Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Académico Profesional de Ingeniería de Sistemas

Metodologías de desarrollo de software orientadas al plan y ágiles (RUP-XP)

TESINA

Para optar el Título Profesional de Ingeniero de Sistemas

AUTOR

Susana Doris RAMOS GONZALES

ASESOR

Roberto Francisco CALMET AGNELLI

Lima, Perú

2006

Referencia bibliográfica

Ramos, S. (2006). *Metodologías de desarrollo de software orientadas al plan y ágiles (RUP-XP).* Tesina para optar el título de Ingeniero de Sistemas. Escuela Académico Profesional de Ingeniería de Sistemas, Facultad de Ingeniería de Sistemas e Informática, Universidad Nacional Mayor de San Marcos, Lima, Perú.

Dedicado a mi amado esposo y

a mi pequeñita Alejandra.

RESUMEN

METODOLOGÍAS DE DESARROLLO DE SOFTWARE ORIENTADAS AL PLAN Y AGILES (RUP - XP)

SUSANA DORIS RAMOS GONZALES

Junio – 2006

Asesor : Mag. Roberto Francisco Calmet Agnelli

Grado : Magíster

A lo largo de los años se ha demostrado que los proyectos exitosos son aquellos que son administrados siguiendo una serie de procesos que permiten organizar y luego controlar al proyecto. Por otro lado, en los últimos años, ha surgido una corriente en la industria del software que considera que las necesidades de los clientes son muy cambiantes, y que si no nos adaptamos rápidamente, corremos el riesgo de estar resolviendo el problema equivocado. Estas dos visiones parecen opuestas, pero las metodologías que reflejan a

esas visiones pueden considerarse como herramientas diseñadas para ser usadas en contextos específicos. En este trabajo se describen los dos exponentes mas representativos de las Metodologías Orientadas al Plan y las Metodologías Ágiles (RUP- XP), pretendiéndose otorgar un marco conceptual y practico para poder seleccionar la metodología mas adecuada para el desarrollo de software.

Palabras Claves :

Ingeniería de Software

Modelo de Proceso de Software

Metodologías

Metodología Orientada al Plan

Metodología Ágil

ABSTRACT

METHODOLOGIES OF DEVELOPMENT OF SOFTWARE ORIENTATED TO THE PLAN AND AGILE (RUP – XP)

SUSANA DORIS RAMOS GONZALES

Junio – 2006

Asesor : Mag. Roberto Francisco Calmet Agnelli

Grado : Magíster

Throughout the years there has been demonstrated that the successful projects are those that are administered following(continuing) a series of processes that they allow to organize and then to control to the project. On the other hand, in the last years, a current has arisen in the industry of the software that he(she) thinks that the needs of the clients are very changeable, and that if we do not

adapt rapidly, we traverse the risk of being solving the wrong problem. These two visions seem to be opposite, but the methodologies that they reflect to these visions can be considered to be tools designed to be used in specific contexts. In this work both exponents are described more representative of the Methodologies Orientated to the Plan and the Agile Methodologies (RUP - XP), a conceptual frame being tried to grant and I practice to be able to select the methodology more adapted for the development of software.

Keywords :

Engineering software

Process model of Software

Methodologies

Methodology Orientated to the Plan

Methodology Agile

INDICE

INTRODUCCIÓN

La información se ha convertido en el activo principal de las empresas, representando en la mayoría de los casos su principal ventaja estratégica. Es por ello por lo que el desarrollo de sistemas de información se ve sometido actualmente a grandes exigencias en cuanto a productividad y calidad, y se hace necesaria la aplicación de un nuevo enfoque en la producción del software, más cercano a una disciplina de ingeniería que a los hábitos y modos artesanales que, desafortunadamente, se han venido aplicando en más de una ocasión.

OBJETIVOS

OBJETIVO GENERAL

El objetivo de la presente tesina es dar algunos alcances para la correcta selección de algunas de las dos metodologías aquí presentadas para su adecuada aplicación en el proceso de desarrollo de software.

OBJETIVOS ESPECIFICOS

- Entender conceptos asociados al desarrollo de software.
- Conocer y aplicar los conceptos y fases de las metodologías de desarrollo de software (RUP – Rational Unified Process y/o XP – Extreme Programming), para el desarrollo de un sistema.
- Capacitar al lector para evaluar la metodología mas adecuada según las características de un proyecto.

CAPITULO I

1. GENERALIDADES

Se ha visto conveniente exponer algunos conceptos básicos relacionados con la metodologías de desarrollo de software, que serán utilizados en el presente trabajo, de igual forma se explica el proceso de desarrollo de software para entender mejor el papel que juega la metodología de desarrollo de software en este proceso.

1.1 DEFINICION DE CONCEPTOS

- **Ingeniería de Software**

  Es la rama de la ingeniera que crea y mantiene las aplicaciones de software aplicando tecnologías y practicas de las ciencias computacionales, manejo de proyectos, ingeniería, el ámbito de la aplicación, y otros campos.

- **Modelo de Proceso de Desarrollo Software**

  Un Modelo de Proceso de Desarrollo de Software es una representación simplificada de un proceso de software que conlleva una estrategia global para abordar el desarrollo de software.

- **Notación**

  Aplicado en el contexto de un proceso de desarrollo de software, es un lenguaje establecido como estándar para visualizar, especificar, construir y documentar un proceso de ingeniería de software.

- **Herramienta Case**

  Herramientas individuales para ayudar al desarrollador de software o administrador de proyecto durante una o más fases del desarrollo de software (o mantenimiento)[9].

  Una combinación de herramientas de software y metodologías de desarrollo" [5] .

- **Metodología de Desarrollo de Software**

  Conjunto de procedimientos, técnicas, herramientas y soporte documental que deben seguirse para el desarrollo de software. En un proyecto de desarrollo de software la metodología define Quién debe hacer Qué, Cuándo y Cómo debe hacerlo.

- **Ciclo de Vida de Software**

  Conjunto de fases por las que pasa el sistema que se está desarrollando desde que nace la idea hasta que el software se retira o se reemplaza.

1.2 PROCESO DE DESARROLLO DE SOFTWARE

La evolución de la disciplina de ingeniería de software ha traído consigo propuestas diferentes para mejorar los resultados del proceso de construcción. De manera paralela, el tema de modelos para el mejoramiento de los procesos de desarrollo ocupa un lugar importante

en la búsqueda de la metodología adecuada para producir software de calidad en cualquier contexto de desarrollo.

1.2.1 Introducción

Un sistema de información es un conjunto de elementos que interactúan entre sí con el fin de apoyar las actividades de una empresa o negocio, entre estos componentes los más importantes son: el hardware, necesario para que el sistema de información pueda operar, el recurso humano que interactúa con el Sistema de Información(el cual está formado por las personas que utilizan el sistema), el sistema de información en sí (una aplicación software, el cual esta diseñado en base a los procesos de la empresa). Con respecto al software, su construcción y resultados han sido siempre cuestionados debido a los problemas asociados, entre ellos podemos destacar los siguientes :

- Los sistemas no responden a las expectativas de los usuarios.
- Los programas “fallan” con cierta frecuencia.
- Los costos del software referidos en la etapas de elaboración e implementación son difíciles de prever y normalmente superan las estimaciones.
- La modificación del software es una tarea difícil y costosa.
- El software se suele presentar fuera del plazo establecido y con menos prestaciones de las consideradas inicialmente.
- El aprovechamiento óptimo de los recursos (personas, tiempo, dinero, herramientas, etc.) no suele cumplirse.

Según el Centro Experimental de Ingeniería de Software (CEIS), el estudio de mercado The Chaos Report realizado por Standish Group

Internactional en 1996, concluyó que sólo un 16% de los proyectos de software son exitosos (terminan dentro de plazos y costos y cumplen los requerimientos acordados). Otro 53% sobrepasa costos y plazos y cumple parcialmente los requerimientos. El resto ni siquiera llega al término.

Algunas deficiencias comunes en el desarrollo de software son:

- Escasa o tardía validación con el cliente.
- Inadecuada gestión de los requisitos.
- No existe medición del proceso ni registro de datos históricos.
- Estimaciones imprevistas de plazos y costos.
- Excesiva e irracional presión en los plazos.
- Escaso o deficiente control en el progreso del proceso de desarrollo.
- No se hace gestión de riesgos formalmente.
- No se realiza un proceso formal de pruebas.
- No se realizan revisiones técnicas formales e inspecciones de código.

El primer reconocimiento público de la existencia de problemas en la producción de software tuvo lugar en la conferencia organizada en 1968 por la Comisión de Ciencias de la OTAN en Garmisch (Alemania), dicha situación problemática se denominó crisis del software. En esta conferencia, así como en la siguiente realizada en Roma en 1969, se estipuló el interés hacia los aspectos técnicos y administrativos en el desarrollo y mantenimiento de productos software. Se pretendía acordar las bases para una ingeniería de construcción de software. Según Fritz Bauer lo que se necesitaba era “establecer y usar principios de ingeniería orientados a obtener software de manera económica, que sea

fiable y funcione eficientemente sobre máquinas reales". Esta definición marcaba posibles cuestiones tales como: ¿Cuáles son los principios robustos de la ingeniería aplicables al desarrollo de software? ¿Cómo construimos el software económicamente para que sea fiable? ¿Qué se necesita para crear programas que funcionen eficientemente no en una máquina sino en diferentes máquinas reales?. Sin embargo, dicho planteamiento además debía incluir otros aspectos, tales como: mejora de la calidad del software, satisfacción del cliente, mediciones y métricas, etc.

El "IEEE Standard Glossary of Software Engineering Terminology" (Stad. 610.12-1990) ha desarrollado una definición más completa para ingeniería del software [1]: "La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento del software; es decir, la aplicación de ingeniería al software. El estudio de enfoques en ". Pressman [1] caracteriza la Ingeniería de Software como "una tecnología multicapa", ilustrada en la Figura 1.

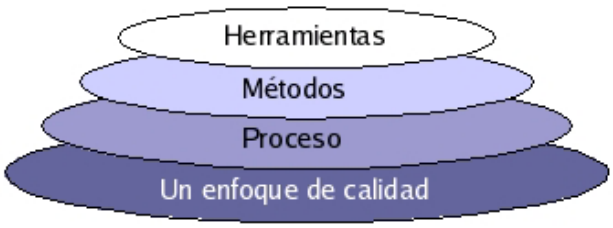

Figura 1: Capas de la Ingeniería de Software.

Dichas capas se describen a continuación:

- Cualquier disciplina de ingeniería (incluida la ingeniería del software) debe descansar sobre un esfuerzo de organización de **calidad**. La gestión total de la calidad y las filosofías similares fomentan una cultura

continua de mejoras de procesos que conduce al desarrollo de enfoques cada vez más robustos para la ingeniería del software.

- El fundamento de la ingeniería de software es la capa proceso. El proceso define un marco de trabajo para un conjunto de áreas clave, las cuales forman la base del control de gestión de proyectos de software y establecen el contexto en el cual: se aplican los métodos técnicos, se producen resultados de trabajo, se establecen hitos, se asegura la calidad y el cambio se gestiona adecuadamente.
- Los métodos de la ingeniería de software indican cómo construir técnicamente el software. Los métodos abarcan una gran gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento. Estos métodos dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.
- Las herramientas de la ingeniería del software proporcionan un soporte automático o semiautomático para el proceso y los métodos, a estas herramientas se les llama herramientas CASE (Computer-Aided Software Engineering).

Dado lo anterior, el objetivo de la ingeniería de software es lograr productos de software de calidad (tanto en su forma final como durante su elaboración), mediante un proceso apoyado por métodos y herramientas.

A continuación nos enfocaremos en el proceso necesario para elaborar un producto de software.

1.2.2 El proceso de desarrollo de software

Un proceso de desarrollo de software tiene como propósito la producción eficaz y eficiente de un producto software que reúna los requisitos del cliente. Dicho proceso, en términos globales se muestra en la Figura 2 . Este proceso es intensamente intelectual, afectado por la creatividad y juicio de las personas involucradas. Aunque un proyecto de desarrollo de software es equiparable en muchos aspectos a cualquier otro proyecto de ingeniería, en el desarrollo de software hay una serie de desafíos adicionales, relativos esencialmente a la naturaleza del producto obtenido. A continuación se explican algunas particularidades asociadas al desarrollo de software y que influyen en su proceso de construcción.

Un producto software en sí es complejo, es prácticamente inviable conseguir un 100% de confiabilidad de un programa por pequeño que sea. Existe una inmensa combinación de factores que impiden una verificación exhaustiva de las todas posibles situaciones de ejecución que se puedan presentar (entradas, valores de variables, datos almacenados, software del sistema, otras aplicaciones que intervienen, el hardware sobre el cual se ejecuta, etc.).

Un producto software es intangible y por lo general muy abstracto, esto dificulta la definición del producto y todos sus requisitos, sobre todo cuando no se tiene precedentes en productos software similares. Esto hace que todos los requisitos sean difíciles de consolidar tempranamente. Así, los cambios o nuevos requisitos son inevitables, no sólo después de entregado el producto sino también durante el proceso de desarrollo.

Además, de las dos anteriores, siempre puede señalarse la inmadurez de la ingeniería del software como disciplina, justificada por su corta vida comparada con otras disciplinas de la ingeniería.

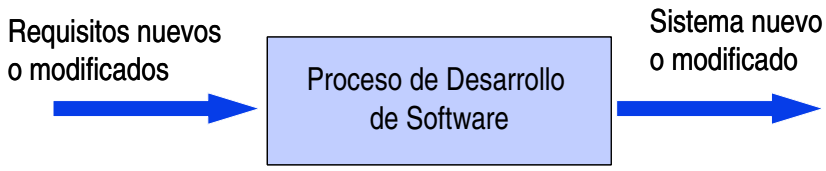

Figura 2: proceso de desarrollo de software.

El proceso de desarrollo de software no es único. No existe un proceso de software universal que sea efectivo para todos los contextos de proyectos de desarrollo. Debido a esta diversidad, es difícil automatizar todo un proceso de desarrollo de software.

A pesar de la variedad de propuestas de proceso de software, existe un conjunto de actividades fundamentales que se encuentran presentes en todos ellos, las cuales se presentan a continuación :

a. **Especificación de software**: Se debe definir la funcionalidad y restricciones operacionales que debe cumplir el software.
b. **Diseño e Implementación**: Se diseña y construye el software de acuerdo a la especificación.
c. **Validación**: El software debe validarse, para asegurar que cumpla con lo que quiere el cliente.
d. **Evolución**: El software debe evolucionar, para adaptarse a las necesidades del cliente.

Además de estas actividades fundamentales, Pressman [1] menciona un conjunto de “actividades protectoras”, que se aplican a lo largo de todo el proceso del software. Ellas se señalan a continuación:

- Seguimiento y control de proyecto de software.
- Revisiones técnicas formales.
- Garantía de calidad del software.
- Gestión de configuración del software.
- Preparación y producción de documentos.
- Gestión de reutilización.
- Mediciones.
- Gestión de riesgos.

Pressman [1] caracteriza un proceso de desarrollo de software como se muestra en la Figura 3. Los elementos involucrados se describen a continuación:

- **Un marco común del proceso**, definiendo un pequeño número de actividades del marco de trabajo que son aplicables a todos los proyectos de software, con independencia del tamaño o complejidad.
- **Un conjunto de tareas**, cada uno es una colección de tareas de ingeniería del software, hitos de proyectos, entregas y productos de trabajo del software, y puntos de garantía de calidad, que permiten que las actividades del marco de trabajo se adapten a las características del proyecto de software y los requisitos del equipo del proyecto.
- **Las actividades de protección**, tales como garantía de calidad del software, gestión de configuración del software y medición, abarcan el modelo del proceso. Las actividades de protección son independientes

de cualquier actividad del marco de trabajo y aparecen durante todo el proceso.

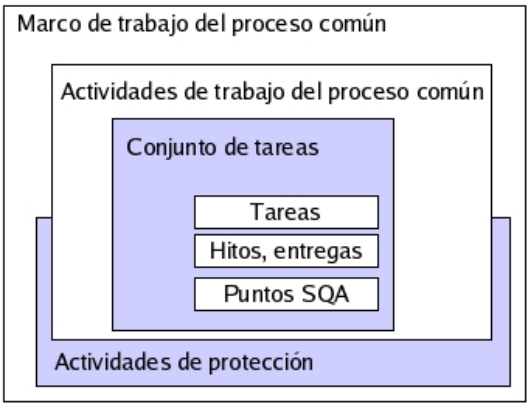

Figura 3: Elementos del proceso del software

Otra perspectiva utilizada para determinar los elementos del proceso de desarrollo de software es establecer las relaciones entre elementos que permitan responder **Quién** debe hacer **Qué**, **Cuándo** y **Cómo** debe hacerlo [5].

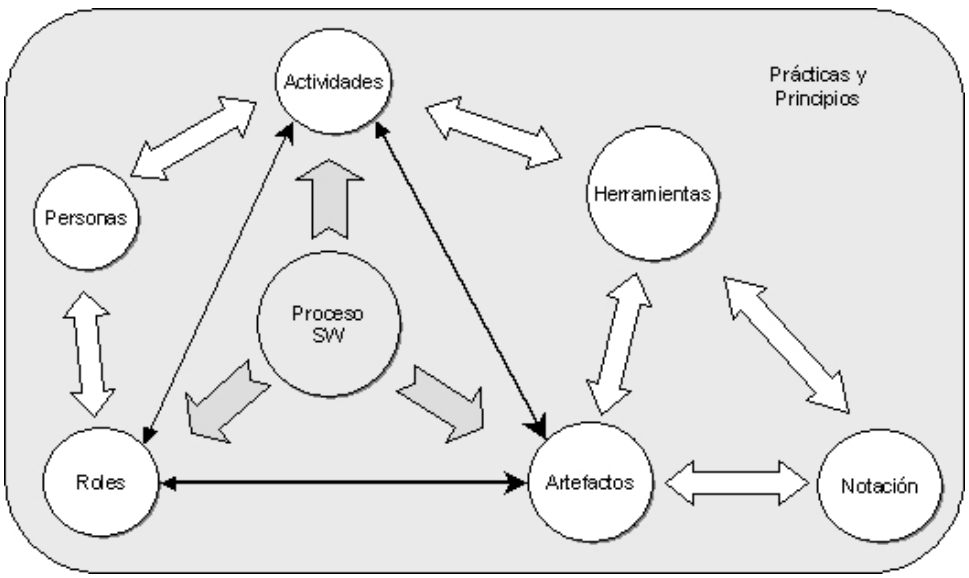

Figura 4: Relación entre elementos del proceso del software

En la Figura 4 se muestran los elementos de un proceso de desarrollo de software y sus relaciones. Así las interrogantes se responden de la siguiente forma:

- **Quién:** Las Personas participantes en el proyecto de desarrollo desempeñando uno o más Roles específicos.
- **Qué:** Un Artefacto es producido por un Rol en una de sus Actividades. Los Artefactos se especifican utilizando Notaciones específicas. Las Herramientas apoyan la elaboración de Artefactos soportando ciertas Notaciones.
- **Cómo y Cuándo:** Las Actividades son una serie de pasos que lleva a cabo un Rol durante el proceso de desarrollo. El avance del proyecto está controlado mediante hitos que establecen un determinado estado de terminación de ciertos Artefactos.

La composición y sincronía de las actividades está basada en un conjunto de Principios y Prácticas. Las Prácticas y Principios enfatizan ciertas actividades y/o la forma como deben realizarse, por ejemplo: desarrollar iterativamente, gestionar requisitos, desarrollo basado en componentes, modelar visualmente, verificar continuamente la calidad, gestionar los cambios, etc.

1.2.3 Modelos de Proceso de Desarrollo de Software

Sommerville define modelo de proceso de software como “Una representación simplificada de un proceso de software, representada desde una perspectiva específica. Por su naturaleza los modelos son

simplificados, por lo tanto un modelo de procesos del software es una abstracción de un proceso real."

Los modelos genéricos no son descripciones definitivas de procesos de software; sin embargo, son abstracciones útiles que pueden ser utilizadas para explicar diferentes enfoques del desarrollo de software.

A continuación explicaremos las principales características de los modelos de procesos de software mas representativos.

- **Codificar y corregir (Code-and-Fix)**

  Este es el modelo básico utilizado en los inicios del desarrollo de software. Contiene dos pasos:

  - Escribir código.
  - Corregir problemas en el código.

  Se trata de primero implementar algo de código y luego pensar acerca de requisitos, diseño, validación, y mantenimiento.

  Este modelo tiene tres problemas principales [7]:

  - Después de un número de correcciones, el código puede tener una muy mala estructura, hace que los arreglos sean muy costosos.
  - Frecuentemente, aún el software bien diseñado, no se ajusta a las necesidades del usuario, por lo que es rechazado o su reconstrucción es muy cara.
  - El código es difícil de reparar por su pobre preparación para probar y modificar.

- **Modelo en cascada**

  El primer modelo de desarrollo de software que se publicó se derivó de otros procesos de ingeniería [8]. Éste toma las actividades

fundamentales del proceso de especificación, desarrollo, validación y evolución y las representa como fases separadas del proceso.

El modelo en cascada consta de las siguientes fases:

a. Definición de los requisitos: Los servicios, restricciones y objetivos son establecidos con los usuarios del sistema. Se busca hacer esta definición en detalle.
b. Diseño de software: Se particiona el sistema en sistemas de software o hardware. Se establece la arquitectura total del sistema. Se identifican y describen las abstracciones y relaciones de los componentes del sistema.
c. Implementación y pruebas unitarias: Construcción de los módulos y unidades de software. Se realizan pruebas de cada unidad.
d. Integración y pruebas del sistema: Se integran todas las unidades. Se prueban en conjunto. Se entrega el conjunto probado al cliente.
e. Operación y mantenimiento: Generalmente es la fase más larga. El sistema es puesto en marcha y se realiza la corrección de errores descubiertos. Se realizan mejoras de implementación. Se identifican nuevos requisitos.

La interacción entre fases puede observarse en la Figura 5. Cada fase tiene como resultado documentos que deben ser aprobados por el usuario. Una fase no comienza hasta que termine la fase anterior y generalmente se incluye la corrección de los problemas encontrados en fases previas.

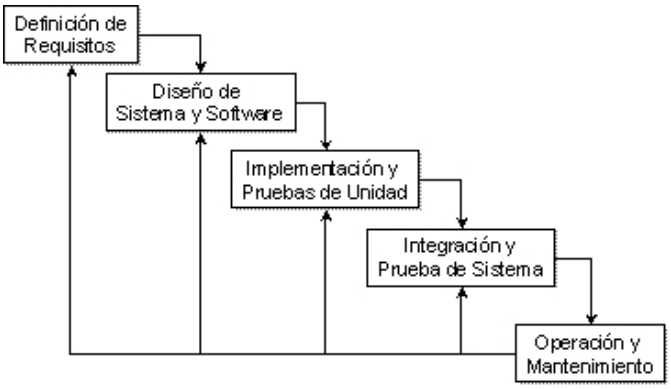

Figura 5: Modelo de desarrollo en cascada.

En la práctica, este modelo no es lineal, e involucra varias iteraciones e interacción entre las distintas fases de desarrollo. Algunos problemas que se observan en el modelo de cascada son:

- Las iteraciones son costosas e implican rehacer trabajo debido a la producción y aprobación de documentos.
- Aunque son pocas iteraciones, es normal congelar parte del desarrollo y continuar con las siguientes fases.
- Los problemas se dejan para su posterior resolución, lo que lleva a que estos sean ignorados o corregidos de una forma poco elegante.
- Existe una alta probabilidad de que el software no cumpla con los requisitos del usuario por el largo tiempo de entrega del producto.
- Es inflexible a la hora de evolucionar para incorporar nuevos requisitos. Es difícil responder a cambios en los requisitos.

Este modelo sólo debe usarse si se entienden a plenitud los requisitos. Aún se utiliza como parte de proyectos grandes.

➢ **Desarrollo evolutivo**

La idea detrás de este modelo es el desarrollo de una implantación del sistema inicial, exponerla a los comentarios del usuario, refinarla en N

versiones hasta que se desarrolle el sistema adecuado. En la Figura 6 se observa cómo las actividades concurrentes: especificación, desarrollo y validación, se realizan durante el desarrollo de las versiones hasta llegar al producto final.

Una ventaja de este modelo es que se obtiene una rápida realimentación del usuario, ya que las actividades de especificación, desarrollo y pruebas se ejecutan en cada iteración.

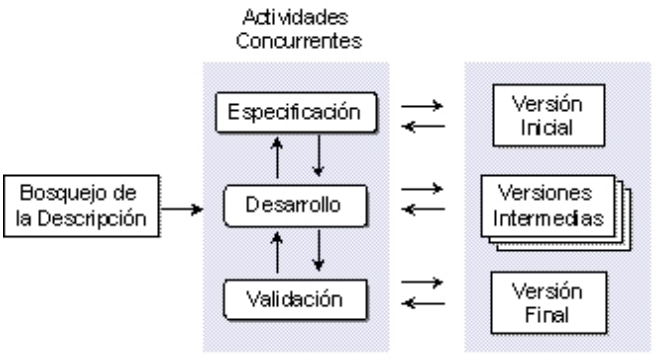

Figura 6: Modelo de desarrollo evolutivo.

Existen dos tipos de desarrollo evolutivo:

- Desarrollo Exploratorio: El objetivo de este enfoque es explorar con el usuario los requisitos hasta llegar a un sistema final. El desarrollo comienza con las partes que se tiene más claras. El sistema evoluciona conforme se añaden nuevas características propuestas por el usuario.
- Enfoque utilizando prototipos: El objetivo es entender los requisitos del usuario y trabajar para mejorar la calidad de los requisitos. A diferencia del desarrollo exploratorio, se comienza por definir los requisitos que no están claros para el usuario y se utiliza un prototipo para experimentar con ellos. El prototipo ayuda a terminar de definir estos requisitos.

Entre los puntos favorables de este modelo están:

- La especificación puede desarrollarse de forma creciente.

- Los usuarios y desarrolladores logran un mejor entendimiento del sistema. Esto se refleja en una mejora de la calidad del software.
- Es más efectivo que el modelo de cascada, ya que cumple con las necesidades inmediatas del cliente.

Desde una perspectiva de ingeniería y administración se identifican los siguientes problemas:

- Proceso no Visible: Los administradores necesitan entregas para medir el progreso. Si el sistema se necesita desarrollar rápido, no es efectivo producir documentos que reflejen cada versión del sistema.
- Sistemas pobremente estructurados: Los cambios continuos pueden ser perjudiciales para la estructura del software haciendo costoso el mantenimiento.
- Se requieren técnicas y herramientas: Para el rápido desarrollo se necesitan herramientas que pueden ser incompatibles con otras o que poca gente sabe utilizar.

  Este modelo es efectivo en proyectos pequeños (menos de 100,000 líneas de código) o medianos (hasta 500,000 líneas de código) con poco tiempo para su desarrollo y sin generar documentación para cada versión.

  Para proyectos de sistemas de largo plazo de entrega, es mejor combinar lo mejor del modelo de cascada y evolutivo: se puede hacer un prototipo global del sistema y posteriormente reimplementarlo con un acercamiento más estructurado. Los subsistemas con requisitos bien definidos y estables se pueden programar utilizando cascada y la

interfaz de usuario se puede especificar utilizando un enfoque exploratorio.

- **Desarrollo formal de sistemas**

  Este modelo se basa en transformaciones formales de los requisitos hasta llegar a un programa ejecutable.

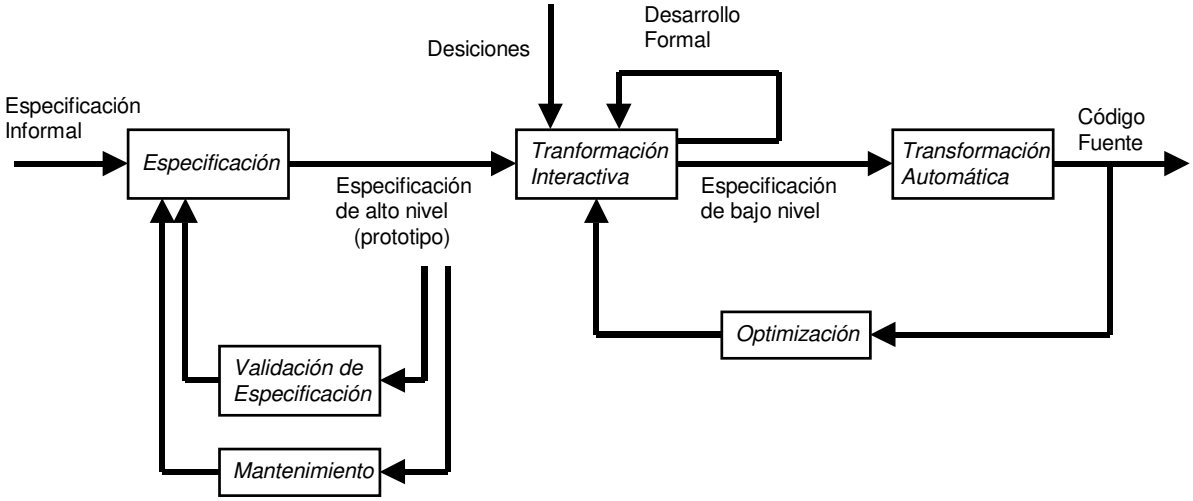

Figura 7: Paradigma de programación automática.

La Figura 7 (obtenida desde [20]) ilustra un paradigma ideal de programación automática. Se distinguen dos fases globales: especificación (incluyendo validación) y transformación. Las características principales de este paradigma son: la especificación es formal y ejecutable constituye el primer prototipo del sistema), la especificación es validada mediante prototipación. Posteriormente, a través de transformaciones formales la especificación se convierte en la implementación del sistema, en el último paso de transformación se obtiene una implementación en un lenguaje de programación determinado, el mantenimiento se realiza sobre la especificación (no sobre el código fuente), la documentación es generada

automáticamente y el mantenimiento es realizado por repetición del proceso (no mediante parches sobre la implementación).

Observaciones sobre el desarrollo formal de sistemas:

- Permite demostrar la corrección del sistema durante el proceso de transformación. Así, las pruebas que verifican la correspondencia con la especificación no son necesarias.
- Es atractivo sobre todo para sistemas donde hay requisitos de seguridad y confiabilidad importantes.
- Requiere desarrolladores especializados y experimentados en este proceso para llevarse a cabo.

➢ **Desarrollo basado en reutilización**

Como su nombre lo indica, es un modelo fuertemente orientado a la reutilización. Este modelo consta de 4 fases ilustradas en la Figura 9. A continuación se describe cada fase:

1. Análisis de componentes: Se determina qué componentes pueden ser utilizados para el sistema en cuestión. Casi siempre hay que hacer ajustes para adecuarlos.
2. Modificación de requisitos: Se adaptan (en lo posible) los requisitos para concordar con los componentes de la etapa anterior. Si no se puede realizar modificaciones en los requisitos, hay que seguir buscando componentes más adecuados (fase 1).
3. Diseño del sistema con reutilización: Se diseña o reutiliza el marco de trabajo para el sistema. Se debe tener en cuenta los componentes localizados en la fase 2 para diseñar o determinar este marco.

4. Desarrollo e integración: El software que no puede comprarse, se desarrolla. Se integran los componentes y subsistemas. La integración es parte del desarrollo en lugar de una actividad separada.

Las ventajas de este modelo son:

- Disminuye el costo y esfuerzo de desarrollo.
- Reduce el tiempo de entrega.
- Disminuye los riesgos durante el desarrollo.

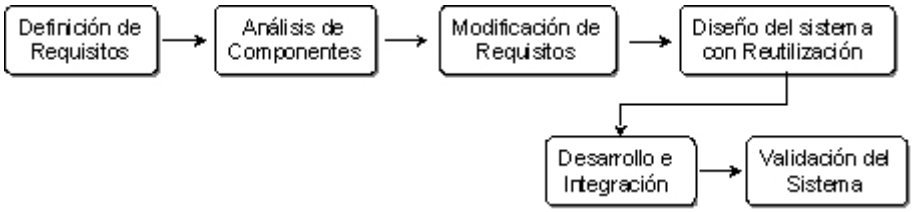

Figura 8: Desarrollo basado en reutilización de componentes

Desventajas de este modelo:

- Los “compromisos” en los requisitos son inevitables, por lo cual puede que el software no cumpla las expectativas del cliente.
- Las actualizaciones de los componentes adquiridos no están en manos de los desarrolladores del sistema.

Procesos iterativos

A continuación se expondrán dos enfoques híbridos, especialmente diseñados para el soporte de las iteraciones:

- Desarrollo Incremental.
- Desarrollo en Espiral.

- **Desarrollo Incremental**

Mills [9] sugirió el enfoque incremental de desarrollo como una forma de reducir la repetición del trabajo en el proceso de desarrollo y dar oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema (ver Figura 10). Es una combinación del Modelo de Cascada y Modelo Evolutivo.

Reduce el rehacer trabajo durante el proceso de desarrollo y da oportunidad para retrasar las decisiones hasta tener experiencia en el sistema.

Durante el desarrollo de cada incremento se puede utilizar el modelo de cascada o evolutivo, dependiendo del conocimiento que se tenga sobre los requisitos a implementar. Si se tiene un buen conocimiento, se puede optar por cascada, si es dudoso, evolutivo.

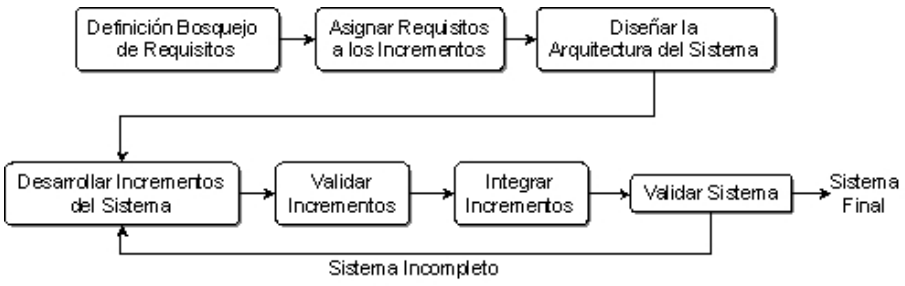

Figura 9: Modelo de desarrollo iterativo incremental.

Entre las ventajas del modelo incremental se encuentran:

- Los clientes no esperan hasta el fin del desarrollo para utilizar el sistema. Pueden empezar a usarlo desde el primer incremento.
- Los clientes pueden aclarar los requisitos que no tengan claros conforme ven las entregas del sistema.

- Se disminuye el riesgo de fracaso de todo el proyecto, ya que se puede distribuir en cada incremento.
- Las partes más importantes del sistema son entregadas primero, por lo cual se realizan más pruebas en estos módulos y se disminuye el riesgo de fallos.

Algunas de las desventajas identificadas para este modelo son:

- Cada incremento debe ser pequeño para limitar el riesgo (menos de 20.000 líneas).
- Cada incremento debe aumentar la funcionalidad.
- Es difícil establecer las correspondencias de los requisitos contra los incrementos.
- Es difícil detectar las unidades o servicios genéricos para todo el sistema.

➢ **Desarrollo en Espiral**

El modelo de desarrollo en espiral (ver Figura 11) es actualmente uno de los más conocidos y fue propuesto por Boehm [7]. El ciclo de desarrollo se representa como una espiral, en lugar de una serie de actividades sucesivas con retrospectiva de una actividad a otra.

Cada ciclo de desarrollo se divide en cuatro fases:

1. Definición de objetivos: Se definen los objetivos. Se definen las restricciones del proceso y del producto. Se realiza un diseño detallado del plan administrativo. Se identifican los riesgos y se elaboran estrategias alternativas dependiendo de estos.
2. Evaluación y reducción de riesgos: Se realiza un análisis detallado de cada riesgo identificado. Pueden desarrollarse prototipos para disminuir

el riesgo de requisitos dudosos. Se llevan a cabo los pasos para reducir los riesgos.

3. Desarrollo y validación: Se escoge el modelo de desarrollo después de la evaluación del riesgo. El modelo que se utilizará (cascada, sistemas formales, evolutivo, etc.) depende del riesgo identificado para esa fase.
4. Planificación: Se determina si continuar con otro ciclo. Se planea la siguiente fase del proyecto.

Este modelo a diferencia de los otros toma en consideración explícitamente el riesgo, esta es una actividad importante en la administración del proyecto.

El ciclo de vida inicia con la definición de los objetivos. De acuerdo a las restricciones se determinan distintas alternativas. Se identifican los riesgos al sopesar los objetivos contra las alternativas. Se evalúan los riesgos con actividades como análisis detallado, simulación, prototipos, etc. Se desarrolla un poco el sistema. Se planifica la siguiente fase.

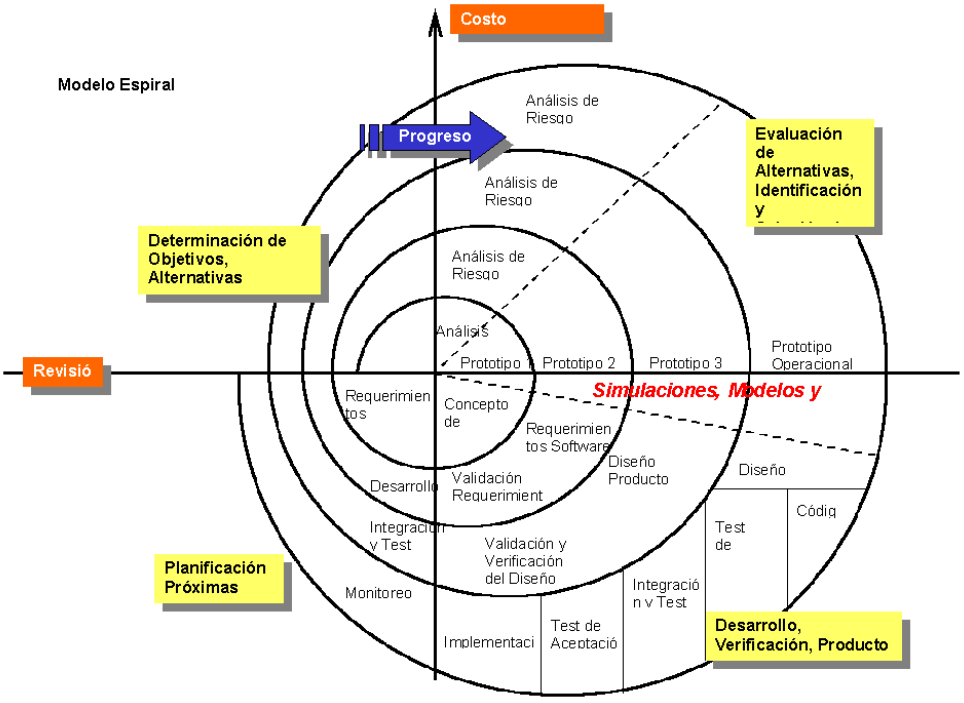

Figura 10: Modelo de desarrollo en Espiral

1.2.4 ¿Cuál Modelo de Proceso es el mas adecuado?

Cada proyecto de software requiere de una forma particular de abordar el problema. Las propuestas comerciales y académicas actuales promueven procesos iterativos, donde en cada iteración puede utilizarse uno u otro modelo de proceso, considerando un conjunto de criterios (Por ejemplo: grado de definición de requisitos, tamaño del proyecto, riesgos identificados, entre otros).

En la Tabla 1 se expone un cuadro comparativo de acuerdo con algunos criterios básicos para la selección de un modelo de proceso [10], la medida utilizada indica el nivel de efectividad del modelo de proceso de acuerdo al criterio (Por ejemplo: El modelo Cascada responde con un nivel de efectividad Bajo cuando los Requisitos y arquitectura no están predefinidos).

Modelo de Procesos y Criterios	Funciona con requisitos y arquitectura no predefinidos	Produce software altamente fiable	Gestión de riesgos	Permite correcciones sobre la marcha	Visión del progreso por el Cliente y el Jefe del proyecto
Codificar y corregir	Bajo	Bajo	Bajo	Alto	Medio
Cascada	Bajo	Alto	Bajo	Bajo	Bajo
Evolutivo exploratorio	Medio o Alto	Medio o Alto	Medio	Medio o Alto	Medio o Alto
Evolutivo prototipado	Alto	Medio	Medio	Alto	Alto
Desarrollo formal de sistemas	Bajo	Alto	Bajo a Medio	Bajo	Bajo
Desarrollo orientado a reutilización	Medio	Bajo a Alto	Bajo a Medio	Alto	Alto
Incremental	Bajo	Alto	Medio	Bajo	Bajo
Espiral	Alto	Alto	Alto	Medio	Medio

Tabla 1: Comparación entre modelos de proceso de software, según criterios de evaluación.

CAPITULO II

2. PLANTEAMIENTO DEL PROBLEMA

2.1 ANTECEDENTES

En el siglo XX, se dio una de las más grandes revoluciones de la humanidad: la revolución informática. A medida que la electrónica evolucionaba, los diseños de sistemas computacionales realizados en el siglo XIX se hacían cada vez más reales. Primero eran computadores análogos, luego gracias al uso de la electrónica digital (Que inventó Claude Shannon en 1937) se pudieron crear los primeros computadores con la capacidad de ser programados. A pesar que no se sabe con certeza cuándo ni donde se construyó el primer computador, se destacan casos como el computador Atanasoff Berry, el ENIAC (1943), el computador British Colossus (1944) y las máquinas Konrad Zuse's Z.

Así fue como en 1957 el término "Software" fue utilizado por primera vez por John W. Tukey, con base a los conceptos planteados por Alan Mathison Turing y Alonzo Church en su célebre Tesis Church-Turing, donde afirman que cualquier cálculo puede ser realizado por un algoritmo que se encuentre corriendo en un computador, siempre y

cuando se le de el tiempo y el almacenamiento necesario para su trabajo.

De esta manera se comenzó a usar el término Ingeniería de Software para referenciar a la profesión encargada de crear y mantener aplicaciones software por medio del uso tecnologías, prácticas y métodos para gestionar un proyecto relacionado con las ciencias de la computación

La ingeniería de software tiene como base fundamental el uso de metodologías de desarrollo de software, las cuales son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software.

Un objetivo de estas ultimas décadas ha sido el encontrar procesos o metodologías predecibles y repetibles que mejoren la productividad y la calidad.

2.2 DEFINICION DEL PROBLEMA

Siendo el desarrollo de software una tarea difícil, su realización trae consigo muchos problemas actuales como son el de no contar con un marco de trabajo claramente definido y estandarizado que apoye a obtener productos de calidad, que no cumplan con las expectativas del cliente, y que no se desarrollen en el tiempo estimado y los costos presupuestados, entre otras cosas. Generalmente estos problemas ocurren por la falta de una metodología de desarrollo de software.

Actualmente se cuenta con una buena cantidad de propuestas que definen este marco de trabajo, escuchamos hablar de metodologías tradicionales como Rational Unified Process (RUP), Microsoft Solutions

Framework (MSF) y Métrica 3.0 o metodologías ágiles como Extreme Programming pero las experiencias indican que no existe una receta mágica y no se trata de solo seleccionar una de las propuestas y seguirla.

La selección de una metodología no es una tarea trivial ni automática. Para hacerla adecuadamente se debe realizar una evaluación de las propuestas existentes buscando que sea flexible y adaptable a las necesidades y naturaleza de la organización y sus proyectos. Una Adecuada selección de una metodología provoca confianza en la misma por parte del grupo de proyecto.

Por otro lado, la selección no adecuada de una metodología podría ocasionar trastornos durante el proyecto e incluso dificultar el logro de los objetivos planteados en el mismo.

El propósito de esta tesina es ofrecer una herramienta inicial que permita evaluar propuestas de marcos de trabajo o procesos para el desarrollo de software (metodologías). Para ello se va a definir claramente qué es un proceso de desarrollo de software, qué elementos lo componen y los diferentes modelos de proceso existentes. Para ejemplificar y evaluar la definición de proceso estudiaremos las metodologías Orientadas al Plan /Tradicionales (RUP) y Ágiles (XP)[13], por ser estas las mas reconocidas además haremos la aplicación de la metodología elegida para un caso practico.

[13] Según Survey de Cuttter Consortium XP, es la mas reconocida y transgresora así como la mas popular de los MAs con 38% del mercado ágil contra 23% de su competidor mas cercano, FDD, luego vienen ASD con 22%, DSDM con 19%, Cristal con 8%, Lean Development con 7% y Scrum con 3%, los demás suman 9%.

2.3 JUSTIFICACION E IMPORTANCIA

La evolución de la disciplina de ingeniería de software ha traído consigo propuestas diferentes para mejorar los resultados del proceso de construcción. Las Metodologías Orientadas al Plan/Tradicionales haciendo énfasis en la planeación, y las Metodologías Agiles haciendo énfasis en la adaptabilidad del proceso, delinean las principales propuestas presentes en la literatura. De manera paralela, el tema de modelos para el mejoramiento de los procesos de desarrollo ocupa un lugar importante en la búsqueda de la metodología adecuada para producir software de calidad en cualquier contexto de desarrollo.

2.4 METODOLOGIAS EXISTENTES

Existen muchos tipos de Metodologías de desarrollo de software. Aquí presentamos algunas de ellos, pudiendo existir en algunos casos combinaciones de ellas.

2.4.1 Metodología Estructurada

Los métodos estructurados son aquellos que enfocan los requerimientos de software hacia la estructura de datos, estos comenzaron a desarrollarse a fines de los 70's con la Programación Estructurada, luego a mediados de los 70's aparecieron técnicas para el Diseño (por ejemplo: el diagrama de Estructura) primero y posteriormente para el Análisis (por ejemplo: Diagramas de Flujo de Datos). Estas metodologías son particularmente apropiadas en proyectos que utilizan para la implementación lenguajes de 3ra y 4ta generación.

Ejemplos de metodologías estructuradas de ámbito gubernamental: MERISE (Francia), MÉTRICA (España), SSADM (Reino Unido).

Ejemplos de propuestas de métodos estructurados en el ámbito académico: Gane & Sarson Ward & Mellor, Yourdon & DeMarco e Information Engineering

2.4.2 Metodología orientada a Objetos

La metodología orientada a objetos son aquellas cuya pieza básica del software es el objeto, su historia va unida a la evolución de los lenguajes de programación orientadas a objeto

En 1995 Booch y Rumbaugh proponen el Método Unificado con la ambiciosa idea de conseguir una unificación de sus métodos y notaciones, que posteriormente se reorienta a un objetivo más modesto, para dar lugar al Unified Modeling Language (UML), la notación OO más popular en la actualidad.

Algunos métodos OO con notaciones predecesoras de UML son: OOAD (Booch), OOSE (Jacobson), Coad & Yourdon, Shaler & Mellor y OMT (Rumbaugh).

Algunas metodologías orientadas a objetos que utilizan la notación UML son: Rational Unified Process (RUP), OPEN MÉTRICA (que también soporta la notación estructurada).

2.4.3 Metodología orientada al plan/tradicional (no ágil)

Las metodologías no ágiles son aquellas que están guiadas por una fuerte planificación durante todo el proceso de desarrollo; llamadas también metodologías tradicionales o clásicas, donde se realiza una intensa etapa de análisis y diseño antes de la construcción del sistema.

Todas las propuestas metodológicas antes indicadas pueden considerarse como metodologías tradicionales. Aunque en el caso particular de RUP, por el especial énfasis que presenta en cuanto a su adaptación a las condiciones del proyecto (mediante su configuración previa a aplicarse), realizando una configuración adecuada, podría considerarse Ágil.

2.4.4 Metodología ágil

Un proceso es ágil cuando el desarrollo de software es incremental (entregas pequeñas de software, con ciclos rápidos), cooperativo (cliente y desarrolladores trabajan juntos constantemente con una cercana comunicación), sencillo (el método en sí mismo es fácil de aprender y modificar, bien documentado), y adaptable (permite realizar cambios de último momento)

CAPITULO III

3. METODOLOGÍAS DE DESARROLLO DE SOFTWARE

3.1 INTRODUCCION

La experiencia de las personas que se encuentran en el ambiente de desarrollo de software , indica que los proyectos exitosos son aquellos que son administrados siguiendo una serie de procesos que permiten organizar y luego controlar al proyecto. Según esta visión , los proyectos que no sigan estos lineamientos corren un alto riesgo de fracasar. Por otro lado, en los últimos años, ha surgido una corriente en la industria del software que considera que las necesidades de los clientes son muy cambiantes, y que si no nos adaptamos rápidamente, corremos el riesgo de estar resolviendo el problema equivocado. En pos de esto se proponen metodologías que parecen contradecir la visión tradicional de los proyectos.

3.2 EVOLUCION

Las metodologías de desarrollo de software surgieron a raíz de la necesidad de controlar y documentar proyectos cada vez más complejos, impulsadas principalmente por instituciones económicamente

importantes y con requisitos de seguridad y fiabilidad en sus sistemas sumamente estrictos. Pero la evolución de estas metodologías, que sugieren o imponen a menudo varias actividades paralelas para cada fase del ciclo de vida de los sistemas y que asimismo pueden requerir más de un modelo para cada actividad, determinó que el costo y el esfuerzo requeridos en producir y mantener los documentos relativos al proceso de desarrollo crecieran considerablemente, a tal punto que hoy en día sólo instituciones muy grandes o que desarrollan sistemas muy complejos las adoptan y cumplen formalmente; es así que surge la necesitad de crear nuevas metodologías para hacer frente a esta problemática.

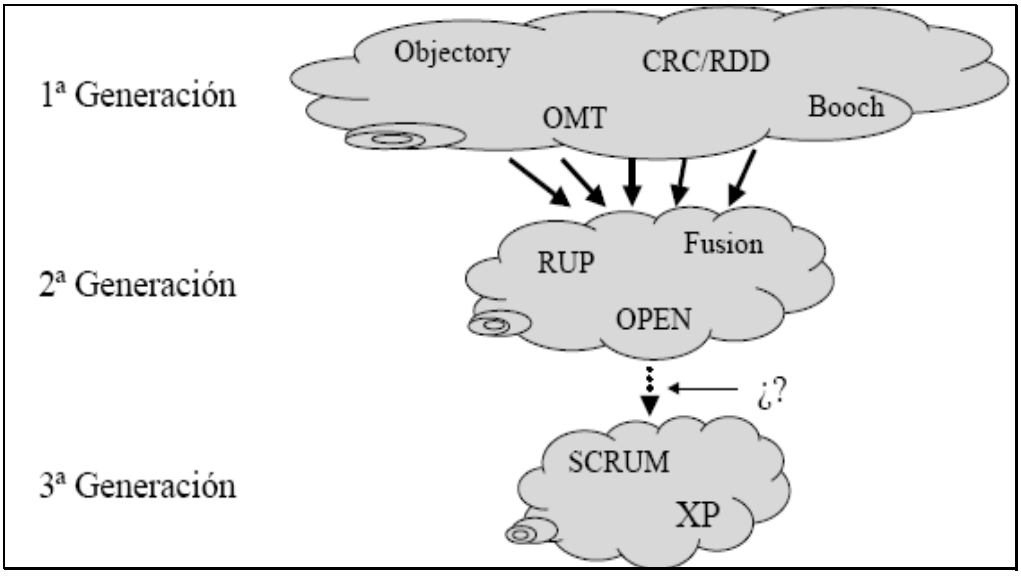

Figura 111: Evolución de las Metodologías de Desarrollo de SW

3.3 CLASIFICACION

Existen varios criterios de clasificación de las metodologías

- ***Basadas en las notaciones utilizadas*** (para especificar artefactos producidos en actividades de análisis y diseño)

Metodologías Estructuradas, Enfocan los requerimientos de software hacia la estructura de datos.

Metodologías Orientadas a Objetos, Son aquellas cuya pieza básica del software es el objeto.

- ***Considerando su filosofía de desarrollo***

Metodologías Orientadas al Plan/Tradicional, Ponen mayor énfasis en la planificación y control del proyecto. Consideradas Predictivas debido a la estrategia de desarrollo de software.

Metodologías Ágiles, Orientadas a la generación de código con ciclos de desarrollo muy corto. Consideradas Adaptativas debido a la estrategia de desarrollo de Software.

Explicadas Cada una de ellas anteriormente.

3.4 ELEMENTOS Y CARACTERÍSTICAS DESEABLES DE UNA METODOLOGÍA

Elementos de una Metodología

- Ciclo de vida: jerarquización ordenada del desarrollo completo.
- Productos intermedios y final.
- Procedimientos y herramientas.
- Criterios de evaluación.

La metodología de desarrollo de software nos permite definir :

- ¿Cómo se divide el proyecto?
- ¿Qué tareas en cada etapa?
- ¿Qué salidas y cuando se producen?
- ¿Restricciones?

- ¿Herramientas?
- ¿Cómo gestionar, cómo controlar?

Características Deseable de una Metodología

- Existencia de reglas predefinidas
- Cobertura total del ciclo de desarrollo
- Verificaciones intermedias
- Planificación y Control
- Comunicación Efectiva
- Utilización sobre un abanico amplio de proyectos
- Fácil Formación
- Herramienta CASE
- Actividades que mejoren el proceso de desarrollo
- Soporte al mantenimiento
- Soporte de la reutilización de software

3.5 VENTAJAS DEL USO DE METODOLOGÍAS

Desde el punto de vista de gestión

- Facilita la planificación
- Facilita el control y seguimiento de los proyectos
- Mejora del ratio costo / beneficio
- Optimiza la gestión de recursos
- Facilita la comunicación entre los participantes
- Facilita la evaluación de los proyectos

Desde el punto de vista de los Ingenieros en el desarrollo

- Comprensión del problema
- Optimización del proceso, y dentro del proceso las fases a seguir

- Facilidad de mantenimiento
- Mejora de la reusabilidad

Desde el punto de vista del cliente/usuario final

- Garantiza en la medida de lo posible la calidad del producto
- Mayor confianza

3.6 METODOLOGÍA ORIENTADAS AL PLAN

3.6.1 Antecedentes

Estas metodologías son desarrolladas a partir de la Ingeniería de Sistemas y de los métodos para mejoras de procesos, lo que requiere procesos definidos. Planificación predictiva definición de tareas e hitos y documentación como producto intermedio entre las etapas. Esto tiende a permitir controlas medir y mejorar el proceso.

Históricamente, las metodologías de este tipo surgieron de grandes proyectos de defensa y aeroespaciales. Es por eso que soportan naturalmente los proyectos en los que el software se desarrolla conjuntamente con el hardware.

En el PMBOK (PMI 2004), establece su relación con las metodologías orientadas al plan ya que las identifica como buenas practicas y conocimientos claves necesarios para administrar exitosamente proyectos de desarrollo de software. En el área de proyectos de software, esto se refleja en metodologías como RUP y en modelos de madurez como SW-CMM. Se caracterizan por exponer procesos basados en planeación exhaustiva. Esta planeación se realiza esperando que el resultado de cada proceso sea determinante y predecible. La experiencia ha mostrado que, como consecuencia de las

características del software, los resultados de los procesos no son siempre predecibles y sobre todo, es difícil predecir desde el comienzo del proyecto cada resultado. Sin embargo, es posible por medio de la recolección y estudio de métricas de desarrollo lograr realizar estimaciones acertadas en contextos de desarrollo repetibles.

3.7 RATIONAL UNIFIED PROCESS - RUP

3.7.1 Introducción

Rational Unified Process (RUP) es un proceso de Ingeniería de Software planteado por Kruchten (1996) cuyo objetivo es producir software de alta calidad, es decir, que cumpla con los requerimientos de los usuarios dentro de una planificación y presupuesto establecidos.

Cubre el ciclo de vida de desarrollo de software.

RUP toma en cuenta las mejores prácticas en el modelo de desarrollo de software en particular las siguientes:

- Desarrollo de software en forma iterativa (repite una acción).
- Manejo de requerimientos.
- Utiliza arquitectura basada en componentes.
- Modela el software visualmente (Modela con Unified Modeling Language, UML)
- Verifica la calidad del software.
- Controla los cambios.

El proceso iterativo de RUP se organiza en fases (Kruchten, 1996), cada fase se concluye con una piedra de milla (mile stone) principal (ver figura 1). Es importante resaltar que la inclusión de piedras de millas o puntos de revisión, es sumamente importante y en estos puntos se revisan los

requerimientos establecidos para cada fase, basados en los controles de calidad . De esta manera, si un producto o proceso no pasa el punto de revisión de calidad, se rediseña o se cancela, evitando así, los problemas de coste extra, de retrabajo, y de productos de mala calidad, que no satisfacen los requerimientos establecidos a nivel educativo, comunicacional, técnico y de diseño gráfico. Los puntos de revisión están basados a su vez en cuestionarios elaborados a partir de métricas establecidas producto de la experiencia y de la investigación (Díaz-Antón, Pérez, Grimmán y Mendoza, 2002). En la figura 2 se puede observar la expresión gráfica equivalente al tiempo y esfuerzo que se dedican a cada una de las fases de RUP. En esta gráfica se puede observar que la inversión de tiempo y esfuerzo en la primera fase y segunda fase es pequeña en comparación con la tercer fase, garantizando así que la mayor parte del trabajo, costo y esfuerzo se realice si y solo si, ha pasado la segunda piedra de milla, o sea, el segundo control de revisión de calidad.

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, divide en 4 fases el desarrollo del software:

3.7.2 Modelo RUP

Es una metodología de ingeniería de software, provee las disciplinas para el desarrollo de sistemas informáticos con tecnología orientada a objetos

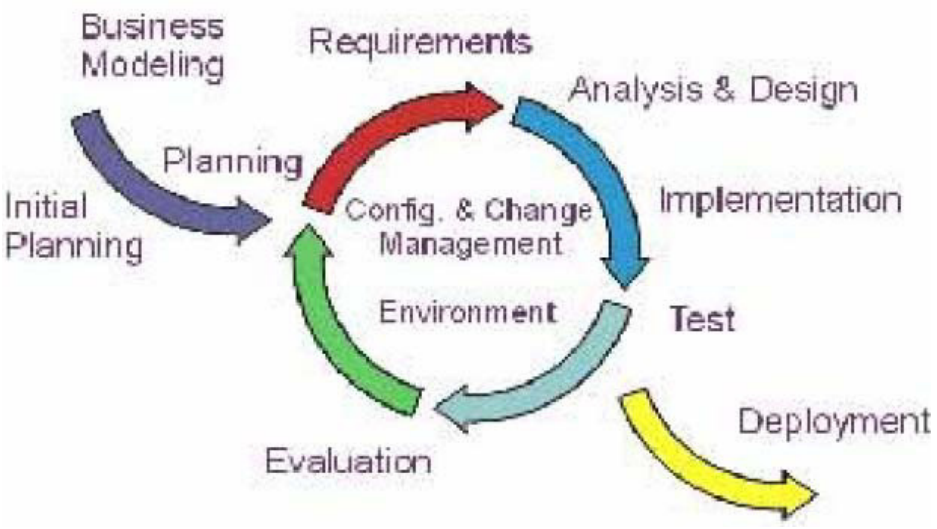

Figura 12: Modelo RUP

3.7.3 Arquitectura RUP

La arquitectura de RUP tiene dos dimensiones.

Verticalmente(Etapas), Representa las etapas para la ejecución de las actividades del proyecto.

- Grupo 1: Etapas para el desarrollo del sistema.
- Grupo 2: Etapas para la gestión del proyecto.

Horizontalmente(Fases), Representa las fases de ejecución del ciclo de vida del proyecto

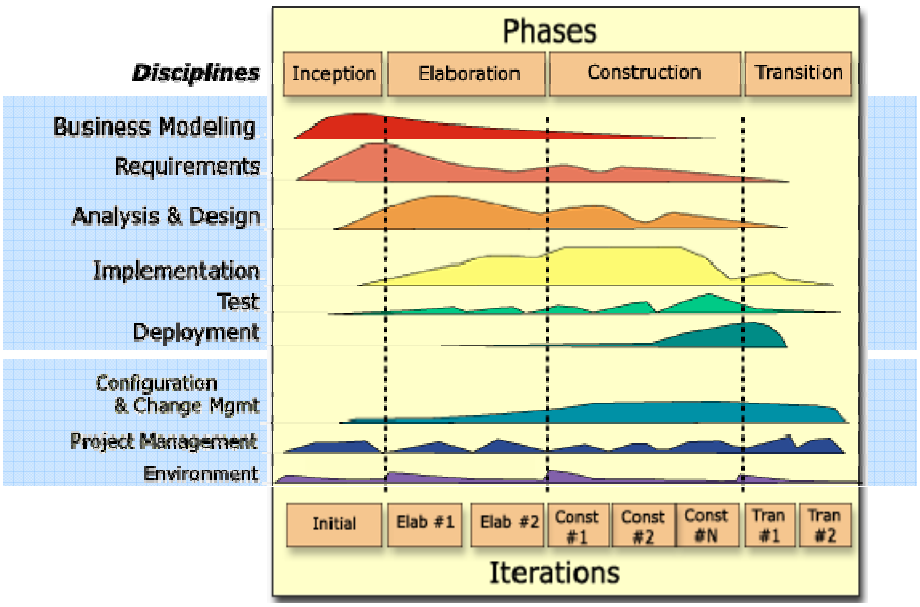

Figura 13: Arquitectura RUP

Grupo 1 : <u>Etapas para el Desarrollo del Sistema</u>

- Muestra el aspecto estático del proceso.
- Se organiza según el contenido del proyecto.
- Se describe en términos de etapas, actividades, workflows, artefactos y personas.

Las etapas para el desarrollo de sistemas son las siguientes :

Modelado del Negocio: Entendiendo las necesidades del negocio.

Requerimientos: Trasladando las necesidades del negocio a un sistema automatizado.

Análisis y Diseño: Trasladando los requerimientos dentro de la arquitectura de software.

Implementación: Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.

Pruebas: Asegurándose que el comportamiento requerido es el correcto y que todo los solicitado esta presente.

Implantación: Hacer todo lo necesario para la salida del proyecto

Grupo 2 : <u>Etapas para la Gestión del Proyecto</u>

- Se organiza según el contenido del proyecto.
- Muestra el aspecto estático del proceso.
- Se describe en términos de soporte y administración del proyectos.

Las etapas para la gestión del proyecto son :

Configuración y Control del cambio: Guardando todas las versiones del proyecto.

Administrando el proyecto: Administrando horarios y recursos.

Administración del Entorno: Administrando el ambiente de desarrollo.

Es recomendable que a cada una de estas iteraciones se les clasifique y ordene según su prioridad, y que cada una se convierte luego en un entregable al cliente. Esto trae como beneficio la retroalimentación que se tendría en cada entregable o en cada iteración

Fases para el Desarrollo de Software - RUP

- Muestra el aspecto dinámico del proceso.
- Lo describe en función del tiempo, ciclos, fases, iteraciones e hitos.

Las Fases de ejecución del ciclo de vida del Proyecto son :

Inicio : El Objetivo en esta etapa es determinar la visión del proyecto.

Elaboración : En esta etapa el objetivo es determinar la arquitectura optima.

Construcción : En esta etapa el objetivo es llevar a obtener la capacidad operacional inicial.

Transmisión : El objetivo es llegar a obtener el release del proyecto.

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

3.7.4 Elementos del RUP

Los elementos del RUP son:

Actividades : Son los procesos que se llegan a determinar en cada iteración.

Trabajadores: Vienen hacer las personas o entes involucrados en cada proceso.

Artefactos: Un artefacto puede ser un documento, un modelo, o un elemento de modelo.

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

3.8 METODOLOGIA AGIL

3.8.1 Antecedentes

En febrero de 2001, tras una reunión celebrada 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.

Tras esta reunión se creó The Agile Alliance3, 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 es fue el Manifiesto Ágil, un documento que resume la filosofía "ágil".

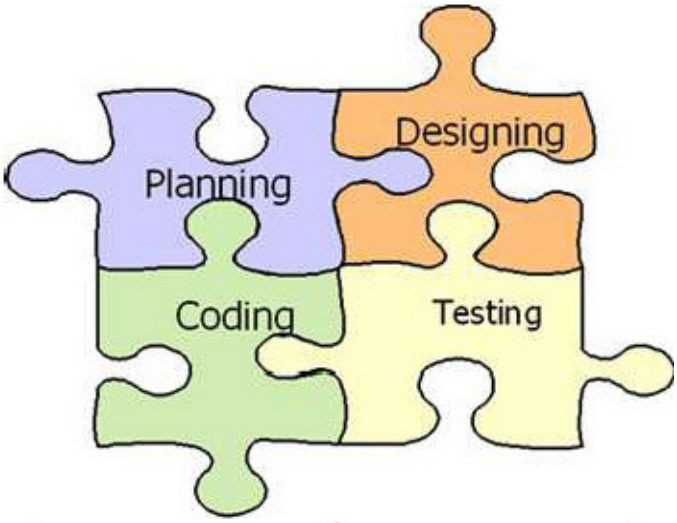

Figura 14: Metodología Ágil

3.8.2 El 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 un 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 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***:

1 La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.

2 Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.

3 Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.

4 La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.

5 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.

6 El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.

7 El software que funciona es la medida principal de progreso.

8 Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.

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

10 La simplicidad es esencial.

11 Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por si mismos.

12 En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

3.8.3 METODOLOGIAS AGILES

Aunque los creadores e impulsores de las metodologías ágiles más populares han suscrito el manifiesto Ágil y coinciden con los principios enunciados anteriormente, cada metodología tiene características propias y hace hincapié en algunos aspectos más específicos. A continuación se resumen otras metodologías ágiles. La mayoría de ellas ya estaban siendo utilizadas con Éxito en proyectos reales pero les faltaba una mayor difusión y reconocimiento.

SCRUM. Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. 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. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. 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.

Crystal Methodologies. Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. Han sido desarrolladas por Alistair Cockburn. 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).

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.

Lean Development(LD). Definida por Bob Charetteís a partir de su experiencia en proyectos con la industria japonesa del automóvil en los años 80 y utilizada en numerosos proyectos de telecomunicaciones en Europa. En LD, los cambios se consideran riesgos, pero si se manejan adecuadamente se pueden convertir en oportunidades que mejoren la

3.9 PROGRAMACIÓN EXTREMA (EXTREME PROGRAMMING, XP)

3.9.1 Introducción

XP [2] es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en 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 como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Kent Beck, el padre de XP, describe la filosofía de XP en [2] sin cubrir los detalles técnicos y de implantación de las prácticas. Posteriormente, otras publicaciones de experiencias se han encargado de dicha tarea. A continuación presentaremos las características esenciales de XP organizadas en los tres apartados siguientes: historias de usuario, roles, proceso y prácticas.

3.9.2 Las Historias de Usuario

Son la técnica utilizada para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas [12]. Beck en su libro [2] presenta un ejemplo de ficha (customer story and task card) en la cual pueden reconocerse los siguientes contenidos: fecha, tipo de actividad (nueva, corrección, mejora), prueba funcional, número de historia, prioridad técnica y del cliente, referencia a otra historia previa, riesgo, estimación técnica, descripción, notas y una lista de seguimiento con la fecha, estado cosas por terminar y comentarios. A efectos de planificación, las historias pueden ser de una a tres semanas de tiempo de programación (para no superar el tamaño de una iteración). Las historias de usuario son descompuestas en tareas de programación (task card) y asignadas a los programadores para ser implementadas durante una iteración.

3.9.3 Roles XP

Los roles de acuerdo con la propuesta original de Beck 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 (Tester). 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 (Tracker). 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 (Coach). 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 (Big boss). Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

3.9.4 Proceso XP

El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos [12]:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.

4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

El ciclo de vida ideal de XP consiste de seis fases [2]: Exploración, Planificación de la Entrega (Release), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

3.9.5. 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.

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.

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.

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).

Diseño simple. Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.

Pruebas . La producción de código está dirigida por las pruebas unitarias. Éstas son son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.

Refactorización (Refactoring). Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplific arlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo [8].

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).

Propiedad colectiva del código. Cualquier programador puede cambiar cualquier parte del código en cualquier momento.

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.

40 horas por semana. Se debe trabaja r 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.

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.

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 .

3.10 QUE METODOLOGÍA ELEGIR ?

El análisis que haremos parte de la premisa que la aplicabilidad de las metodologías depende de las circunstancias. Pero también se debe considerar que algunas decisiones que se toman en cuanto a la forma de encarar el proyecto facilitan o no alguno de los enfoques. Para distinguir esta dos formas de análisis, denominamos Territorios al análisis tomando

Características de los métodos y Factores a las características del problema a resolver.

3.10.1Territorios

Condiciones bajo las cuales cada metodología tiene mas probabilidad de éxito; cuanto mas se aleja el caso en estudio de las características de dicha condición; mas riesgo hay en aplicar tal metodología (Ver Tabla 1). Los territorios son rangos de valores de características. Por ejemplo para las características Tamaño. valores de menos de 10 personas es territorio ágil, mientras que mas de 50 es territorio orientado al plan. Los valores entre 10 y 50 no son claramente territorio de alguna de las metodologías

Características	Ágil	Orientada al Plan
Aplicación		
Objetivo Primario	Obtener valor rápida y continuamente; responder al cambio	Alta seguridad, predecible, repetible, optimizable
Tamaño	Grupo y Proyecto pequeño	Grupo y Proyecto grande
Entorno	Turbulentos, de alto cambio, foco en el proyecto (objetivo inmediato)	Estables, pocos cambios, foco en proyecto y organización, tomar en cuenta al entorno operativo existente
Gerenciamiento del Proyecto		
Relación con Clientes	Clientes en el lugar, focalizados en priorizar requerimientos	Interacción con clientes según se requiera; focalizados en contratos
Planificación y control	Planes internalizados; control cualitativo	Planes documentos; control cuantitativo
Comunicación	Conocimiento tácito e interpersonal	Conocimiento explicito y documentado
Aspectos Técnicos		
Requeri-Mientos	Historias informales y casos de prueba priorizados; con cambios predecibles	Especificaciones formales y completas bajo control de cambio
Desarrollo	Diseño simple; iteraciones cortas; se asume que el costo del cambio es bajo y constante	Arquitectura; iteraciones mayores; se asume que el costo del cambio es creciente
Testing	Casos de prueba ejecutables definen Requerimientos	Plan de procedimientos de prueba
Personal		
Clientes	Dedicados y en el lugar; características CRACK (collaborative,representative, authorized, Committed, knowledgeable)	Características CRACK, y deben tener alta participación en algunos momentos, pero no necesariamente durante toda la duración del proy.
Desarrolla-Dores	Alto Porcentaje de recursos seniors, el resto semi-senior	Alto porcentaje de recursos seniors al inicio, Después los perfiles distribuidos
Cultura	Empowerment a través de autonomía	Empowerment a través de políticas y procedimientos

Tabla 2 : Característica y Territorios, Metodología Ágil y Metodología orientada al Plan.

3.10.2 Factores

Los factores permiten clasificar tomando las restricciones o características impuestas por el problema a resolver y los medios disponibles para hacerlo: consideran al proyecto, al producto, a las organizaciones y personas que ejecutan o están relacionadas y el contexto de negocio (Ver Tabla 2). La definición y cantidad de factores son arbitrarias, y existen otras propuestas. ver por ejemplo (Cockburn, 2000).

Tamaño : Del proyecto. Se mide en numero de personas involucradas. Se busca evaluar con este factor la escala del problema, que afecta en varios sentidos, por ejemplo el nivel de complejidad en la comunicación y los costos de cambio que puedan esperarse. Diferencias de escala provocan que algunas practicas, como la dependencia del conocimiento explicito, no sean aplicables. En este sentido, hay que considerar que el tamaño del producto o la duración del proyecto (medir los meses/hombre) también se deben tomar en cuenta.

Criticidad : Naturaleza del daño ocasionado por defectos no detectado. Un posible clasificación cualitativa es perdida del confort, perdida de dinero discrecional, perdida de dinero fundamental para la empresa, daños a la salud, perdida de vidas.

Personal : Proporción de Personal Senior, Semi-senior y Junior. Este factor no afecta negativamente a los métodos orientados al plan, que pueden ser efectivos tanto con alto como con bajo nivel de seniority.

Cultura : Las organizaciones y las personal intervinientes pueden depender de la confianza, o en la relación contractual. Esto se refleja en

el nivel de ceremonia necesario y aceptado : documentación, control, formalismo en las comunicaciones. Por otro lado, ¿como suele ser la estructura de los grupos de proyectos, jerárquica o grupos de pares?

Factor	Territorio M. Ágiles cuando ...	Territorio M. Orientadas al Plan cuando ...
Tamaño	El producto y el proyectos son pequeños. La Dependencia del conocimiento explicito limita la Escalabilidad	Grupos o productos grandes. Los métodos evolucionaron para solucionar problemas grandes; son difíciles de ajustar a algo pequeño.
Criticidad	No están involucradas vidas o perdida Significativas. Dificultad para asegurar la calidad por el foco en diseños sencillos y poca documentación.	Productos críticos. Los métodos evolucionaron para soportar criticidad, el numero de salvaguardas los hace difíciles de ajustar a casos no críticos.
Dinamismo	Entornos de alto dinamismo: requerimientos o tecnologías. En estos casos, el trabajo es minimizado con diseños simples y refactoreo, poca documentación y otras practicas ágiles.	Entornos relativamente estables. En estos casos, una arquitectura diseñada al inicio, que contemple los cambios previsibles, minimiza el retrabajo.
Personal	El grupo contiene en forma continua un alto porcentaje de Seniors y Semi-Seniors, en todos los roles.	El grupo contiene un alto porcentaje de Juniors, al menos en algunos roles. En esos roles, se requiere mayor participación de Seniors al inicio del proyecto.
Cultura	Organización / Personas acostumbradas a trabajar con bajo nivel de ceremonia. Mayor libertad en cuanto a como lograr los objetivos.	Organización / Personas acostumbradas a procesos y políticas definidas y controlables, roles y tareas claramente definidos.

Tabla 3 : Factores y Territorios, Metodología Ágil y Metodología orientada al Plan.

3.10.3 Relación entre Características, Territorios y Factores

La tabla 3 muestra algunas de las relaciones. Por ejemplo, en un Entorno de negocios cambiantes o en el caso de un producto nuevo, el factor Dinamismo será alto, mientras que en un Entorno operativo correspondiente a un equipo medico de monitoreo de signos vitales la Criticidad es elevada, incluso para una aplicación que en si misma no seria critica. La relación entre los territorios y los factores es compleja e involucra toda la flexibilidad en manejo de riesgos que provee el diseño de la metodología. Por ejemplo, en el caso de un nuevo producto, para disminuir el Dinamismo del proyecto se puede realizar un proyecto previo, mucho mas reducido, que desarrollo un prototipo que llegue a manos de un grupo reducido de usuarios, y permita estabilizar los requerimientos

(Ver Imagen 1). Consideramos que si el problema analizado esta dentro del área punteada interna, estaríamos en territorios ágiles.

Características	Factor Relacionado
Aplicación	
Objetivo Primario	Dinamismo, Criticidad
Tamaño	Tamaño
Entorno	Dinamismo, Criticidad
Gerenciamiento del Proyecto	
Relación con Clientes	Criticidad, Cultura
Planificación y Control	Cultura
Comunicación	Cultura
Aspectos técnicos	
Requerimientos	Cultura ,Criticidad
Desarrollo	Criticidad
Testing	Criticidad
Personal	
Clientes	Personal
Desarrolladores	Personal
Cultura	Cultura

Tabla 4 : Relación Características y Factores.

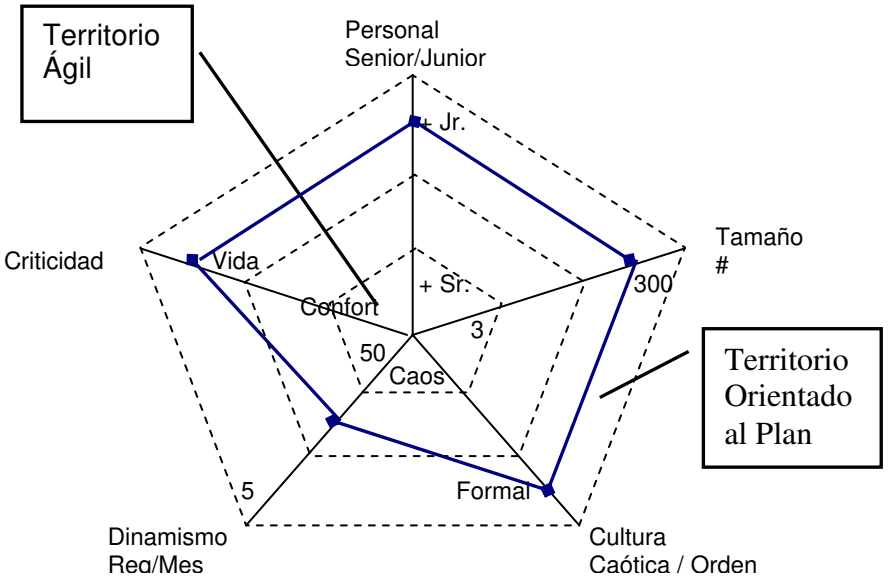

Figura 15: Principalmente Orientado al Plan

Metodologías Ágiles	Metodologías Orientadas al Plan
Basadas en heurísticas provenientes de practicas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia al cambio
Impuestas internamente por el equipo	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho mas controlado, con numerosas políticas y normas
No existe contrato tradicional o al menos es Bastante flexible	Existe un contrato pre fijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Mas artefactos
Pocos roles	Mas roles
Menos énfasis en la arquitectura del software	La arquitectura de software es esencial y se expresa mediante modelos

Tabla 5 : Diferencias entre metodologías Ágiles y Orientadas al Plan.

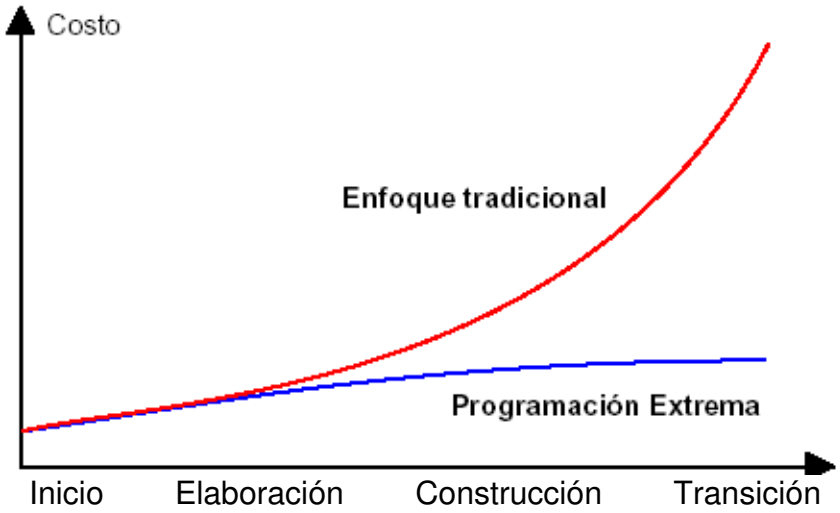

Figura 16: Costo de Introducir Cambios

CAPITULO IV

4. CASO DE ESTUDIO

Este capitulo describe la empresa que se utiliza como caso de estudio. La empresa Seleccionada es Lideres en Servicio SAC., con 5 años de presencia en el mercado. A continuación se describe el caso de estudio, el perfil y el análisis del caso de estudio.

4.1 DESCRIPCION DEL CASO DE ESTUDIO

La empresa Lideres en Servicio S.A. es una empresa dedicada a la comercialización de productos y servicios de Telefónica del Perú (Telefonía Básica, Cable Mágico, Speddy, Telefonía Móvil, Tarjetas de Telefonía así como otros productos) a la cual representa, contando esta con una red de Agencias Multicentros en Lima como en Provincia (Actualmente 11 Agencias en Lima y 3 en Provincia). Esta empresa representa comercialmente a Telefónica del Perú, en calidad de Distribuidor Autorizado, recibe una comisión de ventas por cada producto que vende, el cual esta en función de la forma en que fue realizada la venta (Canal de Venta : Campo, Punto, Call Center, Plataforma, etc). La empresa cuenta con un numero de vendedores

para cada agencia, y estos reciben la comisión de la venta en función del producto que fue vendido, el canal de venta, y el numero de productos vendidos así como un bono si llega a un objetivo definido por la empresa, la cual ha su vez recibió un objetivo de venta de Telefónica del Perú.

Tanto las Reglas del Negocio como en Tecnología se depende directamente de TDP debido que es esta la que crea nuevas campañas de venta, nuevos productos, nuevos planes, modifica comisiones, asigna objetivos de Ventas, etc; en cuanto a tecnología se accede a los sistemas de Telefónica para Registrar los Contratos, Obtener Data, Activar los Equipos Móviles, Acceder a las Bodegas de Equipos (Omega, G-400, Silicom), los cuales queda registrados en la Base de Datos de TDP; a su vez la empresa cuenta con algunas aplicaciones para manejar su información; actualmente se están desarrollando un sistema integrado para la empresa, debido a que fue diseñado para una sola tienda y se ha ido adaptando, pero no cuenta con un análisis y un estructura de datos robusta que soporte todas las particularidades que presenta.

4.1.1 Perfil de la Empresa

Es una empresa privada que desarrolla sus actividades en el rubro de "Telefonía, Telecomunicaciones y Comunicaciones por Internet", dedicándose exclusivamente a comercializar bajo la modalidad de "distribuidor autorizado", todos los productos y servicios del Grupo Telefónica.

4.1.2 Estructura Organizacional de la Empresa en Estudio

La estructura Organizacional es muy dinámica, cambian según el ritmo que lo hace TDP, es decir si TDP crea un Nuevo Negocio de Venta. La Empresa creara también una área nueva, la cual maneje este nuevo negocio.

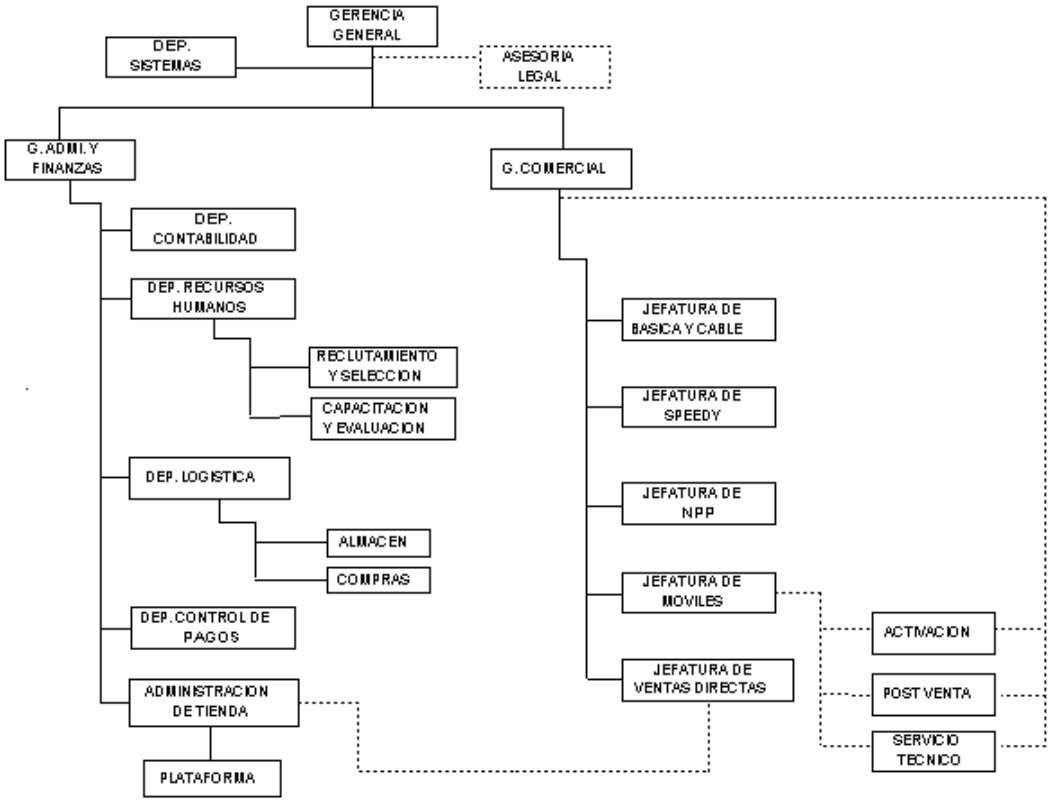

Figura 17: Organigrama de Lideres en Servicio SAC

4.2 Análisis del Caso de Estudio :

Del Análisis realizado a la organización se desprenden las siguientes puntos los cuales nos servirán para elegir una determinada metodología de desarrollo de Software.

- Debido a su calidad de distribuidor autorizado de los productos y servicios de TDP y su dependencia en cuanto a las reglas del negocio que TDP imponga. La empresa tiene la necesidad de responder al cambio en forma continua, así como de obtener resultados en forma

rápida. Por tal motivo nos encontramos en un entorno turbulento de alto cambio enfocado en el proyecto y de resultados inmediatos.

- Existe mucho dinamismo tanto de requerimientos como de tecnología.
- No existen reglas formales tanto de la organización como de las personas, las cuales están acostumbradas a trabajar de manera autónoma para el logro de objetivos.
- Comunicación efectiva, reuniones semanales con el personal y quincenales con los lideres.
- La relación con el cliente es estrecha, encontrándose cerca al Grupo de Desarrollo del Proyecto, llevándose una a cabo continuas reuniones, la cuales están focalizadas a solucionar y priorizar los requerimientos.
- El Grupo del Proyecto esta compuesto por 6 Analistas Programadores y el Proyecto se puede considerar pequeño, ya que estamos frente a una empresa que comercializa productos y/o servicios de terceros.
- El 50% de los Desarrolladores cuentan con mas de 4 años de experiencia, considerándose Senior, y los demás Junior y Semi-Senior.

Según estas observaciones y en base a las Características y Factores definidos en el punto 3.10 del presente trabajo, se realiza el siguiente grafico.

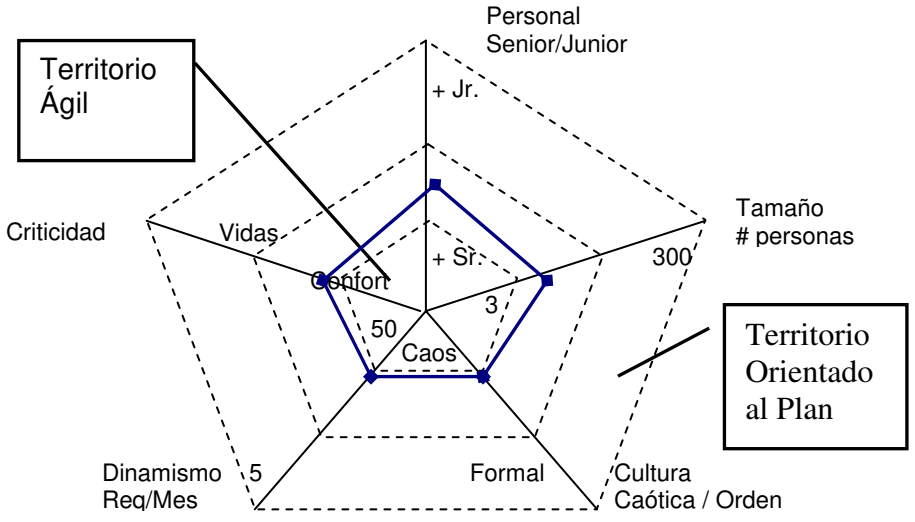

Figura 17: Orientado a la Metodología Ágil

En el cual por las características definidas se observa que se encuentra Orientado a la Metodología de Desarrollo Ágil.

Con los modelos descritos anteriormente podemos identificar si estamos en territorio ágil u orientado al plan, en cuyo caso podemos seleccionar uno u otro enfoque metodológico. Pero, ¿que pasa en el caso que la situación no sea tan clara?. Se propone ejecutar el proyecto con un ciclo de vida espiral controlado por riesgos. El siguiente es el proceso utilizado para definir el detalle de la metodología a utilizar.

Proceso

1. Evaluar los factores para ver en que territorio están: ágil u orientado a plan. Si hay incertidumbre importante, conseguir mas información con prototipos, búsqueda de datos y análisis .
2. ¿Domina alguno de los métodos?, si es así, seguir en 4, utilizando el método dominante: Ágil u Orientado al Plan(Ver Figura 17)
3. Si no domina ninguno de los métodos, diseñar la aplicación (y el proyecto) para encapsular la parte ágil.
4. Establecer una estrategia de proyectos integrando las distintas mitigaciones de riesgos.
5. Monitorear los riesgos (amenazas / oportunidades) y reajustar correspondientemente.

A continuación mostramos las Historias y Tareas para el Caso de estudio, para a 1era Iteración.

Historia de Usuario	
Número:	**Usuario:**
Nombre historia:	
Prioridad en negocio: (Alta / Media / Baja)	**Riesgo en desarrollo:** (Alta / Media / Baja)
Puntos estimados:	**Iteración asignada:**
Programador responsable:	
Descripción:	
Observaciones:	

Historia de Usuario	
Número: 1	**Usuario:** Asistente de Personal
Nombre historia: Registro de Penalidades.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 4	**Iteración asignada: 1**
Programador responsable: Susana Ramos – Iván Sánchez	
Descripción: Registrar las Penalidades de Contratos (Básica, Cable o Speedy) la cual envía Telefónica de forma mensual. Si el contrato fue registrado con anterioridad mostrar los datos respectivos del contrato, en caso no este registrado debe ingresar el área, la agencia, el personal y la promoción para la cual se aplicara la penalidad, para que esta sea considerada en su respectivo pago.	
Observaciones: Penalidad = Monto que telefónica aplica a la empresa, por la baja de un contrato antes de un periodo determinado	

Historia de Usuario	
Número: 2	**Usuario:** Asistente de Personal
Nombre historia: Actualización de Contratos a Pagar (Acreditadas Telefónica)	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 4.5	**Iteración asignada: 1**
Programador responsable: Susana Ramos – Iván Sánchez	
Descripción: Los contratos acreditados por telefónica llega en un Archivo de texto ASCII y es procesado automáticamente siguiendo el formato de la plantilla de contratos a actualizar.	
Observaciones: Acreditadas Telefónica = Son los contratos que Telefónica reconoce como ventas de la empresa, de las cuales pagara su respectiva comisión a la empresa; solo estos deben ser considerados en los pagos de los vendedores.	

Historia de Usuario	
Número: 3	**Usuario:** Asistente de Personal
Nombre historia: Generación del Pago	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 4.5	**Iteración asignada: 1**
Programador responsable: Susana Ramos – Iván Sánchez	
Descripción: La información de los Contratos ingresados a la base de Datos en un determinado periodo (Año - Mes) debe ser procesados para generar el pago respectivo.	
Observaciones:	

Tarea	
Número tarea:	**Número historia:**
Nombre tarea:	
Tipo de tarea : Desarrollo / Corrección / Mejora / Otra (especificar)	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Programador responsable:	
Descripción:	

Tarea	
Número tarea: 1	**Número historia: 1**
Nombre tarea: Comprobación de la base de datos	
Tipo de tarea : Desarrollo	**Puntos estimados: 3**
Fecha inicio: 7 Abril 2006	**Fecha fin: 24 Abril 2006**
Programador responsable: Equipo XP	
Descripción: Comprobación que la definición establecida para la base de datos admite los campos que se leen del Archivo de entrada, realizando las oportunas modificaciones	

Tarea	
Número tarea: 2	**Número historia: 1**
Nombre tarea: Lectura y procesado del Archivo	
Tipo de tarea : Desarrollo	**Puntos estimados: 0.5**
Fecha inicio: 17 Abril 2006	**Fecha fin: 24 Abril 2006**
Programador responsable: Susana Ramos – Iván Sánchez	
Descripción: Se abre el Archivo y se lee línea a línea (lo que corresponde a un registro), en primer lugar se comprueba que tipo de registro es y en función de esto se extraen los datos actualizándolos en las tablas correspondientes de la base de datos.	

Tarea	
Número tarea: 3	**Número historia: 1**
Nombre tarea: Comprobación resultados	
Tipo de tarea : Verificación	**Puntos estimados: 0.5**
Fecha inicio: 24 Abril 2006	**Fecha fin: 24 Abril 2006**
Programador responsable: Susana Ramos – Iván Sánchez	
Descripción: Una vez importados todos los datos extraídos del Archivo se comprueba que se hace de forma correcta y que el proceso no falla debido a alguna restricción no satisfecha. Si el proceso falla en algún pedido, éste no se actualiza en la base de datos y se notifica al usuario.	

CONCLUSIONES

Tanto los metodologías ágiles como los orientados al plan son aplicables eficientemente dentro de sus territorios. Fuera de ellos, se corren riesgos. Varios autores están trabajando en ampliar los territorios tanto en metodologías ágiles como en orientados al plan, manteniéndose dentro de un tipo de metodología. En este sentido, la experiencia indica que es mejor partir de un metodología sencilla, al cual agregarles lo necesario, antes que partir de uno muy completo, al que se debe recortar lo no necesario, ya que esto ultimo pocas veces se hace en la practica.

Presentamos una forma de ampliar los territorios aun mas, para el caso en que el problema tiene factores que impiden considerarlo netamente en territorio ágil o en territorio orientado al plan. En este caso, se selecciona un metodología base y se manejan los riesgos ocasionados por los factores que no están dentro del metodología base.

Los valores de los factores son dependientes del evaluador. Un tema a investigar es la sensibilidad que tiene la metodología ante la obtención de diferentes valores de los factores para un mismo problema. En caso de ser muy sensible, haría necesario explicitar mas la forma de medir factores. Pero puede esperarse que lo importante no sean los valores, sino la comprensión del problema lograda en el proceso de medirlos.

Las metodologías son importantes, pero no deberían hacernos perder la vista que el éxito del proyecto depende mas de la comunicación efectiva con todos los interesados (skateholders), el manejo de las expectativas, el valor generado para el negocio y las personas que participan en el proyecto.

Las organizaciones tienen rangos de aplicabilidad de las metodologías que utilizan; y es mas sencillo trabajar dentro de ese rango de tipos de proyectos. En el caso de proyectos atípicos, debe replantearse las decisiones metodológicas, ya que se corre el riesgo de no responder lo suficientemente rápido y con los apropiados en el caso de un problema en territorio mas ágil que lo acostumbrado, o por lo contrario, intentar escalar formas de comunicación y practicas ágiles mas allá de lo conveniente, en el caso de problemas en territorio mas orientado al plan que lo acostumbrado.

REFERENCIAS BIBLIOGRAFICAS

[1] Anónimo. Metodologías de Desarrollo de Software

[2] CANOS, José y otros “Metodologías Ágiles en el desarrollo de software”. DSIC. Universidad de la Politécnica. http://www.willydev.net/descargas/prev/TodoAgil.Pdf

[3] GARBANZO, Priscilla. “Procesos de desarrollo de software”. Setiembre. 2005. http://www.cit.co.cr/informes/37.html.

[4] LETELIER, Patricio “Introducción al proceso de desarrollo de software ”.

[5] MCCLURE, Carma. “The CASE Experience”. Byte. Abril 1989 p.235.

[6] MENDOZA, María. “Metodologías de desarrollo de software”. Junio 2004 http://www.informatizate.net/articulos/metodologias de_desarrollo_de_software_07062004.html.

[7] PAZ, Diego C. “Evolución de la filosofía del conocimiento en las metodologías de desarrollo de software”. http://creativecommons.org/licenses/by-nc-sa/2.5/ . Febrero, 2006.

[8] REYNOSO, Carlos. “Métodos Heterodoxos en Desarrollo de Software” http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/ heterodox.asp

[9] TERRY,B. “Terminology for Software Engineering and Computer Aided Software Engineering”. Software Engineering Notes. Abril 1990.

[10] TORRES, Araceli. “Metodologías modernas de desarrollo de Sistemas de Información”

[11]ZAVALA, Jesús. “¿Por qué fracasan los proyectos de software?.Un enfoque organizacional”. Congreso Nacional de Software Libre 2004. http://www.consol.org.mx/2004/material/63/por-que-fallan-los-proy-de-soft.pdf

[12] ZAMORA, Hugo. “Modelos de Ciclo de Vida en el desarrollo de Software”. Setiembre 2005. Asociación Colombiana de Ingenieros de Sistemas

[13] CHARETE, Robert. “The decision is in: Agile versus Heavy Methodologies”. Cutter Consortium, Executive Update, 2(19), http://www.cutter.com/freestuff/apmupdate.html, 2004.