Universidad de las Ciencias Informáticas

Facultad 3

Estrategia metodológica para el desarrollo de Software de Gestión a Distancia basado en Programación Extrema.

Trabajo de Diploma

para optar por el título de Ingeniero en Ciencias Informáticas.

Autoras: Leamny Teresa Fernández Carballo.
Betxy Castellanos Rolo.
Tutor: Lic. Jorge Quesada González.
Asesor: Lic. Nelson Cabrera Egues.

Junio, 2007
"Año 49 de la Revolución"

Declaración de Autoría

Declaro que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año 2007.

Betxy Castellanos Rolo

_____
Autora

Leamny Teresa Fernández Carballo

_____
Autora

Lic. Jorge Quesada González

_____
Tutor

La ciencia humana consiste más en destruir errores que en descubrir verdades.
Sócrates.

Agradecimientos

A mi madre y a mi hermano, por ser las personas que más quiero en este mundo. A los dos, gracias por existir.

A mi padre por estar siempre pendiente de mí.

A mi padrastro por ser mi segundo padre y por todo su apoyo incondicional.

A mis tías y abuelas por confiar siempre en mí.

A nuestro tutor por habernos atendido aún en los momentos más malos de su vida.

A nuestro Comandante en Jefe y la Universidad de las Ciencias Informáticas (UCI) por permitirnos lograr este sueño.

A todos nuestros profesores por ser nuestros creadores desde que comenzamos en la Universidad.

A mi novio, por estar siempre presente en todos los momentos buenos y malos.

A Edilberto por tanta ayuda incondicional que nos ha brindado en todos los momentos que lo hemos necesitado.

A nuestros compañeros de aula que son como nuestros hermanos de la universidad en especial a Lily, Soilen y Figueroa.

Dedicatoria

A *mi mamá: que sin ti no hubiera podido realizar este sueño, tu amor, ayuda y dedicación ha sido importante en toda mi vida universitaria, por haberme dado la vida, a ti por ser un buen ejemplo y porque este también es tu sueño.*

A mi hermano: por haberme dado el ejemplo hace 3 años atrás y por tenerme siempre en cuenta como yo a ti.

A mi papá: Julián por darme su apoyo en todo y su amor.

A mi padrastro: Noel por el incondicional cariño y por quererme como una hija más.

A mis abuelas: Virginia y Carmelina por todo su amor y ternura.

A mi tía: porque siempre has estado presente en mi pensamiento.

A mis primas: para que este sueño la sirva de ejemplo.

A mi novio: mi amor por todo tu amor y cariño en estos 5 años de carrera.

En fin a toda mi familia porque este sueño también es parte de ustedes.

Resumen

La Arquitectura de Software constituye tema de actualidad y forma parte de problemáticas inherentes al tema, la pobre información que existe sobre cómo orientar en la modalidad a distancia el uso de metodologías ágiles de modo que se favorezca la preparación del Ingeniero Informático. Los estudios buscan soluciones al proponer una **Estrategia de capacitación a distancia para el uso de la Metodología Ágil XP en proyectos de Ingeniería de Software**. El trabajo consta de tres capítulos, el primero aborda los fundamentos teóricos-metodológicos de la capacitación a distancia por medio de recursos informáticos; el segundo refleja información preliminar del estado actual en el uso de metodologías ágiles para proyectos de desarrollo de software y la propuesta de la estrategia y el tercero contempla los resultados de la aplicación de la estrategia propuesta a partir de un conjunto de instrumentos, donde se incluye el criterio de especialistas.

ÍNDICE

INTRODUCCIÓN

El Gobierno Revolucionario, con gran esfuerzo, ha asignado en la última década gran parte del presupuesto de la nación a sufragar los gastos para las edificaciones de la Universidad de Ciencias Informáticas y las Facultades Territoriales, así como la preparación de decenas de miles de estudiantes de Ingeniería en Informática, de modo que ocupa una posición privilegiada en el mundo debido a los resultados alcanzados en este Programa de la Revolución.

En la nación se dan pasos sólidos en el desarrollo en la Informática y las Comunicaciones creando condiciones para en el futuro, en que además de resolver los problemas internos en lo correspondiente a la demanda de software, para la educación y otras entidades; poder en el futuro brindar asesoramiento técnico a otros países hermanos. Sin embargo se precisa la actualización constante e inserción en proyectos a distancia, que están llamados a ser las alternativas para el futuro próximo, tanto en el comercio, como el intercambio de tecnología y otros avances científico-técnicos. No obstante con estas proyecciones, aún la Informática a escala nacional no ha creado suficientes recursos metodológicos que preparen a los coordinadores de proyectos, y su equipo de guionistas, diseñadores, especialistas de medios audiovisuales y programadores, para el trabajo con Metodologías Ágiles para Arquitectura de Software a Distancia.

En las Universidades se favorece la preparación de los individuos para su desempeño posterior como profesionales en determinadas ramas, tomando en cuenta las condiciones socioeconómicas y necesidades sociales del país en que se encuentran.

La Informática como ciencia del tratamiento racional de la Información, considerada como soporte de los conocimientos humanos en los campos técnico, económico y social, está permitiendo a costos, cada vez más bajos, obtener calidades superiores en un menor tiempo y con un menor esfuerzo.

En Cuba, el desarrollo de software se ha convertido en una actividad fundamental como medio de soporte a la actividad interna de las entidades que conforman la Empresa Cubana. En un mundo en constante

revolución, la búsqueda de soluciones para potenciar el desarrollo de sistemas computacionales ocupa un lugar preponderante en la agenda del personal involucrado.

Es por este motivo, que con el objetivo de introducir su utilización en la empresa de software cubana, se realiza el análisis de uno los exponentes más populares de las metodologías ágiles existentes hoy día: la Programación Extrema [1](XP), ya que en muchas empresas del mundo la metodología que más se utiliza para la realización de software es la XP.

En estudios realizados por las investigadoras en el campo de las Metodologías Ágiles para Arquitectura de Software a Distancia, en dependencias de la UCI y especialistas del territorio de Ciego de Ávila consultados, se pudieron registrar las siguientes irregularidades que conforman la ***problemática****:*

- La Información sobre Metodologías Ágiles para Arquitectura de Software a Distancia es limitada.
- No se conocen estrategias para usar las metodologías ágiles en la producción de software de gestión a distancia.
- Los líderes de proyectos, profesores y estudiantes de UCI no poseen la información sobre Metodologías Ágiles para Arquitectura de Software a Distancia que les permitan enfrentar las demandas de software que se le han asignado a la institución.

Partiendo de la problemática anteriormente planteada se ha determinado el ***problema a resolver****:* ¿Qué requerimientos debe contemplar una estrategia de gestión a distancia para aplicar la metodología ágil en la Arquitectura de Software? La arquitectura de software representa una temática de actualidad que deben tomar en cuenta los directivos que tienen a su cargo la producción de estos recursos informáticos. La investigación en correspondencia con el problema tiene como ***objetivo:*** Proponer una estrategia de gestión a distancia que permita aplicar la Metodología XP Programación Extrema para Arquitectura de Software. En correspondencia con el objetivo y problema planteado se define como ***objeto de estudio:*** proceso de desarrollo a distancia de la arquitectura de software a partir de metodologías ágiles y siendo su ***campo de acción****:* la aplicación de estrategia de gestión a distancia para la Metodología XP Programación Extrema para Arquitectura de Software.

[1] Programación Extrema proviene del término inglés Extreming Programming, dicho término puede aparecer escrito también de la siguiente manera eXtreming Programming resaltando las letras XP, según autores.

Preguntas Científicas

1. ¿Cuáles son los fundamentos teóricos generales de las estrategias de gestión a distancia en el campo de la Informática?
2. ¿Cuál es el estado actual en que se encuentra el uso de las Metodologías Ágiles en el mundo para el desarrollo de la Ingeniera de Software?
3. ¿Qué requerimientos debe poseer una estrategia de gestión a distancia para aplicar una metodología ágil en la Ingeniería de Software?
4. ¿Cómo se podría constatar que esta estrategia es efectiva?

Constituye aspecto esencial en las perspectivas de la Informática desplegar su acción en la modalidad a distancia, dando posibilidad a que se exploten a un mayor nivel los recursos humanos especializados. Para dar respuesta a las interrogantes se desarrollaron las siguientes ***Tareas de la investigación:***

1. Fundamentación teórica general de las estrategias de gestión a distancia en el campo de Informática y la Ingeniería de Software.
2. Diagnóstico del estado actual en el uso de Metodologías Ágiles para Ingeniería de Software en la modalidad de capacitación a distancia.
3. Elaboración de una Estrategia de Gestión a Distancia para la Metodología Ágil XP **Extreming Programming de Ingeniería de Software.**
4. Validación teórico-práctica de la Estrategia de Gestión a Distancia para la Metodología Ágil XP **Extreming Programming de Ingeniería de Software.**

En la investigación se tomó como población 123 estudiantes y 24 profesores de los proyectos S.I.G.I.A.; O.N.E. y Registro y Notaría. La muestra intencional la componen 60 estudiantes y 8 profesores que representan el 48% y 40% de la población respectivamente.

Métodos del Nivel Teórico:

- **Análisis – Síntesis:** en el procesamiento de la bibliografía disponible, con el objetivo de exponer los principios fundamentales para la solución del problema científico.

- **Histórico – Lógico:** con el fin de estudiar las tendencias actuales sobre las Metodologías Ágiles y la producción del Software a distancia.
- **Modelación** para crear abstracciones en aras de explicar la realidad, en el proceso de diseño y concepción de la estrategia.

Métodos del Nivel Empírico.

- **Análisis Documental:** en la revisión de la bibliografía sobre las principales Metodologías Ágiles en el desarrollo de software y la producción de estos a distancia.
- **Encuesta a profesores y estudiantes de la UCI:** para determinar estado actual en el uso de metodologías ágiles de Ingeniería de Software en Cuba.
- **Criterio de Especialistas:** Para la validación teórico-práctica de la estrategia aplicada.

Técnicas de procesamiento matemático-estadístico: Se utilizaron la media aritmética y el análisis porcentual con la intención de agrupar, comparar datos y en consecuencia hacer inferencias para conclusiones parciales y finales en el desarrollo de la investigación.

El aporte **teórico** de los estudios radica en la recopilación y reflexiones teóricas sobre el empleo y la importancia de la aplicación de estrategias para metodologías ágiles a distancia en Ingeniería de Software para la preparación del Ingeniero en Arquitectura de Software. La aplicación de las acciones estratégicas que constituyen su principal **aporte práctico**. Lo **novedoso** del trabajo se aprecia en que no existe en la Facultad 3 de la Universidad de las Ciencias Informáticas una estrategia para equipos de proyectos a distancia, que en consecuencia, sirva de guía en la realización del software al encontrarse dichos equipos de desarrolladores y/o integradores en distantes lugares de trabajo.

Variables conceptuales:

Metodologías Ágiles: Proceso disciplinado sobre el desarrollo de software con el fin de hacerlo predecible y eficiente. Orientada con énfasis en el código, integrada en un proceso disciplinado sobre el desarrollo de software inspirado por otras disciplinas de la ingeniería con el fin de hacer el producto poco burocrático, pero más predecible y eficiente. También se le llama como Metodologías Ligeras.

Software a Distancia: Aplicación informática integrada por un conjunto de programas cuyo desarrollo ha sido dirigido a distancia. Es además un conjunto integrado de herramientas y servicios de Internet (plataformas que se utilizan íntegramente vía Web) y ponen a disposición de los participantes procesos que dan cumplimiento a su objetivo.

Estrategia metodológica de gestión a distancia: En el trabajo se asume la definición de estrategia metodológica de M. A. Rodríguez del Castillo y A. Rodríguez Palacios. Las autoras sin embargo la definen aplicada a la gestión a distancia como: la proyección de un sistema de acciones a corto, mediano y largo plazo que permite vinculación entre dos entidades con el fin de generar rendimientos laborales, coordinar recursos disponibles y tomar decisiones en favor de objetivos estratégicos en un tiempo concreto.

CAPÍTULO I

Fundamentación Teórica

1.1- Introducción.

El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Como respuesta a estas deficiencias aparecen las Metodologías Ágiles que aplicadas correctamente es una alternativa eficiente para el desarrollo de software en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

El movimiento del desarrollo ágil de software es una iniciativa que agrupa una serie de metodologías (XP, FDD, SCRUM, Crystal Clear, etc.) que se basan en la adaptabilidad ante el cambio como medio para aumentar las posibilidades de éxito de un proyecto. En general los procesos ágiles se centran en las personas; en su comunicación directa y sus habilidades.

1.2- Software de Gestión.

El Software de Gestión es el encargado de implantar las aplicaciones de gestión de acuerdo a las necesidades del cliente en el sistema informático de las empresas.

1.2.1- Actividades y fases que despliega un software de gestión.

Consultoría y asesoría de los procesos de negocio del cliente.

Implantaciones de aplicaciones de software estándar de gestión, que conlleva las siguientes fases:

- Definición del procedimiento de implantación.
- Personalización de la aplicación.
- Estudio de migración de datos
- Formación a usuarios.
- Puesta en marcha de la implantación de la aplicación.
- Evaluación de la implantación de la aplicación.

Servicios de mantenimiento de las implantaciones, como pueden ser:

- Resolución de dudas y problemas derivados del uso de la aplicación.
- Implantación de nuevas versiones del software.
- Implantación de nuevas necesidades del cliente.
- Formación a usuarios nuevos de la aplicación.
- Formación de reciclaje a usuarios de la aplicación. (Morato, 2007)

1.2.2- Tendencias en el desarrollo de estrategias de gestión.

Las estrategias de gestión actualmente se encuentran enmarcadas en el desarrollo exitoso y cada vez más proliferante del mundo de la gestión de información y conocimiento. Esta es una expresión de aparición reciente que, sin embargo, hace referencia a una práctica que se apoya en una idea económica clásica: no perder tiempo y dinero dedicándose a actividades en las que no se posee ventaja comparativa.

Las principales tendencias en el campo de la informática vendrán de dadas por:

1. Nuevas modalidades de provisión.
2. Estrategias de suministro de servicio.
3. Estrategias contractuales.
4. Perfil del proveedor.

1.2.2.1- Nuevas modalidades de provisión.

En lo que respecta a las nuevas modalidades de provisión, el mercado está experimentando el surgimiento de una estrategia de gestión que se acerca cada vez más al núcleo de competencia del negocio. Consistente básicamente en delegar en un tercero ciertos procesos de Informática y Telecomunicaciones (IT) implicados de forma intensiva en el negocio, como por ejemplo: la gestión de la relación de clientes, será atractivo debido a la entrada en el mercado de competidores distintos de los tradicionales, a precios más bajos.

Las empresas adoptarán nuevas estrategias de gestión por considerarla:

- **Facilitadora** de la gestión de los cambios resultantes de fusiones y adquisiciones.
- **Capacitadora** de una reducción de costes, o de proporcionar una estructura de costes más predictible.
- **Mejoradora** de la fiabilidad de funciones que son difíciles de gestionar.
- **Generadora** de la conservación de un alto estándar tecnológico sin necesidad de grandes inversiones de capital.

1.2.2.2- Estrategia en el suministro de servicios.

En cuanto a las estrategias de suministro de servicios, se va perfilando, por una parte, la tendencia hacia un modelo de suministro global, con prácticas de gestionabilidad en las que el cliente se implicará cada vez más, y por otro lado un cambio desde concentración en funciones de infraestructura y mantenimiento de back-office a un enfoque en funciones de front-office, atención al cliente y potenciación del negocio.

El suministro global incorporará un amplio rango de servicios con una única fuente, modelo que impondrá un control estricto del cliente. Por una parte, los destinatarios de servicios de externalización han madurado y están en condiciones de imponer al proveedor, a través de modelos de relación utilizados en toda su capacidad, las siguientes normas:

- Voluntad del proveedor en adoptar una franca postura de acercamiento al negocio que se concrete en una transparencia en la presentación y detalle de los niveles de servicio establecidos

- Acuerdos de nivel de servicio establecidos adecuados al servicio que se requiere.
- Mecanismos de tarificación adecuados al coste real del servicio.
- Posibilidad de renegociación de costes.

En lo que respecta a las variables tecno-económicas que servirán de marco al modelo de suministro podemos citar, entre otras, las siguientes:

- Los **requerimientos de E-business** producirán un incremento importante en el tráfico de red, y por consiguiente en los costes de servicios de red, por lo que la externalización de los servicios de comunicaciones tendrá cada vez mayor presencia.
- La **tendencia a la externalización** cambiará poco a poco las relaciones cliente/proveedor de software. Los proveedores de software se dirigirán cada vez más al suministrador del servicio de externalización, que gestionará y optimizará el uso del software para su cliente.
- La mayor presencia de **soluciones empaquetadas** implicará que el proveedor de servicios de externalización será un selector de la mejor solución para su cliente, más que un desarrollador de soluciones,
- Otras áreas tecnológicas con importantes perspectivas de ser externalizadas son: Gestión de sistemas mainframe, recuperación de desastres, gestión de redes y desarrollo de aplicaciones.

1.2.2.3- Estrategias contractuales.

Existe una elevada probabilidad de que las estrategias contractuales sean diseñadas para hacer frente a las nuevas modalidades de provisión y estrategia de suministro existentes anteriormente. Se mantendrá, sin embargo la elevada valoración que el mercado ha mostrado siempre de una relación contractual prolongada y matizada por una estrecha colaboración. Esto impone contratos de larga duración. El largo plazo de la externalización permanece, por tanto como una constante.

Estas implicaciones no descartan contratos con duración a 3 años, pero dado su atractivo en costes, serán los menos frecuentes, realizándose prórrogas si estos han sido satisfactorios. Finalmente la plurianualidad de los contratos de estas nuevas tendencias queda definida por la reducción de costes que se producen en los últimos años de contrato, así como la evolución prevista u ofertada por los proveedores.

1.2.2.4- Selección del proveedor.

El perfil del proveedor se irá acercando cada vez más al de un agente colaborador que se centre en el servicio orientado al negocio, no al servicio como mera infraestructura para el negocio. El proveedor no será tanto apreciado por su penetración en el mercado como por su capacidad de innovación tecno-económica y sus bajos costes, siendo necesarios proveedores con amplia flexibilidad ante un mercado muy dinámico y con incremento de competencia. (Cortés, 2003)

1.3- La Ingeniería de Software y las metodologías ágiles.

Los Métodos Ágiles no surgieron porque sí, sino que estuvieron motivados por una conciencia particularmente aguda de la crisis del software, por la responsabilidad que se imputa a las grandes metodologías en la gestación de esa crisis y por el propósito de articular soluciones. Los organismos y corporaciones han desarrollado una plétora de estándares comprensivos que han ido jalonando la historia y poblando los textos de metodologías e ingeniería de software. Algunos son verdaderamente métodos; otros, metodologías de evaluación o estimación de conformidad; otros más, estándares para metodologías o meta-modelos. Al lado de ellos se encuentra lo que se ha llamado una ciénaga de estándares generales o específicos de industria a los que se someten organizaciones que desean (o deben) articular sus métodos conforme a diversos criterios de evaluación, vehículos de selección de contratistas o marcos de referencia.

En esta década (2000) se ha comenzado con un creciente interés en metodologías de desarrollo. Hasta hace poco el proceso de desarrollo llevaba asociada un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos), donde por lo general se exige un alto grado de ceremonia en el proceso.

Las metodologías ágiles constituyen una solución a medida que se produce desarrollo en el campo de la Informática y que como el caso de Cuba, debido a la Universalización que prepara especialistas para todo el país; se precisa la intervención de recursos humanos con especialización y responsabilidad en los

equipos de desarrollo de software que se encuentran dispersos y se necesita su preparación y debida orientación para el trabajo en la Ingeniería de Software. Para la realización de software estos aportes son importantes y sin descuidar las prácticas esenciales de la calidad del producto final. Las características de los proyectos para los cuales las metodologías ágiles han sido especialmente pensadas se ajustan a un amplio rango de proyectos de desarrollo de software; aquellos en los cuales los equipos de desarrollo son pequeños, con lazos reducidos, requisitos volátiles, y/o basados en nuevas tecnologías.

Las acertadamente conocidas como metodologías ligeras o de poco peso han ido cobrando auge en los últimos tiempos. Estos métodos surgen como contraparte de sus dos antecesores, buscando un punto central entre la no existencia de un proceso y un proceso exageradamente grande. Las metodologías ligeras también conocidas como “metodologías ágiles” realzan más la importancia de la programación del código que la documentación del proceso.

En el país, el desarrollo de software se ha convertido en una actividad fundamental como medio de soporte a la actividad interna de las entidades que conforman la Empresa Cubana. En un mundo en constante revolución, la búsqueda de soluciones para potenciar el desarrollo de sistemas computacionales ocupa un lugar preponderante en la agenda del personal involucrado.

1.4- Bases y antecedentes de las metodologías ágiles.

1.4.1- ¿Qué es Metodología Ágil?

Las metodologías imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar inspirado por otras disciplinas de la ingeniería.

Las metodologías ingenieriles han estado presentes durante mucho tiempo. No se han distinguido precisamente por ser muy exitosas. Aún menos por su popularidad. La crítica más frecuente a estas metodologías es que son burocráticas.

Como una reacción a estas metodologías, un nuevo grupo de metodologías ha surgido en los últimos años. Durante algún tiempo se conocían como las metodologías ligeras, pero el término aceptado ahora es **metodologías ágiles**.

El resultado de todo esto es que los métodos ágiles cambian significativamente algunos de los énfasis de los métodos ingenieriles. La diferencia inmediata es que son menos orientados al documento, exigiendo una cantidad más pequeña de documentación para una tarea dada. En muchas maneras son más bien orientados al código: siguiendo un camino que dice que la parte importante de la documentación es el código fuente.

Sin embargo éste no es el punto más importante sobre los métodos ágiles. Hay diferencias mucho más profundas:

- *Los métodos ágiles son adaptables en lugar de predictivos*. Los métodos ingenieriles tienden a intentar planear una parte grande del proceso del software en gran detalle para un plazo grande de tiempo, esto funciona bien hasta que las cosas cambian. Así que su naturaleza es resistirse al cambio. Para los métodos ágiles, no obstante, el cambio es bienvenido. Intentan ser procesos que se adaptan y crecen en el cambio, incluso al punto de cambiarse ellos mismos.
- *Los métodos ágiles son orientados a la gente y no orientados al proceso*. La meta de los métodos ingenieriles es definir un proceso que funcionará bien con cualquiera que lo use. Los métodos ágiles afirman que ningún proceso podrá nunca maquillar las habilidades del equipo de desarrollo, de modo que el papel del proceso es apoyar al equipo de desarrollo en su trabajo. Explícitamente puntualizan el trabajar a favor de la naturaleza humana en lugar de en su contra y enfatizan que el desarrollo de software debe ser una actividad agradable. (Fowler, 2003)

1.4.2- Antecedentes: el Manifiesto Ágil.

En una reunión celebrada en febrero de 2001 en Utah-EEUU, nace el término “ágil” aplicado al desarrollo de software. En esta reunión participan un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los

cambios que puedan surgir a lo largo del proyecto. Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas. Varias de las denominadas metodologías ágiles ya estaban siendo utilizadas con éxito en proyectos reales, pero les faltaba una mayor difusión y reconocimiento.

Tras esta reunión se creó The Agile Alliance[2], una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida fue el Manifiesto Ágil, un documento que resume la filosofía "ágil".

1.4.3- Valores y principios del Manifiesto Ágil.

Según el Manifiesto se valora:

- Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas. La gente es el principal factor de éxito de un proyecto software. Es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.
- Desarrollar software que funciona más que conseguir una buena documentación. La regla a seguir es "no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante". Estos documentos deben ser cortos y centrarse en lo fundamental.
- La colaboración con el cliente más que la negociación de un contrato. Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.
- Responder a los cambios más que seguir estrictamente un plan. La habilidad de responder a los cambios que puedan surgir a los largo del proyecto (cambios en los requisitos, en la tecnología, en

[2] www.agilealliance.com

el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

Los valores anteriores inspiran los doce principios del manifiesto. Son características que diferencian un proceso ágil de uno tradicional. Los dos primeros principios son generales y resumen gran parte del espíritu ágil. El resto tienen que ver con el proceso a seguir y con el equipo de desarrollo, en cuanto metas a seguir y organización del mismo. Los principios son:

I. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.
II. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
III. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
IV. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
V. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
VI. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
VII. El software que funciona es la medida principal de progreso.
VIII. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
IX. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
X. La simplicidad es esencial.
XI. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
XII. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento. (Canós, y otros, 2004)

1.5- Metodologías ágiles: principales características.

Entre las metodología ágiles más conocidas se encuentran: Scrum; Crystal Methods; FDD (Feature Driven Development) y XP (Programación Extrema), entre otras.

1.5.1- Scrum.

Esta es, después de XP, la metodología ágil mejor conocida y la que otros métodos ágiles recomiendan como complemento. (Reynoso, 2004) Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Procede de la terminología del juego de rugby. Scrum es adaptativo, ágil, auto-organizante y con pocos tiempos muertos. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración. (Canós, y otros, 2004)

1.5.1.1- Ciclo de vida de Scrum. Figura 1.1

1. **Pre-Juego: Planeamiento.** El propósito es establecer la visión, definir expectativas y asegurarse la financiación. Las actividades son la escritura de la visión, el presupuesto, el registro de acumulación o retraso (backlog) del producto inicial y los ítems estimados, así como la arquitectura de alto nivel, el diseño exploratorio y los prototipos. El registro de acumulación es de alto nivel de abstracción.
2. **Pre-Juego: Montaje (Staging).** El propósito es identificar más requerimientos y priorizar las tareas para la primera iteración. Las actividades son planificación, diseño exploratorio y prototipos.
3. **Juego o Desarrollo**. El propósito es implementar un sistema listo para entrega en una serie de iteraciones de treinta días llamadas “corridas” (sprints). Las actividades son un encuentro de planeamiento de corridas en cada iteración, la definición del registro de acumulación de corridas y los estimados, y encuentros diarios de Scrum.
4. **Pos-Juego: Liberación.** El propósito es el despliegue operacional. Las actividades, documentación, entrenamiento, mercadeo y venta.

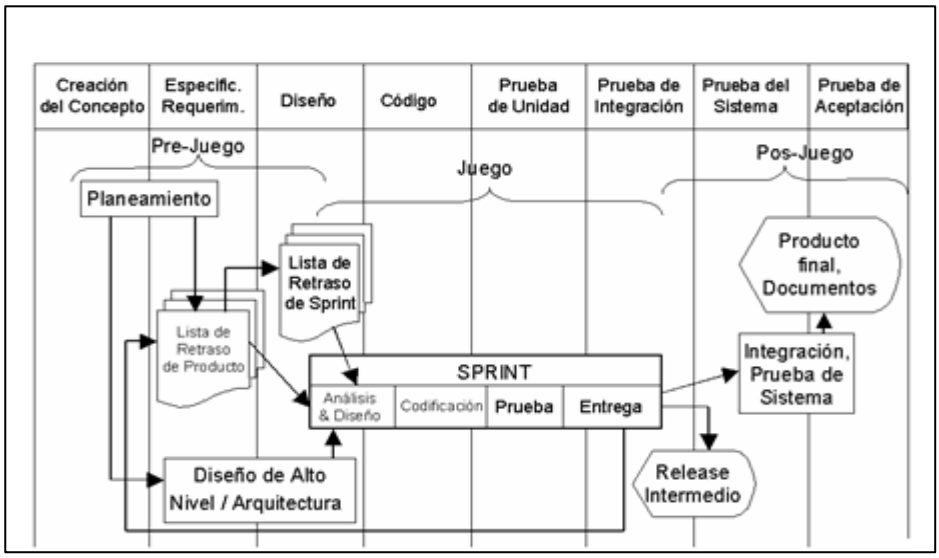

Figura 1.1- Ciclo de Scrum

1.5.1.2- Los valores de Scrum son:

- Equipos auto-dirigidos y auto-organizados. No hay manager que decida, ni otros títulos que "miembros del equipo"; la excepción es el Scrum Master que debe ser 50% programador y que resuelve problemas, pero no manda. Los observadores externos pueden observar, pero no interferir ni opinar.
- Una vez elegida una tarea, no se agrega trabajo extra. En caso que se agregue algo, se recomienda quitar alguna otra cosa.
- Encuentros diarios. Se realizan siempre en el mismo lugar, en círculo. El encuentro diario impide caer en el dilema ¿Cómo es que un proyecto puede atrasarse un año?
- Iteraciones de treinta días; se admite que sean más frecuentes.
- Demostración a participantes externos al fin de cada iteración.
- Al principio de cada iteración, planeamiento adaptativo guiado por el cliente.

1.5.1.3- Scrum define seis roles:

1. **El Scrum Master.** Interactúa con el cliente y el equipo. Es responsable de asegurarse que el proyecto se lleve a cabo de acuerdo con las prácticas, valores y reglas de Scrum y que progrese según lo previsto. Coordina los encuentros diarios, formula las tres preguntas

canónicas y se encarga de eliminar eventuales obstáculos. Debe ser miembro del equipo y trabajar a la par.

2. **Propietario del Proyecto.** Es el responsable oficial del proyecto, gestión, control y visibilidad de la lista de acumulación o lista de retraso del producto. Es elegido por el Scrum Master, el cliente y los ejecutivos a cargo. Toma las decisiones finales de las tareas asignadas al registro y convierte sus elementos en rasgos a desarrollar.
3. **Equipo de Scrum.** Tiene autoridad para reorganizarse y definir las acciones necesarias o sugerir remoción de impedimentos.
4. **Cliente.** Participa en las tareas relacionadas con los ítems del registro.
5. **Management.** Está a cargo de las decisiones fundamentales y participa en la definición de los objetivos y requerimientos. Por ejemplo, selecciona al Dueño del Producto, evalúa el progreso y reduce el registro de acumulación junto con el Scrum Master.
6. **Usuario.**

La dimensión del equipo total de Scrum no debería ser superior a diez ingenieros. El número ideal es siete, más o menos dos. Si hay más, lo más recomendable es formar varios equipos. No hay una técnica oficial para coordinar equipos múltiples, pero se han documentado experiencias de hasta 800 miembros, divididos en Scrums de Scrum, definiendo un equipo central que se encarga de la coordinación, las pruebas cruzadas y la rotación de los miembros. (Reynoso, 2004)

1.5.2- Crystal Methods (CC).

Las metodologías Crystal fueron creadas por el “antropólogo de proyectos” Alistair Cockburn. (Reynoso, 2004) Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros). (Canós, y otros, 2004)

1.5.2.1- Ciclo de vida de Crystal Clear:

Crystal Clear enfatiza el proceso como un conjunto de ciclos anidados. En la mayoría de los proyectos se perciben siete ciclos: (1) el proyecto, (2) el ciclo de entrega de una unidad, (3) la iteración (nótese que CC requiere múltiples entregas por proyecto pero no muchas iteraciones por entrega), (4) la semana laboral, (5) el período de integración, de 30 minutos a tres días, (6) el día de trabajo, (7) el episodio de desarrollo de una sección de código, de pocos minutos a pocas horas.

1.5.2.2- Los siete valores o propiedades de Crystal Clear son:

1. **Entrega frecuente.** Consiste en entregar software a los clientes con frecuencia, no solamente en compilar el código. La frecuencia dependerá del proyecto, pero puede ser diaria, semanal, mensual o lo que fuere.
2. **Comunicación osmótica.** Todos juntos en el mismo cuarto. Una variante especial es disponer en la sala de un diseñador.
3. **Mejora reflexiva.** Tomarse un pequeño tiempo (unas pocas horas cada cierto tiempo una vez al mes) para pensar bien qué se está haciendo, cotejar notas, reflexionar, discutir.
4. **Seguridad personal.** Hablar cuando algo molesta: decirle amigablemente al manager que la agenda no es realista, o a un colega que su código necesita mejorarse. Esto es importante porque el equipo puede descubrir y reparar sus debilidades. Técnicamente, estas cuestiones se han caracterizado como una importante variable de confianza.
5. **Foco.** Saber lo que se está haciendo y tener la tranquilidad y el tiempo para hacerlo. Lo primero debe venir de la comunicación sobre dirección y prioridades, típicamente con el Patrocinador Ejecutivo. Lo segundo, de un ambiente en que la gente no se vea compelida a hacer otras cosas incompatibles.
6. **Fácil acceso a usuarios expertos.** Un encuentro semanal o semi-semanal con llamados telefónicos adicionales parece ser una buena pauta. Otra variante es que los programadores se entrenen para ser usuarios durante un tiempo. El equipo de desarrollo, de todas maneras, incluye un Experto en Negocios.

7. **Ambiente técnico con prueba automatizada, management de configuración e integración frecuente.** Microsoft estableció la idea de los builds cotidianos, y no es una mala práctica. Muchos equipos ágiles compilan e integran varias veces al día.

1.5.2.3- <u>En cuanto a las técnicas, se favorecen:</u>

1. **Entrevistas de proyectos.** Se suele entrevistar a más de un responsable para tener visiones más ricas. La idea es averiguar cuales son las prioridades, obtener una lista de rasgos deseados, saber cuáles son los requerimientos más críticos y cuáles los más negociables. Si se trata de una actualización o corrección, saber cuáles son las cosas que se hicieron bien y merecen preservarse y los errores que no se quieren repetir.
2. **Talleres de reflexión.** El equipo debe detenerse treinta minutos o una hora para reflexionar sobre sus convenciones de trabajo, discutir inconvenientes y mejoras y planear para el período siguiente.
3. **Planeamiento Blitz.** Una técnica puede ser el Juego de Planeamiento de XP. En este juego, se ponen tarjetas indexadas en una mesa, con una historia de usuario o función visible en cada una. El grupo finge que no hay dependencias entre tarjetas, y las alinea en secuencias de desarrollo preferidas. Los programadores escriben en cada tarjeta el tiempo estimado para desarrollar cada función. El patrocinador o embajador del usuario escribe la secuencia de prioridades, teniendo en cuenta los tiempos referidos y el valor de negocio de cada función. Las tarjetas se agrupan en períodos de tres semanas llamados iteraciones que se agrupan en entregas (releases), usualmente no más largas de tres meses.
4. **Estimación Delphi con estimaciones de pericia.** En el proceso Delphi se reúnen los expertos responsables y proceden como en un remate para proponer el tamaño del sistema, su tiempo de ejecución, la fecha de las entregas según dependencias técnicas y de negocios y para equilibrar las entregas en paquetes de igual tamaño.
5. **Encuentros diarios de pie.** La palabra clave es “brevedad”, cinco a diez minutos como máximo. No se trata de discutir problemas, sino de identificarlos. Los problemas sólo se discuten en otros encuentros posteriores, con la gente que tiene que ver en ellos. La técnica se origina en Scrum.
6. **Miniatura de procesos.** Una forma de presentar Crystal Clear puede insumir entre 90 minutos y un día. La idea es que la gente pueda “degustar” la nueva metodología.

7. **Gráficos de quemado.** Se trata de una técnica de graficación para descubrir demoras y problemas tempranamente en el proceso, evitando que se descubra demasiado tarde que todavía no se sabe cuánto falta. Para ello se hace una estimación del tiempo faltante para programar lo que resta al ritmo actual, lo cual sirve para tener dominio de proyectos en los cuales las prioridades cambian bruscamente y con frecuencia.
8. **Programación lado a lado.** La versión de Crystal Clear establece proximidad, pero cada quien se aboca a su trabajo asignado, prestando un ojo a lo que hace su compañero, quien tiene su propia máquina. Esta es una ampliación de la Comunicación Osmótica al contexto de la programación.

1.5.2.4- Roles Crystal Clear.

Hay ocho roles nominados en Crystal Clear: Patrocinador, Usuario Experto, Diseñador Principal, Diseñador-Programador, Experto en Negocios, Coordinador, Verificador, Escritor. A continuación se describen en bastardilla los artefactos de los que son responsables los roles de Crystal Clear, detalladamente descriptos en la documentación.

1. **Patrocinador.** Produce la Declaración de Misión con Prioridades de Compromiso. Consigue los recursos y define la totalidad del proyecto.
2. **Usuario Experto**. Junto con el Experto en Negocios produce la Lista de Actores-Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe familiarizarse con el uso del sistema, sugerir atajos de teclado, modos de operación, información a visualizar simultáneamente, navegación, etcétera.
3. **Diseñador Principal.** Produce la Descripción Arquitectónica. Se supone que debe ser al menos un profesional de Nivel 3. (En los Métodos Ágiles se definen tres niveles de experiencia: Nivel 1 es capaz de “seguir los procedimientos”; Nivel 2 es capaz de “apartarse de los procedimientos específicos” y encontrar otros distintos; Nivel 3 es capaz de manejar con fluidez, mezclar e inventar procedimientos). El Diseñador Principal tiene roles de coordinador, arquitecto, mentor y programador más experto.
4. **Diseñador-Programador.** Produce, junto con el Diseñador Principal, los Borradores de Pantallas, el Modelo Común de Dominio, las Notas y Diagramas de Diseño, el Código Fuente, el Código de

Migración, las Pruebas y el Sistema Empaquetado. Un programa en CC es “diseño y programa”; sus programadores son diseñadores-programadores.

5. **Experto en Negocios.** Junto con el Usuario Experto produce la Lista de Actores-Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe conocer las reglas y políticas del negocio.
6. **Coordinador.** Con la ayuda del equipo, produce el Mapa de Proyecto, el Plan de Entrega, el Estado del Proyecto, la Lista de Riesgos, el Plan y Estado de Iteración y la Agenda de Visualización.
7. **Verificador.** Produce el Reporte de Bugs. Puede ser un programador en tiempo parcial, o un equipo de varias personas.
8. **Escritor.** Produce el Manual de Usuario.

1.5.3- Feature Driven Development (FDD).

Define un proceso iterativo que consta de 5 pasos. Las iteraciones son cortas (hasta 2 semanas). Se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software. Sus impulsores son Jeff De Luca y Peter Coad. FDD es un método ágil, iterativo y adaptativo. A diferencia de otros Métodos Ágiles, no cubre todo el ciclo de vida sino sólo las fases de diseño y construcción y se considera adecuado para proyectos mayores y de misión crítica. FDD no requiere un modelo específico de proceso y se complementa con otras metodologías. Enfatiza cuestiones de calidad y define claramente entregas tangibles y formas de evaluación del progreso.

1.5.3.1- Ciclo de vida FDD. Figura 1.2

FDD consiste en cinco procesos secuenciales durante los cuales se diseña y construye el sistema. La parte iterativa soporta desarrollo ágil con rápidas adaptaciones a cambios en requerimientos y necesidades del negocio. Cada fase del proceso tiene un criterio de entrada, tareas, pruebas y un criterio de salida. Típicamente, la iteración de un rasgo insume de una a tres semanas. Las fases son:

1) **Desarrollo de un modelo general.** Cuando comienza este desarrollo, los expertos de dominio ya están al tanto de la visión, el contexto y los requerimientos del sistema a construir. A esta altura se espera que existan requerimientos tales como casos de uso o especificaciones funcionales. FDD, sin embargo, no cubre este aspecto. Los expertos de dominio presentan un ensayo en el que los

miembros del equipo y el arquitecto principal se informan de la descripción de alto nivel del sistema. El dominio general se subdivide en áreas más específicas y se define un ensayo más detallado para cada uno de los miembros del dominio. Luego de cada ensayo, un equipo de desarrollo trabaja en pequeños grupos para producir modelos de objeto de cada área de dominio. Simultáneamente, se construye un gran modelo general para todo el sistema.

2) **Construcción de la lista de rasgos.** Los ensayos, modelos de objeto y documentación de requerimientos proporcionan la base para construir una amplia lista de rasgos. Los rasgos son pequeños ítems útiles a los ojos del cliente. Se escriben en un lenguaje que todas las partes puedan entender. Las funciones se agrupan conforme a diversas actividades en áreas de dominio específicas. La lista de rasgos es revisada por los usuarios y patrocinadores para asegurar su validez y exhaustividad. Los rasgos que requieran más de diez días se descomponen en otros más pequeños.
3) **Planeamiento por rasgo.** Incluye la creación de un plan de alto nivel, en el que los conjuntos de rasgos se ponen en secuencia conforme a su prioridad y dependencia, y se asigna a los programadores jefes. Las listas se priorizan en secciones que se llaman paquetes de diseño. Luego se asignan las clases definidas en la selección del modelo general a programadores individuales, o sea propietarios de clases. Se pone fecha para los conjuntos de rasgos.
4) **Diseño por rasgo y Construcción por rasgo.** Se selecciona un pequeño conjunto de rasgos del conjunto y los propietarios de clases seleccionan los correspondientes equipos dispuestos por rasgos. Se procede luego iterativamente hasta que se producen los rasgos seleccionados. Una iteración puede tomar de unos pocos días a un máximo de dos semanas. Puede haber varios grupos trabajando en paralelo. El proceso iterativo incluye inspección de diseño, codificación, prueba de unidad, integración e inspección de código. Luego de una iteración exitosa, los rasgos completos se promueven al build principal. Este proceso puede demorar una o dos semanas.

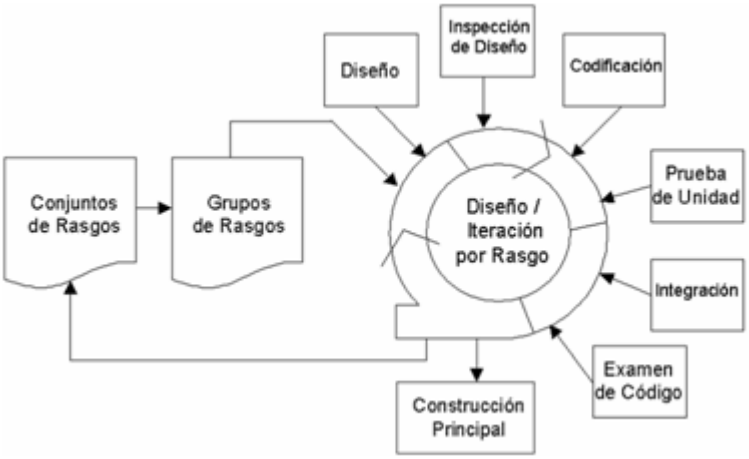

Figura 1.2- Ciclo de FDD

1.5.3.2- Los principios de FDD son pocos y simples:

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

1.5.3.3- Roles de FDD.

Hay tres categorías de rol en FDD: roles claves, roles de soporte y roles adicionales. Los seis roles claves de un proyecto son: (1) administrador del proyecto, quien tiene la última palabra en materia de visión, cronograma y asignación del personal; (2) arquitecto jefe (puede dividirse en arquitecto de dominio y arquitecto técnico); (3) manager de desarrollo, que puede combinarse con arquitecto jefe o manager de proyecto; (4) programador jefe, que participa en el análisis del requerimiento y selecciona rasgos del conjunto a desarrollar en la siguiente iteración; (5) propietarios de clases, que trabajan bajo la guía del

programador jefe en diseño, codificación, prueba y documentación, repartidos por rasgos y (6) experto de dominio, que puede ser un cliente, patrocinador, analista de negocios o una mezcla de todo eso.

Los cinco roles de soporte comprenden (1) administrador de entrega, que controla el progreso del proceso revisando los reportes del programador jefe y manteniendo reuniones breves con él; reporta al manager del proyecto; (2) abogado de lenguaje, que conoce a la perfección el lenguaje y la tecnología; (3) ingeniero de construcción, que se encarga del control de versiones de los builds y publica la documentación; (4) herramientista que construye herramientas o mantiene bases de datos y sitios Web y (5) administrador del sistema, que controla el ambiente de trabajo o productiza el sistema cuando se lo entrega.

Los tres roles adicionales son los de verificadores, encargados del despliegue y escritores técnicos. Un miembro de un equipo puede tener otros roles a cargo, y un solo rol puede ser compartido por varias personas. (Reynoso, 2004)

1.5.4- Programación Extrema (XP).

Algunos autores definen XP como:

“Un proceso ligero, de bajo riesgo, flexible, predecible, científico y divertido de desarrollar software”

“Una metodología ágil que requiere gran disciplina”. (Ferrer Zarzuela, 2001-2006)

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (Canós, y otros, 2004)

1.5.4.1- Ciclo de vida XP. Figura 1.3

El ciclo de vida ideal de XP consiste de seis fases: Exploración, Planificación de la Entrega (*Release*), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

El ciclo de vida es, naturalmente, iterativo. El siguiente diagrama describe su cuerpo principal:

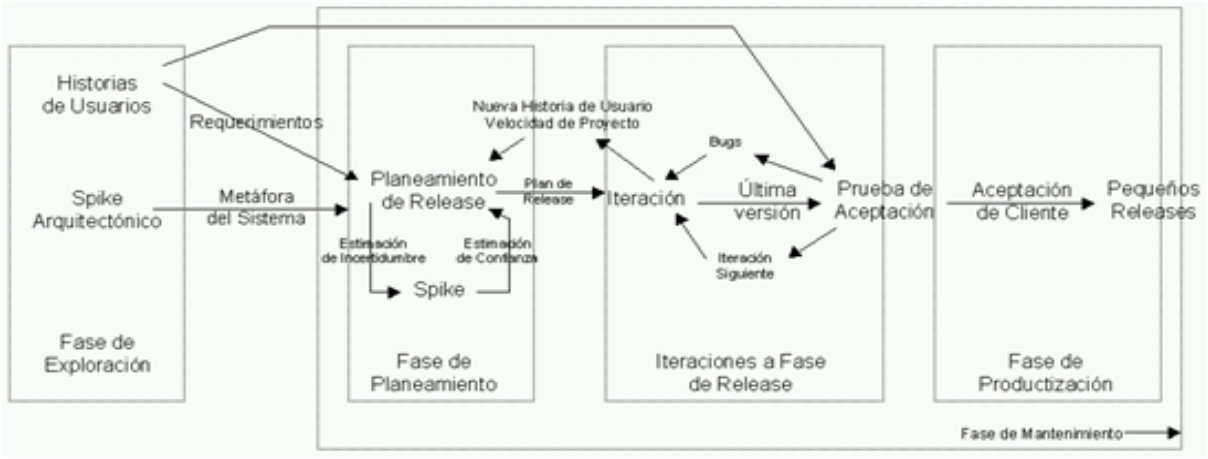

Figura 1.3 Ciclo de vida de XP (Palacio, 2005)

1.5.4.2- Valores XP.

Comunicación

XP pone en comunicación directa y continua a clientes y desarrolladores. El cliente se integra en el equipo para establecer prioridades y resolver dudas. De esta forma ve el avance día a día, y es posible ajustar la agenda y las funcionalidades de forma consecuente.

Feedback rápido y continuo

Una metodología basada en el desarrollo incremental de pequeñas partes, con entregas y pruebas frecuentes y continuas, proporciona un flujo de retro-información valioso para detectar los problemas o desviaciones. De esta forma fallos se localizan muy pronto. La planificación no puede evitar algunos

errores, que sólo se evidencian al desarrollar el sistema. La retro-información es la herramienta que permite reajustar la agenda y los planes.

Simplicidad

La simplicidad consiste en desarrollar sólo el sistema que realmente se necesita. Implica resolver en cada momento sólo las necesidades actuales. “Los costes y la complejidad de predecir el futuro son muy elevados, y la mejor forma de acertar es esperar al futuro.” Con este principio de simplicidad, junto con la comunicación y el feedback resulta más fácil conocer las necesidades reales.

Coraje

El coraje implica saber tomar decisiones difíciles. Reparar un error cuando se detecta. Mejorar el código siempre que tras el feedback y las sucesivas iteraciones se manifieste susceptible de mejora. Tratar rápidamente con el cliente los desajustes de agendas para decidir qué partes y cuándo se va a entregar. (Reynoso, 2004)

1.5.4.3- Prácticas XP.

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las siguientes prácticas.

1. **El juego de la planificación.** Hay una comunicación frecuente el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.
2. **Entregas pequeñas.** Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más 3 meses.

3. **Metáfora.** El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema (conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).
4. **Diseño simple.** Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.
5. **Pruebas.** La producción de código está dirigida por las pruebas unitarias. Éstas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.
6. **Refactorización** (Refactoring)[3]. Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.
7. **Programación en parejas.** Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores).
8. **Propiedad colectiva del código.** Cualquier programador puede cambiar cualquier parte del código en cualquier momento.
9. **Integración continua.** Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.
10. **40 horas por semana.** Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.

[3] El término refactoring, se refiere “al proceso de cambiar un sistema de software [orientado a objetos] de tal manera que no se altere el comportamiento exterior del código, pero se mejore su estructura interna”. existe un amplio catálogo de refactorizaciones más comunes: reemplazo de iteración por recursión; sustitución de un algoritmo por otro más claro; extracción de clase, interfase o método, etcétera.

11. **Cliente in-situ.** El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.
12. **Estándares de programación.** XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible. (Canós, y otros, 2004)

1.5.4.4- Roles XP.

Los roles que implementa la Programación Extrema son:

- **Programador.** El programador escribe las pruebas unitarias y produce el código del sistema.
- **Cliente.** Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.
- **Encargado de pruebas.** Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
- **Encargado de seguimiento.** Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.
- **Entrenador.** Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.
- **Consultor.** Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.
- **Gestor.** Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación. (Reynoso, 2004)

1.6- Comparaciones entre Metodología Ágiles.

Podríamos pensar que no es posible comparar los distintos métodos de desarrollo, y que todo depende de nuestros gustos personales, pero puesto que todos los procesos se centran en la producción de software (orientado a objetos mayormente) y que el proceso se implantará para aumentar la calidad del software producido y la eficiencia de los desarrolladores, es deseable una comparación.

1.6.1- Aspectos positivos y negativos de Scrum.

Positivos

- Indicada para proyectos con un rápido cambio de requisitos.
- Reuniones a lo largo del proyecto de coordinación e integración
- El desarrollo del software es un juego. En el ciclo: prejuego (se planea), en el juego se hace y entrega el proyecto, se hacen pruebas y en el pos-juego se integra todo, se hace prueba final.
- El desarrollo del software es mediante iteraciones (sprints, carreras rápidas en el desarrollo del software).

Negativos

- Método de desarrollo de ciclos largos.
- No hay manager, sólo Scrum Master resuelve problemas pero no manda, asegura que el proyecto se lleve a cabo. El papel de cierta dirección lo hace el **Management.** Está a cargo de las decisiones fundamentales y participa en la definición de los objetivos y requerimientos.

1.6.2- Aspectos positivos y negativos de Crystal Clear.

Positivo

- Centrada en las personas.
- El desarrollo se considera un juego cooperativo de invención y comunicación.

- El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas.
- Amplia comunicación en cuanto a dirección y prioridades
- Un encuentro semanal o semi-semanal con llamados telefónicos adicionales.

Negativo

- Todos los desarrolladores se encuentran juntos en el mismo local de trabajo.
- Ambiente de trabajo en el que la gente se ve obligada a hacer otras cosas incompatibles.
- Se planea en forma de juego lo que debe realizar cada desarrollador y la estimación del tiempo de desarrollo.
- No modela objetos.

1.6.3- Aspectos positivos y negativos de FDD.

Positivos

- Método de desarrollo de ciclos cortos.
- Ideal para proyectos de misión crítica (concentrados en dos fases diseño y construcción de software).
- El cliente no está en el lugar del desarrollo.
- Escrupuloso en inspección, tiene equipos pequeños que valoran varias alternativas para solucionar problemas y tomar decisiones.
- Modela objetos, descompone un problema mayor en sub-problemas, para una mejor solución.
- Es ligera, orientada al cliente y de iteraciones cortas y rápidas.
- La aseguración de la calidad en FDD no se basa en la documentación, si no en controles propios y una comunicación fluida con el cliente. El cliente recibe después de cada iteración un pedazo funcional del programa.
- En esta metodología se entrega bastante libertad a los desarrolladores, pero siempre bajo cierto orden marcado por una jerarquía (arquitecto, programador jefe, etc.), que representa también en nivel de responsabilidad existente en cada caso.

- Las revisiones de código fuente están guiadas por algún programador con más experiencia.
- Define métricas para el estado del proyecto, puesto que al dividirlos en unidades pequeñas es bastante sencillo hacer un seguimiento de las mismas.
- El equipo está distribuido jerárquicamente.

Negativos

- No usa procedimientos detallados de prueba.
- Reducen a segundo plano la implementación.
- Genera demasiada documentación.
- Se centra más en la organización global, y muchas de esas actividades, como ejecución de pruebas, las asumen como obligatorias aunque sin definirlas completamente, dejando libertad a las distintas subunidades del proyecto para implementarlas a su manera (por ejemplo usar la programación por parejas en partes complejas), aunque las directrices de la empresa suelen marcar el camino a seguir.
- Optan por la propiedad del código fuente.

1.6.4- Aspectos positivos y negativos de XP.

Positivos

- Usa procedimientos exhaustivos de pruebas.
- Es ligera, orientada al cliente y de iteraciones cortas y rápidas. No ponen demasiadas tareas organizativas sobre los desarrolladores (como modelado, generación de documentación externa, etc.), minimizan el efecto de esto con un representante del cliente.
- El cliente recibe después de cada iteración un pedazo funcional del programa.
- Si se requieren cambios se ajustará el plan de iteraciones y el de releases y tomará la nueva dirección en el desarrollo.
- Para proyectos cortos y equipos pequeños.
- Debido a que las releases se producen a menudo, se afirma la necesidad de que se puedan realizar pruebas automáticas.

- Está diseñado con los programadores en mente, con facilitar su trabajo y es por ello que define casi todo el proceso de desarrollo al completo, incluido el de pruebas e integración.
- La implementación de las pruebas antes que la propia funcionalidad hace que el desarrollador tenga que pensar pronto sobre lo que tiene que hacer y como probarlo correctamente.
- Se basa en un proceso iterativo. Esto permite acercarse poco a poco a la solución sin entrar demasiado rápido en detalles.
- En XP el desarrollo ve en qué dirección debe ir gracias al feedback del cliente, sin ningún tipo de restricción previa.
- Presenta la compartición del código fuente, pues está pensado para equipos pequeños.

Negativo

- El cliente está en el lugar del desarrollo.
- La aseguración de la calidad no se basa en la documentación, si no en controles propios y una comunicación fluida con el cliente.
- Programación en pares.
- La rotación de los miembros del equipo sobre los componentes y emparejamientos.
- La evaluación del estado del proyecto, los reporting recaen solamente en los jefes de proyecto.
- No requiere ninguna herramienta fuera del ambiente de programación y prueba.

1.7- ¿Por qué utilizar XP como metodología?

Por desgracia ninguno de estos procesos puede ser considerado perfecto, ni ser aplicado en su totalidad en la mayoría de los casos, por lo que también es necesario saber donde están sus puntos débiles para corregirlos, si es necesario.

XP es un proceso muy orientado a la implementación. Debido al bajo número de documentos a generar, se ofrece al desarrollador un escenario ideal para participar en el proyecto. Este proceso es aceptado con el mejor grado por desarrolladores menos experimentados ya que pueden sacar provecho directo de los compañeros más experimentados.

También el cliente está contento porque recibe un software que se adapta a sus deseos exactamente, mientras disponga de tiempo y dinero, pero ya que la funcionalidad exacta del software final nunca se definió formal y contractualmente (definirlo sería un contrasentido para XP, puesto que impediría el transcurso normal del proyecto guiado por el feedback del cliente), este método de desarrollo es quizás más aplicable para desarrollos internos, ya que para proyectos externos existe el problema concreto de que se debe realizar muy pronto una oferta concreta que defina que funcionalidad se va a implementar y en que período de tiempo, lo que también es importante para el cliente, ya que debe estar en la posición de comprobar tanto el rendimiento como la calidad y el contenido del software y estar seguro de recibirlo cuando lo espera.

Se debe, en todo caso, tener una gran confianza entre el cliente y el equipo de desarrollo (o su empresa) para usar el XP para escribir el software. Se podría considerar que la solución (teórica) a este problema es la presencia del representante del cliente junto al equipo de desarrollo, pero es también poco probable que el cliente pueda prescindir de los empleados que reúnen las características requeridas, puesto que suelen ser poco menos que imprescindibles, y la solución habitual de que un miembro del equipo tome el rol de representante del cliente no siempre puede ser acertada, ya que aunque la empresa desarrolladora tenga un experto en el dominio del problema, puede desconocer gran parte de las particularidades de la empresa del cliente, como políticas internas o particularidades remarcables.

La programación por parejas es un interesante punto a discutir de XP. Muchos consideran que quizás debería ser utilizado por cualquier otro método en determinadas ocasiones (algoritmos complejos, nuevos miembros del equipo con poca experiencia o malos hábitos, etc.), pero está por ver hasta qué punto el coste que supone es asumible por una empresa, y hasta qué punto se puede producir el ambiente que presupone XP (a fin de cuentas a nadie le gusta estar mirando mientras otro está delante de su ordenador). En todo caso, como ya se ha dicho en repetidas ocasiones, en muchos momentos del proceso de desarrollo, el trabajo en equipo solo puede ser beneficioso. Miembros del equipo con menos experiencia pueden aprender a partir de discusiones sobre una arquitectura más que cuando ya la reciben definida y solo la tienen que implementar y/o completar. Si se debe seguir este modo de trabajo en todas las fases del desarrollo del software depende en gran medida del problema concreto en cuestión y del propio equipo.

Lo que es *muy poco deseable* en XP es el hecho de *evitar* cualquier tipo de documentación fuera del código fuente (UML juega un papel prácticamente nulo, por ejemplo). Es asumible y aceptable que solo el propio código contenga la documentación actualizada, pero esto supone carencias que debemos tener en cuenta ya que puede no representar todo lo que debería (dependencias entre componentes, por ejemplo) y hace difícil utilizar la experiencia ganada en otros desarrollos (con otros equipos o por otras parejas), ya que no se ha anotado o archivado nada y se debe generar todo desde cero. (Molpeceres, 2003)

1.8- Conclusiones.

En este capítulo se han definido los conceptos fundamentales que se tienen en cuenta para la comprensión y realización de la estrategia metodológica. Se abordó el tema de las bases y antecedentes de las metodologías ágiles donde se puede determinar que a raíz de la creación del manifiesto ágil es que comienzan a llamarse “Metodologías Ágiles” puesto que anteriormente se estaba incursionando en ellas pero no se habían definido sus pautas y principios. Se analizaron las metodologías ágiles más utilizadas, conocidas y documentadas en el mercado mundial del software. A pesar de ser adaptativas y ligeras se hizo indispensable la comparación entre cada una de las metodologías ágiles, señalando sus aspectos positivos y negativos. Con el estudio de los fundamentos teóricos y para dar paso al capítulo dos se analizó a profundidad la aplicación del Método Ágil XP en el desarrollo de software de gestión a distancia, asumiendo posiciones en contra y a favor de la misma, convirtiéndose evidentemente la Metodología XP en la más acertada para en el proceso de gestión en la Universidad de las Ciencias Informáticas (UCI).

CAPÍTULO II

Estrategia metodológica para el desarrollo de Software de Gestión a Distancia aplicando la Programación Extrema.

2.1- Introducción.

En este capítulo se propone la Estrategia Metodológica para el desarrollo de software de gestión a distancia basado en la Metodología Ágil XP, con el fin de hacer el proceso de desarrollo de software de la Universidad de las Ciencias Informáticas (UCI) un poco menos complejo y más rápido. Para esto se valoraron y utilizaron las ventajas de la Metodología XP (Programación Extrema) y se adaptaron los principales problemas, dándole soluciones ágiles; propiciando mejores resultados, mayor eficiencia y menores costos.

Primeramente en el capítulo se identifica el objetivo estratégico general, luego una serie de elementos fundamentales, sistemas de habilidades y una serie de elementos que no se deben pasar por alto hasta llegar a cada una de las etapas con sus acciones fundamentales explicadas. La estrategia se divide en tres etapas fundamentales: creación de condiciones para la implementación, implementación de la estrategia y evaluación de la efectividad de la estrategia metodológica para metodología ágil XP (Programación Extrema) en el desarrollo de software.

2.2- Fundamentos teóricos generales de la estrategia metodológica.

La estrategia diseñada se inspira en los doce principios del manifiesto ágil. Estas son características que permiten diferenciar un proceso ágil de uno tradicional o pesado. En ellos se encuentran recogidas de

manera explícita todas las particularidades que hacen que un proceso sea ligero. Los dos primeros principios son generales y resumen gran parte del espíritu ágil. Todos los demás de encuentran enfocados al proceso y equipo de desarrollo, exaltando la organización y metas a seguir.

Un factor clave en la agilidad de los procesos de encuentra enmarcado en la satisfacción del cliente, se debe mantener contento al mismo mediante la entrega frecuente y temprana de partes funcionales y del producto en general. Se trabajará exhaustivamente en la total aceptación de los cambios que se puedan producir en el proceso.

Se debe enfocar no sólo en las pautas que marcan considerablemente un proceso ágil sino en el equipo de desarrollo, construir un entorno de trabajo favorable, donde cada individuo se sienta motivado y de lo mejor de sí para el proyecto. Se debe atender y dar preparación y superación a los miembros del equipo de desarrollo. La comunicación osmótica es imprescindible e indispensable entre los miembros.

En buena medida será necesario hacer énfasis en los procesos determinando promocionar un desarrollo sostenible. Mantener una adecuada atención a la calidad y al diseño mejora considerablemente la agilidad del proceso de desarrollo, tratando siempre de ser lo más simple posibles.

2.2.1- Características específicas del software de gestión.

El software de gestión debe adaptarse a una regulación jurídica y administrativa concreta. Esta necesidad involucra a las empresas desarrolladoras en el asesoramiento empresarial, actividades y conocimientos que no tienen nada que ver con la actividad tecnológica que está en la base de la creación de software. Esta necesidad de adecuación a las características locales es la razón por la que el software de gestión ha sido desarrollado tradicionalmente por empresas nacionales.

El software de gestión en su origen está fragmentado funcionalmente, según el área de la empresa a la que se dirige, siendo abundantes las aplicaciones estrictamente departamentales. Esta característica está evolucionando muy rápidamente hacia productos orientados a resolver de forma integrada los procesos de gestión.

En la misma medida que la empresa es una realidad cambiante, necesita que su software acompañe esta evolución. Por una parte el crecimiento de volumen de actividad supone en muchos casos la reforma de los procesos de gestión. También los cambios organizativos, la ampliación de actividades y nuevos productos. Por todos estos motivos el software de gestión está en permanente evolución y requiere de mantenimiento para adecuarse en cada momento a la realidad empresarial.

2.3- Objetivo Estratégico General.

Tiene como objetivo estratégico general desarrollar habilidades de la arquitectura de software en los miembros del proyecto para implementar la metodología ágil XP (Programación Extrema) de modo que pueda instrumentarse con la comunicación a distancia con el cliente y lograr el grado de eficiencia requerido en el producto informático.

2.3.1- Sistema de habilidades:

- ✓ **Comunicación-interactiva-emotiva:** Intercambio de información impactante, conmovedora y sensible que permite influir en el usuario para que acepte los valores que se le ofrecen en el producto de modo que su actuación se identifique con los propósitos del equipo de desarrollo del software.
- ✓ **Reflexión-intuitiva:** Meditación profunda de acercamiento a las necesidades proyectadas someramente por el cliente con la intención de lograr éxitos en cada una de las iteraciones del proceso de desarrollo del software.
- ✓ **Destreza motriz:** Conjunto de habilidades complejas del desarrollo del software (planificar-diseñar-programar) que se expresan en la conducta del desarrollador permitiéndole propulsar con el apoyo de los recursos informáticos las particularidades del producto de acuerdo con los intereses del cliente.

2.4- Definición de estrategia metodológica.

La definición de estrategia metodológica en el desarrollo de software por metodologías ágiles que se asume, toma en cuenta los criterios de M. A. Rodríguez del Castillo y A. Rodríguez Palacios y en consecuencia las autoras la definen como **la proyección de un sistema de acciones a corto, mediano**

y largo plazo que permite la transformación de la dirección del proceso de desarrollo de software tomando como base la Metodología Ágil XP (Programación Extrema) cuyos procedimientos permiten el logro de los objetivos determinados en un tiempo concreto.

2.4.1- Caracterización de la definición de estrategia metodológica.

La definición de estrategia metodológica en el desarrollo de software por metodologías ágiles se caracteriza por su:

- ✓ **Objetividad**: Porque toda la proyección estratégica está concebida a partir de los resultados del diagnóstico realizado.
- ✓ En su **desarrollo:** Demuestra que el cambio y la transformación conscientes, posibilitarán el surgimiento de cualidades superiores en los productos informáticos terminados en correspondencia con las necesidades y demandas del cliente.
- ✓ **Interdisciplinariedad:** Posibilita el desarrollo de la habilidades en todos los miembros del equipo de desarrolladores, independientemente de su rol, mediante la utilización de todas las herramientas, textos, códigos y estilos en que se trabaja la programación de software actualmente.
- ✓ **Trabajo cooperativo:** Porque tiene como premisa esencial el trabajo en equipo que parte de la unidad de criterio y de acción, para la producción de software.
- ✓ **Flexibilidad:** Porque se apoya en XP que como metodología ágil es flexible y adaptativa a los cambios.
- ✓ **Actualización:** La estrategia tiene en cuenta las principales concepciones sobre metodología ágiles, el manifiesto ágil y los adelantos de la Informática y las Comunicaciones.
- ✓ Las **exigencias** que deben tenerse en cuenta al aplicar la estrategia metodológica.
- ✓ **Preparación previa del desarrollador:** Es imprescindible que el desarrollador esté preparado con anterioridad para que la puesta en práctica del proyecto sea efectiva.
- ✓ **Disposición de todos los participantes:** El cumplimiento de esta exigencia es fundamental porque garantiza el éxito de las acciones que se desarrollarán como parte de la estrategia y en las que el desarrollador puede ser creativo en la utilización de medios, técnicas y procedimientos.

2.5- Etapas de la estrategia.

2.5.1- Primera Etapa: Creación de condiciones para la implementación.

2.5.1.1- Acción 1.- Precisar desarrollo de talleres.

Objetivo: Elaborar las precisiones metodológicas para la realización de los talleres de trabajo interactivo, con el propósito de contribuir al desarrollo de las habilidades propuestas.

Métodos: Revisión bibliográfica, análisis documental, modelación.

Medios: Fuentes bibliográficas, documentales, esquemas lógicos de contenido, informes, colecciones y muestra de software.

Responsable: Autoras de la tesis.

Participantes: Tutor, especialistas en Informática.

Formas de evaluación: Aplicación de diversas técnicas como: entrevistas y cuestionarios a directivos.

Plazos para la realización: Septiembre 2006.

Instrumentación: El propósito fundamental de estos talleres, es provocar la existencia de nuevas relaciones de los especialistas, a partir de situaciones problémicas que generen desarrollo de software.

2.5.1.2- Acción 2.- Fundamentación teórica y búsqueda de materiales complementarios.

Objetivo: Elaborar las precisiones metodológicas para la realización de los talleres de trabajo interactivo, con el propósito de contribuir al desarrollo de las habilidades propuestas.

Métodos: Revisión bibliográfica, análisis documental, modelación.

Medios: Fuentes bibliográficas, documentales, esquemas lógicos de contenido, informes, colecciones y muestra de software.

Responsable: Autoras de la tesis.

Participantes: Tutor, especialistas en Informática.

Formas de evaluación: Aplicación de diversas técnicas como: entrevistas y cuestionarios a directivos.

Plazos para la realización: Diciembre 2006.

Instrumentación: El propósito fundamental de estos talleres, es provocar la existencia de nuevas relaciones de los especialistas, a partir de situaciones problémicas que generen desarrollo de software.

2.5.1.3- Acción 3.- Reunión metodológica con el equipo de desarrollo.

Objetivo: Ejercitar entre los miembros del equipo sobre los elementos teórico-prácticos que intervienen en el desarrollo de software haciendo uso de los recursos de la comunicación a distancia.

Métodos: Debate, intercambio grupal y análisis científico.

Medios: Computadoras, software, enciclopedias, videos, documentales, láminas y textos seleccionados.

Responsable: Autoras de la tesis.

Participantes: Especialistas en Informática.

Formas de evaluación: Ver tabla 2.1

Tabla 2.1 *Evaluación mediante las dimensiones e indicadores convenidos.*

Dimensión	**Indicadores**
Comunicativa	Funcionalidad
Impacto del Software	Usabilidad Flexibilidad Confiabilidad

Plazos para la realización: Enero-Marzo 2007.

Instrumentación: El trabajo cooperativo es fundamental en el desarrollo del equipo. Promover el debate para posibilitar la participación activa de todos los desarrolladores.

2.5.1.4- Acción 4.- Determinación de factores de riesgo.

Durante el proceso de desarrollo de software en la Universidad de las Ciencias Informáticas (UCI) han sido identificados como factores de riesgo en la producción y/o calidad del producto, los siguientes factores que puede promover la existencia de riesgos y/o retrasos en la entrega del producto. Se toma como referencia algunos proyectos productivos de la Facultad 3, (SIGIA, ONE, Registro y Notaría)

Algunos factores de riesgo significativos en el desarrollo de un software. Ver Anexo 1.

1. No tener determinada la estrategia que se debe aplicar.
2. Insuficiente preparación de los desarrolladores del proyecto en guiones y otras fases del desarrollo del software.
3. Parálisis del proyecto debido a un excesivo tiempo recogiendo requisitos.
4. No disponer de instrumentos o mecanismo eficientes de control de cambio.
5. Trabajar con controladores de versiones obsoletos.
6. En unos casos manifestaciones sustanciales cuando el proyecto está muy avanzado y en otros temor a evitar cambios importantes durante el proyecto.
7. No realizar planificación de las iteraciones (traspasarse a un método supuestamente ágil).
8. Demasiada planificación de las iteraciones.

2.5.1.4.1- No tener determinada la estrategia que se debe aplicar.

En la realización del software a distancia se ha visto que existen varios problemas, hay equipos de desarrollo que no tienen definida ninguna metodología, otros intentan definir todo a priori guiados por metodologías pesadas y otros no hacen análisis de requisitos ninguno pensando que así aplica un método ágil. O lo que es peor, el mismo equipo va dando tumbos de un extremo a otro.

Lo primero que se debe definir en cualquier proceso de desarrollo de software es la Metodología que se va a utilizar. Muchas veces no se sabe que camino tomar y esta indecisión afecta, pues como Arquitectos

de Software, se debe tener la metodología definida, al menos un prototipo de guión que oriente al equipo que va a intervenir en el proyecto. Es a partir de estos antecedentes que se deben diseñar acciones estratégicas.

Siempre que se hable de proyecto, no importa la amplitud, se requiere apoyo en una metodología. Este es el punto de partida de cualquier proyecto, aunque muchas veces no sea la más adecuada y luego sea necesario el diseño de una metodología propia, que cumpla con el objetivo.

Todo desarrollo de software es riesgoso, tanto por la dispersión de los factores como por el costo del producto y difícil de controlar, pero si no se sigue una metodología de por medio, el resultado son clientes insatisfechos y desarrolladores aún más insatisfechos.

En el caso del desarrollo de software a distancia es una buena práctica aplicar algún método ágil, estos responden a los cambios más que seguir estrictamente un plan, un principio muy recomendado para la producción a distancia. Este es un tipo de desarrollo donde los cambios son muy frecuentes, en dependencia de lo que interprete o pueda asumir el programador de los documentos donde se encuentran contenidos los requisitos. A su vez estos documentos deben ser cortos y centrados en lo fundamental, la distancia no recomienda una cantidad excesiva de documentación. Dentro de los principios que proponen los métodos ágiles se destaca una importante pauta para crear software con calidad y con buen diseño son los equipos autodirigidos, equipos que se comuniquen, y reflexionen en torno a su comportamiento. Estos son aspectos que los procesos a distancia deben implementar para el éxito del proyecto. Además la prioridad número uno debe ser producir software de calidad para permitir posteriormente el uso eficiente del producto y satisfacer al cliente y esto se puede lograr mediante pequeñas entregas cada cierto tiempo, que le aporten un valor al cliente, lo que le permitirá al equipo de desarrollo aceptar los cambios, puesto que es más fácil variar poco a poco desde el principio que radicalmente al final del proyecto.

2.5.1.4.2- Insuficiente preparación de los desarrolladores del proyecto en guiones y otras fases del desarrollo del software.

Un factor muy significativo para desarrollar un software con la calidad requerida es que el equipo de desarrollo tenga un entendimiento común, es preciso construir el proyecto en torno a individuos

motivados. Los miembros del equipo deben familiarizarse con el ambiente de trabajo, las herramientas, tecnologías y prácticas que se utilizarán durante el progreso del proyecto. Darles el entorno y el apoyo que necesitan forjará la confianza para conseguir finalizar el trabajo. Muchas veces al iniciar un proceso de desarrollo de software no se tiene en cuenta la preparación que puedan tener los miembros del equipo y es aquí donde hay errores costosos.

Uno de los factores que propicia que la entrega de un software se retrase o alargue, es el enfrentamiento por parte del equipo de desarrollo y en ocasiones del mismo líder del proyecto con una herramienta o lenguaje de programación que le es difícil de entender o manejar por parte de todos los miembros del equipo; la sobrecarga de tareas a algunos miembros y otros con baja carga, la Ingeniería de software es **trabajo de equipo**. Para tratar el problema anteriormente explicado se adoptan dos posibles soluciones:

- Una de ellas es la parálisis temporal del desarrollo del proyecto para impartir cursos que satisfagan las necesidades de los desarrolladores (guiones-diseño-lenguajes de programación, etc).
- Las tareas más complejas y las que aún no han sido concluidas se le asignan a los desarrolladores más ágiles a tiempo, sin esperar que el proyecto corra demasiado para evitar atrasos.

Todas estas soluciones no son más que métodos o decisiones que toman los líderes de proyectos para salvar el proceso de desarrollo del software. Sin embargo la mayoría de las veces esto involucra un cambio significativo en el tiempo de desarrollo del proyecto, hay que variar las fechas de entrega o simplemente agilizar el tiempo en cada una de las iteraciones. En caso que se varíen las fechas de entregas que es lo más frecuente en este tipo de problemas, lo que se obtiene es clientes insatisfechos con el producto final, ya sea por falta de calidad o por retraso en la entrega.

Una experiencia recomendada de las Metodologías Ágiles es el **enfoque directo hacia el individuo**, este es el objetivo fundamental en cualquier labor. Las personas son la fuente de conocimiento y sin ellas no sería posible el desarrollo de acciones concretas para satisfacer las insuficiencias de los sistemas.

El tener un entendimiento claro de que se hará en el proyecto es de vital importancia. Una vez concebida todas las herramientas, prácticas y un buen entendimiento dentro del ambiente de trabajo, a partir de aquí entonces se puede trabajar sin pensar en la posibilidad de interrumpir el proceso por un error humano.

Una buena preparación en el tema permite trabajar cómodamente, saber **qué se realizará y cómo**. Tener base acelera el proceso productivo, se puede mostrar un producto bien definido y con muchas más perspectivas, mantiene al cliente contento, cada iteración muestra un mayor conocimiento del tema. Un análisis y diseño certeros dan la posibilidad de ampliar las opciones del software, definir una arquitectura sólida, minimizar la probabilidad de riesgos, etc.

Para mantener a los desarrolladores con arrojo y dando lo mejor de sí para el trabajo en la entidad productora se hace necesario estimular a los destacados, tanto moral como espiritualmente. Los jefes de proyectos deben ser capaces; conscientes de que el producto requiere de mucho uso del intelecto y no debe pedirle más de lo que pueden dar los especialistas y nunca sobrecargarle el horario laboral. El exceso irracional de horas de trabajo agota significativamente al trabajador y esto afecta la calidad del software.

2.5.1.4.3- Parálisis del proyecto debido a un excesivo tiempo recogiendo requisitos.

La mayoría de los proyectos de ingeniería del software tienen requisitos incompletos o inconsistentes. Unas veces se debe a la poca experiencia de los clientes para especificar requisitos. Otras a que no saben bien qué es lo que necesitan y se limitan a decir "**Lo sabré cuando lo vea**". Incluso los clientes que conocen qué es exactamente lo que quieren encuentran difícil expresarlo en forma de requisitos. Además de estas dificultades suele ser normal que los clientes esperen más de lo que han esperado, o que los documentos de requisitos describan aplicaciones sin una solución práctica informatizable. En ocasiones se encuentran proyectos que pasan mucho tiempo haciendo la captura de requisitos o que simplemente no la hacen por desconocimiento, porque no lo consideran importante, o por cualquier otro motivo.

Una tendencia generalizada, es el excesivo tiempo haciendo la captura de requisitos, esto es un proceso engorroso, pesado y que demanda una gran documentación. Muchas veces los clientes no saben definir las necesidades del negocio, se demoran en la elaboración de las entrevistas, dejan funcionalidades por definir y en ocasiones sus descripciones son cortas y esquemáticas. La realización de un proyecto grande tardaría un tiempo relativo de 4 a 6 meses en la captura de requisitos, una documentación exhaustiva, donde los desarrolladores perderían un valioso tiempo leyendo tratando de entender lo que el usuario desea.

Se ha demostrado teóricamente la posibilidad de capturar y extraer la información conocimiento en un documento sin distorsionar el contenido, esto puede no ser una manera práctica o eficiente de comunicar los requisitos para los sistemas de software complejos. En la determinación del proceso para desarrollar software es importante recordar que el propósito primario de un proyecto es entregar un sistema de software que genere valor ya sea para un negocio o para fines educativos. Los modelos y los documentos son esencialmente subproductos del desarrollo de un software; que no generan valor directamente, pero favorecen su calidad.

Los programadores que tienen la experiencia de trabajar con requisitos tradicionales describen que los documentos son escritos expresados en lenguaje seco, formal, que describen un sistema de software deseado. El documento de los requisitos actúa con dos propósitos: comunicación y almacenaje de datos (que preservan la evidencia de la petición que debe organizar el guionista), lo demás su estética, calidad, su funcionabilidad, lo pone el resto del equipo.

Hay que colocarse en el lugar del guionista como miembro del equipo de desarrollo del software y en el lugar del resto (diseñadores, especialistas en medios audiovisuales y programadores) de modo que se limiten los defectos de los documentos como medio de comunicación. Los documentos son selectivos, encierran los objetivos, son unidireccionales; pero pueden también ser ambiguos.

Los documentos son un medio de comunicación unidireccional; flujos de información del autor al lector. En la modalidad **a distancia** hay pocas oportunidades para que el lector pida preguntas, ideas de la oferta y penetraciones. Un programador que trabaje con este tipo de documento puede encontrar partes difíciles de entender. Esto puede ser debido a la opción pobre de palabras por el autor o la carencia del conocimiento de base del lector. Dado este caso el programador tendrá que asumir muchas de las cosas descritas. Si el trabajo está bajo presión de tiempo, un programador puede hacer su propia conclusión sobre el significado previsto para evitar el retraso. Asumir respuestas puede conducir a desarrollar el sistema de software incorrecto, se debe tratar que en el guionista facilite al resto del equipo lo más claro posible las particularidades de lo que se va a realizar, haciendo un interpretación clara y precisa de lo que se ha pedido.

En los últimos años, el movimiento ágil del desarrollo del software ha creado un cambio del paradigma en cómo trabajamos para entender requisitos del sistema. Los equipos ágiles forman sistemas de software usando un proceso de colaboración. Esto crea un cambio fundamental lejos de las herramientas para manejar los artefactos de los requisitos.

Para desarrollar software de gestión a distancia es necesario implementar la captura de requisitos mediante métodos ágiles, donde el cliente defina bien los objetivos y en caso de dudas este se pueda consultar mediante vías corporativas, e-mails, chats, teléfonos, etc. La autodeterminación por parte del programador de algunos aspectos significativos dentro de la descripción de los requerimientos puede conseguir que el proyecto se retrase o que el diseño del software no sea el esperado por el cliente.

Cuando el trabajo de equipo se basa en el desarrollo del análisis y del software paralelamente, éste crea una oportunidad de trabajar con requisitos de diversas maneras. Específicamente, se puede considerar otros canales de comunicaciones, abrir diálogo. Los medios primarios de los equipos ágiles de comunicar requisitos son conversación. Este medio crea la posibilidad del flujo de información entre todos los partidos. Cuando los equipos de desarrollo son ágiles permite el intercambio de información fluidamente, los equipos hacen la planificación del desarrollo. La conversación fluye alrededor del grupo, el equipo entero se encuentra involucrado en esta conversación. Discuten panoramas y opciones del proceso para explorar los requisitos antes de que se planee el desarrollo. Mediante la conversación se explora una comprensión compartida y se tiene la oportunidad de ofrecer ideas.

2.5.1.4.4- No disponer de instrumentos o mecanismo eficientes de control de cambio.

Los seres humanos cometen errores diariamente ya sea en la vida privada como en el entorno de estudio o trabajo. Uno de los errores típicos que a menudo se comete es cuando a un proyecto en su fase inicial (planificación) es que no se tienen bien definidos los objetivos a tener en cuenta para un desarrollo exitoso, en este caso se refiere al guión del software.

Un mecanismo de control de cambio es un sistema que posibilita la gestión de versiones (revisiones) de todos los elementos de configuración que integran la línea base de un producto o una configuración del mismo. La utilización de este tipo de mecanismo es de gran ayuda para cualquier proyecto pues

proporciona la administración de todos los cambios que se realicen, así como las posibles determinaciones realizadas para algún cliente específico. El mecanismo de control de cambios se adiciona en la práctica al guión del software de modo que todo el equipo vaya participando y tenga conocimiento del proceso de desarrollo del software, ya sea con fines comerciales o con intereses sociales, educativos.

Existen proyectos productivos que aún no están bien claros de la necesidad del control de cambios. En la Universidad (UCI) algunos proyectos que realizan la gestión de cambios mediante reuniones a partir de comité o cuando se hace la entrega de los instaladores se emplean técnicas que (herramienta de control de cambios) pueden inducir un control de cambios, pero son muy viejas y obsoletas, demandan un alto nivel de documentación, exigen mucho tiempo en reuniones y revisiones. Utilizar un mecanismo para controlar todo lo que se haga en el proyecto es elemental puesto que algunos integrantes dependen de lo que otros desarrollen para culminar sus tareas. Por ello todo elemento de control de cambios, que se adiciona a la planificación inicial, proyecto inicial o guión, debe divulgarse a todo el equipo, es decir los informes de estado.

El control de versiones se realiza principalmente para controlar las distintas versiones del código fuente. Sin embargo, esto también puede aplicarse para documentos, imágenes, etc. Suele ser muy útil la generación de informes de estado con los cambios introducidos entre dos versiones.

Un sistema de control de versiones debe proporcionar:

- Mecanismo de almacenaje de cada uno de los ítems que deba gestionarse (archivos de texto, imágenes, documentación).
- Posibilidad de modificar, mover, borrar cada uno de los elementos.
- Histórico de las acciones realizadas con cada elemento pudiendo volver a un estado anterior dentro de ese historial.

Se pueden utilizar diversos medios y herramientas para el control de cambios pero se piensa que la más óptima para implementar en los proyectos productivos de la Universidad es el SVN. Esta herramienta es utilizada por algunos de los proyectos y se han obtenido buenos resultados.

Subversión es un software de sistema de control de versiones. Es software libre una característica importante de subversión es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.

Trabajar con SVN agiliza y mejora el tratamiento de cambios. Esta es una herramienta muy potente las modificaciones (incluyendo cambios a varios archivos) son atómicas, sigue la historia de los archivos y directorios a través de copias y renombrados. Puede utilizar opciones en dependencia del tipo de servidor al que se encuentre integrado. Permite enviar solamente las diferencias en ambas direcciones (cliente-servidor) aunque posee problemas con el manejo de cambio de nombres de archivos, este no es completo.

2.5.1.4.5- En unos casos manifestaciones sustanciales cuando el proyecto está muy avanzado y en otros temor a evitar cambios importantes durante el proyecto.

Muchas veces se realiza el diseño de software de manera rígida, con los requerimientos que el cliente solicitó (el guionista como miembro del equipo debe ajustar, pero no lo hizo) de tal manera que cuando el cliente en la etapa final (etapa de prueba), solicita un cambio se hace muy difícil realizarlo, pues al hacerlo se alteran muchas cosas que se habían previsto, y es justo éste, uno de los factores que ocasiona un atraso en el proyecto y por tanto la incomodidad del desarrollador por no cumplir con el cambio solicitado y el malestar por parte del cliente por no tomar en cuenta su pedido.

La mayoría de los clientes no reciben la versión operativa del software hasta que el mismo se encuentre muy avanzado o en fase de terminación. Un error importante puede ser desastroso, si se descubre al final del proceso. Se requiere dar al cliente tantas versiones operativas como sea posible, para evitar pérdidas de tiempo y de recursos.

Para evitar el retraso de un proyecto cuando se produce repentinamente un cambio se debe utilizar una guía de la metodología que oriente el trabajo por caminos ágiles, llegando a un acuerdo formal con el cliente, al inicio del proyecto, en cada fase importante del proyecto, de tal manera que cada cambio o modificación no perjudique al desarrollo del software; es decir haciendo entregas frecuentes al cliente de

versiones del proyecto con funcionalidades implementadas, que aporten valor al negocio, a la entidad de servicios educacionales u otras según los intereses. Así tendrá el cliente la posibilidad de solicitar cambios y agregar los aspectos que considere importante o que halla omitido en la fase de inicio. Sobre esta base se crea una arquitectura sólida que responda a las necesidades de los clientes y usuarios finales.

2.5.1.4.6- No realizar planificación de las iteraciones (tras pasarse a un método supuestamente ágil).

Las iteraciones se deben planificar en el tiempo preciso, no exceder la cantidad de tiempo pues se estaría atentando contra el éxito del proyecto. Sin embargo, las iteraciones cortas no son siempre la mejor manera de ir. Una longitud corta de la iteración para un tipo dado de trabajo dependerá del horizonte de la previsibilidad para ese trabajo.

Las **iteraciones** son un **período fijo de tiempo** en el que se programan tareas. Una iteración es generalmente un período fijo de entre dos y seis semanas. Las iteraciones generalmente son numeradas consecutivamente y siguen una a otra de manera continua.

Las iteraciones son siempre puntos de control, aprendizaje y planificación. Con independencia de la metodología utilizada, lo que todas las metodologías populares tienen en común, es que al principio de una iteración se planifica el trabajo que se va a abordar a corto plazo y al final de una iteración se sacan conclusiones de la iteración reciente terminada. Este proceso de mirar atrás se debe hacer siempre con el afán de que sirva para ir hacia adelante. El final de una iteración siempre coincide con el comienzo de otra, de manera que en ese punto lo que se ve al mirar atrás son los resultados de una iteración y estos resultados son los que nos deben guiar a la hora de planificar la siguiente iteración.

Evidentemente, el proceso descrito, se puede aplicar de manera anidada. Una iteración puede contener a otras. Esto es así porque al principio de un proyecto es habitual no tener claro en un alto nivel de detalle cuales serán la iteraciones, y por lo tanto se podrá definir 3 o 4 iteraciones que luego se irá refinando en sub-iteraciones.

A veces se tiende a definir iteraciones en función de hitos en lugar de en función de un horizonte temporal. Esto es una mala práctica. Cuando se define erróneamente una iteración y demoramos mucho tiempo, lo

que sucede es, que se acumula retraso. Es claro que parece conveniente realizar periódicamente actividades explicitas de control, aprendizaje y planificación detalla. A la hora de establecer iteraciones es conveniente establecer una meta.

En la planificación de una iteración se debe:

1. Decidir cuanto durará.
2. Establecer que trabajos se realizarán.
3. Estimar en detalle estos trabajos.
4. Asignar quien realizará estos.
5. Establecer una meta para la iteración.

Todo ello sin perder de vista los que logramos en la iteración anterior.

2.5.1.4.7- Otros problemas que se han detectado.

Muchas personas han llegado a la conclusión de que son necesarias las aplicaciones de gestión de software libre. En la historia reciente del software libre se han producido intentos para la creación de aplicaciones de gestión, principalmente para empresas, que no se han acabado materializando en programas. Por ello, se ha considerado interesante analizar las posibles causas que han motivado algunos de estos fracasos y las conclusiones de este análisis son las siguientes:

- El programador se debe encargar del ciclo completo del desarrollo.
- El software aparece como resultado de una serie de casualidades, por lo que no siempre se desarrolla lo que hace falta.
- Problemas de comunicación y mantenimiento en los proyectos grandes.
- Cuando hay problemas de continuidad, se trabaja a tiempo parcial y no existe una documentación adecuada, el proyecto puede desaparecer.
- Los programadores tienen pocos alicientes económicos.
- Cuando se logran aplicaciones suelen aparecer problemas de compatibilidad o de eficacia por carencias en los análisis.

- El modelo evolutivo es poco útil para las aplicaciones de gestión en las que se requieren resultados óptimos desde el principio.
- Faltan herramientas y librerías especialmente diseñadas para la creación de aplicaciones de gestión y para la migración de datos.

2.5.1.5- Acción 5.- Preparación de los desarrolladores.

Elementos a tener en cuenta en la preparación de los desarrolladores:

- **Entendimiento común**.
- **Motivación hacia el proyecto** en los miembros del equipo.
- **Familiarización con el ambiente de trabajo**, las herramientas, tecnologías y prácticas.
- La Ingeniería de Software es **tarea de equipo**, con nivelación de la carga de trabajo.
- **Impartición de cursos** según las necesidades (guiones-diseño-incorporación de multimedia-lenguajes de programación, etc).
- Las tareas más complejas y las que aún no han sido concluidas se le asignan a los desarrolladores más ágiles a tiempo, sin esperar que el proyecto corra demasiado para evitar atrasos.
- Las personas son la fuente de conocimiento (las Metodologías Ágiles deben tener un **enfoque directo hacia el individuo).**
- **Análisis y diseño certeros** amplían las opciones del software, definen una arquitectura sólida, minimizan la probabilidad de riesgos, etc.
- **Jefes de proyectos deben ser capaces** y orientar a tiempo a los desarrolladores.

2.5.2- Segunda Etapa: Implementación de la estrategia.

2.5.2.1- Acción 1.-Desarrollo de talleres a distancia.

- **Taller 1.-** Determinación de elementos esenciales a tener en cuenta en la contratación del proyecto con el cliente. Ver Anexo 2.

  Cuando el cliente hace la solicitud de contrato, debe referirse a dos puntos fundamentalmente: al tipo de producto que desea, sus características, funcionalidades y a los parámetros o cláusulas

que se manifestarán en el contrato. Posteriormente a esto el Jefe de Producción de la Facultad se reúne y valora de acuerdo a las características del proyecto cual puede ser el Líder o Manager, definiendo así cuales son los elementos esenciales que se deben tener en cuenta para la contratación con el cliente. Estos elementos no son más que el tiempo de desarrollo del proyecto, las herramientas a utilizar, las fases, la metodología a seguir, la entrega, entre otros.

- **Taller 2.-** Valoración de teoría y práctica sobre arquitectura de software a distancia. Ver Anexo 3.

  En este taller se tienen en cuenta para desarrollar software a distancia una serie de pasos que son importantes para el desarrollo del mismo. Se valoran todo lo que tiene que ver con la teoría del desarrollo de software a distancia, que es lo que se hace en cada uno de los pasos a seguir y cual es el orden de los mismos para que luego no se forme confusión. Todos estos puntos de gran importancia se aclaran bien para que una vez que estos se lleven a la práctica tenga el éxito que se espera del mismo. Aquí se preparan a los desarrolladores, se les da pie al debate, al conocimiento de todo lo necesario para el desarrollo del software, beneficiándose así la entidad pues disminuyen sus costos. Se evidencian y analizan temas como arquitectura de software y sus estilos, lenguajes de descripción, modelos de proceso y diseño, herramientas, prácticas de diseño sobre arquitecturas orientadas a servicios y modelos de integración.

  En este espacio para la reflexión y el debate, se debe hacer alusión a las características que poseen el software de gestión en su conjunto. Estas son necesidades de este tipo de software que no se pueden omitir, todos deben tener conocimiento de ellas. Algunas se mencionan a continuación:

  - Sistema de Gestión de Base de Datos (cliente / servidor).
  - Multiusuario, multiempresa.
  - Trazabilidad entre los diferentes módulos.
  - Base de datos documental: permite asociar cualquier tipo de documento Word, Excel, PDF, a los clientes con el ahorro de tiempo en búsqueda de información.
  - Importación / Exportación de datos a cualquier sistema externo.

- **Taller 3.-** Diagnóstico del estado actual en que se encuentra la teoría sobre metodologías ágiles. Ventajas y desventajas de XP (Programación Extrema) frente a otras metodologías ágiles.

  Aquí se desarrolla un diagnóstico general de toda la bibliografía consultada para conocer el estado actual en que se encuentran las principales metodologías ágiles, el desarrollo de cada una de ellas y como pueden estas ser aplicadas en dependencia del tipo de proyecto. Se comparan las ventajas y desventajas de XP con cada una de las otras metodologías para así llegar a la conclusión de cual es la más acertada para el trabajo a distancia. XP es la metodología que se propone aunque tenga algunos inconvenientes que no sean del todo favorables en su aplicación, pero esto sería secundario pues posee la característica de ser adaptativa.

- **Taller 4.-** Sobre desarrollo de software y el papel del equipo de desarrolladores: condiciones previas, recursos disponibles, exigencias en el producto, periodicidad en los informes de estado, períodos de prueba, etapa de ajuste de cambios y fecha de entrega del producto. Ver Anexo 4.

  En este taller abordará las condiciones previas que deben existir para el trabajo, confort y comodidad de cada uno de los desarrolladores, así como su capacidad para desarrollar tareas en el proyecto, no solo las asignadas por su rol sino cualquier otra labor de la que requiera ayuda. Además será un aspecto relevante en este encuentro los recursos con que cuenta el equipo de desarrollo para la realización del producto pues se sabrá entonces hasta donde es que el cliente puede exigir en su producto y hasta donde el líder del proyecto podrá aceptar esas exigencias. Un tema muy importante a tocar aquí sería la entrega constantemente del estado en que se encuentra el producto para saber si por el camino en que se va es el correcto y las pruebas que serían buenas que se hicieran desde el inicio para así conocer cada una de las cosas que no se están haciendo como es debido, corregirlo a tiempo y no esperar a la etapa final en que se tengan que hacer ajustes en las fechas de la entrega total y el cliente quede insatisfecho en la puntualidad de su producto.

- **Taller 5.-** Recursos de publicidad para la generalización del producto.

  Los recursos necesarios para generar una adecuada comprensión y comercialización del producto están guiados particularmente por una eficaz comunicación, una forma efectiva de hacer llegar al

usuario cuales son las funcionalidades del software y como debe utilizarlo en su beneficio propio, esto enmarca el aprendizaje de los usuarios. Podemos llamar la atención de varias y disímiles formas las más utilizadas son: la creación de páginas o sitios Web de promoción del producto, artículos comerciales, promociones y una de las más efectivas, la comunicación directa, por medio de las mismas personas y lo que sean capaces de asumir y transmitir.

2.5.2.2- Acción 2.- Desarrollo de talleres de forma presencial.

- **Taller 6.-** Análisis a nivel de equipo de la problemática existente y factores de riesgo para el desarrollo del software.

  Durante el desarrollo de este taller se hace un análisis minuciosamente de la problemática existente y de los factores de riesgo. Aquí se llega a la conclusión de darle respuesta a la problemática y también se realiza un plan de mitigación para así poder contrarrestar todos aquellos elementos que puedan afectar la entrega a tiempo del proyecto. Todas estas acciones las lleva a cabo el jefe del proyecto pero con la ayuda de todos los integrantes del equipo de desarrollo.

- **Taller 7.-** Valoración de los requerimientos para el producto contratado.

  Una vez hecha la captura de los requerimientos conjuntamente con el cliente todos los miembros del equipo se reúnen para llegar a un acuerdo y a partir de las premisas que el cliente desea, llegar a un consenso de cuales son los principales requisitos que se deben implementar, para así comenzar a evaluarlos y ver que si estos requisitos dependen de otros, y también implementarlos aunque los mismos no tengan una prioridad de orden crítica.

2.5.2.3- Acción 3.- Desarrollo del software de acuerdo con el rol de cada miembro del equipo y monitoreo por parte del manager.

Una vez concebido todo lo que en el proyecto se tiene que hacer el jefe del mismo se reúne con todos los estudiantes seleccionados para conformarlo y le asigna a cada uno de ellos la tarea que deben realizar de acuerdo al rol de cada uno y a sus capacidades. En concordancia con el taller 3 que es donde se da a

conocer la metodología a usar se le informa a cada uno cual es la que se va a utilizar, que en este caso sería la Programación Extrema.

Después de haber conocido todos estos aspectos entonces el líder del proyecto se reúne con el equipo de desarrollo para ver que falta, conocimientos no alcanzados para desarrollar exitosamente el proyecto y terminarlo en tiempo y con la calidad requerida. Luego de esta reunión el mismo se pone en función de comenzar a impartirles asesoría individual (capacitación) a aquellos miembros que aún no se encuentren bien preparados para asumir su tarea.

Otra tarea importante que se hace cuando se emprende un proyecto y que es de suma importancia, es la planificación de cada iteración y dentro de cada una de ellas las actividades, las cuales no tienen el mismo tiempo de desarrollo porque algunas son un poco más complejas que otras. Esto es un aspecto muy importante para la entrega final ya que sino se planifica luego el tiempo no va a alcanzar para terminar el proyecto con puntualidad.

2.5.2.4- Acción 4.- Integración de los paquetes o módulos para la concepción final del producto.

Una vez concebido el producto, y desarrollado cada paquete o módulo del producto, se hace la integración de las partes funcionales y se logra la obtención de un beneficio. Se prepara el software para la fase de pruebas finales.

2.5.2.5- Acción 5.- Ejercitación interactiva de pruebas y ajustes para comprobar en el software su nivel de funcionabilidad, usabilidad, flexibilidad y confiabilidad. Ver Anexo 5.

Con la terminación del software se le hacen al mismo una serie de pruebas pertinentes para comprobar que todo esté correctamente bien terminado, es decir que no suceda ningún error sintáctico o semántico. Esto lleva una serie de pasos a parte de las pruebas que se le vienen haciendo con el transcurso del proyecto en cada fase. Cada vez que se pinche o se de clic en cualquier botón o link el producto debe mostrar lo que se desee para que el mismo tenga la funcionalidad esperada, el software debe tener una buena usabilidad para todas aquellas personas que lo vayan a usar teniendo en cuenta que el mismo puede ser usado por un universitario o por otro que no tenga un nivel escolar elevado. El mismo debe

contar con la confiabilidad requerida, es decir que cada uno de los que accedan a la aplicación sea por su usuario o por el rol al que pertenece, no que todos puedan acceder a todos los links de la aplicación que aparezcan si los mismos no están autorizado.

El desarrollo guiado por pruebas es una de las principales prácticas de Programación Extrema (XP), que propone una serie de pasos para probar antes de programar.

El proceso a realizarse es el siguiente:

- Se crea un caso de prueba que verifica una pequeña funcionalidad del sistema
- Se ejecuta el caso de prueba y deberá tener un resultado no exitoso, ya que la funcionalidad que intenta probar aún no está concebida.
- Una vez que se observa el fallo se desarrolla únicamente el código que hará que la prueba sea exitosa.
- Se hace, un refactoring del código para asegurar que se tiene el diseño más simple para la funcionalidad que acaba de agregarse.

Una vez que estos pasos son llevados a cabo, se realiza lo que motiva a la metodología: la ejecución de todas las pruebas automatizadas que se tienen construidas hasta el momento.

2.5.3- Tercera Etapa: Evaluación de la efectividad de la estrategia metodológica para metodología ágil XP (Programación Extrema) en el desarrollo de software.

2.5.3.1- Acción 1.- Evaluación del trabajo realizado por cada desarrollador en las distintas fases del proceso de desarrollo del software mediante: Exploración-Planificación de la entrega-Iteraciones-Producción-Mantenimiento-Muerte del proyecto.

En esta acción se evalúa a cada uno de los miembros del equipo por las actividades que ellos realizaron en cada fase como son: la construcción de artefactos, la documentación del negocio y del sistema, la implementación de cada módulo o paquete, entre otras. Para evaluar a cada uno de los integrantes del proyecto se verifica la planificación, si esta se cumplió a cabalidad por parte de ellos y si la realizaron con la calidad requerida, esta es una actividad muy importante puesto que se tiene que llevar un control

estricto en el tiempo de desarrollo de cada actividad que componen las diferentes fases, ya que si una de estas actividades se retrasa entonces la fase en la que se está desarrollando no termina en el tiempo planificado y así sucesivamente todas las que le siguen, esto conlleva a un atraso en la entrega del proyecto final y a su vez clientes insatisfechos con el trabajo. *Ver Anexo 6.*

Fase I: Exploración.

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.

Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

Fase II: Planificación de la Entrega.

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días.

Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la “velocidad” de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración.

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el

número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

Fase III: Iteraciones.

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción.

Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores.

Fase IV: Producción.

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento).

Fase V: Mantenimiento.

Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

Fase VI: Muerte del Proyecto.

Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

2.5.3.2- Acción 2.- Reunión de análisis para evaluar el desarrollo de software. Ver Anexo 7.

A medida que se valla desarrollando el proyecto es necesario hacer informes de estado para evaluar la situación real en la que se encuentra el proceso. Estos informes contendrán de forma sencilla y directa la evaluación del proyecto a partir de una serie de pasos. Además en la misma se dejará plasmado todos los problemas resultantes de este proceso de evaluación, y se procesarán dándoles la mejor solución posible, marcando sugerencias y alternativas.

2.5.3.3- Acción 3.- Validación de la aplicación de la estrategia por Criterio de Especialistas. Ver Anexo 8.

En la valoración de la estrategia se aplicó el Criterio de Especialistas. Para este tipo de validación se aplicó un instrumento de medición, con dimensiones e indicadores, tales como: Comunicación, impacto del software y funcionalidad, usabilidad, flexibilidad, confiabilidad. Las primeras dos expresarían las dimensiones y el resto los indicadores.

La validación es una parte importante de todo proyecto, aquí se definen esquemas y tablas que representan la visión de las tesistas y el tutor, los cuales serían más válidos si esa visión particular es respaldada por un grupo de expertos o jueces. La cantidad de especialistas consultados debe ser elevada

y un número impar, pues así se determinará la validez y calidad de nuestra estrategia, cuánto aporta y que deficiencias puede tener.

Para realizar esta consulta con los especialistas se dispuso de una sala en donde los especialistas trabajaron a juicio personal de forma individual e independiente y luego se efectuó una reunión de consenso con las autoras de la tesis, para efectuar el trabajo de puesta en común de sus opiniones.

2.6- Conclusión.

La Estrategia Metodológica, da la medida de la cantidad de etapas y acciones que se deben tener en cuenta para la realización e implementación del software a distancia. Potenciamos la utilización de la metodología ágil XP (Programación Extrema) para optimizar el tiempo de desarrollo, aumentar la calidad y disminuir los costos.

Finalmente se logró elaborar la estrategia metodológica a partir de la metodología ágil XP para desarrollar software a distancia, y se especificó en cada una de las etapas dividas a su vez en acciones las actividades sobre las cuales se debe encausar el desarrollo del software y las solución a los problemas de manera ágil.

CAPÍTULO III

Evaluación de estrategia metodológica para la aplicación de la Metodología Ágil XP.

3.1- Introducción.

Según la estrategia metodológica planteada en el capítulo anterior, en el presente se propone una evaluación a partir de los especialistas en el tema para saber cual es el impacto de la misma en la producción de software a distancia, con el fin de darle solución a los problemas encontrados para un desarrollo exitoso. Para esto se consultaron una gran cantidad de especialistas con un alto nivel de conocimiento y unos cuantos años de experiencia en la rama de la informática, los cuales se pusieron de acuerdo para evaluarla a partir de una serie de indicadores y dimensiones y así obtener un cálculo aproximado del impacto de la misma.

3.2- Planificación de la Estrategia.

Para la evaluación de la calidad de la estrategia metodológica por el Criterio de Especialistas, en que mediante la valoración profesional se tuvo consenso de opiniones sobre la propuesta, incluyendo dimensiones e indicadores para la medición de impacto de software. Los pasos seguidos en su utilización fueron:

- Determinación de objetivos.
- Elaboración de la Estrategia.
- Selección de los especialistas.
- Determinación de la metodología ágil a utilizar de acuerdo con criterios recogidos sobre las potencialidades de los distintos tipos: SCRUM, CC, FDD y XP.

- Determinación de las dimensiones e indicadores de medición de impacto de software a considerar dentro de la estrategia de aplicación a partir de la Metodología Ágil XP (Programación Extrema).
- Elaboración de instrumentos (cuestionarios) y pilotaje.
- Evaluación integral de la Estrategia.
- Procesamiento estadístico de la información y análisis de los resultados.
- Informe de resultados.

3.3- Resultados de la Evaluación de la Estrategia.

En el pilotaje: Se tomaron 20 estudiantes y 2 profesores de los distintos proyectos (SIGIA, ONE y RN) y la información arrojó lo siguiente:

- ✓ La Estrategia Metodológica propuesta a partir de la Metodología Ágil XP (Programación Extrema) es superior para los objetivos de desarrollo de Software en la Universidad de las Ciencias Informáticas (UCI) por ser ligera, requiere menos documentación, utiliza cómo método secuencia de informes de estado al cliente, proceso enfocado al desarrollador, entre otros aspectos (19 para un 86%).
- ✓ Se tomó como base el criterio de 15 (68%) para la modificación del objetivo, dándole un carácter estratégico.
- ✓ Se incorporó un indicador más a la dimensión dos: Confiabilidad (14, 63%).

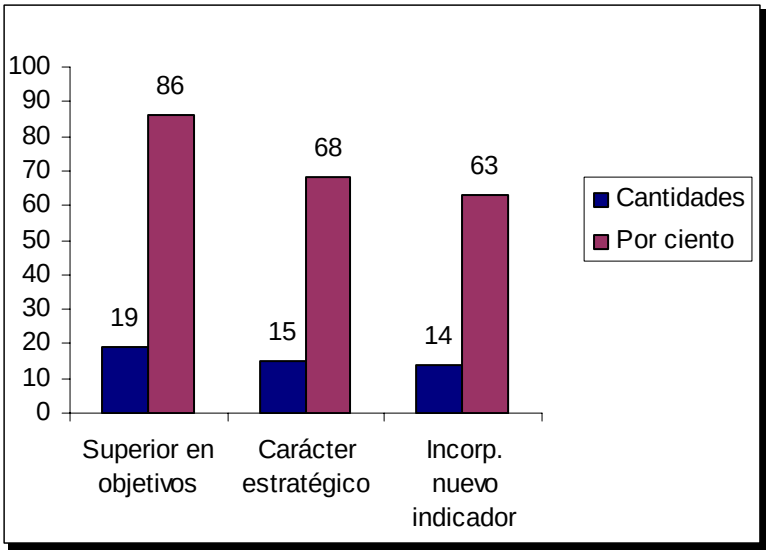

Figura 3.1 Evaluación Estratégica.

3.4- Evaluación por Criterio de Especialistas. Ver Anexo 8.

- **Selección de especialistas:** Para la selección de los especialistas se tomaron en cuenta profesores de Informática y otras disciplinas del Instituto Politécnico de Informática (IPI) "Rafael Morales González; profesores de la Universidad de Ciego de Ávila (UNICA) ; Instituto Superior Pedagógico (ISP) "Manuel Ascunce Domenech"; directivos de Informática de la Dirección Municipal y Provincial de Ciego de Ávila, de los cuales se seleccionaron 25 especialistas entre ellos: 10 doctores en ciencias; 8 másters; 4 auxiliares y 3 profesores asistentes, con un promedio de experiencia docente en el campo de la Informática de 7 años. De ellos 21 trabajan y viven en la provincia, 4 trabajan en Ciego de Ávila, pero residen en las provincias de Camagüey y Sancti Spíritus.
- **Preparación de los desarrolladores y criterios sobre dimensiones e indicadores de medición de impacto de software:**

Los criterios de los especialistas se resumen en las siguientes ideas:

- ✓ El 52% (13) de los especialistas los consideró muy adecuados**;** el 32% (8) adecuados; el 8% (2) poco adecuados y el 8% (2) no dijeron opinión al respecto. Ver figura 3.2.

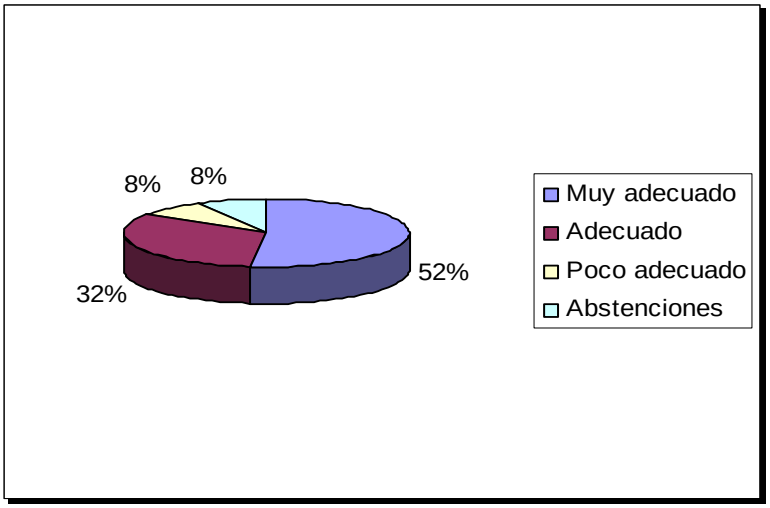

Figura 3.2 Consideraciones de Especialistas

- ✓ El 32% (8) considera que se deben ajustar los criterios a la hora de medir los rangos en que se expresan los indicadores propuestos.
- ✓ El 72% de los especialistas (18) tienen la opinión que los talleres deben tener una mayor descripción en la estrategia, por presentarse en la modalidad a distancia.
- ✓ El 84% (21) de los encuestados consideran factible brindar la preparación de los desarrolladores a distancia, aunque luego para desarrollo de software se integren; lo que es positivo por aligerar los costos del producto. Ver figura 3.3.

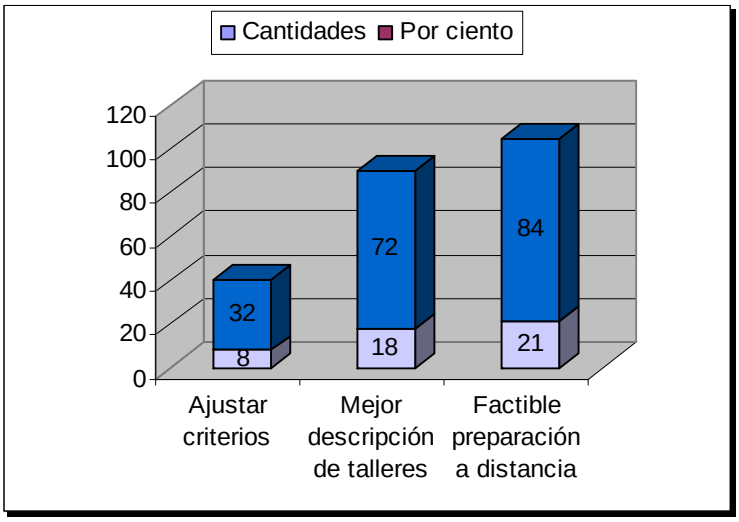

Figura 3.3 Valoraciones de Especialistas.

3.4.1- Evaluación integral de la Estrategia: Criterio de Especialistas.

- ✓ La estrategia metodológica para el uso de la Metodología Ágil XP (Programación Extrema) se considera muy adecuada por los especialistas consultados (21, para el 84%).
- ✓ Los 25 especialistas (100%) ratificaron con su aprobación los ajustes hechos de acuerdo con el pilotaje.
- ✓ La evaluación otorgada por los especialistas a cada elemento de la propuesta permiten valorarla como factible de aplicar para el uso de la Metodología Ágil XP (Programación Extrema), de acuerdo con los valores promedios otorgados en todos los casos son menores que el valor del primer punto de corte como se muestra en la tabla que se presenta a continuación. Ver Tabla 3.1.

Tabla 3.1 *Evaluación integral de la estrategia (valores otorgados en una escala de 0 a 10)*

	VALOR PROMEDIO OTORGADO	**MAYOR VALOR**	**MENOR VALOR**
A) Objetivo integral y sistema de habilidades de la estrategia.	Frecuencia 25 Valor promedio 8.20	10	4
B) Estructura de la estrategia.	Frecuencia 25 Valor promedio 6.92	10	4
C) Dimensiones e indicadores de medición de impacto de software.	Frecuencia 25 Valor promedio 9.20	10	6

Referencia: De **7 a 10 puntos** se considera Adecuado, de **4 a 6 puntos** Poco Adecuado y menos de **4 puntos** No Adecuado.

Por tanto no hubo necesidad de ir a otra ronda de evaluación por los especialistas debido a que ninguno de los ítems fue evaluado de **poco o no adecuado** en su promedio. Lo anterior significa que la categoría evaluativo correspondiente a cada aspecto, es la de adecuada, por lo que se considera un resultado altamente significativo. Se observa que las calificaciones han estado por encima de 6 en los promedios.

Tabla 3.2 *Evaluación integral de la estrategia (valores otorgados en una escala de 0 a 10)*

	A	B	C
1	8	6	10
2	10	4	8
3	8	6	10
4	7	5	8
5	6	6	10
6	10	10	10
7	10	8	8
8	9	7	8
9	8	8	8
10	5	6	6
11	10	10	10
12	8	8	8
13	6	4	8
14	10	6	10
15	8	8	8
16	10	6	7
17	4	6	6
18	8	8	10
19	8	6	10
20	10	10	10
21	10	8	10
22	7	5	5
23	8	8	8
24	7	6	6
25	10	8	8

Referencias:

A.- Objetivo integral y sistema de habilidades de la estrategia.

B.- Estructura de la estrategia.

C.- Dimensiones e indicadores de medición de impacto de software.

Los datos reflejan que los especialistas otorgaron calificaciones de mayor calidad en el objetivo-sistema de habilidades y dimensiones e indicadores, que en estructura de la estrategia, donde el 48% (12) consideraron que requería modificaciones en algunos de sus elementos, principalmente en la descripción de los talleres.

Tabla 3.3 *Determinación de dimensiones e indicadores de medición de impacto de software por parte de los especialistas.*

Dim.	Comunicativa	Otras	Impacto del Software			
Ind.	Funcionabilidad		Usabilidad	Flexibilidad	Confiabilidad	Otros
1	A		A	PA	PA	X
2	A		A	A	A	
3	A		A	A	A	
4	NA	X	NA	NA	A	X
5	NA	X	NA	NA	A	X
6	A		A	A	A	
7	A		A	PA	A	X
8	A		A	PA	A	X
9	A		A	PA	A	X
10	PA		A	NA	A	X
11	A		A	A	A	
12	A		A	A	A	
13	NA	X	A	NA	A	X
14	A		A	A	A	
15	A		A	A	A	
16	A		A	PA	A	X
17	PA	X	A	NA	A	X
18	A		A	A	A	
19	A		A	A	A	
20	A		A	A	A	
21	A		A	A	A	
22	NA		A	NA	PA	X
23	A		A	A	A	
24	A		A	PA	A	X
25	A		A	A	A	

Los datos reflejan que el indicador más cuestionado es flexibilidad 12 especialistas (48%) lo consideran no adecuado o poco adecuado para medir impacto de software y en su defecto proponen **(incidencia en el conocimiento o aprendizaje de algo 8 (32%); interés por el objeto que divulga el software 10 (40%), los datos más coincidentes).**

3.5- Resultados de aplicación de otros instrumentos:

- ✓ Guía de observación a las actividades de los talleres a distancia para la preparación de desarrolladores: Con el objetivo de evaluar las necesidades de los miembros de equipos de desarrollo; asimilación de contenidos sobre factores de riesgo en el Proyecto Registro y Notaría (Software Registro y Notaría); dominio de herramienta Java, C#, etc.
- ✓ Entrevista individual para profundizar, mediante el criterio de los profesores y estudiantes de la Universidad de las Ciencias Informáticas (UCI) sobre metodologías ágiles. Es significativo que 42 el 62% de los consultados de la muestra consideran que la Metodología Ágil XP es la más idónea para el desarrollo de software, sin embargo los directivos orientan el uso de RUP, que es más pesada.

3.6- Conclusiones.

Este capitulo demuestra que el uso de las Metodologías Ágiles es mucho más efectiva, en el caso que nos ocupa XP (Programación Extrema). La mayor parte de los Especialistas seleccionados estuvo de acuerdo con la Estrategia propuesta. Se aprueba la efectividad y aplicación de la misma. Así damos cumplimiento a la última tarea de la investigación, terminando así exitosamente nuestro trabajo.

CONCLUSIONES

La realización de este Trabajo de Diploma ha aportado importantes conocimientos a las autoras, ofreciendo una guía de cómo se debe desarrollar un software a distancia aplicándole una metodología ágil.

A partir de dicha realización se llegó a las siguientes conclusiones:

- ✓ Para darle respuesta al objetivo general se estudiaron las últimas tendencias sobre todo el desarrollo de software a distancia y las metodologías ágiles identificando así las cuatro principales para este tipo de software entre las consultadas.
- ✓ Con el objetivo de realizar una estrategia metodológica se identificaron las etapas y dentro de ellas las acciones a seguir a partir de los fundamentos teóricos generales de la estrategia y los objetivos generales de la misma.
- ✓ Llegando por último a la validación de los especialistas para comprobar el impacto y la utilidad de la estrategia en la Universidad de las Ciencias Informáticas (UCI) para el desarrollo de software a distancia utilizando una metodología ágil.

✓ En último lugar la investigación ha interesado, pues a partir de ella se han consolidado ideas que se tenían en el aire logrando entre los desarrolladores de software e ingenieros informáticos que realizan software la unidad y creación de nuevos criterios.

RECOMENDACIONES

- ✓ Continuar con la investigación para mejorar el desarrollo de software a distancia mediante la metodología ágil.
- ✓ Realizar una Estrategia para el desarrollo de software a distancia entre los integrantes del equipo de desarrollo.
- ✓ Implementación de la estrategia en el software a distancia que se desarrollan en la Universidad de las Ciencias Informáticas (UCI).

BIBLIOGRAFIA CITADA

- ✓ **Cortés, Angel. 2003.** PROFit GESTIÓN INFORMÁTICA . [Online] 2003. http://banners.noticiasdot.com/termometro/boletines/docs/ti/profit/2003/Outsourcing_previsiones_2003.pdf.
- ✓ **Canós, Jose H., Letelier, Patricio and Penadés, Mª Carmen. 2004.** Métodologías Ágiles en el Desarrollo de Software. [Online] 2004. http://www.willydev.net/descargas/prev/TodoAgil.Pdf.
- ✓ **Ferrer Zarzuela, Jorge. 2001-2006.** Metodologías Ágiles. [Online] 2001-2006. http://libresoft.urjc.es/downloads/ferrer-20030312.pdf.
- ✓ **Fowler, Martin. 2003.** La nueva metodologia . [Online] marzo/abril 2003. http://www.programacionextrema.org/articulos/newMethodology.es.html#tth_sEc5.
- ✓ **Molpeceres, Alberto. 2003.** Procesos de desarrollo: RUP, XP y FDD. [Online] Febrero 15, 2003. http://www.willydev.net/descargas/articulos/general/cualxpfddrup.PDF.
- ✓ **Palacio, Juan. 2005.** Gestion y porcesos en empresas de software. [Online] Noviembre 2005. http://www.navegapolis.net/files/articulos/gestion_y_procesos.pdf.
- ✓ **Reynoso, Carlos. 2004.** Metodologías de Desarrollo de Software Ágiles. [Online] 2004. http://www.sel.unsl.edu.ar/ApuntesMaes/2004/Metodologias%20Agiles.doc.
- ✓ **—. 2004.** Métodos Heterodoxos en Desarrollo de Software. [Online] 2004. http://www.willydev.net/descargas/Heterodoxia.pdf.

BIBLIOGRAFIA CONSULTADA

- ✓ **Arboleda, Hugo F. 2005.** Modelos de ciclo de vida en desarrollo de software. [Online] Octubre 5, 2005. http://www.acis.org.co/index.php?id=551.
- ✓ **Connell, S. 1997.** Desarrollo y Gestión de Proyectos Informáticos. s.l. : McGraw-Hill Iberoamericana, 1997.
- ✓ **Canós, Jose H., Letelier, Patricio and Penadés, Mª Carmen. 2004.** Métodologías Ágiles en el Desarrollo de Software. [Online] 2004. http://www.willydev.net/descargas/prev/TodoAgil.Pdf.
- ✓ **Cortés, Angel. 2003.** PROFit GESTIÓN INFORMÁTICA . [Online] 2003. http://banners.noticiasdot.com/termometro/boletines/docs/ti/profit/2003/Outsourcing_previsiones_2003.pdf.
- ✓ **Ferrer Zarzuela, Jorge. 2001-2006.** Metodologías Ágiles. [Online] 2001-2006. http://libresoft.urjc.es/downloads/ferrer-20030312.pdf.
- ✓ **Fowler, Martin. 2003.** La nueva metodologia . [Online] marzo/abril 2003. http://www.programacionextrema.org/articulos/newMethodology.es.html#tth_sEc5.
- ✓ **Gómez-Senet Martínez, E. 1992.** Las fases del proyecto y su Metodología. Valencia : s.n., 1992.
- ✓ **Herranz, Paloma and Sánchez, Esperanza. 2005.** Nuevas estrategias de gestion documental. [Online] 2005. http://www.soluziona.es/htdocs/global/de_interes/revistas/articulos_voice2/articulo5.pdf.
- ✓ **Molpeceres, Alberto. 2003.** Procesos de desarrollo: RUP, XP y FDD. [Online] Febrero 15, 2003. http://www.willydev.net/descargas/articulos/general/cualxpfddrup.PDF.
- ✓ **Morato, Garcia. 2007.** Software de Gestión. [Online] 2007. http://www.cic.es/VerPagina.asp?IDPage=46&MenuPinchado=139.
- ✓ **Palacio, Juan. 2005.** Gestion y porcesos en empresas de software. [Online] Noviembre 2005. http://www.navegapolis.net/files/articulos/gestion_y_procesos.pdf.
- ✓ **Pavón, J.M. 2004.** Ingeniería del Software de Gestión 1. Gestión del Riesgo. Madrid : s.n., 2004.
- ✓ **Reifer, Donald J. 2002.** Administrador. [Online] Julio/Agosto de 2002. http://www.salta.softwarelibre.org.ar/programacion/agil/Manager.pdf
- ✓ **Reynoso, Carlos. 2004.** Metodologías de Desarrollo de Software Ágiles. [Online] 2004. http://www.sel.unsl.edu.ar/ApuntesMaes/2004/Metodologias%20Agiles.doc.

- ✓ **—. 2004.** Métodos Heterodoxos en Desarrollo de Software . [Online] 2004. http://www.willydev.net/descargas/Heterodoxia.pdf.
- ✓ **Ruiz, F. 2004.** Palnificación y Gestión de Sistemas de Información. Gestión de Riesgos en Proyectos Informáticos. 2004.
- ✓ **Santamaría Uriarte, Juan. 2005.** El Balanced Scorecard (BSC): Distintas perspectivas, un mismo rumbo. [Online] Enero 7, 2005. http://www.profit.es/41artdet.asp?NoticiaID=113.
- ✓ **Straub, Pablo. 2004-2005.** Desarrollo de Software Orientado a los Negocios con Métodos Ágiles. [Online] 2004-2005. http://www.agileshift.cl/Tutorial/DesarrolloAgilParte4.pdf....DesarrolloAgilParte4.pdf
- ✓ **Valdarrama, Satiago L. 2005.** Planificacion Extrema en pocos minutos. [Online] Octubre 12, 2005. http://svpino.blogspot.com/2005/10/programacin-extrema-en-pocos-minutos.html.