Diciembre
2014
Edición Nº 8

pág
37 - 45

# BUENAS PRÁCTICAS, UNA SOLUCIÓN PARA UN MEJOR DESARROLLO DE SOFTWARE

# GOOD PRACTICE, A SOLUTION FOR BETTER SOFTWARE DEVELOPMENT

*Liliana del Carmen Ramírez M[1], Anderson Smith Flórez Fuentes[2]*

## RESUMEN

Este articulo tiene como objetivo principal dar a conocer una serie de buenas prácticas planteadas para un mejor desarrollo de software, basadas en las metodologías agiles más representativas se han propuesto unos valores, unas características, unos roles, unas prácticas y un ciclo de vida para el desarrollo de un proyecto de software, con lo cual se busca abarcar todo el proceso desde que surge una necesidad por parte de un cliente, hasta el momento en se tiene la seguridad de que el cliente está totalmente satisfecho con el producto. Estas prácticas planteadas buscan comprometer al equipo de trabajo e integrarlo al proyecto de una manera beneficiosa para las personas participantes, ya que pueden estar desde el inicio hasta la fase final del proyecto cumpliendo un rol u otro específicamente, logrando con esto reducción de tiempo, costos y otros.

**PALABRAS CLAVE**: Agil, caracteristicas, desarrollo, metodologias, roles, software

## ABSTRACT

This article's main objective is to provide a set of best practices suggested for better software development, based on the most representative agile methodologies have been proposed values, characteristics, certain roles, practices and life cycle for the development of a software project, which seeks to cover the entire process from which arises a need for a customer, until such time as it is certain that the client is fully satisfied with the product. These practices engage the raised look team and integrate the project in a manner beneficial to the participants, as they may be from the beginning to the final phase of the project fulfilling a role or another specifically, achieving this reduction of time, costs and others.

**KEYWORDS:** Agile, characteristics, Development, Methodologies, Roles, Software.

*1. Universidad de Pamplona, Facultad de Ingenierías y Arquitecturas, Ingeniería de Sistemas (Villa del Rosario), Email: lilianarm2906@gmail.com*
*2. Universidad de Pamplona, Facultad de Ingenierías y Arquitecturas, Grupo de Investigación Ciencias Computacionales (CICOM), Ingeniería de Sistemas (Villa del Rosario), Email: andersonflorezf@unipamplona.edu.co*

# BUENAS PRÁCTICAS, UNA SOLUCIÓN PARA UN MEJOR DESARROLLO DE SOFTWARE

## 1. INTRODUCCIÓN

En la actualidad existen muchas metodologías ágiles de desarrollo de software, las cuales tienen 4 aspectos en común:

- Los individuos y sus interacciones, sobre los procesos y las herramientas.
- El software que funciona, más que la documentación exhaustiva.
- La colaboración con el cliente y no tanto la negociación del contrato.
- Responder al cambio, mejor que apegarse a un plan.

Estos son los 4 valores del manifiesto ágil que debe seguir una metodología para poder ser denominada metodología ágil [1].

Cuando hablamos metodología ágil se puede decir que es una combinación de una filosofía con un conjunto de lineamientos de desarrollo. La filosofía pone el énfasis en: la satisfacción del cliente y en la entrega rápida de software incremental, los equipos pequeños y muy motivados para efectuar el proyecto, los métodos informales, los productos de trabajo con mínima ingeniería de software y la sencillez general en el desarrollo.

Los lineamientos de desarrollo enfatizan la entrega sobre el análisis y el diseño y la comunicación activa y continua entre desarrolladores y clientes [2]. En este artículo se proponen una serie de estas buenas prácticas que siguen estos lineamientos de las metodologías agiles, para así, finalmente lograr tener un mejor desempeño de un equipo en el desarrollo de un producto de software.

## 2. METODOLOGÍAS

Los métodos ágiles universalmente dependen de un enfoque iterativo para la especificación, desarrollo y entrega del software. Principalmente fueron diseñados para apoyar el desarrollo de aplicaciones de negocio donde los requerimientos del sistema normalmente cambiaban rápidamente durante el proceso de desarrollo.

Están pensados para entregar software funcional de manera rápida a los clientes quienes pueden entonces proponer que se incluyan en iteraciones posteriores del sistema nuevos requerimientos o cambios en los mismos [3]. A continuación una breve descripción de las metodologías estudiadas:

### 2.1 Metodología XP

XP 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 [4].

#### 2.1.1 proceso XP

El ciclo de desarrollo consta de los siguientes pasos:

a) El cliente define el valor de negocio a implementar.
b) El programador estima el esfuerzo necesario para su implementación.
c) El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
d) El programador construye ese valor de negocio.
e) Vuelve al paso 1.

#### 2.1.2 prácticas XP

Las prácticas seguidas en la metodología XP son las siguientes:

- La planificación del juego.
- Versiones pequeñas.
- Diseño simple.
- Pruebas continuas.
- Refactorización.
- Programación por parejas.
- Posesión colectiva del código.
- Integración continúa
- Semana laboral de 40 horas.
- Cliente en el sitio.
- Estándares de codificación.

## 2.2 Metodología SCRUM

Esta metodología centra su atención en las actividades de Gerencia y no especifica prácticas de Ingeniería. Fomenta el surgimiento de equipos autos dirigidos cooperativos y aplica inspecciones frecuentes como mecanismo de control; Scrum parte de la base de que los procesos definidos funcionan bien sólo si las entradas están perfectamente definidas y el ruido, ambigüedad o cambio es muy pequeño. Por lo tanto, resulta ideal para proyectos con requerimientos inestables, ya que fomenta el surgimiento de los mismos [5].

### 2.1.1 Características de SCRUM

- Equipos autodirigidos.
- Colaboración estrecha con el cliente.
- Predisposición y respuesta al cambio.
- Prefiere el conocimiento tácito de las personas al explícito de los procesos.
- Desarrollo incremental con entregas funcionales frecuentes.
- Comunicación verbal directa entre los implicados en el proyecto.
- Motivación y responsabilidad de los equipos por la auto-gestión, auto-organización y compromiso.
- Simplicidad. Supresión de artefactos innecesarios en la gestión del proyecto [6].

### 2.2.2. Principales elementos de la metodología SCRUM

- Herramientas
- Product Backlog.
- Sprint Backlog.
- Prácticas
- Sprints.
- Sprint planning meeting (reunión de planificación del sprint).
- Daily meetings (reuniones diarias).
- Sprint review meeting (reunión de revisión del sprint.)
- Design review meeting (diseño de reunión de revisión).
- Stabilization sprints.
- Meta scrums.

## 2.3 Metodología Crystal Clear

Esta metodología maneja iteraciones cortas con feedback frecuente por parte de los usuarios/clientes, minimizando de esta forma la necesidad de productos intermedios. Otra de las cuestiones planteadas es la necesidad de disponer de un usuario real aunque sea de forma part time para realizar validaciones sobre la Interface del Usuario y para participar en la definición de los requerimientos funcionales y no funcionales del software [7].

### 2.3.1 Características

- Hace menos hincapié en la documentación que otras metodologías más tradicionales.
- Persigue el desarrollo de versiones del proyecto que puedan ser probadas.
- Basada en SCRUM.
- Énfasis especial en pruebas y correcciones al final de cada iteración.

### 2.3.2 Valores de Crystal Clear

- Entrega frecuente.
- Comunicación osmótica.
- Mejora reflexiva.
- Seguridad personal.
- Foco.
- Fácil acceso a usuarios expertos.
- Ambiente técnico con prueba automatizada, management de configuración e integración frecuente.

## 2.3 Metodología DSDM (Método de Desarrollo de Sistema Dinámico)

La metodología DSDM es un método que utiliza una serie de conceptos, prácticas y criterios para el desarrollo ágil de software. Tiene como objetivo desa-

rrollar un sistema con las necesidades de la empresa tanto en tiempo como en presupuesto, por lo que es un método ideal para proyectos de sistemas de información con restricciones temporales o requerimientos cambiantes [8].

La idea fundamental de DSDM se basa en que fija primero el tiempo y el coste, fija primero el tiempo y el coste y con esto fijado, determina las funcionalidades que se pueden implementar en el producto, en vez de fijar las funcionalidades de un producto primero y después el tiempo y el coste [9].

### 2.3.1 Características

- El desarrollo es iterativo e incremental.
- Cambios reversibles.
- El equipo toma decisiones sin autorización de superiores.
- Se realizan entregas del producto frecuentemente.
- Pruebas de calidad a lo largo del proceso de desarrollo integradas en el ciclo de vida.
- Cooperación entre los desarrolladores, usuarios y Stakeholders.

### 2.3.2 Fases de DSDM

- Fase 1 el pre-proyecto.
- Fase 2 el ciclo vital del proyecto.

Etapas en el ciclo vital del proyecto DSDM: Estudio de viabilidad, Estudio de la empresa, Iteración del modelo funcional, Diseño e iteración de la estructura, Implementación, Fase 3 el post-proyecto.

## 2.4 Metodología ASD

Desarrollo Adaptable de Software – ASD James Highsmith, consultor de Cutter Consortium, desarrolló ASD hacia el año 2000 con la intención primaria de ofrecer una alternativa a la idea de que la optimización es la única solución para problemas de complejidad creciente [10].
ASD incorpora el principio de la adaptación continua, que el proceso de adaptación al trabajo en cuestión es el estado normal de cosas. Es decir que su principio es adaptarse al cambio en lugar de luchar contra él [11].

### 2.4.1 Características de ASD

Sus principales características son:

- Iterativo: Ya que la base de esta metodología se enfoca en un ciclo de vida.
- Orientado en los componentes del software: Definiendo este como un grupo de funcionalidades a ser desarrolladas durante un ciclo de vida.
- Tolerante a cambios: Fácil adaptación a cambios repentinos de requerimientos, ya sean estos implantados por el usuario o el análisis del grupo de trabajo.
- Guiado por los riesgos.
- La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.

### 2.4.2 Fases en ASD

Las fases en las que se trabaja la metodología ASD son tres:

- Especular.
- Colaborar.
- Aprender.

Esta fase es la última en cada ciclo y consiste en la revisión de calidad donde se analizan 4 categorías:

- Calidad del resultado de la desde la perspectiva del cliente.
- Calidad del resultado de la desde la perspectiva técnica.
- Funcionamiento del equipo de desarrollo y las prácticas que este utiliza Status del proyecto.

## 2.5 Metodología FDD (Desarrollo Basado en Funcionalidades)

FDD es un enfoque á gil para el desarrollo de sistemas. Fue desarrollado por Jeff De Luca y Peter Coad. Como las otras metodologías adaptables, se enfoca en iteraciones cortas que entregan funcionalidad tangible Dicho enfoque no hace énfasis en la obtención de los requerimientos sino en cómo se realizan las fases de diseño y construcción. Sin embargo, fue diseñado para trabajar con otras actividades de desarrollo de software y no requiere la utilización de ningún modelo de proceso específico. Además, hace

énfasis en aspectos de calidad durante todo el proceso e incluye un monitoreo permanente del avance del proyecto. Al contrario de otras metodologías, FDD afirma ser conveniente para el desarrollo de sistemas críticos [12].

### 2.6 Metodología TDD (Desarrollo orientado a las pruebas)

TDD es un estilo de desarrollo donde se mantiene un juego de pruebas del programador exhaustivo, ninguna parte del código pasa a producción a no ser que pase sus juegos asociados, escribir primero las pruebas y las pruebas determinan el código que se necesita escribir [13].

## 3. PRÁCTICAS PROPUESTAS

Estas prácticas se han propuesto con el fin de mejorar el rendimiento a la hora de desarrollar un producto de software, dando un poco de libertad a los integrantes del equipo en cuanto a reglas de horario y sitio de trabajo. La siguiente es la descripción de cada uno de los componentes de estas buenas prácticas.

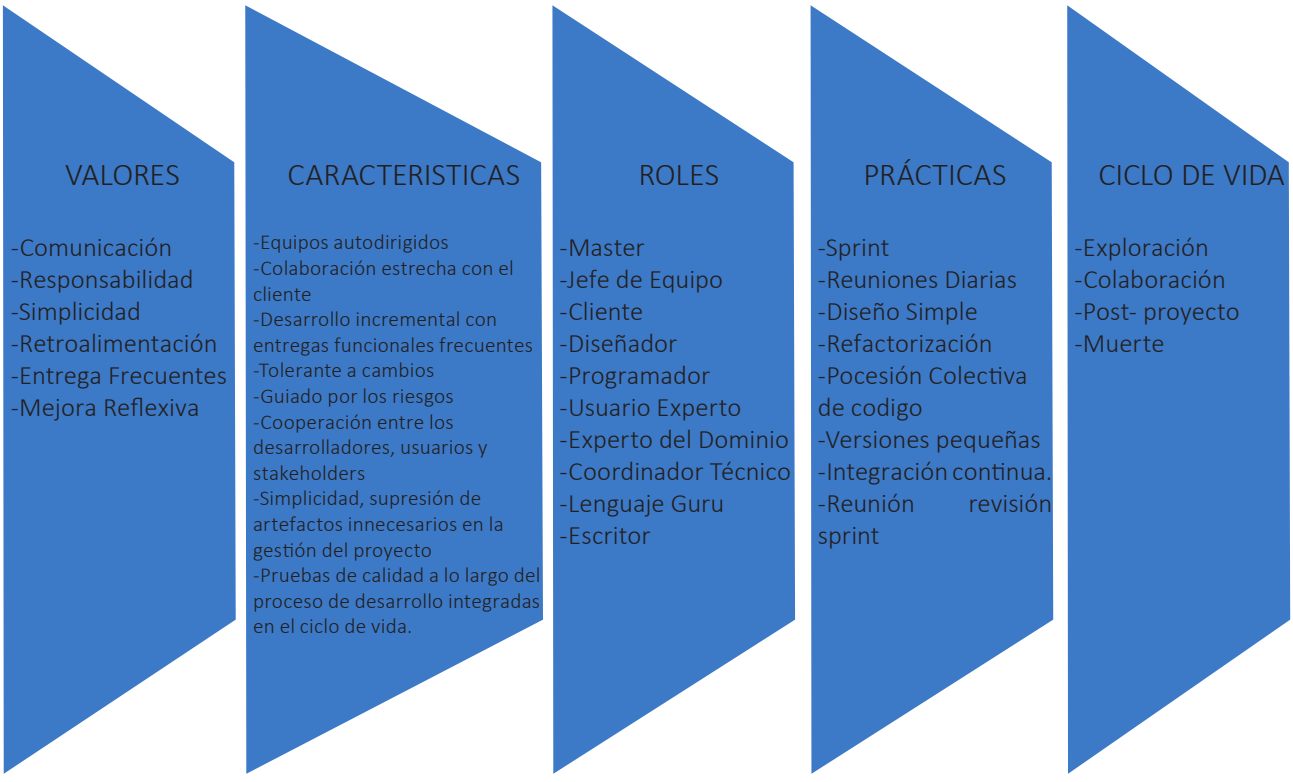
### 3.1 Valores

- Comunicación
- Responsabilidad
- Simplicidad
- Retroalimentación
- Entrega frecuente
- Mejora reflexiva

### 3.2 Características

- Equipos autodirigidos.
- Colaboración estrecha con el cliente.
- Desarrollo incremental con entregas funcionales frecuentes.
- Tolerante a cambios.
- Guiado por los riesgos.
- Cooperación entre los desarrolladores, usuarios y Stakeholders.
- Simplicidad. Supresión de artefactos innecesarios en la gestión del proyecto.
- Pruebas de calidad a lo largo del proceso de desarrollo integradas en el ciclo de vida.

*Figura 1: Prácticas propuestas*



| ROL | DESCRIPCION |
|---|---|
| Master | Tiene la habilidad de unir ideas, personas y recursos. Así como tener facilidad para la toma de decisiones, además representa la cara visible del proyecto. |
| Jefe del equipo | Permite reforzar la adherencia al proceso de desarrollo por parte de los integrantes del equipo y tiene la responsabilidad de entregar el proyecto en las fechas estipuladas. |
| Cliente | Es la persona o entidad que tiene la necesidad del producto. Además crea una estrecha relación con el equipo por la razón de retroalimentar el producto en desarrollo. |
| Diseñador, programador | Posee un amplio conocimiento de las herramientas de desarrollo, del lenguaje de programación, de los aspectos técnicos involucrados y demás. |
| Expertos del dominio | Permite al equipo aprender sobre el negocio para el cual se construye el software, se encarga de resolver cualquier evento relacionado con el software o con la funcionalidad que se esté desarrollando en ese momento. |
| Coordinador técnico | Es el experto en la parte técnica del desarrollo y debe mantener a todo el equipo en conocimiento de los lineamientos fundamentales de la construcción, tiene en cuenta no solo los requerimientos funcionales, también los no funcionales. |
| Tester | Tiene las prácticas y el conocimiento para garantizar la calidad en el producto desde el punto de vista técnico. |
| Language GURU | Aporta conocimientos al equipo, lo cual genera en los involucrados un aumento de nivel en cuanto a conocimientos, reflejándose así en un buen desarrollo. |
| Escritor | Produce el manual de usuario |

3.4 Prácticas

• Sprints: Son ciclos iterativos en los cuales se desarrolla o mejora una funcionalidad para producir nuevos incrementos. Durante un Sprint el producto es diseñado, codificado y probado.

• Reuniones diarias: Reunión no mayor a 15 minutos, en la que se presenta que hizo cada miembro del equipo el día anterior, que va a hacer el día de hoy y que problemas se han encontrado.

• Diseño simple: El sistema se diseña con la máxima simplicidad posible.

• Refactorización: Es posible reestructurar el sistema sin cambiar su comportamiento

• Posesión colectiva del código: Todos los programadores pueden cambiar cualquier parte del sistema en determinado momento, siempre se utilizan estándares.

• Versiones pequeñas: Periódicamente, se producen nuevas versiones agregando en cada iteración aquellas funciones consideradas valiosas para el cliente.

• Integración continua: Los cambios se integran en el código base varias veces por día.

• Reunión de revisión del sprint: Se realiza al final del Sprint. Se indica qué ha podido completarse y qué no, presentando el trabajo realizado a los implicados.

## 3.5 Ciclo de Vida

Como se observa en la figura se ha planteado un ciclo de vida que consta de 4 fases la cuales serán descritas más adelante:

- Exploración
- Colaboración
- Post-proyecto
- Muerte

### 3.5.1. Fase I: Exploración: Roles que intervienen

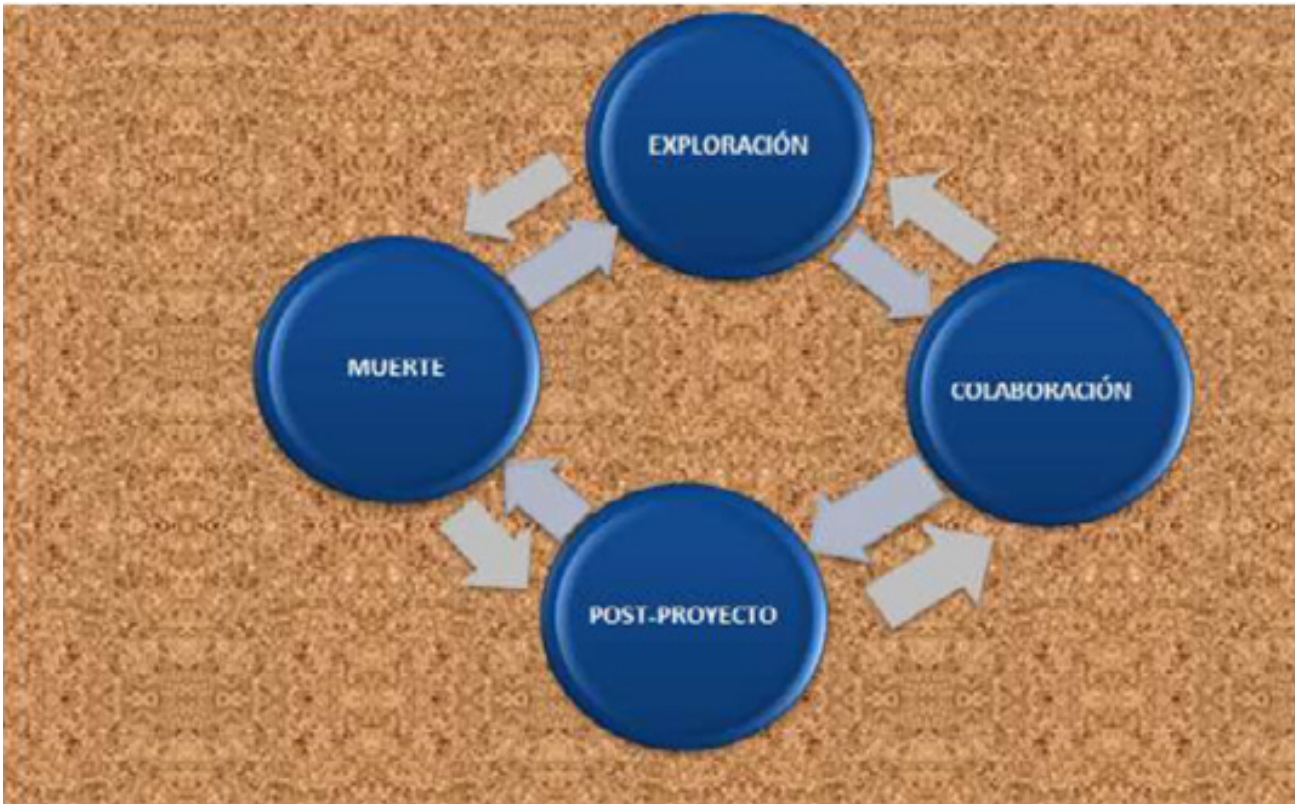
- El cliente
- El master
- El jefe del equipo
- El diseñador programador
- El experto del dominio
- El coordinador técnico

Descripción de la fase:

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

La fase inicial es la más importante en cualquier proyecto, ya que en ella debe quedar claro la idea de lo que se quiere realizar, también es importante la familiarización del equipo entre ellos y con las herramientas a utilizar, metodología a seguir, su entorno

*Figura 2. Fases de la Metodología*



de trabajo para con ello lograr tener una integración de todas las partes involucradas de una manera continua, lo que le es provechoso al desarrollo del proyecto, ya que entre más rápido se realice este proceso más fácil será el desarrollo del producto.

3.5.2. Fase II: Colaboración

Roles que intervienen

- El cliente
- El master
- El jefe del equipo
- El diseñador programador
- Usuario experto
- El experto del dominio
- El coordinador técnico
- Tester
- Language guru

Descripción de la fase:

En esta Fase se construirá la funcionalidad del proyecto. Durante cada Iteración el equipo se preocupa de colaborar fuertemente para que de esta manera se logre liberar la funcionalidad planificada. En esta parte también es posible explorar nuevas alternativas pudiendo alterar fuertemente el rumbo del proyecto.

La fase de colaboración es el punto donde se realiza todo el proceso funcional del proyecto o dicho de otra manera es el punto vital del ciclo de vida del software, ya que en ella se integran el 99% de los roles existentes para lograr un objetivo común que es el producto funcionando en su totalidad. Juega aquí un papel muy importante la relación y comunicación del equipo de trabajo para tener una claridad sobre lo que se está desarrollando, realimentación por parte del cliente y demás compañeros del grupo para superar los puntos de embotellamiento que se dan en el proceso de desarrollo del software.

3.5.3. Fase III: el post-proyecto

Roles que intervienen

- El cliente
- El master
- El jefe del equipo
- El diseñador programador
- Usuario experto
- El experto del dominio
- El coordinador técnico
- Tester
- Language guru

Descripción de la fase:

La fase posterior al proyecto asegura que el sistema funcione de manera eficaz y eficiente. Esto se realiza por el mantenimiento, mejoras y correcciones; El mantenimiento puede ser visto como desarrollo continuo basado en el carácter iterativo e incremental. En lugar de terminar el proyecto en un ciclo, por lo general el proyecto puede volver a las fases o etapas previas para que el paso anterior y los productos a entregar se puedan refinar.

Esta fase es muy importante ya que en ella se realiza una revisión general y detallada del producto de software a entregar, y al igual que en la anterior fase intervienen casi todos los roles a excepción del escritor, se empieza con analizar si se cumplieron todos los requerimientos del cliente, se tienen en cuenta también los requerimientos no funcionales, se realizan las respectivas correcciones en caso de que se detecte algún error y si se pueden hacer mejoras al producto se realizan sin afectar su buen funcionamiento.

3.5.4. Fase IV: Muerte

Roles que intervienen

- El cliente
- El master
- El jefe del equipo
- El diseñador programador
- Escritor

Descripción de la fase:

Ocurre cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura.

La muerte del proceso es el final del ciclo de vida, en este punto se tiene un producto de software que cumple a cabalidad con los requerimientos hechos por el cliente. Se tienen en cuenta los requerimientos no funcionales para lograr una satisfacción completa del usuario final. Se realiza el respectivo manual de usuario para explicar al usuario el manejo y procedimientos a seguir en la utilización del software.

## 4. CONCLUSIONES

Las metodologías ágiles para el desarrollo de software son unas de las mejores maneras de trabajar en el desarrollo de un producto de software actualmente. Se puede escoger entre las diferentes metodologías ágiles que existen la que más se adapte al proyecto que se quiere desarrollar, y según la metodología seguir sus lineamientos y prácticas para lograr un mejor desarrollo de software.

A pesar de las muchas metodologías existentes en algunos casos nos encontramos en que ninguna de ellas se adapta al tipo de proyecto que se va a desarrollar, por lo que se hace necesario investigar las necesidades del proyecto ejecutando una serie de prácticas tomadas de las metodologías ágiles que se quieren seguir pero que de un modo u otro no se adaptan al proyecto a desarrollar. Estas prácticas planteadas se pueden utilizar tanto en proyectos pequeños, como en grandes proyectos de desarrollo de software, porque siguen un lineamiento básico de todas las metodologías agiles para el desarrollo de software.

Estas prácticas se encuentran estructuradas a partir de una planificación inicial de lo que se quiere hacer, siguiendo un proceso de ciclo de vida en el cual la realimentación es un punto muy importante hasta llegar a la culminación total del proyecto.

## 5. BIBLIOGRAFÍA


[1]. Fowler, Martin and Highsmith, Jim. The Agile Manifesto. (August 2001) (http://www.agilemanifesto.org ).

[2]. Pressman, Roger S. Ingeniería del Software Un Enfoque Práctico. Séptima edición. Mc Graw Hill.

[3]. Ian Sommerville. Ingeniería De Software, Séptima Edición. Pearson.

[4]. Beck, K. “Extreme Programming Explained. Embrace Change”, Pearson Education, 1999. Traducido al español como: “Una explicación de la programación extrema. Aceptar el cambio”, Addison Wesley, 2000.

[5]. Peralta, Adriana. Universidad ORT Uruguay. (2003).

[6] Agile Software Development with Scrum - Pearson International Edition

[7]. Amaro Calderón, Sarah Dámaris. Valverde Rebaza. Jorge Carlos. Metodologías Ágiles. Universidad Nacional de Trujillo. Perú (2007).

[8]http://rasmultimedia.blogspot.com/2012/03/metodologias-dsdm-dynamic-systems.html, (Consultado 18 Noviembre 2012)

[9]. Lic. Gimson, Loraine. Metodologías Ágiles y Desarrollo Basado en Conocimiento. Universidad Nacional de la Plata. (2012).

[10]. Heterodoxia. Página de Microsoft: MSDN en Español, Métodos Heterodoxos en Desarrollo de Software, Artículo de Carlos Reynoso – Universidad de Buenos Aires, Abril del 2004. Disponible en: http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/heterodox.asp#12

[11]. Dr. Cerpa, Narciso. Ingeniería de Software Informe de Metodología. Universidad De Talca. (2007).

[12]. Collorana, Jorge Rafael. FDD (Feature Driven Development). Universidad Unión Bolivariana. La Paz-Bolivia (2009).

[13]. David Astels. Test-Driven Development. A practical guide. Prentice Hall PTR 2003.



Diciembre 2014
Edición Nº 8

pág 32 - 45