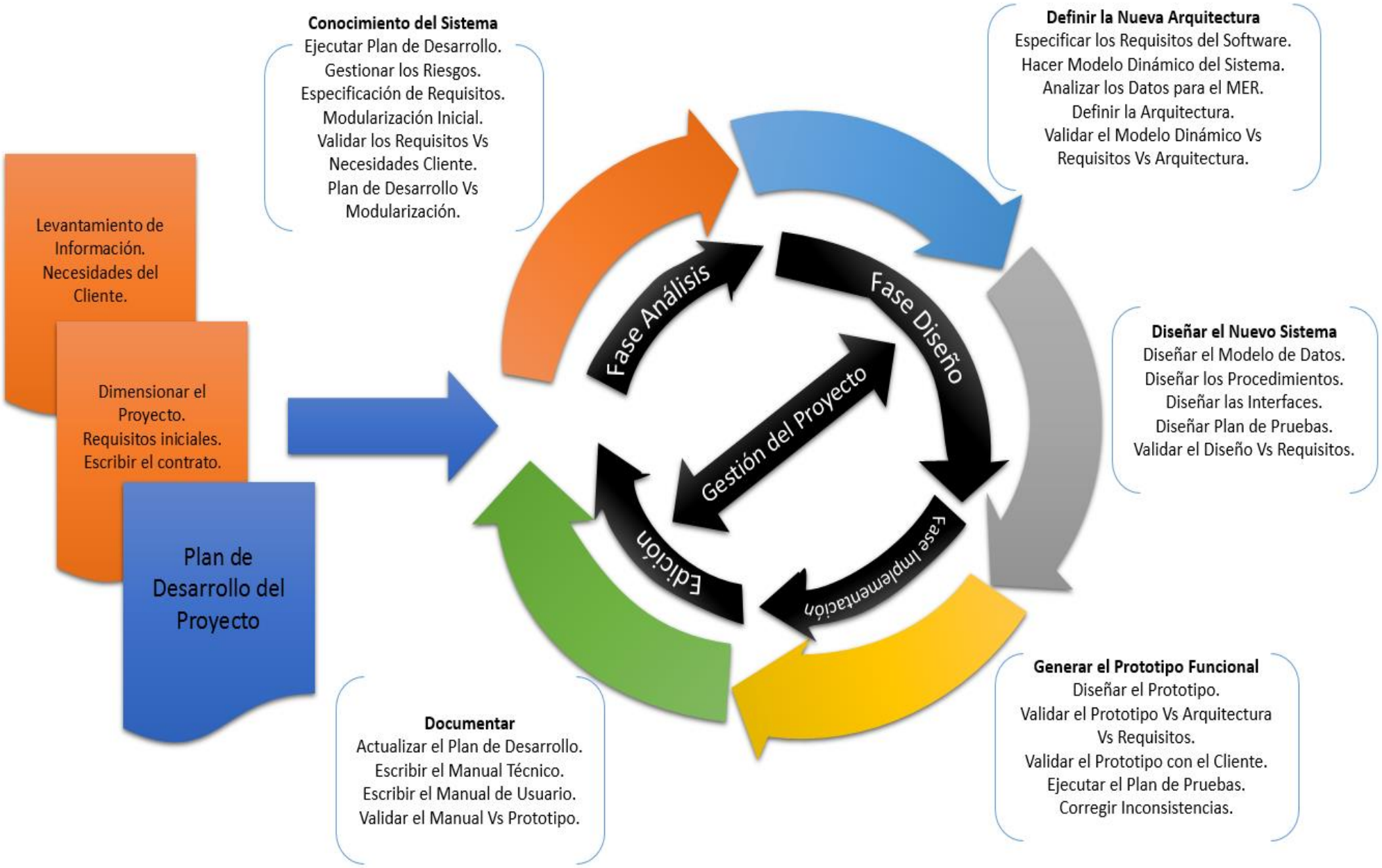

Imagen 10. Modelo formulado nueva metodología ágil de desarrollo
FUENTE: Los autores

Descripción de las Disciplinas / Artefactos que conformaran la nueva metodología de desarrollo de software ágil:

Análisis - Conocimientos del Sistema

Levantamiento de Información: Construir instrumentos para la recolección de información.

Necesidades Cliente: Realizar entrevista con el cliente para conocer sus necesidades.

Dimensionar Proyecto: Dimensionar el ámbito del proyecto.

Listado Requisitos Iniciales: Listado de requisitos extraídos de la entrevista con el cliente.

Construcción Contrato: Construcción del contrato para formalizar el nuevo proyecto con los requisitos iníciales.

Comunicación Cliente: Mantener la comunicación constante con el cliente para informar cómo va el proceso y solucionar inconvenientes.

Plan Desarrollo Proyecto: Realizar el Plan Desarrollo del Proyecto como guía del proyecto.

Ejecutar Plan Desarrollo Proyecto: Iniciar el Plan de Desarrollo del Proyecto para definir los pasos a realizar en el proyecto.

Gestionar Riesgos: Gestionar los posibles riesgos que se pueden presentar en el proyecto.

Especificación Requisitos: Formalizar los requisitos del proyecto por medio del Formato Especificación de Requisitos.

Modularización Inicial: Dividir el proyecto en módulos para ejecutar las respectivas iteraciones por cada módulo.

Validar Requisitos Vs Necesidades Cliente - Plan Desarrollo Proyecto Vs Modularización: Con estos procesos se finaliza la fase de análisis de cada módulo.

Diseño – Definición Nueva Arquitectura

Especificar Requisitos Software: Revisar y detallar los requisitos del software e incluirlos en su respectivo formato.

Hacer Modelo Dinámico del Sistema: Hacer los diagramas de clases, casos de uso y secuencia.

Analizar Datos para el MER: Inferir los datos desde cada requisito.

Definir Arquitectura: Definir arquitectura del proyecto por medio del Documento Arquitectura del Software. Definir patrón, modelo y diseño de la arquitectura.

Validar Modelo Dinámico Vs Requisitos Vs Arquitectura: Con este proceso se finaliza el diseño de la arquitectura.

Modelo de Despliegue (opcional)

Modelo de Componentes (opcional)

Modelo de Actividad (opcional)

Modelo de Colaboración (opcional)

Diseñar Nuevo Sistema

Diseñar Modelo Datos: Construir el modelo entidad relación (MER).

Diseñar Procedimientos: Diseñar los métodos.

Diseñar Interfaces: Diseñar las GUI's.

Diseñar Plan Pruebas: Diseñar responsable, tiempos y métodos a probar.

Validar Diseño Vs Requisitos: Con este proceso se finaliza la fase de diseño.

Implementación – Generar Prototipo Funcional

Diseñar Prototipo: Se construye el prototipo del módulo.

Validar Prototipo Vs Arquitectura Vs Requisitos: Comprobar que el prototipo se ajusta a la arquitectura definida.

Validar Prototipo con el Cliente: Mostrar el prototipo al cliente y validar su funcionalidad.

Ejecutar Plan de Pruebas: Probar las funcionalidades.

Corregir Inconsistencias: Ajustar el prototipo. Con este proceso se finaliza la fase de implementación del módulo.

Edición - Documentar

Actualizar Plan Desarrollo Proyecto: Actualizar las tareas ejecutadas del plan, gestionar los riesgos y verificar la planeación.

Escribir Manual Técnico: Escribir el manual técnico del módulo.

Escribir Manual Usuario: Escribir el manual de usuario del módulo.

Validar Manual Vs Prototipo: Comprobar la correspondencia entre el prototipo y el manual. Con este proceso se finaliza el módulo de edición.

8.4.3.2 Calendario del Proyecto

Para este proyecto se ha establecido el siguiente calendario. La fecha de aprobación indica cuándo el artefacto en cuestión tiene un estado de completitud suficiente para someterse a revisión y aprobación, pero esto no quita la posibilidad de su posterior refinamiento y cambios.

Disciplinas / Artefactos generados o modificados durante todas las fases.	Comienzo	Fin
Análisis - Conocimientos del Sistema		
Levantamiento de Información	Semana 1	Semana #

Necesidades Cliente	Semana #	Semana #
Dimensionar Proyecto	Semana #	Semana #
Listado Requisitos Iniciales	Semana #	Semana #
Construcción Contrato	Semana #	Semana #
Comunicación Cliente	Semana #	Semana #
Plan Desarrollo Proyecto	Semana #	Semana #
Ejecutar Plan Desarrollo Proyecto	Semana #	Semana #
Gestionar Riesgos	Semana #	Semana #
Especificación Requisitos	Semana #	Semana #
Modularización Inicial	Semana #	Semana #
Validar Requisitos Vs Necesidades Cliente Plan Desarrollo Proyecto Vs Modularización	Semana #	Semana #
Diseño – Definición Nueva Arquitectura		
Especificar Requisitos Software	Semana #	Semana #
Hacer Modelo Dinámico del Sistema	Semana #	Semana #
Analizar Datos para el MER	Semana #	Semana #
Definir Arquitectura	Semana #	Semana #
Validar Modelo Dinámico Vs Requisitos Vs Arquitectura	Semana #	Semana #
Modelo de Despliegue (opcional)	Semana #	Semana #
Modelo de Componentes (opcional)	Semana #	Semana #
Modelo de Actividad (opcional)	Semana #	Semana #

Modelo de Colaboración (opcional)	Semana #	Semana #
Diseñar Nuevo Sistema		
Diseñar Modelo Datos	Semana #	Semana #
Diseñar Procedimientos	Semana #	Semana #
Diseñar Interfaces	Semana #	Semana #
Diseñar Plan Pruebas	Semana #	Semana #
Validar Diseño Vs Requisitos	Semana #	Semana #
Implementación – Generar Prototipo Funcional		
Diseñar Prototipo	Semana #	Semana #
Validar Prototipo Vs Arquitectura Vs Requisitos	Semana #	Semana #
Validar Prototipo con el Cliente	Semana #	Semana #
Ejecutar Plan de Pruebas	Semana #	Semana #
Corregir Inconsistencias	Semana #	Semana #
Edición - Documentar		
Actualizar Plan Desarrollo Proyecto	Semana #	Semana #
Escribir Manual Técnico	Semana #	Semana #
Escribir Manual Usuario	Semana #	Semana #
Validar Manual Vs Prototipo	Semana #	Semana #

Se debe de tener en cuenta que estos tiempos son estimados y pueden variar dependiendo del tamaño de los diferentes proyectos, a los cuales se llega con base en personas con experiencia en el ámbito de desarrollo de software, que darán su opinión en cuanto a tiempo de cada fase. También se logra aplicar el

concepto de sala limpia o sus siglas en inglés “Clean Room”, el cual hace un enfoque en la necesidad de incluir la validación en el software a medida que se desarrolla, por esta razón se integrará la fase de pruebas al final de cada fase y no al final del proyecto como se aplica en las diferentes metodologías.

8.4.4 Seguimiento y Control del Proyecto

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

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

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

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

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

8.4.5 Resumen de características

A continuación se mostrará un listado con los beneficios que obtendrá el cliente a partir del producto:

Beneficio del cliente	Características que lo apoyan
Mayor agilidad en los pedidos dando la posibilidad de hacerlo vía servicios web.	Aplicación web desde la cual poder realizar los pedidos.
Gestión automatizada del stock del almacén.	Sistema de optimización del stock en el almacén y previsión de pedidos
Mayor facilidad para la gestión de los recursos humanos.	Base de datos centralizada con la información de todo el personal.
Posibilidad de cancelación de órdenes por parte del cliente dando la posibilidad de hacerlo vía servicios web.	Aplicación web desde la que poder cancelar pedidos.
Automatización de la cancelación de estas órdenes.	Sistema automatizado de anulación de órdenes.
Mayor facilidad para el control de catálogos para el área de marketing.	Base de datos con acceso remoto desde la que poder controlar ofertas y políticas de ventas.
Automatización del sistema de nóminas	Sistema automático de generación de nóminas.

8.5 Otros Requisitos del Producto

8.5.1 Estándares Aplicables

8.5.2 Requisitos de Sistema

8.5.3 Requisitos de Desempeño

8.5.4 Requisitos de Entorno

8.6 Requisitos de Documentación

8.6.1 Manual de Usuario

8.6.2 Ayuda en Línea

8.6.3 Guías de Instalación, Configuración, y Fichero Léame

El instrumento Plan General de Desarrollo de Software incluido en este trabajo fue realizado con base al propuesto por Patricio Letelier titulado como Deportes LSI 03 – Sistema para Gestión de Artículos Deportivos LSI 03 – Plan Desarrollo de Software Versión 3.0[49]

[49] Deportes LSI 03 – Sistema para Gestión de Artículos Deportivos LSI 03 – Plan Desarrollo de Software Versión 3.0, Disponible en: http://www.slideshare.net/ntvp/plan-de-desarrollo-software

CONCLUSIONES

- De acuerdo al estudio realizado a las diferentes metodologías de desarrollo de software se puede concluir que los diagramas de casos de uso, diagramas de clase, diagramas de secuencia y diagramas de entidad relación son los más utilizados en las metodologías para el desarrollo de software, por lo tanto serán la base al momento de proponer una nueva metodología ágil de desarrollo de software.
- En el estudio nombrado anteriormente se logra identificar que los diagramas de componentes, robustez, implementación, estado, colaboración, actividad y despliegue son utilizados más en unas metodologías que en otras, por lo tanto serán considerados opcionales, pero resaltando que no son menos importantes y que pueden llegar a ser vitales en cualquier momento del desarrollo de software y el cual será decisión del desarrollador utilizarlos en el momento necesario.
- Se puede llegar a formular una nueva metodología la cual puede estar compuesta por 3 fases que son análisis, diseño e implementación y que tendrían una serie de artefactos y actividades en cada una de ellas como se muestra en la imagen 10.
- Las metodologías de desarrollo de software están compuestas por cuatro fases generalmente: análisis, diseño, implementación y pruebas, al momento de proponer una nueva metodología se puede determinar que la fase de pruebas y enfocándonos en el concepto de sala limpia el cual consiste en ir solucionando o corrigiendo los problemas a medida de que se presentan, esta fase se integrará al final de las fases anteriores, en la cual se probará y corregirá al final de cada una de ellas. Se aclara que las pruebas Alfa y Beta no desaparecen del marco metodológico.
- El estudio sobre las diferentes empresas desarrolladoras de software nos deja como resultado que estas compañías a la hora de desarrollar sus proyectos prefieren en un gran porcentaje adaptarse a una serie de pasos o procedimientos y esto se debe a que las metodologías existentes no se acoplan a sus necesidades, ya sean económicas, de personal o de tiempos, lo que nos lleva a fortalecer más nuestra idea que es necesario la implementación de una nueva metodología la cual se adapte más a las empresas de nuestro país.

COLABORADORES

Ing. José Gabriel Pérez Canencio
Docente Uceva

Ing. Mary Luz Ojeda Solarte
Docente Uceva

Ing. Andrés Rey Piedrahita
Docente Uceva

BIBLIOGRAFÍA

[1] Pressman, Roger S. Ingeniería de Software, un enfoque práctico. Mac GrawHill. Sexta Edición.

[2]Jeffrey A. Livermore, *Factors that Impact Implementing an Agile Software Development Methodology,* Walsh College.

[3]Sebastian Dyck & Tim A. Majchrzak, *Identifying Common Characteristics in Fundamental, Integrated, and Agile Software Development Methodologies,* Department of Information Systems, University of Munster

[4] Boehm, B. & Turner, R. Management challenges to implement agile processes in traditional development organizations. *IEEE Software*. 22(5), 30-40. 2005.

[5] Revista Antioqueña de las Ciencias Computacionales y la Ingeniería de Software, *Formación Profesional en Ingeniería de Software: Una Necesidad Crítica en los Estados Unidos***,** RACCIS, 2(1), 6-10, 2012.

[6] Beck, K., *Extreme programming explained*. 2000: Addison Wesley.

[7] Schatz, B. & Abdelshafi, I. Primavera gets agile: A successful transition to agile development. *IEEE Software*. 22(3). 2005

[8] R.F. Roggio, "A model for the software engineering capstone sequence using the Rational Unified Process", *Proceedings of ACM SE'06, March 10-12, 2006*, Association for Computing Machinery, p 306-311.

[9] Keefe, K. and M. Dick. *Using extreme programming in a capstone project*. in *6th Conference on Australian Computer Education*. 2004: ACM.

[10] J. Tessem and F. Maurer, "Job satisfaction and motivation in a large agile team," in *Agile Processes in Software Engineering and Extreme Programming, Proceedings*. vol. 4536, G. Concas, E. Damiani, M. Scotto, and G. Succi, Eds., ed Berlin: Springer-Verlag Berlin, 2007, pp. 54-61.

[11] Canós José H., Letelier Patricio, y Penadés, Mª Carmen, "Metodologías Ágiles en el Desarrollo de Software"; http://www.willydev.net/descargas/prev/TodoAgil.pdf.

[12] IEEE, The Institute of Electrical and Electronics Engineers, Inc., "IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology," New York, 1990.

[13] Lindstrom, L. & Jeffries, R. Extreme programming and agile software development methodologies. *Information Systems Management*. 21(13), 41-53. 2005.

[14] Layman, L., L. Williams, and L. Cunningham. *Exploring extreme programming in context: An industrial case study*. in *Agile Development Conference*. 2004.

[14] Roberts, T., Gibson, M., Fields, K., and Rainer, R. Factors That Impact Implementing a System Development Methodology*. IEEE Transactions on Software Engineering*. 24(8), 640-649. 1998

[15] Mackenzie, A. and S. Monk. *From cards to code: How extreme programming re-embodies*
programming as a collective practice. in *Computer Supported Cooperative Work*. 2004.

[16] B. Schatz, K. Schwaber and R.C. Martin, "Best Practices in Scrum Project Management and XP Agile Software Development", White Paper, Object Mentor, Inc. http://www.objectmentor.com/resources/articles/Primavera July 2004.

[17]Ming Huo, June Verner, Liming Zhu, Mohammad Ali Babar, "Software Quality and Agile Methods", COMPSAC '04, IEEE 2008.

[18] Muller, M.M. and W.F. Tichey. *Case study: Extreme programming in a university environment*. In *23rd International Conference on Software Engineering*. 2001: IEEE.

[19]J. Grenning, "Launching extreme programming at a process-intensive company," Software, IEEE, vol. 18, pp. 27- 33, 2001.

[20] Williams, L., et al. *Toward a framework for evaluating extreme program*. in *Empirical Assessment in Software Engineering (EASE)*. 2004.

[21]Arthur English, "Extreme programming, it's worth a look", IT Professional, Volume 4, Issue 3, May-June 2002 Page(s):48–50,IEEE.

[22] Kent Beck et ai, Manifesto for Agile Software Development Accessed from http://agilemanifesto.org/

[23] P. Kruchten, *The Rational Unified Process: An Introduction*. Boston, MA: Addison-Wesley Professional, 2003.
[24] Wäyrynen, J., Bodén, M. & Boström, G.(2004) Security Engineering and eXtreme Programming: An Impossible Marriage? In Proceedings of the 4th

Conference on Extreme Programming and Agile Methods. 2004, Springer-Verlag, Lecture Notes in Computer Science. p. 117 .

[25] R. Ramsin, The Engineering of an Object-Oriented Software Development Methodology. York, UK: University of York, 2006.

[26] Schwaber K., Beedle M., Martin R.C. “Agile Software Development with SCRUM”. Prentice Hall. 2001.

[27] [McBreen, 2002] McBreen, Pete, *Questioning Extreme Programming*, Addison-Wesley The XP Series, 2002.

[28] Shenone Marcelo Hernán, “Diseño de una Metodología Ágil de Desarrollo de Software”; http://materias.fi.uba.ar/7500/shenone-tesisdegrado.pdf.

[29] Mendoza Sánchez, María A, “Metodologías de Desarrollo de Software,” Jun.2004;
http://www.willidev.net/InsiteCreation/v1.0/descargas/cualmetodología.pdf.

[30] de San Martin Oliva, Carla Rebeca Patricia, “Uso de la metodología ICONIX”; http://www.unsj-cuim.edu.ar/portalzonda/seminario08/archivos/UsodeICONIX.pdf

[31] *Patton P, Jayaswal B., Design for trustworthy Software, Prentice Hall, pp 7, 2006*

[32] [Martin, 1991] Martin, J., *Rapid Application Development*, Macmillan Inc., New York, 1991.

[33] López Villatoro, Marco René, “Desarrollo de software utilizando proceso unificado y extreme programming, Ene. 2009; http://www.revistaciencia.info/papers/v01n01_02.pdf.

[34]Rueda Chacón, Julio César, “Aplicación De La Metodología Rup Para El Desarrollo Rápido De Aplicaciones Basado En El Estándar J2EE,” Mar. 2006; http://biblioteca.usac.edu.gt/tesis/08/08_7691.pdf.

[35]Alejandro Martínez y Raúl Martínez, “Guía a Rational Unified Process”; http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo-Guia%20RUP.pdf.

[36] “Introducción al modelo Scrum de desarrollo de Software”; http://www.navegapolis.net/files/s/NST-010_01.pdf.

[37] Molpeceres, Alberto, “Procesos de desarrollo: RUP, XP y FDD”; http://www.javahispano.org/contenidos/archivo/71/metodos_desarrollo.pdf.

[38] Anaya, Víctor y Letelier, Patricio, “Trazabilidad de Requisitos Adaptada a las Necesidades del Proyecto: Un Caso de Estudio Usando Alternativamente RUP y XP”; www.willydev.net/InsiteCreation/v1.0/descargas/prev/traza.pdf.

REFERENCIAS

[1] “Introducción a la programación extrema”; http://www.cristalab.com/blog/introduccion-a-la-programacion-extrema-c44013l/

[2] “ICONIX”; http://www.iconixsw.com/

[3] “Metodología XP vs Metodología RUP”; http://metodologiaxpvsmetodologiarup.blogspot.com/

[4] “SVMasterPlan”; https://sites.google.com/site/svmasterplan/10peii/recursos

[5] “xprogramming”; http://xprogramming.com/index.php

[6] “Extreme Programming: a gentle introduction”; http://www.extremeprogramming.org/

[7] “Calcula tus emisiones de CO2 y toma acciones para reducirlas”; http://www.calculatusemisiones.com/main.html

[8] “Special Offer: Extreme Programming”; http://www.cutter.com/content-and-analysis/resource-centers/agile-project-management/sample-our-research/ead0002.html

[9] ”Object Mentor Industry experts in software best practices”; http://www.objectmentor.com/

[10] “Bayer calculadora climática”; http://www.bayerandina.com/responsabilidad_social/calculadora.htm

[11] “Lo que le cuestan sus electrodomésticos y cuanto CO2 emiten”; http://ec.europa.eu/clima/sites/campaign/pdf/table_appliances_es.pdf

[12]”Todo un mundo de energía”; http://www.slideshare.net/EndesaEduca/endesa-educa-co2centralesyhogar#btnNext

[13]”La industria de las TIC genera el 2% de las emisiones de CO2”; http://www.um.es/docencia/barzana/DIVULGACION/INFORMATICA/Despilfarro-energia-informatica.html

[14]”Consumo de energía en servidores y los impactos ambientales asociados”; http://www.tcogreen.com/index.php?option=com_content&view=article&id=49&Itemid=57&lang=es

[15]http://www.trox.es/xpool/download/es/technical_documents/IT_Cooling_System/CO2OL_RAC.pdf

[16] “Programación Extrema”; http://eisc.univalle.edu.co/materias/WWW/material/lecturas/xp.pdf

[17] “Centro de datos verde”; http://www.webecologica.com/index.php/blog/14-cpd-verde.html

[18] “Scrum”; http://www.scrum.org/

[19] “Scrum Alliance”;http://scrumalliance.org/

[20] “IBM Rational Unified Process (RUP)”; http://www-01.ibm.com/software/rational/rup/

[21] Meza Martínez, Jorge Iván, “Introducción a la implementación de Scrum”; http://www.jorgeivanmeza.com/.

[22]Pliego de Cláusulas Técnicas para la Definición y Análisis de los Procedimientos del ES-NIC.

[23]Desarrollo de una aplicación informática para el cálculo del personal necesario para la fabricación de carrocerías, utilizando la metodología RUP. – P.F.C. de Ponz Lillo, Daniel.

[24]Visual Modeling with Rational Rose and UML, Terry Quatrani. - Addison-Wesley.

ANEXOS

Anexo A. Manuel Fernando Dávila Sguerra

Imagen 11. Manuel Fernando Dávila Sguerra
FUENTE: http://www.acis.org.co/fileadmin/Revista_117/Editorial.pdf

Ingeniero de Sistemas de la Universidad de los Andes, nacido hace 64 años en el municipio de Boavita en el departamento de Boyacá. De padre inventor, heredó la inquietud por hacerse preguntas, sobre todo en el campo de la computación. Director del Departamento de Informática y Electrónica de Uniminutó, Gerente de desarrollo de Grupo Linux S.A. Uno de los más grandes promotores del software libre. Cofundador de ACIS, INDUSOFT Y REDIS (Red de Decanos y Directores de Ingeniería de Sistemas). El software que usa para sus gestiones lo desarrolló él mismo usando Linux, Perl, y bases de datos relacionales; se llama "e-Genesis - El Generador de sistemas" y obtuvo la mención especial en el Premio Colombiano de Informática 2006 de Acis[50].

Hoja de vida:
http://201.234.78.173:8081/cvlac/visualizador/generarCurriculoCv.do?cod_rh=0001165780

[50] Canal Universitario Nacional ZOOM, Manuel Dávila Sguerra, disponible en: http://www.zoomcanal.com.co/Producciones/Cuenteaver/Invitados/ManuelD%C3%A1vilaSguerra/tabid/562/Default.aspx

Imagen 12. Encuentro REDIS 2011
FUENTE: Los autores.

Anexo B. FRAMEWORK

La elevada complejidad de muchas de las aplicaciones informáticas de hoy en día hace prácticamente inviable el desarrollo de aplicaciones sin mecanismos de reutilización, los cuales permitan a los programadores evitar partir desde cero en cada proyecto.

Para ello, una solución ampliamente extendida es la utilización de Frameworks de desarrollo.

¿Qué es?

Podríamos definir un Framework como una estructura conceptual y tecnológica, formada por un conjunto de bloques predefinidos de software, cuya utilización permite la organización y el desarrollo de proyectos software de forma mucho más ágil.

Los desarrolladores pueden utilizar, extender o personalizar estos bloques con el fin de ajustarlos a las necesidades de su proyecto. De esta forma los Frameworks actúan como mecanismos de reutilización permitiendo al programador emplear menos tiempo en la escritura de código de bajo nivel.

Los Frameworks se basan en el Modelo Vista Controlador (MVC), un patrón de diseño que separa las aplicaciones en tres componentes:

- **Modelo**: son los datos o la información que se manejan en la aplicación.

- **Vista**: normalmente representada por una interfaz de usuario, presenta el modelo en un formato elegido.

- **Controlador**: es la capa intermedia. Se encarga de gestionar las peticiones recibidas desde la vista, interactuando con la capa de modelo.[51]

Ventajas

- **Velocidad de desarrollo**. Una vez que aprendemos los aspectos básicos de utilización de un framework, la velocidad de desarrollo aumenta considerablemente. Frameworks como Ruby on Rails son increíblemente rápidos y ágiles, sin exagerar Rails utiliza como uno de sus principios el paradigma CoC (Convención sobre Configuración) ganando simplicidad, a la vez que no perdemos flexibilidad. Solo con

[51] TALLER DIGITAL, Universidad de Alicante, 20 de febrero del 2013, Disponible en http://blog.eltallerdigital.com/2010/11/frameworks-de-desarrollo-un-metodo-agil-para-el-desarrollo-de-software/.

esa mentalidad y el soporte provisto por todo el framework, desarrolla un sitio convencional (y no tanto) es cuestión de horas.

- **Código optimizado**. Si bien esto no es una ley, en general un framework que cuenta con una comunidad importante siempre se encuentra optimizado, al menos, sabemos que hay personas trabajando en mejoras constantemente. Lo mismo ocurre con aspectos fundamentales como la seguridad. Utilizar un framework puede ahorrarnos muchos dolores de cabeza. Obviamente queda en nosotros también hacer un buen uso de esta plataforma, pero en general y si programamos con cierta lógica y cuidado, no tendremos demasiados problemas.

- **Reducción de costos**. Utilizar un framework muchas veces nos permite abstraernos a un nivel de programación low-level. No tenemos que preocuparnos de los aspectos de desarrollo más básicos, de esta manera podemos centrarnos en los aspectos más esenciales que se encuentran relacionados directamente con el trabajo que debemos realizar. Esto no siempre va a ocurrir, pero en general si el proyecto no es sobre algo muy específico el framework va a tener mucho del trabajo resuelto.

- **Estándares y convención de código**. Dos aspectos que tranquilamente podemos utilizar en nuestras aplicaciones sin la necesidad de un framework. No obstante tener que utilizarlos de manera casi obligatoria nos empujan a mejorar nuestros programas. Y perdón que vuelva con Rails, pero programar para este frameworks es un placer, código limpio y sencillo, fácil de mantener y todo gracias a la implementación de convenciones.[52]

- Los Frameworks utilizan patrones de diseño, lo cual permite que el código resultante sea limpio y extensible para futuras ampliaciones. Entre ellos destaca el Modelo Vista Controlador que comentamos anteriormente.

  - Aumenta la facilidad de depuración del código gracias al MVC.[53]

- **Eficiencia.** Las tareas que normalmente le tomaría horas y cientos de líneas de código, ahora se pueden hacer en cuestión de minutos con funciones pre-construidas. El Desarrollo llega a ser mucho más fácil, por lo que es más rápido y en consecuencia eficiente.

- **Seguridad.** Un marco ampliamente utilizado tiene grandes implementaciones de seguridad. La gran ventaja es la comunidad detrás de él, donde los usuarios se convierten a largo plazo testers. Si usted encuentra una vulnerabilidad o un agujero

---

[52] KABYTES, Usar o no un Framework, Febrero 6 de 2012, Disponible en http://www.kabytes.com/programacion/usar-o-no-un-framework/.

[53] TALLER DIGITAL, Frameworks de desarrollo: Un método ágil para el desarrollo de software, Universidad de Alicante, Noviembre 17 de 2010, Disponible en http://blog.eltallerdigital.com/2010/11/frameworks-de-desarrollo-un-metodo-agil-para-el-desarrollo-de-software/.

de seguridad, usted puede ir a la página web del marco y hacerle saber al equipo para que pueda arreglarlo.

- **Costo.** Los marcos más populares son gratis, y ya que también ayuda al programador a codificar más rápido, el coste para el cliente final será menor.

- **Soporte.** Como cualquier otra herramienta distribuida, un marco general viene con la documentación, un equipo de apoyo, o grandes foros de la comunidad donde se puede obtener respuestas rápidas.[54]

Existen Frameworks para la gran mayoría de lenguajes utilizados en el desarrollo de aplicaciones. Algunos de los Frameworks más conocidos son Spring o Struts para aplicaciones en Java; ASP.NET para C# o Zend para PHP.

Uno de los framework más utilizados a nivel mundial es **"Symfony"**, es un framework Open Source para desarrollar aplicaciones web en PHP. Originalmente fue concebido por la agencia interactiva Sensio Labs para el desarrollo de sitios web para sus propios clientes. Symfony fue publicado por la agencia en el año 2005, bajo la licencia MIT, de código abierto y hoy se encuentra entre los frameworks líderes de PHP.

Apoyado por Sensio Labs - pero también por una gran comunidad- Symfony tiene una amplia gama de recursos que facilitan su aprendizaje: detallada documentación, soporte de la comunidad (listas de e-mail, IRC, etc.), soporte profesional (consultoría, entrenamientos, etc.), y mucho más.[55]

Este framework es utilizado en empresas como 3Creatives innóvate solutions[56], entre otras.

Para más información ingresa a los siguientes enlaces:

- http://www.springsource.org/
- http://www.grails.org/
- http://struts.apache.org/

---

[54] D'Olivera Ruben, WEBDESIGNER, Pros and cons of using frameworks, http://www.1stwebdesigner.com/design/pros-cons-frameworks/. 20de febrero del 2013. Traducida por el autor.

[55] http://symfony.com/es/about

[56] http://www.3creatives.com/index.php?page=tecnologias

Anexo C. Instrumento Formato de Especificación de Requisitos

Logo	Nombre del Proyecto	
Dependencia:		**Fecha:**
Usuario que genera el requerimiento:		**Cargo del Usuario:**
Analista Responsable:		**Requerimiento Nº:**
Pegar el caso de uso como una imagen		
Descripción del Requerimiento:		
Tipo de Requerimiento	**Funcional**	**No Funcional** ☐
Restricciones del Funcionamiento:		
Requerimientos relacionados		
Curso Normal		
Actor(es)	**Sistema**	
Curso Alterno		
Excepciones		
Localización en la aplicación final.		
Módulo de primer nivel : **TABLAS BASICAS**	**Módulos de nivel inferior** **Ruta: MODELOS** **Opción: Crear_modelo**	
Nombre de Interfaz: **Gui_CrearModelo**	**Perfil de usuario:**	
Aceptación del usuario		
Nombre y firma	**Fecha de aceptación**	
Copyright © **Laboratorio Investigación Ingeniería de Software.**		

Tabla 9. Formato de especificación de requisitos
FUENTE: Los autores.

Anexo D.

Encuesta para la Determinación De Los Factores Que Conllevan A La Formulación De Una Nueva Metodología Ágil De Desarrollo De Software				
Nombre: MILLERLANDY MORENO T.			**Empresa**: ARQUITECSOFT	
Email: millerlandymoreno@gmail.com				
Cargo: INGENIERO DE PROYECTOS			**Título**: INGENIERO DE SISTEMAS	
1. ¿En la empresa donde labora se desarrolla software siguiendo una metodología?				
a) Si [X]			b) No	
2. ¿De ser afirmativa la respuesta anterior, que metodología de desarrollo de software es utilizada?				
a) RUP	b) ICONIX	c) SCRUM	d) XP	e) OTRA. [X] CUAL: METODOLOGIA INTERNA, CONSISTE EN LA ADAPATACION DE LAS METODOLOGIAS A LOS PROCESOS DE DESARROLLO SEGÚN EL TIPO DE PROYECTO.
3. ¿Al momento de utilizar la metodología, se cumple al pie de la letra los pasos propuestos por esta?				
a) Si[X]			b) No	
4. ¿De ser negativa la respuesta anterior, en cual fase es donde se omiten más pasos?				
a) Análisis	b) Diseño	c) Implementación	d) Pruebas	
5. ¿Los clientes eligen con que metodología desean que desarrollen sus productos?				
a) Si		b) No [X]		
6. ¿De ser afirmativa la respuesta anterior, cuáles de las siguientes metodologías prefieren los clientes?				
a) RUP	b) ICONIX	c) SCRUM	d) XP	e) OTRA. CUAL_____
7. ¿Según su experiencia y criterio que metodología de desarrollo prefiere utilizar?				
a) RUP	b) ICONIX	c) SCRUM	d) XP	e) OTRA. [X] CUAL: POR PROCESOS DE EFICIENCIA SEGUIR UN METODOLOGIA AL PIE DE LA LETRA TIENE UN COSTO ALTO EN TIEMPO QUE AFECTA LA EFECTIVIDAD DE UN PROYECTO, POR ESTE MOTIVO SE REALIZO INTERNAMENTE EL ESTUDIO Y SE DESARROLLO LA METODOLOGIA, QUE A PARTE DE CUMPLIR LAS FASES DE ANALISIS, DISEÑO IMPLEMENTACION Y PRUEBAS, ES AGIL Y EFECTIVA.
Copyright © **Laboratorio Investigación Ingeniería de Software.**				

Anexo E. Documento de Arquitectura del Software

<Logo de la Empresa>

<Logo de la Empresa Cliente>

Documento de Arquitectura del Software
Proyecto: <Nombre del Proyecto>

Versión: <x.y.z>

Nota: El texto incluido en rectángulos azules y el exhibido en cursiva azul se incluye con el fin de proporcionar una guía para el llenado de este documento y debe ser eliminado antes de publicar el documento.

Descripción del proceso: En este documento se da una descripción de la arquitectura del sistema y contiene varios modelos que muestran aspectos distintos del sistema como son: modelo de diseño, modelo de datos, modelo de casos de uso y el modelo del despliegue. Es un resumen de las ideas principales del diseño.

Historial de Revisiones

Versión	Fecha	Autor	Descripción
<x.y.z>	*<dd/mm/aaaa>*	*<nombre>*	*<especificaciones>*

Índice de Contenido

Documento de Arquitectura del Software

1 Introducción

1.1 Objetivo

Describa el objetivo (esto debería ser relativamente corto).

1.2 Alcance

Describir el alcance, mencionar los proyectos asociados y determinar que se ve afectado por este documento.

1.3 Documentos relacionados

Para poder visualizar las referencias a otros documentos, se debe de llenar la tabla que se muestra a continuación:

Título	Fecha	Organización	Identificador del documento
<título>	*<dd/mm/aaaa>*	*<nombre>*	*<Id documento>*

2 Resumen Arquitectónico

2.1 Hechos más Importantes

En este apartado se debe responder la siguiente pregunta:
¿Cuáles son los hechos más importantes que un desarrollador debería saber acerca de esta arquitectura de sistema?
Párrafo o viñetas.

2.2 Estilo Arquitectónico

En este apartado se debe responder la siguiente pregunta:
¿Qué estilo de arquitectura de software está siendo usado?
Aplicación de escritorio para proceso simple (con módulos de extensión de plugins). Cliente-servidor con clientes delgados y servidor personalizados.
Aplicación Web de 2-puertos: servidor web/servidor de aplicaciones, base de datos.

Aplicación Web de 3-puertos: servidor web/servidor de aplicaciones, base de datos.
Servicio web simple: servidor de aplicaciones, base de datos.
Servicios de Red o Web.
Cliente-a-cliente sin servidor central.
Con tuberías y filtros.
Malla de computadoras / servidores distribuidos.

2.3 Objetivos de la Arquitectura

2.3.1 Facilidad de Integración

Mencione los componentes que trabajarán juntos y que pueden funcionar con otros componentes no pertenecientes al sistema

2.3.2 Expansibilidad

Menciones las nuevas características o componentes que podrán ser añadidos fácilmente después que haya sido desplegado el sistema.

2.3.3 Ajuste a la capacidad

Mencione si los componentes que utilizará la arquitectura propuesta, utilizará equipos que proveen los recursos necesarios con un gasto total razonable.

3 Componentes Significativos de la Arquitectura del Sistema

Los componentes de este sistema deben estar definidos claramente en los diagramas de componentes hechos con UML.
Describa brevemente cada componente del sistema que sea relevante para la arquitectura del sistema. Enfóquese en los detalles arquitectónicos tales como mecanismos de comunicación, aspectos de entorno que afecten el desarrollo, y concurrencia. Observe los aspectos claves de cada interfaz, pero evite duplicar los detalles de las interfaces que se especifican en los diagramas de clase de UML u otros documentos.
Los componentes de este sistema se encuentran listados abajo por tipo:

3.1 Presentación/Componentes de la Interfaz de Usuario

C-00: NOMBRE DEL COMPONENTE	
Descripción:	*Descripción*
Requerimientos:	*Sistema operativo, RAM, etc.*
Interfaces Disponibles:	*Describa brevemente las interfaces*

3.2 Componentes Lógicos de la Aplicación

C-10: NOMBRE DEL COMPONENTE	
Descripción:	*Descripción*
Requerimientos:	*Sistema operativo, RAM, etc.*
Interfaces Disponibles:	*Describa brevemente las interfaces*

3.3 Componentes de Almacenamiento de Datos

C-20: NOMBRE DEL COMPONENTE	
Descripción:	*Descripción*
Requerimientos:	*Sistema operativo, RAM, etc.*
Interfaces Disponibles:	*Describa brevemente las interfaces*

4 Vista Conceptual

La vista conceptual es usada para definir los requerimientos funcionales y la visión que los usuarios del negocio tienen de la aplicación y describir el modelo de negocio que la arquitectura debe cubrir. Esta vista estará descripta en términos de casos de uso, diagramas de actividad, procesos de negocio, entidades del negocio, etc. que definen la funcionalidad que la aplicación deberá brindar.
Esta vista muestra los subsistemas y módulos más significativos arquitectónicamente en los que se divide la aplicación, así mismo se debe describir la funcionalidad que se brinda dentro de cada uno de ellos.

5 Vista Lógica

Muestra los componentes principales de diseño y sus relaciones de forma independiente de los detalles técnicos y de cómo la funcionalidad será implementada en la plataforma de ejecución. Se describe la solución en términos de paquetes y clases de diseño. Dentro de esta vista se describe los casos de uso, subsistemas, paquetes y clases de los casos de uso más significativos arquitectónicamente, así como también se señala las funciones, relaciones, atributos y operaciones de estos.

6 Vista de Desarrollo

Esta vista ilustra la distribución del procesamiento entre los distintos equipos que conforman la solución, incluyendo los servicios y procesos de base. Entre algunos de los aspectos a los que puede hacer referencia están:

6.1 Distribución del Procesamiento

Se debe ilustrar la distribución del procesamiento entre los distintos equipos que conforman la solución, incluyendo los servicios y procesos de base. Los elementos definidos en la vista lógica se
"mapean" a componentes de software (servicios, procesos, etc.) o de hardware que definen más precisamente como se ejecutará el sistema.

6.2 Aspectos/Recursos del Ambiente que Serán Compartidos

En este apartado se debe responder la siguiente pregunta:
¿Qué aspectos/recursos del ambiente se encontrarán compartidos?

todo se encuentra en un solo servidor de forma que todos los recursos de los Equipos se encuentran compartidos por todos los componentes.
Todos los equipos comparten el mismo ancho de banda hacia Internet. Todos los equipos acceden el mismo servidor de archivos. Así que, si un componente hace uso de los recursos intensivamente, los otros componentes tendrán que esperar.

6.3 Equipos Utilizados para la Redundancia y/o el Balanceo de Cargas

En este apartado se debe responder la siguiente pregunta:
¿Cómo son manejadas las respuestas en los servidores redundantes o de balance de carga? No estamos haciendo ninguna clase de balance de carga o redundancia en caso de fallas.
El balance de la carga es manejado en los servidores frontales por un dispositivo de balance de carga del que podemos pocas suposiciones. De cualquier forma, una vez que se ha establecido una sesión de usuario, el mismo servidor frontal se utilizará para todas las solicitudes durante esa sesión.

6.4 Alternativas de Configuraciones de Despliegue Posibles

En este apartado se debe responder la siguiente pregunta:
¿Qué alternativas de configuración de despliegue son posibles?
Solo hay una forma de entrega posible. La base de datos podría ser movida a un equipo diferente con un cambio relativamente sencillo a un archivo de configuración. De otra forma no se puede cambiar nada respecto a la entrega.
El sistema tiene la capacidad de mover el proceso de la base de datos a un equipo separado. Además tiene la capacidad de añadir más servidores frontales. La lógica de la aplicación que corre en el servidor de aplicaciones no puede ser dividida o balanceada.

7 Vista de Implementación

La vista de implementación muestra en general las dependencias y cómo se implementan los componentes físicos del sistema, agrupándolos en subsistemas organizados en capas y jerarquías.

7.1 Capas

Uno de los patrones de diseño más utilizado para cualquier tipo aplicaciones es el de Capas donde, básicamente, se divide los elementos de diseño en paquetes de Interfaz de Usuario, Lógica de Negocio y Acceso a Datos y Servicios.
Describa las diversas capas del sistema y su contenido, detalle los límites entre las capas y las reglas preestablecidas para cada una.
Incluya Para cada capa un diagrama de componentes.

7.2 Relación entre las Capas

Incluya un diagrama que muestre la relación entre las capas.

8 Integración

8.1 Integración de los Componentes y su Comunicación

En este apartado se debe responder la siguiente pregunta:
¿Cómo serán integrados los componentes? Específicamente, ¿cómo se comunicarán?
Todo nuestro código utiliza llamadas directas a procedimiento. La base de datos es accesada con un controlador.
Los componentes dentro del mismo proceso usan llamadas directas a procedimientos o eventos Java estándar. Los plugins son accesados también por medio de una API de llamadas directas a procedimientos y eventos estándar. La comunicación con la base de datos utiliza un controlador JDBC. La comunicación entre los servidores frontales y los de procesos utiliza XML-RPC.

8.2 Mecanismos de la Arquitectura para Futuras Modificaciones o Extensiones

En este apartado se debe responder la siguiente pregunta:
¿Qué mecanismos arquitecturales se utilizan para facilitar futuras extensiones o modificaciones? Podríamos cambiar la base de datos cambiando los controladores. De otra forma las extensiones y las modificaciones solo pueden ser hechas a nivel de diseño. Nuevos componentes de plugin pueden ser cargados dinámicamente, mientras satisfagan la API de plugins. De otra manera, no será posible añadir o cambiar componentes, debido a que esta arquitectura utiliza dependencias directas entre sus componentes en lugar de invocación implícita.
Las extensiones y modificaciones pueden ser hechas a nivel de diseño, pero añadir estos cambios requiere re compilación y tiempo fuera de línea.

9 Aseguramiento de la Calidad

9.1 Alcance del Plan de Calidad

Se debe identificar los componentes y aspectos del sistema que deberán ser evaluados para asegurar que los objetivos de calidad se han alcanzado.

- *Componente-1*
- *Componente-2*
- *Componente-3*
- *Característica-1*
- *Característica-2*

9.2 Objetivos de Calidad

Añada los objetivos para ajustarlos a su proyecto. Agrúpelos por prioridades de acuerdo a los lineamientos de su proyecto.

9.2.1 Esenciales

- *Funcionalidad > Corrección*
- *Funcionalidad > Robustez*

9.2.2 Esperados

- *Funcionalidad > Exactitud*
- *Funcionalidad > Compatibilidad*
- *Funcionalidad > Corrección medible*
- *Usabilidad > Comprensibilidad y Legibilidad*
- *Usabilidad > Apoyo para tareas*
- *Usabilidad > Eficiencia*
- *Usabilidad > Seguridad*
- *Usabilidad > Consistencia y Familiaridad*
- *Usabilidad > Satisfacción Subjetiva*

9.2.3 Deseados

- *Confiabilidad > Consistencia en carga*
- *Confiabilidad > Consistencia bajo concurrencia*
- *Confiabilidad > Disponibilidad bajo carga*
- *Confiabilidad > Longevidad*
- *Eficiencia*
- *Escalabilidad*
- *Escalabilidad > Desempeño bajo carga*
- *Escalabilidad > Grandes volúmenes de datos*
- *Operabilidad*
- *Capacidad de mantenimiento > Comprensibilidad*
- *Capacidad de mantenimiento > Capacidad de evolución*
- *Capacidad de mantenimiento > Capacidad de prueba*

El Documento de Arquitectura del Software fue basado en el realizado por Jason Robbins en el 2003 y la CNTI 2006.