# METODOLOGÍAS ÁGILES, UNA ALTERNATIVA PARA EL DESARROLLO DE SOFTWARE


Monografía presentada como requisito parcial para optar el título de Ingenieros en Electrónica y Telecomunicaciones

**PAOLA ANDREA MANQUILLO MANQUILLO**
**JESUS EMILIO RIVERA DAZA**

Director
Ing. Carlos Alberto Ardila Albarracín

**UNIVERSIDAD DEL CAUCA**
**FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES**
**Departamento de Sistemas**
**Línea de Investigación en Ingeniería de Software**
**POPAYÁN**
**2006**

# CONTENIDO

# INDICE DE TABLAS Y GRÁFICOS

## Índice de tablas



## Índice de gráficos

# 1. INTRODUCCIÓN

La industria de software colombiana atraviesa un momento importante en su historia y es por eso que los diferentes actores que conforman el escenario de esta industria no deben ser ajenos a su realidad. La universidad, como agente creador y transformador del conocimiento, debe cumplir un papel encaminado a fortalecer dicha industria brindando fundamentos teóricos y prácticos de la ingeniería en que se basa, la ingeniería de software.

Uno de los conceptos fundamentales es el de proceso de desarrollo, pues involucra las diferentes áreas de la ingeniería de software y es el elemento guía para la gestión de los proyectos de software. Sin embargo, a dicho proceso de desarrollo no se le ha dado la importancia que se merece. Teniendo en cuanta el valor del proceso para asegurar la competitividad de la industria del software, se ha trabajado en el área de tecnologías de procesos.

En la búsqueda de aproximaciones propias a los procesos de desarrollo de software caracteristicos de la región, nace este proyecto, el cual fue dividido en las fases que mencionamos a continuación: la primera de ellas fue la exploración tecnológica, donde se recopiló y asimiló información acerca de las metodologías a utilizar. La segunda fue la definición del marco del proceso de desarrollo de la aplicación. Dicho proceso estuvo guiado por prácticas y principios de programación extrema, complementándolos en distintas fases del ciclo de vida con modelamiento ágil. Para enriquecer este proceso, durante el mismo se consignaron las conclusiones relevantes obtenidas durante el desarrollo. La tercera etapa fue el desarrollo del aplicativo, donde se empleó el modelo propuesto en la etapa anterior. La cuarta fase corresponde a las conclusiones ya que, una vez realizado el aplicativo, se analizaron las experiencias y resultados obtenidos a partir de la utilización del proceso y se generaron una serie de recomendaciones y conclusiones que pueden ser aprovechadas en proyectos similares.

## 1.1. Antecedentes

A continuación se describen los hechos que sirven para comprender y valorar la situación existente en el desarrollo de software a nivel mundial, nacional y regional, de donde se desprende la pertinencia de explorar nuevas alternativas de desarrollo encaminadas a aumentar la competitividad de la industria de software regional y nacional.

En el contexto mundial se observan países que han tenido un gran éxito en la venta de productos y servicios de software como es el caso de Irlanda, India, Israel e incluso algunos de América Latina como México, los cuales se han posicionado como jugadores de importancia en el negocio de la informática. Colombia, a pesar de sus esfuerzos, no ha tenido el éxito esperado. Según los más importantes analistas extranjeros del mercado de informática y, de acuerdo con las opiniones de los ejecutivos de las principales empresas de equipos y software, se avecina un enorme crecimiento para la contratación de software en el extranjero por parte de compañías de países industrializados. Algunas predicciones [4] indican que, para el año 2015, cerca de 3.3 millones de puestos de trabajo en el sector informático se desplazarán de Estados Unidos a otros países que cuenten con compañías de calidad en el área de software.

Colombia está ingresando un poco tarde a este mercado, puesto que algunos países cuentan con corporaciones integradas por un número considerable de ingenieros y empleados que trabajan con éxito en la industria del software. El Reporte de NASSCOM-McKinsey del 12 de diciembre de 2005 [5] muestra que el mercado global de software alcanzará los 110 mil millones de dólares, de los cuales la India tiene capacidad para capturar más del 50% de ese potencial, con un crecimiento anual de 25% hasta el año 2010.

En el panorama nacional se encuentra que Colombia cuenta con unas 850 empresas de software tributando ante la Dirección de Impuestos y Aduanas Nacionales (DIAN), de las cuales aproximadamente 650 corresponden a personas jurídicas y 200 a empresas unipersonales [1]. Se debe considerar también que, en el ámbito de desarrollo de software en Colombia, las empresas son predominantemente medianas, pequeñas y microempresas (MiPyMEs); de las 850 empresas que se estiman registradas, encontramos que éstas se clasifican en: 3.75% medianas y 96.25% MiPyMEs [2], por lo tanto, éste tipo de empresas cuentan con muy pocos profesionales. Además, los procesos de desarrollo de software

tradicionales como el Proceso Unificado, espiral, o modelo en cascada, no aportan soluciones prácticas a muchas de las necesidades actuales como la agilidad, la escalabilidad o el enfrentar la naturaleza cambiante de los requisitos, entre otras.

El gobierno nacional ha reconocido la importancia del sector informático como un generador de conocimiento local y como una alternativa significativa para producir empleo de calidad. Igualmente las entidades que promueven las exportaciones colombianas han mostrado interés por convertir la exportación de productos y servicios informáticos en un área prioritaria.

En el contexto regional, el sector productivo de software se enfrenta a dificultades como la utilización de procesos de desarrollo de software diseñados para otros países con mayor grado de desarrollo (como puede notarse a partir de las encuestas, ver anexo 3) y la ausencia de método en la construcción de software, lo cual da como resultado un considerable número de proyectos cancelados o no terminados.

Las firmas colombianas tienen un notable potencial, preparado por excelentes Universidades y centros de entrenamiento de categoría mundial, sin embargo, la gran mayoría de las firmas colombianas son pequeñas en términos internacionales, y el mercado colombiano es insuficiente para garantizar su crecimiento y progreso continuo [4].

## 1.2. Planteamiento del problema

El proyecto busca una primera aproximación a la pregunta de investigación: ¿Cómo motivar a las empresas de la región a encaminarse hacia la definición y adecuación de sus procesos, a través de mecanismos sencillos, livianos, implementables y acordes a las necesidades de la industria regional?

Con el fin de conocer la situación actual de la industria de software regional, el programa de Ingeniería de Sistemas de la Universidad del Cauca realizó una encuesta a veinte empresas distribuidas en Cali, Popayán y Pasto. El análisis de estos datos revela información importante (el resumen de estas encuestas se muestra en el anexo 3).

Las empresas de desarrollo de software de la región cuentan, en su mayoría, con muy pocos empleados, pues de las veinte empresas encuestadas catorce cuentan con máximo diez empleados, y a su vez son éstas pequeñas empresas en las que se encuentra el mayor número de proyectos cancelados, proyectos que, por algún motivo, no se terminaron. Todo lo anterior muestra que es pertinente, desde la academia, proponer alternativas de solución y emprender estrategias que contribuyan a aumentar la eficiencia y competitividad de estas empresas.

De las encuestas se deducen dos factores que afectan la eficiencia y la productividad de las empresas más pequeñas, o sea las que cuentan con cuatro, cinco, seis y hasta ocho empleados: el primero es que algunas de esas empresas no han adoptado método alguno de desarrollo, es también en estas empresas donde se presentan unas ventas muy bajas y gran cantidad de proyectos cancelados, en otras palabras, la ausencia de método no es la manera de ser exitoso en la producción de software. El segundo factor que afecta la productividad es que empresas de poco personal (de hasta ocho empleados) emplean métodos de desarrollo de software como cascada y espiral, y presentan también gran cantidad de proyectos cancelados. Lo anteriormente expuesto muestra que es oportuno explorar métodos alternos de desarrollo, adaptados a entornos donde se cuenta con poco personal. Esos métodos a explorar deben tener presente la realidad en que serán aplicados: el tamaño de la organización, talento humano y expectativas.

En muchos casos (especialmente en las empresas más pequeñas), enfocar la mejora sólo en las áreas más relevantes del proceso con prácticas de algunas metodologías ágiles tales como programación extrema, puede aportar niveles de mejoría adecuados a sus estrategias de desarrollo.

Para intentar resolver este problema de investigación es posible que existan alternativas diversas pero, dada la situación actual de la industria informática colombiana que muestra un sector predominantemente PYME (cuentan con poco personal y bajo presupuesto), la solución más adecuada se encuentra en las nacientes metodologías ágiles. Éstas, además de involucrar poco personal, presentan la ventaja de incluir prácticas que reducen el tiempo de entrega y además son ideales para procesos en los que los requerimientos cambian, dichos cambios estén previstos o no, ocurrirán, y lo mejor es estar preparados para cuando ocurran. Por ello, en el interior del departamento de sistemas, se planteó el presente

proyecto, enmarcado dentro del proyecto "Sistema Integral de Mejoramiento de Procesos de Software" (SIMEP-SW), el cual estuvo guiado por prácticas y principios de programación extrema alimentada con modelamiento ágil. Ambas metodologías pertenecen a las llamadas "metodologías ágiles".

## 1.3. Justificación

Cada día es más frecuente la aplicación de procesos y soluciones software en todos los escenarios en que el hombre se desenvuelve, lo cual contribuye a que la complejidad de los problemas que el software debe enfrentar esté creciendo más rápidamente que la habilidad de la humanidad para resolverlos.

En la ingeniería de software existen dos corrientes para guiar los procesos, las cuales intentan satisfacer los cada vez más complejos requisitos del mundo de hoy: una de estas corrientes la conforman los modelos clásicos de desarrollo de software como son el modelo en cascada, desarrollo rápido (RAD), modelo en espiral, entre otros. Dichos modelos han sostenido que la inversión realizada en las etapas de planeación, análisis y diseño se compensa con los posteriores beneficios que ésta genera. Estos métodos tradicionales basados en estrictos esquemas de diseño y modelado, son dirigidos a proyectos de cierta envergadura en los que intervienen grandes grupos de trabajo y que, por tanto, requieren cuantiosos recursos. La segunda corriente la conforman las nacientes metodologías ágiles, que son más indicadas para organizaciones que cuentan con poco personal y que tienen exigencias de entrega de resultados tempranos y, como es sabido a partir de las encuestas realizadas, la situación por la que atraviesa la industria regional exige adoptar medidas que contribuyan a su fortalecimiento como reducir los tiempos de entrega, de tal modo que se logren disminuir los proyectos cancelados y no terminados.

Como es sabido, dentro de la industria de software no se puede ignorar la importancia que tiene CMMI[1], por eso al interior del Proyecto SIMEP-SW se ha trabajado en examinar la

[1] CMMI - Capability Maturity Model Integrated CMMI. Método de definir y gestionar los procesos a realizar por una organización. Fue desarrollado inicialmente para los procesos relativos al software por la Universidad Carnegie-Mellon para el SEI (Software Engineering Institute).

posibilidad de que empresas desarrolladoras de software puedan certificar sus empresas con CMMI empleando prácticas ágiles como lo ilustra en el artículo "alcanzando CMMI a través de métodos ágiles" [10], donde se indica la relación de implementación de los requisitos de CMMI con los activos de proceso ágiles. Con esto, los autores obtienen una valoración inicial de cómo los activos ágiles pueden posibilitar la certificación de CMMI de una empresa, hasta qué punto, qué nivel de capacidad y cuáles son los requisitos que los activos de procesos ágiles no alcanzan a cubrir. Una importante conclusión de este artículo es que, superando algunas dificultades, es posible alcanzar el nivel 2 de madurez utilizando métodos ágiles; por tanto, los métodos ágiles son una buena alternativa no sólo para empresas que aún no han adoptado procesos, sino que pueden serlo también para aquellas que aún no están certificadas a nivel de CMMI.

Otro factor a tener en cuenta en el contexto nacional y regional es que el sector productivo del software colombiano se enfrenta a dificultades como la utilización de procesos de desarrollo inapropiados y la ignorancia de la importancia que tiene el proceso de desarrollo sobre la calidad del producto [3]. Todo lo anterior tiene como consecuencia una baja calidad del producto terminado, un tiempo de desarrollo y pruebas que no satisface al cliente y unos costos que no son competitivos en el mercado global. Por esto, se hace necesario explorar nuevos métodos tendientes a minimizar los tiempos de entrega de forma económica y sencilla, manteniendo la calidad en el producto terminado. Una alternativa para lograrlo y que contribuye a mejorar la calidad del software colombiano es explorar las nacientes metodologías ágiles a través de su aplicación a proyectos reales como alternativa para ganar productividad y calidad en el desarrollo.

Para comercializar de manera efectiva y exitosa los productos y servicios colombianos de software, es necesario elevar la competitividad de este sector de la industria, de tal modo que sea visto en el exterior como un país en el cual se puede contratar productos software y servicios asociados de calidad, para lograrlo se requieren varios ingredientes: la voluntad de las firmas de trabajar conjuntamente en procesos de mejoramiento continuo, un capital apreciable que pueda representarlas ante el mercado externo con seriedad y respaldo [4], un conocimiento de dicho mercado y un factor muy significativo: la vinculación de la academia, contribuyendo con investigación y desarrollo en este proceso; también se requiere la difusión de dichos resultados y conclusiones para que puedan ser asimilados y puestos en práctica en las empresas. En ello radica la importancia de este primer caso de

estudio en el que se abordan las metodologías ágiles en un entorno de pequeñas organizaciones, con el que se busca aportar los fundamentos, adaptación, resultados y conclusiones de la implementación de programación extrema y modelamiento ágil que se propone, de esta manera, las empresas o desarrolladores que deseen probarlas cuentan una base en la cual apoyar su decisión.

# 2. OBJETIVOS

## 2.1. Objetivo General

Generar una serie de lineamientos que puedan ser útiles para desarrolladores y empresas que se encaminen en proyectos similares al que se desarrolla en este trabajo, o en la adopción de metodologías ágiles. Dichos lineamientos contemplarán los siguientes aspectos: ventajas, desventajas, recomendaciones y conclusiones de la metodología empleada.

## 2.2. Objetivos específicos

- Generar una serie de recomendaciones que tengan en cuenta los aspectos de calidad, productividad y competitividad en el desarrollo de software en el Cauca a partir de la aplicación de Programación Extrema y Modelamiento Ágil en un caso concreto.

- Desarrollar un aplicativo software adaptable y flexible, que permita administrar de forma eficiente, cómoda y segura los recursos de la granja agroindustrial de la Universidad del Cauca en la vereda La Rejoya, aplicando Programación Extrema y Modelamiento Ágil.

# 3. MARCO TEÓRICO

El proceso de desarrollo de software es considerado como un conjunto de actividades, personas, componentes de software, estructuras organizacionales, reglas, políticas, metodologías y herramientas utilizadas o creadas específicamente para conceptualizar, desarrollar, ofrecer soluciones y servicios, innovar o extender un producto de software. Es decir, es la forma en que la organización realiza sus distintos proyectos de generación de software.

En el campo informático, en los años sesenta, se identificó una problemática que fue denominada "crisis del software", la cual se caracterizó por el retraso en las entregas de los proyectos, sobrecosto y una gran cantidad de fallos y defectos. Para enfrentar dicha problemática surge la ingeniería de software y con ella las metodologías tradicionales de desarrollo de software (un proceso dividido en planeación, análisis, diseño, programación y ciclo de pruebas), con el fin de imponer cierta disciplina al proceso y lograr que los proyectos de software terminen exitosamente, lo cual no se logra en la totalidad de los casos.

Estas metodologías tradicionales se caracterizan por su rigidez, por un gran énfasis en planificación y diseño y porque no son tolerantes al cambio, algo que no es coherente con la realidad del desarrollo de software ni con el mundo real.

Dadas esas circunstancias, surgen unas nuevas metodologías, que inicialmente recibieron el nombre de metodologías ligeras, pero actualmente se conocen con el nombre de metodologías ágiles. Las metodologías ágiles para el desarrollo de software estuvieron originadas por:

(a) Una conciencia particularmente aguda de la crisis del software.
(b) La responsabilidad que se atribuye a las grandes metodologías en la gestación de esa crisis.
(c) El propósito de articular soluciones.

## 3.1 Metodologías Ágiles y sus principios.

Como una reacción a las metodologías tradicionales, un nuevo grupo de metodologías ha surgido en los últimos años. Durante algún tiempo se conocían como metodologías ligeras, pero el término aceptado ahora es metodologías ágiles [6].

El término "ágil" aplicado al desarrollo de software nace en febrero de 2001, tras una reunión celebrada en Utah, Estados Unidos. Después de esta reunión se creó The Agile Alliance –La Alianza Ágil- [7], una organización sin ánimo de lucro dedicada a impulsar y difundir los conceptos relacionados con la aplicación de metodologías ágiles para el desarrollo de software. En esta misma reunión se firma el Manifiesto Ágil, un documento que resume la filosofía ágil y según el cual se valora:

- **Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas:** el movimiento ágil enfatiza la relación y comunicación de los desarrolladores de software. En las prácticas ágiles existentes, éstas manifiestan estar muy cercanas a la interrelación de los miembros del equipo, disposición del ambiente de trabajo, y otros procedimientos que impulsan el espíritu de equipo.

- **Desarrollar software que funciona más que conseguir una buena documentación:** el gran objetivo del equipo de desarrollo es realizar pruebas de funcionamiento de software al final de cada lanzamiento. Los lanzamientos se producen frecuentemente, y en algunos casos diariamente, pero usualmente cada dos o tres semanas. Adicionalmente, los desarrolladores deben mantener el código tan simple y técnicamente avanzado como sea posible.

- **La colaboración con el cliente más que la negociación de un contrato:** a la relación y cooperación entre desarrolladores y clientes se le da preferencia sobre el contrato estricto. Desde el punto de vista del negocio, el desarrollo ágil está enfocado a entregar valor al negocio desde la iniciación del proyecto, así se reduce inmediatamente el riesgo de no cumplimiento del contrato.

- **Responder a los cambios más que seguir estrictamente un plan:** el grupo de desarrollo, compuesto por desarrolladores y representantes del cliente, debe estar

bien informado, ser competente y estar autorizado para considerar posibles ajustes y necesidades que surgen durante el ciclo del proceso de desarrollo. Esto significa que los contratos existentes contienen herramientas que permiten y apoyan la realización de estos cambios y que los participantes están preparados para realizarlos.

Los firmantes del manifiesto ágil fueron: Kent Beck (XP), Mike Beedle, Arie van Bennekum (DSDM), Alistair Cockburn (Crystal), Ward Cunnimgham (XP), Martin Fowler (XP), James Grenning (XP), Jim Highsmith (ASD), Andrew Hunt (Pragmatic Programming), Ron Jeffries (XP), Jon Kern (FDD), Brian Marick, Robert C. Martin (XP), Steve Mellor, Ken Schwaber (Scrum), Jeff Sutherland (Scrum) y Dave Thomas (Pragmatic Programming).

### 3.1.1. Principios de las metodologías ágiles

Junto al Manifiesto Ágil se han especificado los principios que rigen al conjunto de metodologías ágiles [8].

- La prioridad más alta es satisfacer al cliente a través de la entrega temprana y continua de software valioso.
- Los requerimientos cambiantes son bienvenidos, incluso cuando llegan tarde en el desarrollo. Los procesos ágiles se adaptan al cambio en procura de una ventaja competitiva para el cliente.
- Entregar frecuentemente software que funcione, desde un par de semanas hasta un par de meses, con preferencia por las escalas de tiempo más breves.
- La gente de negocios y los desarrolladores deben trabajar juntos cotidianamente a través de todo el proyecto.
- Construir proyectos en torno a individuos motivados. Darles la oportunidad y el respaldo que necesitan y procurarles confianza para que realicen las tareas.
- La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
- El software que funciona es la medida primaria del progreso.
- Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante indefinidamente.
- La atención continua a la excelencia técnica enaltece la agilidad.

- La simplicidad (el arte de maximizar la cantidad de trabajo que no se hace) es esencial.
- Las mejores arquitecturas, requerimientos y diseños emergen de equipos que se auto-organizan.
- A intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo, y en consecuencia ajusta su conducta.

El resultado de todo esto es que los métodos ágiles cambian significativamente algunos de los énfasis de los métodos en ingeniería de software. La diferencia inmediata es que son menos orientados al documento, exigiendo una cantidad más pequeña de documentación para una tarea dada. En muchas maneras están orientados al código siguiendo un camino que dice que la parte importante del desarrollo es el código fuente. La falta de documentación es un síntoma de diferencias mucho más profundas:

- **Los métodos ágiles son adaptables en lugar de predictivos.** Con los métodos ingenieriles se intenta planear una gran parte del proceso del software detalladamente para un largo plazo de tiempo, esto funciona bien hasta que los requisitos cambian, así que su naturaleza es resistirse al cambio. Para los métodos ágiles, no obstante, el cambio es bienvenido. Intentan ser procesos que se adaptan y crecen en el cambio, incluso al punto de modificarse ellos mismos.

- **Los métodos ágiles son orientados a la gente y no orientados al proceso.** Los métodos ágiles afirman que ningún proceso podrá maquillar las habilidades del equipo de desarrollo, de modo que el papel del proceso es apoyar al equipo de desarrollo en su trabajo, explícitamente, puntualizan el trabajar a favor de la naturaleza humana en lugar de en su contra y enfatizan que el desarrollo de software debe ser una actividad agradable [9].

La Tabla 3.1 recoge esquemáticamente las principales diferencias de las metodologías ágiles con respecto a las tradicionales. Estas diferencias afectan no sólo el proceso en sí, sino también el contexto del equipo así como a su organización [7].

| **Metodologías ágiles** | **Metodologías Tradicionales** |
|---|---|
| Basadas en heurísticas provenientes de prácticas de producción de código. | Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo. |
| Especialmente preparadas para cambios durante el proyecto. | Cierta resistencia a los cambios. |
| El coste de los cambios es prácticamente constante en cualquier fase de desarrollo. | El coste de los cambios aumenta de forma exponencial. |
| Proceso menos controlado, con pocos procesos. | Proceso mucho más controlado, con muchas políticas/normas. |
| No existe contrato tradicional o al menos es bastante flexible. | Existe un contrato prefijado. |
| Interacción constante del cliente con el equipo de desarrollo. | Interacción del cliente con el equipo de desarrollo mediante reuniones. |
| Grupos pequeños trabajando en parejas en el mismo sitio. | Grupos grandes y posiblemente distribuidos. |
| Pocos artefactos | Más artefactos |
| Pocos roles | Más roles |
| Menos énfasis en la arquitectura del software. | La arquitectura del software es esencial y se expresa mediante modelos. |

**Tabla 3.1. Diferencia entre metodologías ágiles y tradicionales.**

## 3.2 Programación Extrema (eXtreme Programming - XP)

A mediados de la década de 1980, Kent Beck y Ward Cunningham trabajaban en un grupo de investigación de Tektronix; allí idearon las tarjetas CRC y sentaron las bases de lo que después serían los patrones de diseño y XP. CRC significa "Clase-Responsabilidad-Colaboración", y es una técnica que reemplaza a los diagramas en la representación de modelos. En las tarjetas se escriben las responsabilidades, una descripción de alto nivel del propósito de una clase y las dependencias primarias. En su economía sintáctica y en su abstracción, prefiguran lo que más tarde serían las tarjetas de historias de usuario de XP.

La programación extrema es la más popular entre las metodologías ágiles, su gran logro ha sido revolucionar los problemas causados por los largos ciclos de desarrollo de software de los modelos tradicionales.

XP es una metodología liviana, diseñada para equipos de pequeño o mediano tamaño. Parte de su filosofía es aceptar el cambio como una realidad presente y constante en los proyectos de desarrollo de software, y dentro de su proceso se involucran entregas tempranas, lo cual hace posible el manejo del riesgo de modo que las metas se cumplan. Todos los anteriores son motivos que, sumados, convierten a XP en la metodología indicada para realizar el presente caso de estudio. Otro factor que se tuvo en cuenta a la hora de la elección de la metodología a utilizarse en este proyecto fue que XP está siendo usado por numerosos grupos de desarrollo.

Aunque existe una gran cantidad de literatura disponible, dicha literatura habla de la disciplina que hay que seguir y no mucho sobre qué prácticas y criterios aplicar; ésta es una falencia no sólo en XP sino en otras metodologías que suponen escenarios ideales de aplicación. Por eso se aprovecha este estudio para abordar la metodología con una visión crítica y desde una perspectiva más realista, empleándola en aplicaciones pequeñas, las cuales son más usadas en nuestro entorno; afrontar la metodología de esta manera permite evaluar las prácticas empleadas, bajo qué criterios deben ser empleadas y obtener una serie de recomendaciones y conclusiones que, se espera, sean de utilidad para quienes se decidan a emplear la metodología en escenarios similares.

### 3.2.1 Proceso XP

El ciclo de vida de XP consta de cinco fases: exploración, planeación, iteraciones, producción, mantenimiento y muerte, a continuación se describen según Fowler [6].

**Fase de exploración.** Se solicita al cliente que realice las historias de usuario, las cuales no son otra cosa que una descripción de lo que él necesita. Cada historia de usuario describe una funcionalidad simple que debe ser desarrollada en pocos días, en algunos casos en pocas horas. Si una historia de usuario es compleja o su implementación demanda mucho tiempo, ésta se debe dividir. Al mismo tiempo se realiza el spike[2] arquitectónico, que consiste en asimilar las herramientas a utilizar en el proceso de desarrollo, como los lenguajes de programación, la tecnología y las prácticas; además se comprueba si las

[2] Un spike es un experimento dinámico de código, realizado para determinar cómo podría resolverse un problema. Es la versión ágil de la idea de prototipo. Se lo llama así porque "va de punta a punta, pero es muy fino" y porque en el recorrido de un árbol de opciones implementaría una opción de búsqueda depth-first [12] (SpikeSolution).

herramientas seleccionadas son las adecuadas, para ello se realiza un prototipo donde la tecnología que se utilizará es probada.

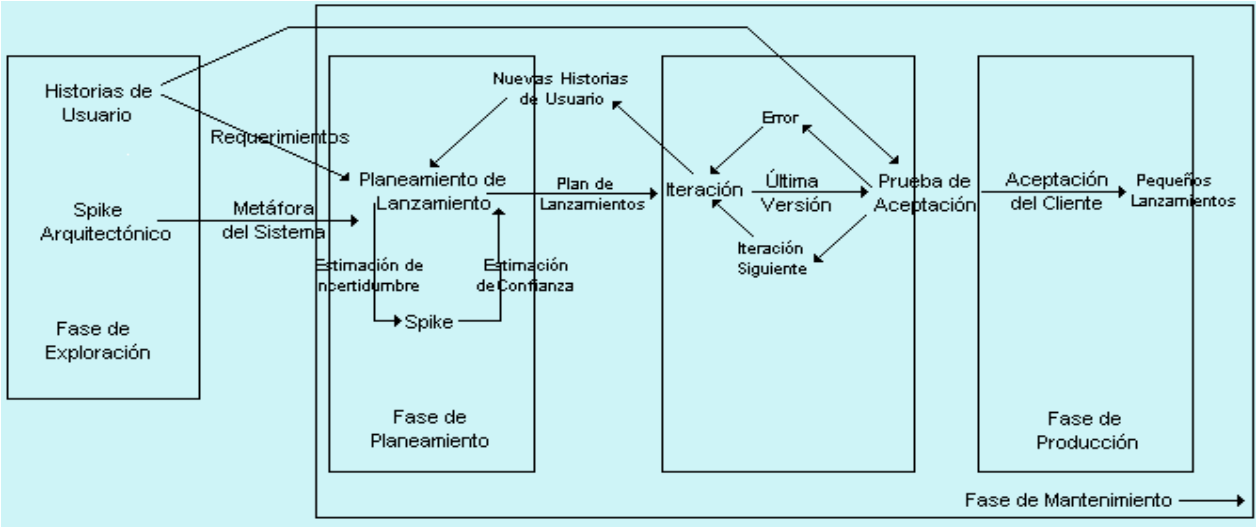



**Figura 1. Ciclo de vida de XP, adaptado de Beck [11]**

**Fase de planeamiento.** En esta fase se realiza un plan de trabajo, fijando un orden de prioridad a las historias de usuario. La metodología indica que en esta fase también se debe llevar a cabo una estimación de la duración de cada una de las historias de usuario, ésta estimación implica tener en cuenta una serie de aspectos como contar con unos datos históricos que permitan conocer la productividad en términos de historias de usuario y líneas de código; además, supone que los programadores saben exactamente qué hacer y que el tiempo de programación ocurre sin interrupciones.

Dado que no se disponía de un registro de datos históricos y los programadores no son expertos en java, es de esperarse que surjan dudas y, debido a otras ocupaciones, no es posible que el tiempo de programación esté desprovisto de interrupciones. En estas circunstancias no se está en capacidad de elaborar un plan de trabajo tan detallado como lo expone XP, lo que se realizó fue un plan de trabajo estimando la duración de cada una de las fases o iteraciones en que se dividió el proyecto en lugar de realizar estimaciones de cada una de las historias de usuario.

**Fase de iteraciones.** Se acordó un plan de trabajo dividido en etapas; cada etapa comprendía un conjunto de funcionalidades que al final fueron mostradas al cliente (el

administrador de la granja agropecuaria de la Universidad del Cauca), quien las daba por aprobadas o sugería los cambios a realizarse. Las etapas, además de incluir funcionalidades, estaban guiadas por las metodologías ágiles, por lo cual utilizaban en cada una de ellas algunas de las prácticas de programación extrema y modelamiento ágil. Esto permitió, al término de cada una de las iteraciones, tener una versión funcional del sistema y analizar el resultado obtenido de la aplicación de prácticas de programación extrema y modelamiento ágil. Al cierre de la última etapa, el sistema estuvo listo para entrar en la fase de producción cumpliendo con los requisitos pactados con el cliente.

**Fase de producción.** Requiere comprobación extra del funcionamiento del sistema. En esta fase el cliente puede sugerir nuevos cambios y, en conjunto, debe decidirse si se incluyen o no en el lanzamiento. Las ideas y sugerencias se deben documentar para una puesta en práctica posterior, por ejemplo en la fase de mantenimiento.

A pesar de que al cliente le fueron presentadas versiones funcionales que cumplían con la mayoría de los requisitos, él optó por la utilización del producto cuando estuviera completamente finalizado. Durante la creación del producto, se tuvieron en cuenta las sugerencias y modificaciones hechas por el cliente, que se implementaron en la siguiente etapa.

**Fase de mantenimiento.** Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento, al mismo tiempo que desarrolla las iteraciones correspondientes a las funcionalidades que se acordaron agregar en la fase anterior. De esta forma, la velocidad de desarrollo puede bajar después de la puesta en producción del sistema. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

Dada la decisión final del cliente de no usar el producto hasta su completa finalización, esta fase no fue aplicada como tal. Lo que se hizo fue prestar asesoría al cliente para una eficiente utilización del producto.

**Fase de muerte.** Es cuando el cliente no tiene más historias para ser incluidas en el sistema, esto requiere que se satisfagan, además, los requerimientos del cliente en otros aspectos como rendimiento y confiabilidad. Se genera la documentación final del sistema y

no se admiten más cambios. La muerte del proyecto también ocurre cuando el sistema no satisface las expectativas del cliente o cuando no hay presupuesto para mantenerlo.

En esta fase ya se había entregado al cliente la versión final, por lo tanto no fue posible agregar nuevas funcionalidades ni sugerir cambios. Durante esta fase se escribió la documentación faltante y algunos anexos.

### 3.2.2 Roles y responsabilidades

Los roles que plantea XP son:

- **Programador.** Los programadores escriben las pruebas y mantienen el código del programa tan simple y definido como sea posible. El primer aspecto para tener éxito con XP es comunicarse y coordinar con los otros miembros del equipo. Este proyecto sólo contó con una pareja de programadores, que fueron los encargados del código y las pruebas.

- **Cliente.** Es quien escribe las historias y pruebas funcionales, y decide cuándo cada requerimiento es satisfecho. También selecciona el orden de prioridad para los requerimientos. El cliente en este caso fue el administrador de la granja agroindustrial de la Universidad del Cauca, localizada en la vereda la Rejoya.

- **Probador.** El probador o tester ayuda al cliente a escribir las pruebas funcionales, realiza pruebas funcionales regularmente, difunden los resultados de las pruebas y mantienen las herramientas de pruebas. Dentro del proyecto no hubo una persona que desempeñara este rol de forma explícita. Aunque el cliente no tenía el conocimiento suficiente para crear las pruebas, los programadores, como responsables del proyecto, se encargaron también de su creación.

- **Tracker/rastreador.** Proporciona realimentación en XP. Él rastrea los estimativos hechos por el equipo, por ejemplo: esfuerzo estimado, y proporciona realimentación de cómo mejorar estimativos futuros. También rastrea el progreso de cada iteración y evalúa si el objetivo es alcanzable con los recursos dados y en el tiempo límite o si son necesarios cambios en el proceso. Este rol no fue necesario en este caso debido a

que no se llevó a cabo la estimación de la duración de las historias de usuario en las iteraciones.

- **Coach/entrenador.** Es la persona responsable por el proceso completo y está en capacidad de guiar a otros miembros del equipo en el seguimiento del proceso. No hubo alguien que asumiera este rol, pero se podría considerar que el director de tesis (director del proyecto), asumió la guía de los otros miembros del equipo, de modo que puede decirse que él fue también el entrenador.

- **Consultor.** Es una persona externa al proyecto que posee el conocimiento técnico necesario, y guía al equipo en la resolución de sus problemas específicos. El profesor Francisco Pino, por su experiencia con metodologías ágiles, fue la persona escogida para este rol.

- **Director.** Es quien toma las decisiones. Para poder hacer esto, él se comunica con el equipo del proyecto para determinar la situación actual y resolver dificultades o deficiencias en el proceso. En algunos casos se realizan reuniones para decidir en conjunto el rumbo del proyecto. Este rol fue asumido en su mayor parte por el director de tesis.

En este caso, el equipo se compone de cuatro personas, por lo tanto no pueden ser asignados todos los roles planteados por XP a diferentes personas, así que hubo personas que tuvieron más de un rol, generándose una sobrecarga de trabajo y dificultad para cumplir el cronograma.

### 3.2.3 Valores de la Programación Extrema

XP se basa en doce prácticas que toman como punto de partida los cuatro valores:

**Comunicación.** XP involucra comunicación extrema. El cliente tiene que hacerse parte del equipo de trabajo, haciendo la definición de los requerimientos, lo cual facilita producir versiones cada 2 a 4 semanas. Esto también permite aclarar dudas en los requerimientos al programador. La programación de pares y la propiedad colectiva del código (prácticas de XP

que se verán en la siguiente sección) promueven la comunicación de conocimiento técnico a través de todo el equipo. Cuando aparecen retos técnicos, el equipo es más capaz de solucionar el problema [13].

Este valor no estuvo totalmente presente en este caso de estudio, pues hubo comunicación frecuente dentro del equipo de trabajo pero no del cliente hacia el grupo de desarrollo, por ello, tampoco no se obtuvieron algunos beneficios que genera el valor de la comunicación, como entregas al cliente cada pocas semanas ni aclaración de dudas en los requisitos.

**Realimentación.** Mientras más rápido se identifique el cambio, más rápido se le puede manejar. En general, la intención es encontrar el error lo más próximo al instante en que fue introducido. XP se esfuerza para que se pueda recibir la retroalimentación lo más rápido posible en todos los aspectos del proyecto.

El valor de la realimentación estuvo presente en el proyecto y fue útil para responder a los cambios, aunque no siempre el cambio era identificado por el cliente con la prontitud deseada.

**Simplicidad.** El cliente es la principal fuerza conductora en XP. Se busca hacer únicamente lo que el cliente necesita, de la manera más simple posible. El propósito es reducir la complejidad desde el principio, por ello, todo el código debe ser refactorizado tan frecuentemente como sea posible [13]. Este valor fue aplicado durante el desarrollo del proyecto y eso evitó que hubiera que realizar trabajo extra.

**Valentía.** Se requiere valor para realizar modificaciones a un software que funciona. XP permite que el programador, de forma responsable y sin temor a que los cambios que realice afecten la funcionalidad existente o causen errores en otras partes, modifique el código cuando lo considere necesario. Esto se da gracias a las pruebas unitarias, que exigen que cada vez que se cambie algo, antes de que se pueda subir al repositorio central, debe pasar todas las pruebas.

Este valor fue aplicado de forma parcial, pues no siempre se estuvo modificando y refactorizando un código que funcionaba. Esta labor se intensificó en las iteraciones finales.

### 3.2.4 Las doce prácticas de la Programación Extrema

La principal suposición que se hace en XP es la posibilidad de disminuir la curva exponencial del costo de cambio a lo largo del proyecto lo suficiente para que el diseño evolutivo funcione. Eso 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.** La planificación en XP permite contestar dos preguntas clave en el desarrollo de software: predecir lo que se hará ahora, y determinar qué se hará después. El énfasis se pone en la dirección del proyecto más que en hacer una predicción exacta de qué será necesario y cuánto se tardará en hacerlo.

  Se elaboró un plan de trabajo que fue la directriz del proyecto en el que se contemplaban los avances para cada fase. Además, se intentó mantener una comunicación frecuente con el cliente como lo sugiere XP, pero la comunicación fue mucho menor de la necesaria; por esto, el cliente no siempre decidía sobre el ámbito y tiempo de las entregas, lo cual afectó profundamente el cronograma previsto.

- **Entregas pequeñas.** Tal como lo establece la metodología, se realizaron entregas pequeñas pero funcionales. Éstas no cumplían con todas las necesidades del cliente pero le servían para verificar el avance del proyecto y para sugerir cambios o mejoras desde etapas tempranas del desarrollo.

- **Metáfora.** Con esta práctica se busca dar una descripción informal de la arquitectura del sistema, una analogía del sistema con la cotidianidad. Se encontró que la aplicación presenta similitudes con los libros de contabilidad en los que se registran ingresos, gastos y, al final de un determinado periodo de tiempo, es posible determinar las utilidades generadas.

- **Diseño simple.** Aunque se busca la solución más simple que funcione, ésta siempre estará limitada por las características deseadas por el cliente. Kent Beck dice que, en cualquier momento, el diseño adecuado para el software es aquel que: supera con éxito todas las pruebas, no tiene lógica duplicada, refleja claramente la intención de

implementación de los programadores y tiene el menor número posible de clases y métodos. En el presente caso de estudio se usaron diagramas de clases, de casos de uso y una arquitectura en capas (para más detalle, ver anexo 5 y la carpeta "Modelos" del CD adjunto a este trabajo), aunque la simplicidad se vio afectada por los constantes cambios y adiciones de requisitos por parte del cliente.

- **Pruebas.** Las pruebas son una práctica crucial en un proyecto de XP. Uno de los fuertes de XP es que hace que los proyectos sean flexibles. La flexibilidad se basa en la retroalimentación exacta y frecuente, y las pruebas proveen esta retroalimentación. En el presente caso de estudio, las pruebas se realizaron cada vez que se generaba nuevo código; cada nueva funcionalidad era probada de forma individual y después, cuando se integraban con el programa existente, se verificaba que no se generaran errores en conjunto.

- **Refactorización.** Consiste en hacer modificaciones en el código que no alteren la funcionalidad del sistema y que impliquen una mejora de alguna cualidad no funcional: simplicidad, flexibilidad, claridad o desempeño. El código debe ser limpio, legible y la duplicación de código debe ser evitada. Se mejora la estructura interna del código sin alterar el comportamiento externo [13]. La refactorización se llevó a cabo en la iteración final, cuando se había cumplido con los requisitos del cliente.

- **Programación en parejas.** Toda la producción de código debe realizarse como trabajo de programadores en parejas. Entre las principales ventajas de introducir este estilo de programación se pueden citar: muchos errores son detectados conforme son introducidos en el código (inspecciones de código continuas), por consiguiente la tasa de errores del producto final es más baja, los diseños son mejores y el tamaño del código es menor (continua discusión de ideas de los programadores), los problemas de programación se resuelven más rápido, se posibilita la transferencia de conocimientos de programación entre los miembros del equipo, varias personas entienden las diferentes partes sistema. Esto implica comunicación por parte de los programadores, lo cual mejora el flujo de información y la dinámica del equipo. Dichos beneficios se consiguen después de varios meses de practicar la programación en parejas (en los estudios realizados por Cockburn y Williams este lapso de tiempo varía de 3 a 4 meses).

Para esta práctica, inicialmente se pensó en un proceso de nivelación con el fin de lograr en el equipo un conocimiento y unas habilidades de programación similares, pero fue complicado y poco fructífero, así que se optó por hacer pruebas programando algunas funcionalidades en pareja y otras de manera individual. Se obtuvo como resultado que la diferencia de conocimiento y experiencia en los lenguajes de programación daban una clara desventaja a la programación en parejas con respecto a la programación individual, ya que el programador con menor conocimiento no estaba en capacidad de corregir errores que es lo que pretende la programación por pares, sino que hacía mas lenta la labor al ser necesario explicarle algunos aspectos del código.

- **Propiedad colectiva del código.** Cualquier programador puede cambiar cualquier parte del código en cualquier momento. Ésta práctica motiva a todos a contribuir con nuevas ideas, evitando a la vez que algún programador sea indispensable para realizar cambios en algún segmento del código. En este caso particular, no cabe duda que el código es de propiedad colectiva ya que, al ser sólo una pareja de programadores trabajando sobre el mismo, el código es conocido por ambos y no es modificado por más personas, mientras que, en grupos más numerosos, los estándares de codificación son necesarios para conservar la "uniformidad" del código y facilitar que todos los miembros del equipo estén en capacidad de entenderlo y modificarlo sin que ello genere confusión entre los demás programadores.

- **Integración continua.** Cada porción de código es integrada al sistema una vez que está lista. La integración continua a menudo reduce la fragmentación de los esfuerzos de los desarrolladores por falta de comunicación sobre lo que puede ser reutilizado o compartido. En este caso, se dio la integración de una forma diferente a la sugerida por XP, ya que no se poseía una máquina de integración, sino que se llevó a cabo en la misma máquina donde se iba produciendo el código. De esta manera, el código realizado se probó de forma individual hasta que funcionó correctamente, y después se procedió a hacer las integraciones con el prototipo general para probarlos en conjunto.

- **Cuarenta horas por semana.** XP promueve contar con un equipo de trabajo bien descansado. Los desarrolladores cansados tienen más propensión a cometer errores y

a empezar a desear un nuevo trabajo [14]. En las iteraciones finales, debido a la mayor presencia del cliente, fue necesario aumentar un poco el tiempo de programación por encima de cuarenta horas por semana; el aumento no fue drástico, así que la calidad del software producido no se vio afectada. Además, hay que tener en cuenta que, en anteriores ocasiones, no se pudo cumplir a cabalidad el propósito de trabajar las cuarenta horas por semana debido a ocupaciones múltiples de los programadores, lo cual compensa las horas "extras" que se trabajaron en las iteraciones finales.

- **Cliente en el sitio de trabajo.** El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Gran parte del éxito del proyecto XP se debe a que el cliente es quien conduce constantemente el trabajo hacia lo que le aportará mayor valor de negocio y los programadores pueden resolver, de manera inmediata, cualquier duda asociada.

  La necesidad del cliente en ciertas fases de la implementación fue evidente, sin ebargo, hubo dificultad para conseguir su presencia en el sitio de trabajo. La ausencia del cliente, en muchos casos, implica dudas en cuanto a los requisitos que no se pueden satisfacer. Ésto se ve reflejado en errores y confusiones en cada una de las iteraciones, que se habrían podido evitar o disminuir considerablemente si el cliente hubiese estado presente para resolverlas en la medida en que se presentaban. Para suplir la ausencia del cliente se contó con un estudiante del programa de Agrozootecnia, quien realizaba su pasantía en la granja agroindustrial de la Universidad del Cauca, pero se pudo verificar que, en instancias decisivas como la validación de las entregas, el cliente debería estar presente, evitando así el riesgo de que su representante valide funcionalidades que no satisfagan al cliente, lo cual implicaría modificaciones. De este modo se evita un posible estancamiento o el retraso de las entregas.

- **Estándares de codificación.** XP enfatiza la comunicación de los programadores a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación que, al aplicarse, propician una integración exitosa del código, lo que a la vez tiene como consecuencia que el código sea comprendido por todos los programadores y lo hace de propiedad colectiva. En este caso, los estándares de

codificación no fueron indispensables porque sólo había una pareja de programadores trabajando sobre todo el código.

El mayor beneficio de las prácticas se consigue con una aplicación conjunta y equilibrada, puesto que se apoyan unas con otras. La mayoría de las prácticas planteadas por XP no son novedosas sino que, de algún modo, ya habían sido propuestas en ingeniería de software e incluso demostrado su valor en la práctica; el mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde las perspectivas del negocio, los valores humanos y el trabajo en equipo [13].

## 3.3 Modelamiento Ágil (Agile Modeling, AM)

Modelamiento Ágil es una metodología para modelado y documentación efectiva de sistemas basados en software. AM es una colección de prácticas, guiadas por principios y valores, para ser aplicadas día a día por los profesionales de software. AM no define detalladamente procedimientos para crear un tipo de modelo dado, en cambio, proporciona consejos para hacer efectivo el modelado [15]. Los modelos ágiles son más efectivos que los modelos tradicionales porque los modelos ágiles son justo lo que se necesita, no tienen que ser perfectos. AM puede ser aplicado para modelar los requisitos, el análisis, la arquitectura y el diseño.

Las técnicas de modelamiento ágil son recomendables, en especial, para equipos de trabajo que están siguiendo algún proceso ágil. Por ésta razón se decidió alimentar XP con técnicas que proporcionen un modelo simple, las cuales fueron otorgadas por AM. Sin embargo, las técnicas de modelamiento ágil no están supeditadas al uso de metodologías ágiles, sino que pueden ser usadas para mejorar y simplificar los resultados del modelamiento en proyectos donde no se usa este proceso. El secreto de AM no son las técnicas de modelado en sí mismas -tales como modelos de casos de uso, modelos de clase, modelos de datos, o modelos de interfaz de usuario- sino cómo se aplican.

En el presente caso de estudio se realizó seleccionando los artefactos indicados para aquello que se iba a modelar. Por ejemplo, los requisitos se modelaron empleando historias de usuario, en cuanto al diseño se optó por emplear artefactos de UML que no impliquen poseer

un amplio conocimiento en esa área a quienes intenten hacer un acercamiento a modelamiento ágil y tomen este trabajo como referente. Por ello se decidió emplear diagramas de clases, de casos de uso y una arquitectura en capas (para más detalle, ver anexo 5 y la carpeta "Modelos" del CD adjunto a este trabajo). En la elección de los artefactos se tuvo en cuenta que se buscó realizar un modelamiento simple, eficiente, y que se debe modelar aquello que se considera necesario y útil, por ello no se consideraron otros artefactos adicionales.

### 3.3.1 Principios de Agile Modeling (AM)

AM define una serie de principios básicos y suplementarios que, al aplicarse en un proyecto de desarrollo de software, fijan la base para una colección de prácticas de modelamiento. Algunas de las prácticas han sido adoptadas de programación extrema y ya han sido bien documentadas.

Los principios de AM son clasificados como:

a) Principios básicos, los cuales se deben adoptar para poder proclamar que verdaderamente se está empleando un enfoque AMDD *Agile Model Driven Development* -Desarrollo Dirigido por Modelamiento Ágil-

b) Principios suplementarios, que se deben considerar y adaptar en un proceso software al conocer las necesidades exactas del entorno.

En enero de 2005 se adicionó una tercera lista, principios desaprobados, que el creador de AM ha decidido eliminar en la segunda versión de AMDD para simplificarla [17].

En las tablas 3.2, 3.3 y 3.4 se presentan respectivamente un resumen de los principios básicos, suplementarios y desaprobados de Agile Modeling. Para complementar la información relacionada con las metodologías ágiles, puede consultar el anexo 1.

| | |
|---|---|
| **Modelar con un propósito** | Muchos desarrolladores se preocupan de que sus artefactos tales como modelos, código fuente o documentos, sean demasiado detallados. Lo que no hacen es dar un paso atrás y preguntarse por qué están creando ese artefacto y para quién lo están creando. El primer paso es identificar un propósito válido para crear un modelo y las audiencias para dicho modelo. |
| **Luz del recorrido** | Cada vez que se decide mantener o agregar un modelo, se intercambia menos agilidad por la conveniencia de tener la información de manera abstracta disponible para el equipo. Nunca subestime la seriedad de esta compensación. Un equipo de desarrollo que decide mantener un documento de requisitos detallado, una colección detallada de modelos de análisis, de modelos arquitectónicos y de modelos de diseño, rápidamente descubrirá que está gastando la mayor parte del tiempo en actualizar dichos documentos y modelos en lugar de escribir el código. |
| **Modelos Múltiples** | Considerando la complejidad del software hoy en día, es necesario tener un amplio rango de técnicas en el kit de herramientas de modelamiento para ser efectivo. Es importante tener presente que no será necesario desarrollar todos estos modelos para un sistema dado pero, dependiendo de la naturaleza del software a desarrollar, será necesario al menos un subconjunto de modelos. |
| **Realimentación Rápida** | El tiempo entre una acción y la realimentación a esa acción es crítico. Particularmente cuando se está trabajando con una tecnología compartida para el modelamiento (como tarjetas CRC), se está obteniendo realimentación rápida en las ideas. En el trabajo de cerca con el cliente, para entender los requisitos, para analizar esos requisitos o para desarrollar una interfaz de usuario que resuelva sus necesidades, es necesaria la realimentación rápida. |
| **Simplicidad** | Durante el desarrollo se debe asumir que la solución más simple es la mejor. No sobredimensione el software o, en caso de que el modelamiento ágil no describa las características adicionales en los modelos que usted no necesite hoy, tenga el valor para no sobremodelar su sistema ahora, ya que se puede modelar basado en los requerimientos existentes hoy y refactorizar el sistema cuando los requerimientos evolucionen. Mantenga el modelo tan simple como sea posible. |
| **Aceptar el Cambio** | El cliente del proyecto puede cambiar características del proyecto a medida que éste avanza, nuevo personal puede ser adicionado al proyecto o algunos existentes lo pueden dejar. El cliente puede cambiar los puntos de vista, potencialmente cambiando los objetivos y los criterios de éxito para su esfuerzo. La implicación es que el ambiente del proyecto cambia mientras progresan sus esfuerzos y que, consecuentemente, su enfoque de desarrollo debe reflejar esta realidad. |
| **Cambio Incremental** | Un concepto importante a entender acerca de modelar es que no es necesario conseguir el modelo correcto la primera vez, lo que se necesita es conseguir un buen modelo a tiempo. |

**Tabla 3.2: Principios Básicos de Modelamiento Ágil. [17]**

| | |
|---|---|
| **Contenido es más importante que documentación** | Un modelo determinado tiene múltiples formas de representarse, mientras que un prototipo se construye utilizando un lenguaje de programación. El modelo no necesariamente es un documento, incluso un completo conjunto de diagramas creados utilizando herramientas CASE[3] puede no hacer parte de un documento, en cambio éstas se utilizan como entradas en otros artefactos, muy comúnmente código fuente, pero nunca formalizadas como documento inicial. Lo importante es que se debe aprovechar las ventajas de los beneficios del modelado sin incurrir en los costos de crear y mantener una documentación. |
| **Comunicación abierta y honesta** | La gente necesita ser libre y percibir que es libre para ofrecer sugerencias. Esto incluye ideas pertinentes a uno o más modelos. La comunicación abierta y honesta capacita al personal para tomar mejores decisiones porque la calidad de la información en la que se están basando es más exacta. |

**Tabla 3.3: Principios Suplementarios de Agile Modeling [17]**

| | |
|---|---|
| **Cada uno puede aprender de cada uno** | Esta es una gran idea, que parece ser seguida por la mayoría de agilistas, pero es muy general y por lo tanto no necesita ser un principio de una metodología de modelamiento específico |
| **Conocer sus modelos** | Conocer aquello que se está haciendo es siempre una buena idea, pero ¿realmente necesita ser un principio explícito? Probablemente no. |
| **Adaptación Local** | El proceso de desarrollo de software se debe adaptar para resolver las necesidades exactas. Sin embargo, esto no significa que esta idea necesite ser parte de los principios, en su lugar necesita ser parte de su estrategia total de mejora de proceso de software. |
| **Trabaje con los instintos de la gente** | Similar a cada uno puede aprender de cada uno. |

**Tabla 3.4: Principios Desaprobados de Agile Modeling V.1**

### 3.4.2 Prácticas de Agile Modeling

AM define una colección de prácticas básicas y suplementarias, fundamentadas en los principios de AM. Algunas de las prácticas han sido adoptadas de programación extrema y ya han sido bien documentadas. Al igual que los principios de AM, las prácticas son presentadas enfocadas en un esfuerzo de modelamiento. Las prácticas básicas se deben adoptar para poder proclamar que verdaderamente se está siguiendo un enfoque AMDD y las prácticas suplementarias se deben considerar adoptar en un proceso de desarrollo de software al conocer las necesidades exactas del ambiente. En Agile Modeling Practices V.2

[3] Herramientas CASE: las herramientas CASE (computer arded software engineering, ingeniería de software asistida por computador) son diversas aplicaciones destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

de enero del 2005 [18], el autor adicionó una tercera lista, prácticas desaprobadas que el creador de AM decidió eliminar en la segunda versión de AMDD para simplificarla. Las prácticas básicas, suplementarias y desaprobadas se resumen en las tablas 3.5, 3.6 y 3.7 respectivamente.

| | |
|---|---|
| **Participación Activa del Cliente** | Es una extensión de la practica XP del cliente en el sitio de trabajo, que describe la necesidad de tener acceso en el sitio a usuarios que tienen la autoridad y habilidad de proveer información pertinente al sistema que está siendo desarrollado, y tomar decisiones a tiempo teniendo en cuenta los requisitos y la prioridad de éstos. |
| **Modele con otros** | Cuando se modela con un propósito, frecuentemente se está modelando para comunicar sus ideas con otros, o se está buscando desarrollar una visión común de un proyecto. Ésta es una actividad del grupo, una en la cual se desea la participación de varias personas que trabajen juntas con eficacia. A menudo será necesario un grupo para desarrollar una solución tan simple como sea posible. Para conseguir esto, la mayoría de las veces el mejor camino es hablar de ello con una o más personas. |
| **Aplique el artefacto correcto** | Cada artefacto tiene sus propias aplicaciones específicas. Es necesario conocer las fortalezas y debilidades de cada tipo de artefacto, de modo que sea claro cuándo utilizarlos y cuándo no utilizarlos. Esto puede ser difícil, puesto que se tienen múltiples modelos disponibles destinados a los modelos ágiles. |
| **Utilice las herramientas más simples** | La gran mayoría de modelos pueden dibujarse simplemente con lápiz y papel o marcadores y tablero. Si se desea ahorrar alguno de éstos dibujos, es posible tomar una fotografía del diagrama con una cámara digital. La mayoría de éstos diagramas, una vez resuelta la aplicación, no ofrecen mucho valor. Como resultado, un tablero y marcadores, o simplemente papel y lápiz, pueden convertirse en su mejor herramienta de modelado. Es recomendable utilizar una herramienta de dibujo para presentar un proyecto importante a los clientes y, ocasionalmente, usar una herramienta de modelado sólo si ello provee valor al objetivo de programación, como la generación de código. |
| **Modele en incrementos pequeños** | AM está enfocado para aplicarse en metodologías ágiles donde se utiliza desarrollo incremental, en el que se organiza una gran labor en pequeñas porciones que se lanzan en un cierto plazo, preferiblemente en incrementos de semanas o un mes o dos, lo que incrementa la agilidad al permitir entregar software más rápidamente. |
| **Fuente de información simple** | La información se debe guardar solamente en un lugar. En otras palabras, no solamente se debe aplicar el artefacto correcto, también se debe modelar un concepto sólo una vez, almacenando la información en el mejor lugar posible. A veces, el mejor lugar para almacenar la información es un documento ágil, frecuentemente el código fuente. |
| **Exhibición de Modelos Públicamente** | Los modelos deben mostrarse públicamente, frecuentemente sobre algo denominado "pared de modelado". Ésta permite una comunicación abierta y honesta en el equipo de desarrollo porque todos los modelos son rápidamente accesibles al grupo, igualmente permite que la información esté disponible para los clientes. La pared de modelado es donde se colocan los modelos para que cada uno los vea, ésta debe ser accesible al equipo de desarrollo y a los clientes. La pared de modelado puede ser un lugar físico, un tablero donde se fijan los modelos, o un lugar virtual, como una pagina Web en Internet que sea actualizada con imágenes escaneadas. |

**Tabla 3.5: Prácticas básicas de Agile Modeling**

| | |
|---|---|
| **Aplicar Estándares de Modelado** | Esta práctica se deriva de la práctica "estándares de codificación" de XP, la idea básica es que los desarrolladores deben permitir y seguir un conjunto común de estándares de modelado en un proyecto de software. |
| **Descarte Modelos Temporales** | La gran mayoría de modelos que se crean son modelos temporales, diseño de bosquejos, prototipos de baja fidelidad, potenciales alternativas de arquitectura/diseño, etcétera. Son modelos que han satisfecho un propósito pero no adicionan valor después de que ya lo han hecho. Los modelos están rápidamente fuera de sincronización con el código, por lo que se debe tomar la decisión de sincronizar los modelos o desecharlos, simplemente porque la inversión para poner al día los modelos no será recuperada. Ésta práctica es particularmente importante para la documentación ágil. |
| **Actualice solamente cuando sea necesario** | Los modelos deben ser actualizados solamente cuando sea absolutamente necesario, cuando no actualizar el modelo sea más doloroso que actualizarlo. Con este enfoque se descubre que se actualizan un número menor de modelos de los que se tenía en el pasado, y que la realidad es que los modelos no tienen que ser perfectos para proporcionar valor. Demasiado tiempo y dinero es desperdiciado al intentar mantener sincronizados modelos y documentación con código fuente, convirtiéndose una tarea imposible de llevar. Ésta práctica es particularmente importante para una documentación ágil. |

**Tabla 3.6: Prácticas Suplementarias de Agile Modeling**

| | |
|---|---|
| **Modele para Comunicar** | Ésta es una gran idea, pero no realmente una práctica, por lo tanto fue desaprobada. |
| **Reutilizar Recursos Existentes** | Ésta es una gran idea que todos los desarrolladores deben practicar, pero es un concepto que va más allá del modelar y de la documentación. |

**Tabla 3.7: Prácticas Desaprobadas de Agile Modeling V.1**

# 4. PROCESO DE PROGRAMACIÓN EXTREMA DOCUMENTADO CON MODELAMIENTO ÁGIL V 1.0

En este capítulo se presenta un resumen del proceso propuesto, para ampliar lo expuesto en el presente capítulo consulte el anexo 4.

## 4.1 Fase 1: Exploración

**Objetivo:**

Obtener los requisitos del sistema en las palabras del cliente, los cuales pueden variar en el transcurso del desarrollo.

El resultado de esta fase es un documento en el que el cliente deja por escrito sus necesidades y cómo deben suplirse. Éste caso de estudio estuvo centrado en obtener un Sistema de Información para Granjas Agroindustriales (en adelante lo designaremos mediante la abreviatura SIGA), el cual debe estar en capacidad de:

- Gestionar la contabilidad de gastos e ingresos de cada microproyecto en periodos mensuales.

- Los proyectos bovinos y porcinos deben considerarse como proyectos especiales, esto es, la contabilidad de gastos e ingresos no se hace en su totalidad para todo el proyecto sino por cada unidad de producción (por cada porcino o por cada bovino); además, en estos proyectos la contabilidad de gastos e ingresos se lleva para todo el ciclo de producción y no para un período de un mes.

- Debe permitir agregar o eliminar proyectos según sean las disposiciones de la granja
- Debe tener una base de datos que permita la recolección de información de las ventas o los gastos realizados, de modo que se permita además de ver los resultados por cada macroproyecto, ver también los resultados de la totalidad de los macroproyectos, y al final del año ver el resultado total de la suma de los meses.

- Debe permitir imprimir los resultados de la contabilidad, esto con el fin de que sirvan como anexos a los informes que deben presentarse a la Vicerectoría administrativa de la Universidad del Cauca.

### 4.1.1 Actividades fase de exploración:

- **Escribir historias de usuario**: el cliente describe las características y especificaciones de aquello que necesita sea implementado.

- **Realizar el spike arquitectónico**: se lleva a cabo una exploración y prueba de las posibles alternativas de solución para el cliente y se selecciona la más adecuada.

- **Realizar la metáfora del sistema**: la metáfora es una descripción sencilla y que todo el mundo puede contar con facilidad de cómo funciona el sistema. La aplicación se puede definir por una o varias metáforas compartidas por el cliente y el equipo de desarrollo, consiste en formar un conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema.

### 4.1.2 Entradas fase de exploración:

Ninguna

### 4.1.3 Salidas fase de exploración:

- **Historias de usuario:** Se busca obtener las necesidades del cliente (que deben ser traducidas a requerimientos del sistema) escritos por su propia mano y explicadas en sus palabras. El formato empleado para este fin se muestra en la tabla 4.1.

| **Historia de Usuario** | |
|---|---|
| **Número:** | **Prioridad en Negocio:** |
| **Nombre historia:** | |
| **Iteración Asignada:** | Riesgo en Desarrollo:<br>**(Alto / Medio / Bajo)** |
| **Programador responsable:** | |
| **Descripción:** | |
| **Observaciones:** | |

**Figura 4.1: Formato Historias de Usuario**

- **Spike arquitectónico:** se genera un programa simple para identificar los riesgos y problemas que se pueden presentar en el desarrollo del sistema. Este programa es basado en los requerimientos del cliente tomados de las historias de usuario. Con esto se busca determinar si el sistema es viable o no. Si el sistema es viable, se desarrolla una metáfora del sistema.

- **Metáfora del sistema:** la metáfora se describió en la sección 4.1.1

## 4.2 Fase 2: Planeamiento

**Objetivo:**

Obtener un estimativo del número de iteraciones a realizar y del tiempo a emplear en cada una de ellas.

### 4.2.1 Actividades fase de planeamiento

- **Priorizar (asignar prioridad a) historias de usuario**: el cliente asigna el orden y la iteración en que las historias de usuario se deben implementar.

- **Definir pruebas de aceptación:** Una vez el equipo de trabajo ha definido los diferentes roles en la reunión de planeamiento, y teniendo en cuenta las historias de usuario ordenadas prioritariamente, el probador ayuda al cliente en la definición de las diferentes pruebas de aceptación correspondientes a cada iteración.

- **Establecer tareas de programación:** Una vez se tienen las pruebas de aceptación a satisfacer en cada iteración los programadores deben determinar las tareas de programación necesarias para satisfacerlas.

- **Estimar duración por iteración:** Los programadores a partir de las tareas de programación a realizar en cada iteración estiman la duración de cada una de ellas.

- **Realizar plan de lanzamientos:** La estimación de duración de cada una de las iteraciones es de utilidad para la realización del plan de lanzamientos, donde se definen las fechas de las posibles entregas correspondientes a cada iteración. En la reunión de planeamiento también se realiza la asignación de los diferentes roles.

### 4.2.3 Entradas fase de planeamiento:

- Historias de usuario

### 4.2.4 Salidas fase de planeamiento:

- **Plan de lanzamientos:** Es el plan de trabajo dividido en etapas, o iteraciones, una iteración corresponde a un grupo de historias de usuario a desarrollar. El plan de lanzamientos define las historias de usuario incluidas en cada iteración y la fecha de las entregas, también conocidas como lanzamientos. Cuando se cumple una iteración, ésta es revisada por el cliente quien la aprueba o sugiere modificaciones. Previamente se había convenido un plan de lanzamientos que consistía en seis

iteraciones, aunque el grupo de trabajo fue conciente de la naturaleza cambiante de los requisitos y del alcance del proyecto, por ello se planeó que la última iteración sea básicamente para corregir errores o asuntos pendientes o nuevos que resulten de todo el proceso de desarrollo, esto permitió en cierta medida, que los requisitos no cambiaran ni se prolongaran de forma indefinida, sino que el cliente sabía hasta que momento podía sugerir modificaciones o agregar historias de usuario.

**Realimentación:**

- **Nueva historia de usuario:** Se toman las historias de usuario que siguen en el orden de prioridad.

## 4.3 Fase 3: Iteraciones

**Objetivo:**

En esta fase se busca desarrollar las historias de usuario prioritarias para el cliente siguiendo el plan de lanzamientos.

### 4.3.1 Actividades fase de iteraciones

- **Realizar reunión de planificación de iteración:** Al inicio de cada iteración se realiza una reunión en la que se asigna el orden en que serán solucionadas las historias de usuario pertenecientes a la iteración.

- **Diseñar pruebas de aceptación:** Se definen las pruebas que deben cumplir las funcionalidades para que pueda considerarse que esta ha sido correctamente implementada. Esto se hace en conjunto con el cliente.

- **Codificar historias de usuario:** Los programadores implementan cada una de las historias de usuario pertenecientes a la iteración, se considera que el desarrollo termina cuando todas las historias de usuario han sido programadas y además satisfacen las pruebas de aceptación.

- **Verificar pruebas:** Las pruebas se dan por satisfechas si una vez desarrolladas cumplen con cada uno de los casos considerado previamente para cada una de ellas. En caso de que no se certifiquen en su totalidad se retorna nuevamente el desarrollo, donde se corrige los fallos existentes hasta que no se encuentren errores.

- **Lanzar prototipo:** Una vez culminado el desarrollo y verificación de pruebas de cada iteración se obtiene un prototipo que es mostrado al cliente, quien puede sugerir cambios o ajustes a esta versión parcial del sistema a desarrollar.

### 4.3.2 Entradas fase de iteraciones

- **Plan de lanzamientos:** Pone en ejecución el trabajo teniendo en cuenta el orden de prioridad de las historias de usuario y la duración estimada de la implementación de cada iteración. Posteriormente se ejecutan las pruebas de aceptación para cada iteración.

- **Prueba de aceptación de fallos (Entrada: Fallos):** En el caso de que las pruebas de aceptación no sean superadas de forma satisfactoria, se procede a corregir los fallos existentes y se vuelve a ejecutar las pruebas de aceptación. Este proceso se repite hasta que las pruebas de aceptación sean superadas en su totalidad.

### 4.3.3 Salidas fase de iteraciones:

- **Nueva funcionalidad:** Al ser superadas las pruebas de aceptación de forma satisfactoria, el sistema en general cuenta con nuevas funcionalidades o nuevas capacidades.

- **Fallos corregidos:** Cuando las pruebas de aceptación no son superadas, se procede a buscar los fallos que impiden su ejecución correcta, y, al encontrarlos, se los corrige. La fase de iteraciones termina cuando las pruebas de aceptación son aprobadas.

## 4.4 Fase 4: Producción

**Objetivo:**

Obtener un sistema adaptable y flexible, acorde a las necesidades del cliente y evaluar la metodología aplicada.

### 4.4.1 Actividades fase de producción

- **Verificar funcionamiento del sistema integrado**: se comprueba el cumplimiento de las pruebas funcionales para el sistema al final de las iteraciones planeadas.

- **Documentar sugerencias o modificaciones**: (en caso de que éstas existan), se documentan las indicaciones del cliente para que puedan ser tenidas en cuenta en posteriores versiones del sistema. Responsable: desarrolladores.

### 4.4.2 Entradas fase de producción:

- **Prototipo funcional:** Cuando las pruebas de aceptación son aprobadas, se verifica el correcto funcionamiento de la totalidad del sistema (el ensamblaje de las funcionalidades obtenidas) hasta que esté listo para ser lanzado al cliente.

### 4.4.3 Salidas fase de producción:

- **Producto liberado:** Se entrega al cliente el producto software en funcionamiento, que en este instante debe satisfacer la totalidad de especificaciones.
- **Nuevas funcionalidades:** Las sugerencias finales del cliente se documentan para ser tenidas en cuenta en la fase de mantenimiento y muerte.

## 4.5 Fase 5: Mantenimiento y Muerte

**Objetivos:**

- Capacitar al usuario en la instalación y manejo del software desarrollado.
- Satisfacer las nuevas funcionalidades aceptadas del prototipo funcional.
- Realizar un informe sobre la experiencia con el modelo seguido, al igual que documentar los manuales para el usuario.

### 4.5.1 Actividades fase de mantenimiento y muerte:

- **Implementar nuevas historias de usuario**: se codifican las nuevas historias de usuario sugeridas por el cliente que hayan sido aceptadas.

- **Realizar nuevas pruebas funcionales**: se realizan las pruebas funcionales correspondientes a las nuevas historias de usuario y se determina el grado de satisfacción de los requerimientos del cliente.

- **Capacitar al cliente**: se entrena al cliente en la instalación y manejo del producto terminado.

- **Realizar documentación:** Se completa la documentación relacionada con el proceso.

### 4.5.2 Entradas fase de mantenimiento y muerte

- **Prototipo Funcional**: la versión en funcionamiento obtenida en la etapa de producción.

- **Nuevas funcionalidades:** correspondientes a las nuevas historias de usuario y las modificaciones que de común acuerdo programadores y cliente deciden poner en funcionamiento para el producto final.

### 4.5.3 Salidas fase de mantenimiento y muerte

- **Producto en funcionamiento:** como resultado de la capacitación al cliente se obtiene el sistema trabajando en manos del cliente.

- **Monografía y Anexos:** Al finalizar todo el proceso se realiza la monografía (documento que expone la metodología y sus resultados) y la documentación relacionada que amplia y explica en mayor nivel de detalle aspectos que no se profundizan en la monografía.

    1. Metodologías Ágiles
    2. Historias de Usuario y pruebas
    3. Estado de la industria del software en el suroccidente colombiano
    4. Manuales de usuario e instalación: contienen las instrucciones, que deben ser seguidas por el cliente, para instalación y funcionamiento correctos del software desarrollado.
    5. Otros anexos

## 4.6 Consideraciones de Aplicación

En esta sección se describen los lineamientos a tener en cuenta para la aplicación del proceso XP adaptado.

- **Fase de exploración:** debe tenerse en cuenta que en la región, la metodología es poco conocida y su aplicación hasta el momento se limita a un escaso número de organizaciones, por lo tanto, quienes estén interesados en aplicar XP deben capacitarse para lograr el conocimiento que les permita la comprensión y aceptación del proceso, los valores y las prácticas que involucra.

Ante la posibilidad de que el cliente no siempre tenga un nivel de comprensión que le permita escribir las historias de usuario, (que fue lo ocurrido en este caso), uno de los integrantes del equipo de desarrollo (programador o probador), debe estar dispuesto a instruir al cliente, o de ser necesario refinar las historias de usuario, ya que es fundamental una total claridad de lo que se debe implementar en las etapas posteriores.

Es probable que esto implique un poco más de tiempo, pero con toda seguridad esto se verá recompensado en las etapas futuras.

Una ventaja de esta fase es la realización del spike arquitectónico, pues la exploración de las diferentes alternativas de solución permite seleccionar la más adecuada.

- **Fase de planeamiento:** actividades como estimación de duración de implementación de cada historia de usuario, implica que los programadores sean experimentados. Para el caso de grupos de programación no avanzados, puede servir de guía un plan de lanzamiento menos detallado teniendo en cuenta los siguientes aspectos: asignar prioridad a las historias de usuario (el cliente es quien decide la prioridad), los programadores implementan en las primeras iteraciones las historias de mayor prioridad, el plan de lanzamientos debe contener las historias asignadas a cada iteración, pero no cuenta con una estimación tan detallada, lo que se puede hacer y también es útil es estimar en qué fecha finalizará cada una de las iteraciones.

  Una ventaja que se desprende de esta etapa es que el desarrollo posterior estará enfocado a dar solución a aquellas funcionalidades prioritarias para el cliente, adicionalmente permite una aproximación del tiempo, recursos y alcance del proyecto.

  La desventaja más notable es que la metodología es tolerante al cambio de requisitos, por ello, en el transcurso del desarrollo pueden ser agregadas o modificadas algunas historias de usuario, con la correspondiente alteración del plan de lanzamientos realizado.

  Otra desventaja en esta etapa es que las estimaciones realizadas suponen unos escenarios bastante ideales, los cuales es difícil que se cumplan en su totalidad. Por eso, aunque el plan de lanzamiento es importante para dimensionar el proyecto, probablemente presente modificaciones en el transcurso del mismo.

- **Fase de Iteraciones:** el entorno regional está compuesto sobre todo por empresas pequeñas, por ello, en algunos casos no será posible que cada uno de los roles implicados en esta etapa sea desempeñado por una persona diferente. También debe considerarse que el equipo de programadores lo pueden conformar un número impar

de personas y en esos casos no se puede poner en práctica la programación por pares en su totalidad.

Para el caso de equipos de menor número de personas, una alternativa posible es que uno de los programadores sea también probador. En la medida de lo posible se debe intentar realizar el código por pares. Cuando el número de programadores es impar, una alternativa para mantener la programación por pares es que una de las personas trabaje sola y se rote de manera que luego trabaje en pareja, de tal modo que todos los programadores estén en capacidad de implementar cualquier sección del código.

El mayor beneficio de la metodología se logra de articular y complementar preferiblemente la totalidad de las doce prácticas, pero algunas están fuera del dominio de las empresas desarrolladoras de software, como es el caso del cliente en el sitio de trabajo. Las posibles alternativas para suplir la ausencia del cliente pueden ser: un representante del cliente con capacidad de decisión disponible, utilización de medios de comunicación como teléfono, servicios de mensajería instantánea; programar reuniones diarias de corta duración en las que tanto el cliente como los programadores aclaren posibles dudas o realicen sugerencias.

La programación extrema es muy abierta y tolerante al cambio, al punto que esto puede convertirse en un obstáculo para la medida del avance y alcance real del proyecto, y aunque parte de su filosofía es darle mayor importancia a la colaboración con el cliente, más que a la negociación de un contrato, es fundamental establecer unos parámetros claros de en que circunstancias y tiempos son aceptables los cambios y adiciones de funcionalidades, de no hacerse así, las organizaciones desarrolladoras corren el riesgo de terminar invirtiendo mucho más de lo estimado en la implementación.

El modelamiento ágil es una buena práctica para complementar el proceso XP, no se debe buscar que estos modelos sean los más completos y detallados, para evitar la inversión en tiempo que significa mantener unos modelos actualizados con los requisitos son cambiantes. Los modelos deben estar enfocados en el propósito del modelamiento y a quien van dirigidos, así es posible evitar la realización de modelos que no tendrán utilidad pero que implican tiempo y recursos.

Una ventaja de esta fase del proceso es que al poner en práctica las entregas frecuentes de historias de usuario implementadas se identifican a tiempo los posibles cambios necesarios, al realizarse los cambios en etapas tempranas, se reduce el costo para las organizaciones. Una ventaja adicional de las entregas frecuentes es que proporcionan agilidad al proceso de desarrollo y brindan seguridad y confianza a ambas partes.

Otra ventaja adicional es que todo el código producido esta siendo probado, se realiza pruebas a las nuevas funcionalidades de forma individual y posteriormente cuando son integradas, adicionalmente también se debe refactorizar, esto hace que el código producido sea de calidad.

La desventaja más notable para el proceso se presenta cuando el cliente o un representante autorizado no pueden estar disponibles, porque sin ellos no es posible realizar las entregas frecuentes deseadas, ni tampoco se puede cumplir con el plan de lanzamientos establecido.

- **Fase de producción:** en esta etapa el cliente puede sugerir nuevos cambios, por ello el equipo de desarrollo debe ser muy cuidadoso en aceptar o no estos últimos cambios. Adicionalmente debe comprometerse al cliente a una mayor presencia en el sitio de trabajo ya que es necesaria una comprobación extra del sistema.

  Una desventaja se presenta si el cliente no puede estar disponible para la aceptación de la totalidad de las pruebas y de la aceptación de todo el producto desarrollado, pero el proceso incluye la presencia del cliente como una de las prácticas.

  No se tiene consideraciones especiales para la fase de mantenimiento y muerte.

# 5. CASO DE ESTUDIO

En este capítulo se describe el proceso de desarrollo de software basado en las metodologías ágiles programación extrema y modelamiento ágil. Las metodologías se adaptaron a las características del proyecto y las limitaciones del equipo de trabajo, por eso no todas las prácticas se adoptaron de forma total. En los casos en que alguna práctica no se pudo llevar a cabo, se propusieron alternativas para que puedan ser apropiadas de forma gradual en una organización. Lo expresado a continuación corresponde al conocimiento obtenido a partir de la aplicación del método propuesto.

En el desarrollo del capítulo inicialmente se describen cómo están aplicadas las distintas fases del proceso propuesto en la construcción de un producto software y, posteriormente, se describe dicho proceso con los resultados obtenidos al aplicar la adaptación del proceso.

## 5.1 EL PROYECTO

El proyecto plantea la alternativa que representan las metodologías ágiles para las pequeñas y medianas industrias de desarrollo de software de la región, para ello se propone un proceso de desarrollo a seguir, el cual es programación extrema alimentado con elementos de modelamiento ágil. Con el fin de servir como referente a quienes estén interesados en implementar el proceso en entornos similares se lleva a cabo una aplicación siguiendo dicho proceso propuesto y se dan a conocer los resultados obtenidos.

## 5.2 EL PRODUCTO SIGA

La aplicación desarrollada es un Sistema de Información para Granjas Agroindustriales –SIGA-, cuyo objetivo es ser útil en las labores de administración y gestión de recursos del Parque Temático Agroindustrial de la Universidad del Cauca, ubicado en la vereda La Rejoya. Este caso de estudio, surge de la necesidad de contar con una herramienta que facilite el manejo de información relacionada con los distintos proyectos y productos existentes, los registros de producción, ventas e inventario.

Se decidió que la solución se desarrollaría en lenguaje de programación Java, aprovechando que éste es de libre distribución y el cierto conocimiento previo por parte de los programadores debido a otras prácticas anteriores con este lenguaje de programación. Además Java es un lenguaje bastante usado actualmente, lo cual facilitaría futuras actualizaciones del software. Dentro de los requisitos del cliente se encuentra el almacenamiento y la posterior recuperación de datos, por lo cual se decidió utilizar bases de datos en MySQL, dado que es el motor más usado por las organizaciones de la industria del software.

### 5.2.1 Especificaciones de la aplicación

La primera definición de los objetivos y características de la aplicación se hizo en una reunión con el cliente, en este caso el administrador de la granja agroindustrial de la universidad. El proceso a seguir plantea que el cliente presente por escrito lo que el espera sea desarrollado, por tanto se solicitó al cliente un documento que contenga los requisitos de SIGA, se reprodujo copia del manuscrito al final de la monografía.

Dada la vaguedad de los requerimientos consignados, y, pretendiendo tener los requerimientos lo más claros posibles, lo mas viable fue que los programadores escribieran las especificaciones a partir de las respuestas a las preguntas que se le hicieron al cliente en las reuniones, encaminadas a aclarar las dudas que generó el primer documento. En resumen, las funcionalidades que se esperaban del programa, recolectadas tanto del documento recibido como de las respuestas del cliente fueron:

- El programa busca tener control eficiente y manejo seguro y rápido de cada uno de los microproyectos que se ejecutan y clasificarlos según el caso en agrícolas o pecuarios.

- Cada uno de los microproyectos debe contar con un registro de egresos, que incluye gastos de inversión, mantenimiento, producción y otros gastos. Así mismo debe contar con un registro de ingresos, a partir de los cuales es posible obtener la utilidad de cada microproyecto, en una fecha o un rango de fechas determinada por el usuario. De la misma manera, se debe contar con la posibilidad de calcular los

egresos, ingresos y utilidades de la totalidad de los microproyectos existentes, dando lugar así al balance anual (P y G), el cual es de gran importancia para el usuario.

- Se debe poseer registros de inventario, tanto para bienes e inmuebles, como para bienes agropecuarios (animales, plantas y todo lo relacionado como alimentos, vacunas, entre otros.).

- El usuario debe tener la posibilidad de generar reportes con la información necesaria para la Vicerectoría administrativa de la Universidad del Cauca, los cuales, por solicitud del usuario se crearon en formato Excel

- Debe existir la posibilidad de añadir los nuevos microproyectos que la administración decida poner en ejecución, o de eliminar del sistema los microproyectos que la administración decida dar por finalizados.

### 5.2.2. Análisis

El proceso de desarrollo indica que los requisitos deben ser escritos por el cliente a través de historias de usuario, (pero que en este caso, fue necesario que los desarrolladores las extrajeran)

- Historias de usuario: como se dijo, las escribe el cliente en su lenguaje propio, sin términos técnicos, también son útiles para las estimaciones de tiempo en la reunión de planeación del lanzamiento. El formato que se empleó para escribir las historias de usuario corresponde al mostrado en el apartado Proceso programación extrema documentado con modelamiento ágil, dicho formato es el mostrado en la figura 2, página 26. Las historias de usuario a desarrollar se encuentran en el Anexo 2.

### 5.2.3 Diseño

En XP, a diferencia de las metodologías tradicionales, no se pide un diseño lo más completo posible previo a la codificación, lo que se hace es acordar un diseño preliminar para lo que se implementará en cada iteración. En XP intentar hacer optimizaciones en diseño para contemplar funcionalidades que se implementarán en iteraciones futuras no se considera

una buena práctica. Este concepto en diseño es opuesto al observado por otras metodologías conocidas, pero tiene su fundamento en que se considera que solo el 10% de las soluciones previstas para el futuro realmente se utilizan, por ello lo adecuado es no cargarse de funcionalidades adicionales que provocan retrasos y malgastarán recursos.

Lo que se pretende es que esta etapa sea corta y que el diseño se vaya depurando en el momento de la codificación. Las dos herramientas principales de ayuda en esta fase, propuestos por XP son la metáfora del sistema y tarjetas CRC, que se describen a continuación.

#### 5.2.3.1 Metáfora del sistema

La metáfora del sistema es un elemento muy útil para optimizar la comunicación entre todos los integrantes del grupo de trabajo, al crear una visión global y común del sistema a desarrollar. La metáfora se debe expresar en términos de dominio para todos los integrantes del equipo, esto se puede lograr comparando lo que se va a desarrollar con algo que se presente en la vida real. El objetivo de la metáfora del sistema es llegar a la forma más simple de estructurar el sistema. Esta actividad se realiza en una reunión de todo el equipo de trabajo para que todos los integrantes tengan la idea general y todos puedan proponer ideas de posibles metáforas y escoger al final la metáfora mas indicada.

En reunión de los miembros del proyecto se acordó que la metáfora más apropiada eran los libros de contabilidad, donde se lleva un control de ingresos, egresos y utilidades a la fecha.

#### 5.2.3.2 Tarjetas CRC

Las tarjetas CRC (Class, Responsability and Collaboration- Clase, Responsabilidad y colaboración) se utilizan para facilitar la comunicación y documentar los resultados. El proceso para llenar estas tarjetas es el siguiente: Se identifican las clases y se especifican sus finalidades (responsabilidad), así como otras clases con las que interactúe (colaboración). La forma de trabajo es la siguiente: Se realiza una reunión del personal encargado del desarrollo del proyecto, en la que los miembros aportan ideas de las clases y sus funcionalidades, de éste modo se van agregando las tarjetas, (cuando una funcionalidad ya está dentro de una clase, dicha funcionalidad no se repite), al final se organizan las

tarjetas y lo que se tiene plasmado es algo similar a lo que seria un diagrama de clases inicial.

### 5.2.4 Desarrollo

En XP el desarrollo se hace por iteraciones, estas son fases en las que se divide el proceso de desarrollo, que cumplen con ciertas funcionalidades definidas en el plan de lanzamiento. Al finalizar cada iteración estás se entregan al cliente y éste las acepta o sugiere posibles modificaciones; cada una de las iteraciones se verán más adelante en el proceso XP.

### 5.2.5 Arquitectura

Se optó por una arquitectura de tres capas, cuyo esquema se muestra a continuación. Ver figura 5.1.

### 5.2.6 Pruebas

Tradicionalmente las pruebas se realizan al final, una vez se haya realizado la codificación, pero en XP las pruebas se realizan durante el desarrollo, cada vez que se vaya a implementar una funcionalidad o clase se escribe una prueba para dicha clase y a continuación se la codifica, una vez se ha pasado la prueba se continua con otra y se repite el mismo proceso con todas las funcionalidades. Esta manera de aplicar las pruebas unitarias contribuye a codificar pensando previamente en las pruebas y contribuye a disminuir los errores y posibilita la realimentación inmediata. Otro aporte de la metodología es que ya no son necesarios dos equipos diferenciados para desarrollo y pruebas cada uno por su cuenta. Ahora el ciclo se basa en implementar una prueba unitaria, codificar la solución y pasar la prueba, con lo que se consigue un código simple y funcional de manera bastante rápida. Por eso es importante que las pruebas se pasen siempre al 100% [16].

Cuando las funcionalidades ya han sido implementadas el cliente realiza la prueba de aceptación, que consiste en verificar que el sistema cumple con todas las funcionalidades, en caso de existir alguna falla el cliente decide la prioridad con que éstas deben resolverse.

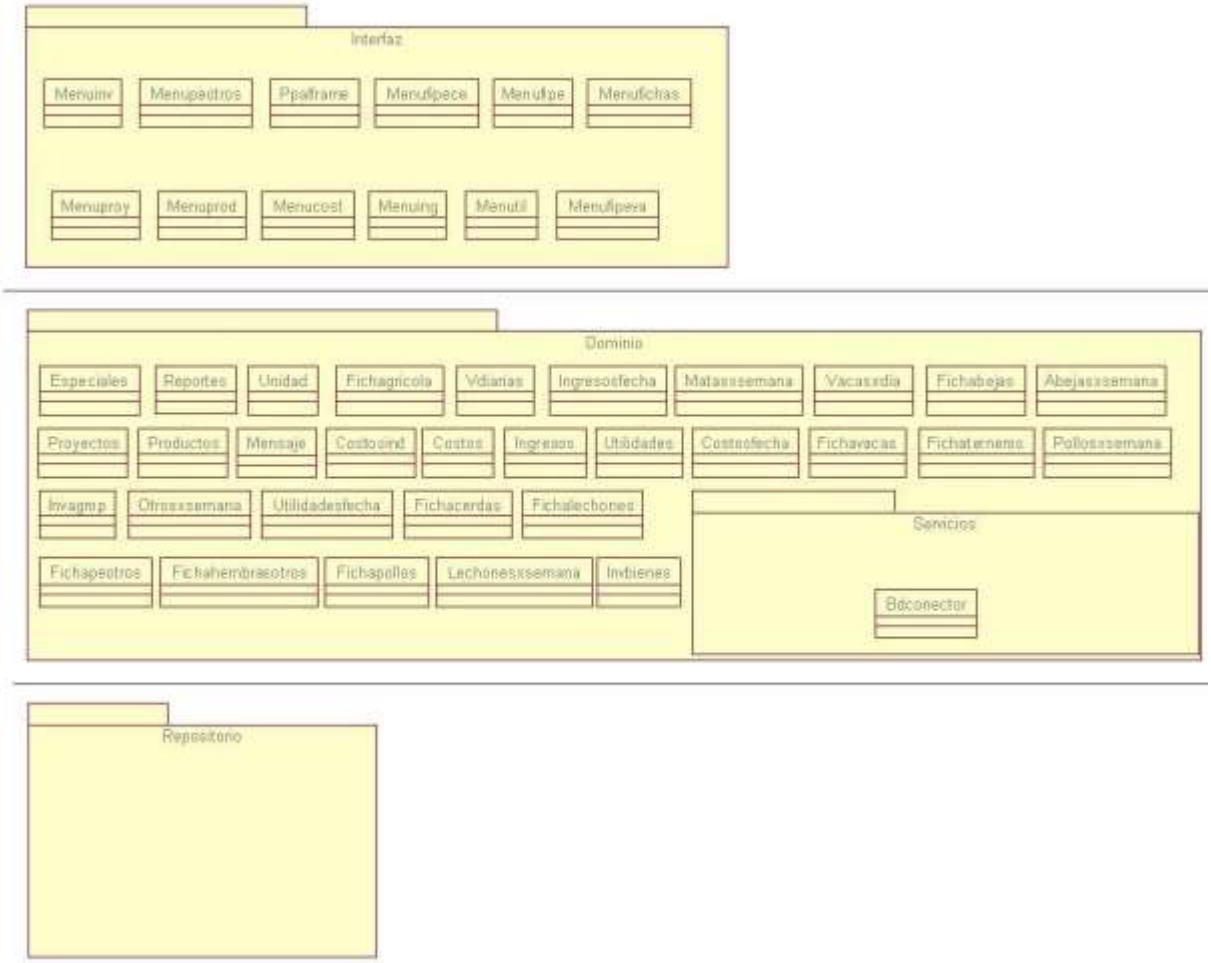



**Figura 5.1 Diagrama de arquitectura en capas**

## 5.3 El proceso XP alimentado con modelamiento ágil

A continuación se describe la aplicación y el conocimiento adquirido, producto del modelo propuesto, programación extrema documentado con modelamiento ágil.

### 5.3.1 Fase de Exploración

Esta fue una de las fases que mayor tiempo consumió, ya que en este caso, implicó una etapa previa de recopilación y asimilación de información acerca de las metodologías a utilizar en el proyecto: sus características, prácticas, principios, valores y el estado del arte de estas metodologías hasta la fecha. Una vez realizado la anterior se está en capacidad de

efectuar las actividades propias de la fase de exploración de programación extrema. Esta fase busca recopilar los requisitos de la aplicación a desarrollar y explorar el camino más adecuado de darle solución. Para ello el proceso de programación extrema se vale de dos herramientas que se explican a continuación: historias de usuario y spike arquitectónico. Los integrantes y roles durante esta fase se resumen en la siguiente tabla:

| **Miembros y Roles Fase de Exploración** | | | |
|---|---|---|---|
| **Miembro** | **Grupo** | **Rol XP** | **Metodología** |
| Paola Andrea Manquillo | A | Programador/Tester | XP |
| Julio Ariel Hurtado | A | Director/entrenador | XP |
| Jesús Emilio Rivera | A | Programador/Tester | XP |
| Carlos Quintín | A | Cliente | XP |
| Francisco Pino | A | Consultor | XP |

**Tabla 5.1: Miembros y Roles Fase de Exploración**

#### 5.3.1.1 Historias de Usuario

Las historias de usuario surgen de una reunión del cliente con el equipo de desarrollo, el fin es obtener los requisitos del cliente en sus propias palabras y cómo deben suplirse. Los autores de XP sostienen que las historias de usuario deben ser escritas por el cliente en su lenguaje, inicialmente se intentó hacerlo de ese modo, pero los desarrolladores detectaron que el cliente no tenía ninguna experiencia en la formulación de requisitos para proyectos de desarrollo de software, tampoco conocía los alcances y limitaciones de las herramientas de desarrollo y, por tanto al final las historias de usuario se fueron refinando a partir de las reuniones con el cliente. Un factor que puede afectar la fluidez de la comunicación y el entendimiento en la captura de requisitos es la naturaleza del proyecto, en este caso se trataba de la administración de una granja agropecuaria, algo con lo que los desarrolladores no estaban familiarizados, por ello en la práctica lo que ocurre es que algunas veces esos requisitos expresados por el cliente no se entienden desde el inicio por los desarrolladores.

La incomprensión entre el cliente y los desarrolladores se pudo convertir en un obstáculo que afecta etapas posteriores del proceso, por lo tanto es algo que se debe atender desde muy temprano, más en una metodología que sustenta algunas de sus prácticas en el valor de la comunicación. Para lograr un mejor entendimiento que se vea reflejado en un

refinamiento de las historias de usuario, se hizo la depuración de las historias de usuario recibidas inicialmente del cliente directamente en la granja, conociendo en vivo como funcionan cada uno de los microproyectos, esto realmente fue de gran ayuda.

La metodología aplicada admite el cambio en los requisitos como bienvenido. En este proyecto los requisitos iniciales presentaron cambios en el transcurso del desarrollo de las iteraciones, estas modificaciones en los requisitos implicaban entonces una nueva definición de pruebas para las nuevas funcionalidades y un nuevo desarrollo de la funcionalidad. Una de las prácticas que hace que los cambios no resulten costosos para las organizaciones es la entrega frecuente de versiones del prototipo en funcionamiento, es allí donde el cliente comprueba exactamente si los requisitos plasmados reflejan exactamente sus necesidades, las entregas frecuentes también posibilitan que las correcciones al código se hagan sobre la ultima versión de código producida, y no sobre el producto totalmente terminado, cuando se ha finalizado todo el desarrollo, pues entre más avanzado el desarrollo es mayor el coste de los cambios.
Otra práctica importante en el proceso de depuración de requisitos es la presencia del cliente en el sitio de trabajo. Más presencia del cliente o su representante posibilita una mayor claridad de los requisitos, lo cual es sumamente útil ya que evita hacer correcciones a historias de usuario desarrolladas, pero en este caso no fue posible contar siempre con el cliente de forma presencial.

### 5.3.2 Fase de planeamiento

En esta fase fue necesario adaptar el proceso XP que proponen los autores, la razón por la que no se siguió el proceso propuesto por los creadores de XP al pie de la letra es porque los desarrolladores de la aplicación no eran desarrolladores de profesión y no contaban con registros históricos que sean de utilidad para llevar a cabo unas estimaciones acertadas para la implementación de cada historia de usuario. Además estas estimaciones que proponen los autores se hacen sobre supuestos bastante ideales que en la práctica no estaban en capacidad de cumplir: Los autores sostienen que se debe estimar el tiempo de implementación de cada historia de usuario en "días ideales de programación", un día ideal de programación es aquel en que el desarrollador hace su trabajo sin interrupciones y sabe exactamente qué hacer. Durante esta fase los roles asignados son iguales a los de la fase anterior, mostrados en la tabla 5.1.

El proceso XP adaptado a este caso de estudio procedió así:

#### 5.3.2.1 Plan de lanzamientos

En el plan de lanzamientos se especifica exactamente que historias de usuario van a ser implementadas en cada una de las iteraciones, incluye una estimación de la duración de cada una de las historias en términos de días ideales de programación, la prioridad de cada una de ellas y las fechas en que se realizaran las entregas de los prototipos correspondientes a cada iteración. La diferencia del plan de lanzamientos realizado con el propuesto por los autores es que no se realizaron estimaciones para cada una de las historias de usuario por las razones anteriormente expuestas, se realizaron estimaciones para el conjunto de funcionalidades a implementarse en cada iteración y se fijaron la fecha de las entregas y así se obtuvo el plan de lanzamientos mostrado en la tabla 5.2.
Para la elaboración del plan de lanzamientos se procedió así: se tomaron las historias de usuario obtenidas de la etapa anterior y se realizó la reunión de planeación de lanzamientos, o sea, una reunión entre el cliente y desarrolladores para acordar la prioridad de cada una de las historias de usuario, y la iteración en la que cada una de ellas será implementada. Se asigna un conjunto de historias de usuario para que el cliente elija durante la reunión de planificación de la iteración lo que se pondrá en ejecución durante la iteración siguiente.

Algo que se evidenció a partir de este plan de lanzamientos, es que es prácticamente imposible crear una planificación invariable de las entregas, esta plan aquí mostrado presentó modificaciones al igual que debieron ser agregadas nuevas historias de usuario y se modificaron algunas ya existentes. Cuando el cambio en los requisitos es algo normal dentro del proceso de desarrollo, más valioso que realizar un plan perfecto es buscar estrategias que permitan ajustar los planes definidos desde el inicio y prepararse para responder a los cambios. Con XP esto se consigue ya que las entregas frecuentes permiten al cliente descubrir si lo que se está implementado es lo que realmente necesita o se requieren modificaciones, y a los desarrolladores esto les permite realizar los cambios en etapas tempranas no al final como en las metodologías tradicionales.

| | **Fecha de Inicio** | **Fecha de finalización** |
|---|---|---|
| **Iteración 1** | 25 abril 2005 | 10 de junio de 2005 |
| **Iteración 2** | 13 junio de 2005 | 22 de julio de 2005 |
| **Iteración 3** | 25 de julio de 2005 | 2 de septiembre de 2005 |
| **Iteración 4** | 5 septiembre de 2005 | 14 de octubre de 2005 |
| **Iteración 5** | 18 octubre de 2005 | 25 noviembre de 2005 |
| **Iteración 6** | Marzo 23 de 2006 | 4 de julio de 2006 |

**Tabla 5.2: Plan de lanzamientos -SIGA**

#### 5.3.2.2 Spike Arquitectónico

El spike es una herramienta válida para explorar las diferentes alternativas en cuanto a tecnología a utilizar, elección del lenguaje de programación y selección del sistema gestor de base de datos necesario. Se evalúan las distintas alternativas con experimentos dinámicos de código que se realizan para encontrar cómo se solucionaría el problema.
Al realizar pequeños prototipos se observó que se obtenían resultados satisfactorios al emplear el lenguaje de programación java y el sistema gestor de base de datos My Sql. Además de validarse la conveniencia de la herramienta para el desarrollo de la aplicación también se tuvo en cuenta que ya eran conocidas y que son de código abierto.

Los desarrolladores consideraron de gran utilidad el spike arquitectónico porque permite vislumbrar la viabilidad de una alternativa de solución desde una etapa muy temprana.

#### 5.3.2.3 Mitigación de riesgos

Con el propósito de visualizar los posibles riesgos y guiar al equipo de trabajo si la situación de riesgo se llega a presentar se incluye esta sección donde se clasifican los distintos riesgos. En la tabla 5.3 se resume la lista de riesgos.

| **Lista de riesgos** | | | |
|---|---|---|---|
| **Descripción** | **Impacto** | **Responsabilidad** | **Contingencia** |
| Riesgo de no conseguir los requisitos correctos: el cliente puede omitir detalles o características deseables del sistema | Los programadores trabajarán implementando funcionalidades que no van a ser de satisfacción para el cliente. Las pruebas y los prototipos obtenidos al final de cada iteración no serán satisfactorios | Equipo de trabajo, cliente | Identificar e implementar historias de usuario clave para identificar si los requisitos son correctos. |
| Riesgo de incluir nuevas herramientas en el ciclo de desarrollo | El desarrollo puede verse retrasado, y probablemente se afecte el plan de lanzamientos | Programadores | Capacitar a los programadores, buscar alternativas entre las herramientas conocidas |
| Riesgo de estimación en la duración de las iteraciones | Posible incumplimiento en las entregas previstas | Programadores y cliente | Poner en práctica el juego de la planificación y lograr nuevos acuerdos con el cliente |
| Riesgo de no presencia del cliente en el sitio de trabajo | Desarrollo, pruebas funcionales, plan de lanzamientos se ven afectados | Cliente | Desplazarse en la medida de lo posible hacia donde se encuentra el cliente, otros medios de comunicación distintos del persona a persona |
| Riesgo de comunicación escasa entre el cliente y los programadores | Puede incidir en cualquier fase del proceso | Cliente y equipo de trabajo | Crear espacios o circunstancias para una mayor comunicación abierta y eficaz. |

**Tabla 5.3: Lista de riesgos-SIGA**

### 5.3.3 Fase de iteraciones

Esta fase comprende un número de iteraciones que depende del tamaño y la complejidad del proyecto, a mayor tamaño y complejidad son necesarias un mayor número de iteraciones para lograr el producto final. Esta división en iteraciones permitió obtener un entregable al final de cada una de ellas, y trajo consigo ventajas como la reducción de la complejidad, control sobre el tiempo de desarrollo, una constante realimentación entre los requisitos y el prototipo al final de cada una de las fases, también posibilita hacer un seguimiento y una gestión del tiempo y la calidad en el desarrollo.

En esta fase se desarrollaron las historias de usuario de acuerdo al plan de lanzamientos, definido en la etapa previa. El desarrollo iterativo puesto en práctica proporcionó realimentación y agilidad al proceso de desarrollo. Teniendo en cuenta que el proceso

propuesto incluyó prácticas de programación extrema y modelamiento ágil, se decidió adoptarlas de forma gradual, se partió inicialmente con prácticas pertenecientes sólo a programación extrema y a partir la segunda iteración se alimentó el proceso con prácticas de modelado ágil, lo anterior además permitió determinar cual es el efecto del modelado en el proceso de desarrollo.

#### 5.3.3.1 Primera iteración

Durante esta iteración se buscó implementar aquellas historias de usuario que permiten una visión general del sistema a desarrollar. En esta fase se conservó la asignación de roles de la fase anterior, mostrado en la tabla 5.1.

**Resultados obtenidos primera iteración:** La tabla 5.4 resume las historias de usuario desarrolladas durante esta fase, para un mayor detalle de las historias de usuario puede consultar el anexo 2.

| **Iteración 1** | |
|---|---|
| **Historias** | **Descripción** |
| 1 | Crear/eliminar Microproyectos estándar: El usuario ingresará los distintos microproyectos estándar con que cuenta la granja, con sus respectivos datos. Tareas que se llevaron a cabo:<br>-Diseño de interfaz menú principal.<br>-Diseño de interfaz Menú Proyectos, permite al usuario seleccionar el ingreso de proyectos estándar o especiales.<br>-Diseño de interfaz Proyectos Estándar<br>-Inserción y almacenamiento de los datos del proyecto en la base de Datos |
| 2 | Crear/eliminar Microproyectos especiales: El usuario ingresará los distintos microproyectos especiales con que cuenta la granja, con sus respectivos datos. Las tareas que se llevaron a cabo en esta historia<br>-Diseño de interfaz Proyectos Especiales<br>-Inserción y almacenamiento de los datos del proyecto en la base de Datos |
| 3 | Crear/eliminar Productos: El usuario ingresará cada uno de los productos extraídos en cada uno de los microproyectos. Las tareas que se llevaron a cabo en esta historia de usuario fueron:<br>-Diseño de menú productos<br>-Diseño de interfaz "crear productos"<br>-Inserción y almacenamiento de datos |
| 4 | Crear/eliminar Unidades: El usuario ingresará las unidades de medida en que se comercializan los productos. Tareas que se llevaron a cabo<br>-Diseño de interfaz "crear unidades"<br>-Inserción y almacenamiento de datos |

| | |
|---|---|
| 5 | Introducir costos individuales: Una vez el usuario ha creado proyectos, productos y unidades podrá calcular los costos individuales de cada uno de los productos de un determinado microproyecto. Tareas que se llevaron a cabo:<br>-Diseño de menú gastos<br>-Desarrollo de interfaz para ingreso de costos individuales<br>-Inserción y almacenamiento de costos individuales en la base de datos |
| 6 | Calcular costos totales: Una vez se han introducido costos individuales se pueden calcular los costos totales para un producto perteneciente a un microproyecto determinado. Tareas que se llevaron a cabo:<br>- Desarrollo de interfaz para cálculo de costos totales<br>- Lectura base de datos<br>- Obtención de costos totales y almacenamiento en base de datos |

**Tabla 5.4: Historias de usuario iteración 1**

En la primera iteración se aplicaron sólo prácticas de programación extrema, una metodología con la que no se tenía experiencia previa por lo que surgieron muchas dudas metodológicas, por eso en esta primera iteración fue en la que más tiempo se consumió. Al inicio de la iteración se llevo a cabo una reunión de planeación de iteración en la que se acordó una reunión semanal con el cliente o su representante, con el fin de mostrar los avances del proyecto y que a su vez el cliente expresara sus inquietudes o conformidad con el avance de la aplicación, desafortunadamente el cliente no asimiló la importancia de la comunicación como parte del equipo de trabajo y no se hizo presente a las reuniones.

**Cifras:**

Horas análisis y diseño: 18

Horas refactorización: 2

Horas codificación: 102

Horas pruebas: 62

Total horas hombre: 184

El diseño se realizó decidiendo las clases a implementar y asignando a cada una de ella las funcionalidades, esto ayudó a identificar si las clases que se piensan implementar en realidad tenían responsabilidades (una clase que no cuente con ellas debe eliminarse), y que a su vez no haya responsabilidades presentes en más de una clase.

Se definieron las pruebas al inicio de la iteración, las pruebas consistían en la verificación dinámica del comportamiento del software a partir de un conjunto de casos de prueba. Los casos de prueba son las posibles entradas que evalúan las funcionalidades definidas en las historias de usuario. Cuando las funcionalidades han sido desarrolladas se ejecuta la

aplicación para identificar posibles errores, los cuales se definen como un resultado distinto del resultado esperado [19]. Los errores se identifican en función de las pruebas unitarias definidas, por lo tanto, las pruebas solo garantizan la existencia o no de los errores que se buscan, es decir, una aplicación que pase todas las pruebas es una aplicación que no tiene ninguno de los errores que dichas pruebas buscan, pero puede tener otros errores. Al encontrarse un error, antes de intentar corregir se debe generar la prueba de aceptación relacionada, así se evita que dicho error se vuelva a presentar, y probablemente sea más fácil corregirlo. En caso de que no todas las pruebas de aceptación sean exitosas, se debe crear nuevas pruebas de aceptación en el inicio de la siguiente iteración, estas pruebas de aceptación fallidas se suman a las pruebas de aceptación de las historias de usuario correspondientes a la siguiente iteración y constituyen las pruebas para la próxima iteración. Para ilustrar mejor el proceso de pruebas se expone como ejemplo la historia de usuario 1, ingreso/eliminación de microproyectos estándar, el proceso seguido es el siguiente:

**Descripción**

En este caso hay que comprobar la introducción correcta de los datos correspondientes a los microproyectos. Un microproyecto consta de múltiples datos y puede ocurrir que algunos sean válidos mientras que otros no. En el proceso se guardan sólo aquellos microproyectos que ingresen los datos correctos. Si los datos ingresados no son válidos, es decir se ingresa un tipo de dato distinto al válido, se genera un mensaje de error y el proyecto no será guardado y si los datos son incompletos se le solicita al cliente que introduzca los datos completos. Cuando la introducción de datos sea correcta debe comprobarse que sean almacenados en la base de datos.

**Ingreso correcto de microproyectos**

**Descripción**

El usuario (administrador agropecuario) una vez haya entrado en el sistema selecciona la opción "Proyectos" del Menú principal. En la ventana proyectos selecciona "Proyectos Estándar". Tras esa elección se le muestra la interfaz, en la que ingresa los datos correspondientes al proyecto, internamente se procesan los datos y si no hay ningún error (datos completos y válidos), el proyecto es guardado.

**Condiciones de ejecución**

Instalación y ejecución de SIGA

**Entrada**

-El Administrador Agropecuario seleccionará "proyectos" del Menú Principal.

-Del Menú Proyectos seleccionará "Proyectos Estándar"

-A continuación se muestra una interfaz donde debe ingresar los siguientes datos: Nombre del proyecto, seleccionar el tipo de proyecto (agrícola o pecuario), área del proyecto, dada ya sea en metros cuadrados o hectáreas, unidades cultivadas y fecha de inicio del microproyecto. Finalmente debe seleccionar la opción guardar para que el microproyecto esté almacenado en la base de datos,

-El proceso de ingreso de microproyectos estándar se considera como finalizado.

**Resultado Esperado**

El proyecto es almacenado en la base de datos y el usuario podrá verlo en la interfaz y, haciendo click en él, podrá visualizar los datos correspondientes a dicho microproyecto.

**Evaluación de la Prueba:** Prueba satisfactoria

Siguiendo este mismo procedimiento el resumen de las pruebas realizadas para esta iteración se muestran en la tabla 5.5. Las pruebas en detalle se pueden ver en el anexo 2.

**Lecciones primera iteración**

Es importante tener en cuenta que entre más numerosas son las pruebas, mayor es la calidad del sistema en desarrollo, pero también esto implicaría un mayor costo en tiempo y en recursos para las organizaciones, por ello el equipo en conjunto debe negociar las pruebas de aceptación de modo que no impliquen funcionalidades extras, o que conlleven una gran demanda de tiempo, de manera que pueda darse cumplimiento al contrato. Eran posibles otras pruebas adicionales, como generar un mensaje de error cada vez que ingresará un dato no válido en el ingreso de los microproyectos, pero dada la cantidad de trabajo extra que implicaba y que no era un requisito del cliente se descartó la idea de dicha prueba. Algo muy importante para quienes estén interesados en adoptar la programación extrema es familiarizar al cliente con la importancia de la comunicación y que

éste verdaderamente asuma que es parte activa del equipo del proyecto, ya que sin la oportuna presencia del cliente es muy difícil un análisis de requisitos completo y detallado, tampoco se pueden llevar a cabo las pruebas de aceptación, lo que afecta las entregas y todo el plan de entregas inicialmente elaborado. Dada la importancia que representa el desarrollo de las historias de usuario que aportan mayor valor al cliente (las de mayor prioridad), y ante la ausencia del cliente, la única alternativa fue que los desarrolladores se desplazaran hasta su sitio de trabajo, la granja agropecuaria de la Universidad.

| **Pruebas de aceptación iteración 1** | | |
|---|---|---|
| **Prueba** | **Resultado Esperado** | **Evaluación de la Prueba** |
| Introducción de microproyectos estándar con datos correctos | El microproyecto estándar es almacenado en la base de datos | Prueba satisfactoria |
| Introducción de microproyectos estándar con datos erróneos | Se produce un mensaje de error y el microproyecto estándar no es guardado | Prueba satisfactoria |
| Introducción de microproyectos estándar con datos incompletos | Se produce un mensaje que solicita introducir todos los datos | Prueba satisfactoria |
| Eliminación de microproyectos estándar | El microproyecto estándar es eliminado de la base de datos | Prueba satisfactoria |
| Introducción de microproyectos especiales con datos correctos | El microproyecto especial es almacenado en la base de datos | Prueba satisfactoria |
| Introducción de microproyectos especiales con datos erróneos | Se produce un mensaje de error y el microproyecto especial no es guardado | Prueba satisfactoria |
| Introducción de microproyectos especiales con datos incompletos | Se produce un mensaje que solicita introducir todos los datos | Prueba satisfactoria |
| Eliminación de microproyectos especiales | El microproyecto especial es eliminado de la base de datos | Prueba satisfactoria |
| Introducción correcta de producto | El producto es guardado en la base de datos | Prueba satisfactoria |
| Introducción incompleta de producto | Se produce un mensaje que solicita introducir todos los datos | Prueba satisfactoria |
| Eliminación de producto | El producto seleccionado es eliminado de la base de datos | Prueba satisfactoria |
| Introducción correcta de unidades | La unidad es guardada en la base de datos | Prueba satisfactoria |
| Introducción incompleta de unidades | Se produce un mensaje que solicita introducir todos los datos | Prueba satisfactoria |
| Eliminación de unidades | La unidad seleccionada es eliminado de la base de datos | Prueba satisfactoria |
| Introducción correcta de costos individuales | Los costos individuales son guardados en la base de datos | Prueba satisfactoria |
| Introducción errónea de costos individuales | Se despliega u mensaje de error y el costo individual no es guardado | Prueba satisfactoria |
| Introducción incompleta de costos individuales | Se despliega un mensaje que solicita ingresar todos los datos | Prueba satisfactoria |
| Cálculo correcto de costos totales | Se calcula el costo total y es mostrado en pantalla | Prueba satisfactoria |
| Cálculo incompleto de costos totales | Se produce un mensaje que solicita ingresar todos los datos | Prueba satisfactoria |

**Tabla 5.5: Pruebas de aceptación iteración 1**

Una de las prácticas que más cuestionamiento causó en el interior del grupo fue la de la programación por pares, en este caso particular, el par de programadores poseen distintos niveles de conocimiento y, empezó a ser evidente que la programación por pares no siempre es ventajosa, ya que la persona que está sentada al lado de quien está programando no tiene total claridad del código que se está desarrollando y tampoco está en capacidad de corregir los errores por eso para iteraciones posteriores se decidió replantear esta práctica y hacerla cuando realmente se considere que representa una ventaja.

#### 5.3.3.2 Segunda iteración

En esta iteración se aplicaron prácticas de modelamiento ágil dentro del proceso, y se continuó con el desarrollo de las historias de usuario. El modelamiento resultó bastante útil para la codificación, los modelos realizados se muestran en el anexo 2. El desarrollo de esta iteración fue más corto que la primera, esto en parte debido al conocimiento que se adquirió de la metodología y de las funcionalidades implementadas (del programa), después de una primera iteración. Se percibió que se desarrollan más historias de usuario en un tiempo menor que el de la primera iteración y esto induce a pensar también en lo complejo que puede ser la labor de cumplir un plan cuando se está trabajando con nuevas metodologías, pues la velocidad del proyecto es variable.

Hacia el final de la iteración cuando se debía hacer la entrega y se habían acordado horarios de reunión, el cliente no se hizo presente y fue retrasando aún más el desarrollo normal del proyecto. Ante la dificultad que representaba para el curso del proyecto la ausencia del cliente, en la adopción de un proceso en que es importante su presencia, se convocó una reunión con el Director del proyecto para definir la conveniencia de la continuidad del proyecto. La opción más viable fue que los desarrolladores se desplazaran hasta el sitio de trabajo del cliente, lo cual se hizo y fue la única manera de poder entregar la versión correspondiente a la segunda iteración. La asignación de roles durante esta iteración fue la mostrada en la tabla 5.6.

| **Miembros y roles segunda iteración** | | | |
|---|---|---|---|
| **Miembro** | **Grupo** | **Rol XP, AM** | **Metodología** |
| Paola Andrea Manquillo | A | Programador, Modelador | XP, AM |
| Julio Ariel Hurtado | A | Director/entrenador | XP, AM |
| Jesús Emilio Rivera | A | Tester, Modelador | XP, AM |
| Carlos Quintín | A | Cliente | XP |
| Francisco Pino | A | Consultor | XP, AM |

**Tabla 5.6: Miembros y roles segunda iteración**

**Resultados segunda iteración:**

**Cifras:**

Horas análisis y diseño: 18
Horas refactorización: 3
Horas codificación: 68
Horas pruebas: 46
Total horas hombre: 135

Para modelamiento ágil existen múltiples opciones en cuanto a los artefactos a utilizar, todos ellos válidos, por eso corresponde a los modeladores decidir cuáles se usarán para el modelado. Los artefactos escogidos para el modelado de SIGA fueron diagramas de secuencias de UML, con los que se busca comunicar de manera simple lo que se pretende implementar a otros que no hacen parte del equipo.

Las tablas 5.7 y 5.8 resumen las historias de usuario desarrolladas y algunas de las pruebas que se llevaron a cabo durante esta iteración. Para mayor detalle tanto de pruebas como de historias de usuario consulte el anexo 2.

| **Iteración 2** | |
|---|---|
| **Historia** | **Descripción** |
| 7 | Introducir ingresos individuales: Una vez el usuario ha creado proyectos, productos y unidades podrá calcular los ingresos individuales de cada uno de los productos de un determinado microproyecto. Las tareas fueron:<br>-Diseño de menú Ingresos<br>-Desarrollo de interfaz para entrada de ingresos individuales<br>-Inserción y almacenamiento de ingresos individuales en la base de datos |
| 8 | Calcular ingresos totales: Una vez se han introducido ingresos individuales se pueden calcular los ingresos totales para un producto perteneciente a un microproyecto determinado. Tareas que se llevaron a cabo:<br>- Desarrollo de interfaz para cálculo de ingresos totales<br>- Lectura base de datos<br>- Obtención de ingresos totales y almacenamiento en base de datos |
| 9 | Calcular costos por fecha: El administrador agropecuario podrá calcular los costos de un determinado producto de un microproyecto, para un período de tiempo comprendido entre una ficha inicial y una fecha final escogidos por Él mismo. Las tareas fueron:<br>- Desarrollo de interfaz para cálculo de costos por fecha<br>- Lectura base de datos<br>- Obtención de costos por fecha y almacenamiento en base de datos |
| 10 | Calcular ingresos por fecha: El administrador agropecuario podrá calcular los ingresos de un determinado producto de un microproyecto, para un período de tiempo comprendido entre una ficha inicial y una fecha final escogidos por Él mismo. Las tareas fueron:<br>- Desarrollo de interfaz para cálculo de ingresos por fecha<br>- Lectura base de datos<br>- Obtención de ingresos por fecha y almacenamiento en base de datos |
| 11 | Calcular utilidades totales: Una vez calculados los costos e ingresos totales es posible calcular las utilidades totales para un producto perteneciente a un determinado microproyecto. Las tareas necesarias fueron:<br>- Desarrollo de interfaz para cálculo de utilidades<br>- Desarrollo de interfaz para cálculo de utilidades totales<br>- Lectura a base de datos<br>- Obtención de utilidades totales y almacenamiento en base de datos |
| 12 | Calcular utilidades por fecha: El administrador agropecuario podrá calcular las utilidades de un determinado producto de un microproyecto, para un período de tiempo comprendido entre una ficha inicial y una fecha final escogidos por él mismo. Las tareas fueron:<br>- Desarrollo de interfaz para cálculo de utilidades por fecha<br>- Lectura base de datos<br>- Obtención de utilidades por fecha y almacenamiento en base de datos |

**Tabla 5.7 Historias de usuario iteración 2**

Al mismo tiempo que se avanzó en la implementación de las historias de usuario se adelantó también en la implementación de la base de datos, ya que la configuración de la base de datos va ligada a la codificación de las historias de usuario.

### Lecciones segunda iteración

El hecho de modelar tiene un impacto positivo en la codificación de las historias de usuario, si bien la primera iteración incluía historias de usuario de poca complejidad, en ésta la complejidad era un poco mayor y el modelamiento proporcionó una mayor precisión en los requerimientos del cliente y a través del modelamiento se describen también de forma más

completa los aspectos del prototipo a entregar en esta iteración (en comparación a cuando no se utilizó modelamiento, como en la primera iteración).

| **Pruebas de aceptación iteración 2** | | |
|---|---|---|
| **Prueba** | **Resultado Esperado** | **Evaluación de la Prueba** |
| Introducción correcta de ingresos individuales | Los ingresos son guardados en la base de datos | Prueba satisfactoria |
| Introducción errónea de ingresos individuales | Se produce un mensaje de error y los ingresos no son guardados | Prueba satisfactoria |
| Cálculo correcto de ingresos totales | Se calcula el total de los ingresos | Prueba satisfactoria |
| Cálculo incompleto de ingresos totales | Se produce un mensaje que solicita ingresar todos los datos | Prueba no satisfactoria |
| Cálculo correcto de costos por fecha | Se calculan los costos comprendidos entre las fechas ingresadas | Prueba no satisfactoria |
| Cálculo correcto de ingresos por fecha | Se calculan los ingresos entre las fechas seleccionadas | Prueba satisfactoria |
| Cálculo correcto de utilidades totales cuando costos totales son distintos de cero e ingresos totales son iguales a cero | Calcula correctamente el valor de la utilidad | Prueba satisfactoria |
| Cálculo de utilidades cuando costos son mayores que ingresos, e ingresos mayores que cero | Calcula correctamente el valor de la utilidad | Prueba satisfactoria |
| Cálculo de utilidades cuando ingresos son mayores que costos y costos diferentes de cero | Calcula correctamente el valor de la utilidad | Prueba satisfactoria |
| Cálculo de utilidades cuando ingresos son mayores que costos y costos iguales a cero | Calcula correctamente el valor de la utilidad | Prueba satisfactoria |
| Cálculo correcto de utilidades por fecha | Se calculan las utilidades entre las fechas seleccionadas | Prueba satisfactoria |
| Cálculo de Utilidades por fecha, cuando ingreso de datos es incompleto | Se despliega un mensaje que solicita ingresar todos los datos | Prueba satisfactoria |

**Tabla 5.8: Pruebas de aceptación iteración 2**

En la primera iteración se expresó lo inconveniente que puede resultar la programación en parejas cuando el par de programadores no tiene el mismo nivel de conocimiento, por ello para esta iteración decidimos experimentar el desarrollo de forma diferente. Puede decirse que la principal ventaja de la práctica de programación por pares que es producir código de mayor calidad, desaparece cuando quien está sentado observando al lado de quien está

programando, no tiene el conocimiento suficiente para corregir los posibles errores que puede cometer quien está programando, por ello se decidió que uno de los desarrolladores se dedicaría a la codificación de las historias de usuario y el otro se encargaría de las pruebas de aceptación. En este caso particular el resultado de trabajar individualmente resultó muy apropiado, ya que al final de haber realizado todo el desarrollo de la iteración en parejas (codificar las historias y probarlas), se hubiese gastado el mismo tiempo pero con dos personas, eso se traduce en 228 horas hombre, pero al hacerlo de forma individual se realizó en la mitad del tiempo, o sea, 114 horas hombre.

Durante la mayor parte del transcurso de la iteración, no fue posible dar total cumplimiento a la práctica del cliente en el sitio de trabajo, por lo tanto se recurrió a la comunicación telefónica en circunstancias en que se pretendía una mayor claridad en ciertos requisitos. Aunque es obvio que hay aspectos del desarrollo que no se pueden aclarar vía telefónica, como el proceso de pruebas de aceptación al final de cada iteración, para lo cual el cliente se hizo presente, la comunicación telefónica es aceptable en la medida que se logre expresar correctamente con palabras aquellos aspectos del sistema que generan dudas, la mayoría relacionados con la interfaz de usuario. No se logró dar cumplimiento a todos los requisitos del cliente, por eso quedaron dos historias de usuario con pruebas de aceptación fallidas, con las que se inició la iteración siguiente.

#### 5.3.3.3 Tercera iteración

Esta iteración comenzó por intentar resolver las pruebas de aceptación no satisfactorias de la segunda iteración, además se codificaron las historias de usuario previstas para la tercera iteración desde el plan de lanzamientos. Inicialmente el cliente plasmó en los requisitos que se necesitaba que en los proyectos bovinos y porcinos se manejara cada bovino y cada porcino como una sola unidad de producción, o sea como si cada animal fuera un proyecto y no un producto, pero cuando se terminó de programar las historias de usuario y se hizo la entrega del prototipo correspondiente el cliente cambió los requisitos, para entonces el cliente solicitó lo siguiente: los bovinos conforman cada uno un solo proyecto productivo y necesitan que se lleve un control diario de los costos e ingresos de cada animal y que los porcinos se manejan por cocheras o "camadas".

**Resultados tercera iteración**

Para efectos de simplificar la documentación no se documentaron las diferentes versiones que tuvieron las historias de usuario y las pruebas, por ello la tabla 5.9 muestra el resumen de las historias de usuario finales que se llevaron a cabo en esta iteración, análogamente la tabla 5.9, muestra el resumen de las pruebas de aceptación finales. Debido a los cambios solicitados por el cliente, esta iteración fue una de las más largas, ya que se programó en su totalidad las historias de usuario pero, al momento de la entrega, el cliente cambio los requisitos, estos cambios implicaron que gran parte del código fuera desechado e implementado nuevamente.

La implementación de la base de datos de acuerdo a las especificaciones del cliente ha requerido buena parte de la iteración, además sufrió bastantes cambios que tuvieron efecto en las historias de usuario posteriores a los cambios. Se aplicó modelamiento ágil, ya que modelar el diseño es de gran ayuda para la codificación, más aun cuando la implementación de los requisitos demanda bastante esfuerzo. La asignación de roles fue la misma que se presentó durante la segunda iteración. La tabla 5.9 presenta un resumen de las historias de usuario implementadas y la tabla 5.10 muestra el resumen de pruebas, para verlas en detalle puede consultar el anexo 2.

**Cifras**

Horas análisis y diseño: 28
Horas refactorización: 3
Horas codificación: 106
Horas pruebas: 66
Total horas hombre: 203

Durante esta iteración se continuó con el desarrollo individual, uno de los programadores implementó las historias de usuario y el otro se encargó de las pruebas, ya que esto ha significado una mayor productividad, aunque la cifra del total de horas hombre aumentó bastante con respecto a la iteración anterior no se debe el ritmo de trabajo haya disminuido sino a que fue necesario rehacer algunas funcionalidades debido al cambio de los requisitos.

| **Iteración 3** | |
|---|---|
| **Historia** | **Descripción** |
| 13 | Introducción de fichas técnicas agrícolas: el usuario podrá introducir y almacenar los datos correspondientes a las fichas técnicas agrícolas. Las tareas para esta historia fueron:<br>-Diseño de menú fichas técnicas<br>-Diseño de interfaz de usuario para ingreso de datos fichas agrícolas.<br>-Inserción de datos en la interfaz y almacenamiento en la base de datos. |
| 14 | Ingreso y almacenamiento de datos fichas técnicas porcinos, etapa lactancia: El usuario podrá ingresar y almacenar los datos correspondientes a la ficha técnica del proyecto porcinos en su etapa de lactancia. Tareas:<br>-Diseño de menú fichas técnicas pecuarias<br>-Diseño de menú fichas técnicas porcinos<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica porcinos etapa lactancia<br>-Inserción de datos y almacenamiento de los mismos |
| 15 | Ingreso y almacenamiento de datos fichas técnicas porcinos, otras etapas: El usuario podrá ingresar y almacenar los datos correspondientes a la ficha técnica del proyecto porcinos en otras etapas (distintas de la etapa de lactancia). Tareas:<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica porcinos en otras etapas<br>-Inserción de datos y almacenamiento de los mismos |
| 16 | Introducción y almacenamiento de datos fichas técnicas vacas lecheras: El usuario podrá ingresar y almacenar los datos correspondientes a la ficha técnica del proyecto vacas lecheras. Tareas<br>-Diseño de menú bovinos<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica vacas lecheras<br>-Inserción de datos y almacenamiento de los mismos |
| 17 | Introducción y almacenamiento de datos fichas técnicas vacas lecheras, registro diario: El usuario podrá ingresar y almacenar los datos correspondientes a la ficha técnica de cada animal del proyecto vacas lecheras diariamente. Tareas realizadas<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica vacas lecheras por día<br>-Inserción de datos y almacenamiento de los mismos |

**Tabla 5.9 Historias de usuario iteración 3**

**Lecciones tercera iteración**

El desarrollo de esta iteración dejó claro la importancia y la necesidad de contar con la presencia del cliente, es de anotar que lo ocurrido aquí no fue un error de interpretación de los requisitos del cliente.

Las historias de usuario a desarrollar estaban aprobadas y estaba claro lo que el cliente necesitaba, y esto fue lo que se desarrolló en la primera versión de la iteración. En el transcurso de la iteración el cliente decidió que para él era más conveniente un manejo no individual sino por corrales y por etapas de los porcinos y el cliente sólo lo comunicó en el momento de la entrega, porque en su imaginario suponía que ese era un cambio menor. Como se mencionó no fue posible contar con la presencia del cliente en vivo en el sitio de trabajo, sino que se le llamaba por teléfono cuando surgían dudas, por ello el contacto con el cliente fue cuando se terminó de programar las historias de usuario.

| **Pruebas de aceptación iteración 3** | | |
|---|---|---|
| **Prueba** | **Resultado Esperado** | **Evaluación de la Prueba** |
| Ingreso correcto de fichas técnicas agrícolas | La ficha técnica es guardada en la base de datos | Prueba satisfactoria |
| Introducción de fichas técnicas agrícolas con datos erróneos | Se despliega un mensaje de error y la ficha técnica no es guardada | Prueba satisfactoria |
| Ingreso incompleto de datos en fichas técnicas agrícolas | Se despliega un mensaje que solicita ingresar todos los datos | Prueba Satisfactoria |
| Ingreso de datos correctos en fichas técnicas porcinos etapa lactancia | La ficha técnica es guardada en la base de datos | Prueba satisfactoria |
| Introducción de fichas técnicas porcinos etapa lactancia con datos erróneos | Se despliega un mensaje de error y la ficha técnica no es guardada | Prueba satisfactoria |
| Ingreso incompleto de datos en fichas técnicas porcinos etapa lactancia | Se despliega un mensaje que solicita ingresar todos los datos | Prueba satisfactoria |
| Eliminación ficha técnica porcinos etapa lactancia | La ficha técnica es almacenada en la base de datos | Prueba satisfactoria |
| Ingreso de datos correctos en fichas técnicas porcinos otras etapas | La ficha técnica es guardada en la base de datos | Prueba satisfactoria |
| Ingreso de datos erróneos en fichas técnicas porcinos otras etapas | Se despliega un mensaje de error y la ficha técnica no es guardada | Prueba satisfactoria |
| Ingreso de datos incompletos en fichas técnicas porcinos otras etapas | Se despliega un mensaje que solicita ingresar todos los datos | Prueba satisfactoria |
| Eliminación ficha técnica porcinos otras etapas | La ficha técnica es almacenada en la base de datos | Prueba satisfactoria |
| Ingreso de datos correctos en fichas técnicas vacas lecheras | La ficha técnica es guardada en la base de datos | Prueba satisfactoria |
| Ingreso de fichas técnicas vacas lecheras con datos incompletos | Se despliega un mensaje que solicita ingresar todos los datos | Prueba satisfactoria |
| Eliminación ficha técnica vacas lecheras | Eliminación ficha técnica vacas lecheras | Prueba satisfactoria |
| Ingreso de datos correctos en registro diario de vacas lecheras | El registro diario es guardada en la base de datos | Prueba satisfactoria |
| Ingreso de datos erróneos en registro diario de vacas lecheras | Se despliega un mensaje de error y la ficha técnica no es guardada | Prueba satisfactoria |
| Ingreso de datos incompletos en registro diario de vacas lecheras | Se despliega un mensaje que solicita ingresar todos los datos | Prueba satisfactoria |

**Tabla 5.10: Pruebas de aceptación Iteración 3**

La anterior situación deja claro la importancia del valor de la comunicación cuando existen cambios en los requisitos, el cliente no debería esperar hasta el momento final, el de la entrega de un prototipo para comunicar que el requerimiento se ha modificado porque el tiempo y los recursos humanos se están destinando a producir algo que finalmente será desechado. Una suposición establecida en ingeniería del software es que el coste de los cambios en un programa crece exponencialmente con el tiempo [20], XP busca que los

cambios no tengan un costo tan elevado y por ello la razón de ser de sus prácticas que combinadas en su totalidad lograrían el objetivo. Los desarrolladores consideran que el inconveniente aquí no fue en gran parte por la ausencia del cliente, pues las dos primeras iteraciones se realizaron del mismo modo y se consiguió implementar correctamente lo que el cliente buscaba, la falla fue la escasa comunicación del cliente hacia el equipo de trabajo y su percepción equivocada del bajo impacto que tienen los cambios en la labor de desarrollo, no se puede decir que esta sea una falla de la metodología, porque XP enuncia la comunicación abierta y franca entre todos los miembros del equipo, e implementa prácticas para que se realice (programación por parejas, propiedad colectiva del código, cliente en el sitio de trabajo, entregas frecuentes), el asunto es que cada proyecto tiene un escenario propio que limita que todas esas prácticas se puedan llevar a cabo, para este caso el cliente por razones laborales no podía permanecer en el lugar donde se desarrolló el trabajo de grado.

El costo en tiempo de esta pobre comunicación fue de casi el 40% del tiempo total de la iteración, muy considerable para el total de la iteración, pero hubiese sido muchísimo más elevado si se está trabajando con alguna de las metodologías tradicionales donde sólo se hace una entrega del producto terminado al final de todo el proceso de desarrollo. Para evitar que una situación similar se vuelva a producir se le explicó al cliente que un pequeño cambio por pequeño que parezca a nivel de programación puede ser fundamental y que es necesario que se comuniquen a la mayor brevedad posible.

Se negoció con el cliente la corrección de las historias de usuario que presentan errores y por ello estas quedan pendientes. Se continuó aplicando modelamiento ágil bajo los mismos parámetros de la iteración anterior y la labor de desarrollo no se hace en parejas porque dadas las características del par de programadores esto nos estaba representando una baja productividad, por ello se continuó dividiendo las labores de desarrollo y cada uno hace su labor y se complementa con el otro. Esta iteración se prolongó mucho más de lo previsto, inicialmente porque el cliente no se presentaba a realizar las pruebas de aceptación, entonces ante la incertidumbre de la aceptación de la versión no se podía continuar, cuando finalmente el cliente se presentó, comunicó a los desarrolladores el cambio de requisitos, con todo lo que ello implica: cambiar historias de usuario, eliminar parte del código, programar las nuevas historias de usuario, cambios en la base de datos, nuevas pruebas.

#### 5.3.3.4 Cuarta Iteración

Inicialmente se había previsto que todo el proyecto se realizaría en 5 iteraciones pero al terminar la tercera iteración e iniciar la cuarta el cliente adicionó nuevas historias de usuario. Anteriormente no estaba previsto que se llevara una contabilidad del inventario de los bienes e insumos de la granja ni tampoco se había solicitado un inventario agropecuario, algo adicional que el cliente pidió fue tener la posibilidad de imprimir las utilidades de cada proyecto, junto con el detalle de los costos, todo esto en formato Excel para que él pueda presentarlo como un informe al final de cada mes, a estas funcionalidades adicionales se suma que se tenían historias pendientes por corregir, por ello se replanteó el plan de lanzamientos a 6 iteraciones.

**Resultados cuarta iteración**

En esta iteración se conservó la misma asignación de roles con la que se trabajó desde la iteración 2. Las funcionalidades que debían satisfacerse requerían de un gran esfuerzo de programación pero los programadores consideran que se hizo de una forma muy conveniente para el cliente, de modo que se le da total libertad en el manejo de las especies menores que podrá administrar a través de SIGA, inicialmente él cliente sugirió que las especies menores fueran conejos y cuyes, pero en el futuro pueden ser otras más, por ello se decidió que la mejor manera de hacerlo era acceder a otras especies desde el menú pecuarios y una vez en la interfaz de otras especies, el usuario tenga la flexibilidad de crear o eliminar cuantas especies menores desee. Las historias de usuario desarrolladas en esta iteración se muestran en la tabla 5.11, para verlas completas consulte el anexo 2.

**Cifras**

Horas análisis y diseño: 16
Horas refactorización: 2
Horas codificación: 60
Horas pruebas: 42
Total horas hombre: 120

| **Iteración 4** | |
|---|---|
| **Historia** | **Descripción** |
| 18 | Introducción y almacenamiento de datos fichas técnicas terneros: El usuario podrá ingresar y almacenar los datos correspondientes a la ficha técnica del proyecto terneros. Tareas<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica terneros<br>-Inserción de datos y almacenamiento de los mismos |
| 19 | Introducción y almacenamiento de datos fichas técnicas aves de postura: El usuario podrá ingresar y almacenar los datos correspondientes a la ficha técnica de cada galpón. Tareas<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica aves de postura<br>-Inserción de datos y almacenamiento de los mismos |
| 20 | Introducción y almacenamiento de datos fichas técnicas aves de postura/semana: El usuario podrá ingresar y almacenar los datos correspondientes a la ficha técnica de cada galpón semanal. Tareas<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica aves de postura por semana.<br>-Inserción de datos y almacenamiento de los mismos |
| 21 | Introducción y almacenamiento de datos fichas técnicas proyecto apiarios: El usuario podrá ingresar y almacenar los datos correspondientes a cada uno de los productos del proyecto apiarios. Tareas realizadas<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica proyecto apiarios.<br>-Inserción de datos y almacenamiento de los mismos |
| 22 | Introducción y almacenamiento de datos fichas técnicas proyecto apiarios/labores semanales: El usuario podrá ingresar y almacenar los datos correspondientes a cada uno de los productos del proyecto apiarios. Tareas realizadas<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica proyecto apiarios.<br>-Inserción de datos y almacenamiento de los mismos |

**Tabla 5.11 Historias de Usuario Iteración 4**

La tabla 5.12 resume algunas de las pruebas de aceptación realizadas durante esta iteración, para mayor detalle consulte el anexo 2.

Para este punto ya no era posible cumplir con el plan de lanzamientos previsto, no sólo por las nuevas historias agregadas sino porque uno de los miembros del equipo de desarrollo entró a trabajar de tiempo completo y eso afectó drásticamente el desempeño en un equipo de desarrollo conformado por dos personas, por tal motivo se negoció con el cliente que en la fecha prevista para la entrega final del producto se le entregaría una versión del prototipo que cumplía con más del 80% de los requisitos.

**Lecciones cuarta iteración**

Después de un tiempo trabajando con la metodología el equipo de desarrollo descubrió sus fortalezas y debilidades y así se logró una asignación de labores de desarrollo más adecuada para el aumento de la productividad, aunque en algunos casos la programación por pares es una opción válida para aumentar la calidad del código producido. Al adoptar la metodología poco a poco se apreció que no todas las labores ameritan hacerse en pares, y que en últimas lo mejor es que cada organización adopte la metodología con una disposición a adaptarla

según su entorno: tamaño, características del personal, tipo de proyectos que maneja, etc. Esto le permitirá un mejor aprovechamiento de sus recursos y mayor eficiencia.

| **Pruebas de aceptación iteración 4** | | |
|---|---|---|
| **Prueba** | **Resultado Esperado** | **Evaluación de la Prueba** |
| Ingreso correcto de datos en fichas técnicas terneros | Ficha técnica es guardada en base de datos y podrá ser visualizada | Prueba satisfactoria |
| Ingreso de datos erróneos en ficha técnica terneros. | Se produce un dato de error, por ingreso de dato no valida y la ficha técnica no es guardada | Prueba satisfactoria |
| Eliminación de ficha técnica terneros | La ficha técnica es eliminada de la base de datos | Prueba satisfactoria |
| Ingreso correcto de datos en fichas técnicas aves de postura. | Ficha técnica es guardada en base de datos y podrá ser visualizada. | Prueba satisfactoria |
| Ingreso de datos erróneos en ficha técnica aves de postura. | No se guarda dicha ficha técnica | Prueba satisfactoria |
| Eliminación de ficha técnica aves de postura | La ficha técnica es eliminada de la base de datos | Prueba satisfactoria |
| Ingreso correcto de datos en fichas técnicas aves de postura/registro semanal. | Ficha técnica es guardada en base de datos y podrá ser visualizada. | Prueba satisfactoria |
| Ingreso erróneo de datos en ficha técnica aves de postura/registro semanal. | Se despliega un mensaje que sugiere introducir los datos correctos y la ficha técnica no es guardada. | Prueba satisfactoria |
| Ingreso correcto de datos en fichas técnicas apiarios | Ficha técnica es guardada en base de datos y podrá ser visualizada. | Prueba satisfactoria |
| Ingreso erróneo de datos en ficha técnica apiarios. | No se guarda dicha ficha técnica | Prueba satisfactoria |
| Eliminación de ficha técnica apiarios | La ficha técnica es eliminada de la base de datos | Prueba satisfactoria |
| Ingreso correcto de datos en fichas técnicas apiarios/labores realizadas. | Ficha técnica es guardada en base de datos y podrá ser visualizada. | Prueba satisfactoria |
| Ingreso incompleto de datos en ficha técnica apiarios/labores semanales. | Se muestra en pantalla un mensaje que sugiere al usuario introducir todos los datos. | Prueba satisfactoria |

**Tabla 5.12: Pruebas de aceptación Iteración 4**

Algo importante de XP es que desde un inicio se trabaja en dar solución a aquellas necesidades que dan mayor valor al cliente. Es el cliente quien sugiere que historias de usuario se desarrollan primero, esta es una ventaja no sólo para el cliente sino también para los desarrolladores, que en caso de no poder cumplir con el plan de lanzamientos pueden entrar a negociar un nuevo plan, con la ventaja que el cliente ya conoce los distintos prototipos y que seguramente aunque no todo está listo para la fecha, si estará aquello que es más urgente para él, eso fue lo que ocurrió en este caso de estudio y eso permitió negociar la fecha de entrega del producto final. Esto evidencia una ventaja de XP: la flexibilidad y la capacidad de negociar un plan de lanzamientos.

Para este momento se había obtenido un buen conocimiento de la aplicación que se está desarrollando y se obtiene la solución más simple que satisface los requerimientos, así mismo se planeó la iteración a dos entregas, así de logramos un contacto más frecuente con el cliente. Esto hizo que hacia al final de la iteración se tuviera muy claro lo que el cliente buscaba. Se continuó haciendo el modelado de la base de datos y diseño de interfases previo a la codificación, estas tareas no se hicieron en grupo, porque ya se conocían las habilidades de cada desarrollador.

#### 5.3.3.5 Quinta iteración

Durante esta iteración fue necesario aumentar el ritmo de trabajo, eso debido a que en la iteración anterior, por razones laborales, no se pudo disponer de uno de los desarrolladores y por ello se redujo la intensidad de trabajo. Esta reducción en la velocidad de trabajo implicó negociar un nuevo plan de lanzamientos, en el plan acordado, se concedió más tiempo para la entrega del producto final si se finalizaba con los requisitos de esta iteración para antes del 20 de enero de 2006.

**Resultados quinta iteración:**

Durante esta iteración se conservó la asignación de roles vigente desde la segunda iteración. Para lograr finalizar la programación de las historias de usuario dentro del tiempo estimado se solicitó al cliente una mayor presencia y, se planeó además una entrega intermedia para asegurar una mayor presencia del cliente y que efectivamente no se hayan presentado cambios en los requisitos.

Las historias de usuario desarrolladas se muestran en la tabla 5.13. y en la tabla 5.14 se muestran las pruebas de aceptación correspondientes a la iteración 5 Estas se muestran en detalle en el anexo 2.

| **Iteración 5** | |
|---|---|
| **Historia** | **Descripción** |
| 23 | Introducción y almacenamiento de datos ficha técnica hembras de cría otras especies: El usuario podrá ingresar y almacenar los datos correspondientes a la ficha técnica de hembras de cría de otras especies diferentes a las ya tratadas. Tareas relacionadas.<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica otras especies<br>-Inserción de datos y almacenamiento de los mismos |
| 24 | Introducción y almacenamiento de datos ficha técnica hembras de cría otras especies/labores semanales. El usuario podrá ingresar y almacenar los datos correspondientes al registro de la ficha técnica de hembras de cría de otras especies diferentes a las ya tratadas. Tareas relacionadas.<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica otras especies/labores semanales<br>-Inserción de datos y almacenamiento de los mismos |
| 25 | Introducción y almacenamiento de datos ficha técnica crías otras especies: El usuario podrá ingresar y almacenar los datos correspondientes a la ficha técnica de crías otras especies diferentes a las ya tratadas. Tareas relacionadas.<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica otras especies<br>-Inserción de datos y almacenamiento de los mismos |
| 26 | Introducción y almacenamiento de datos ficha técnica crías otras especies/labores semanales. El usuario podrá ingresar y almacenar los datos correspondientes al registro de la ficha técnica de crías otras especies diferentes a las ya tratadas. Tareas relacionadas.<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica otras especies/labores semanales<br>-Inserción de datos y almacenamiento de los mismos |
| 27 | Introducción y almacenamiento de datos bienes e insumos. El usuario podrá ingresar y almacenar los datos correspondientes a los bienes e insumos que se encuentran en la finca y el estado en que éstos se encuentran. Tareas realizadas:<br>-Diseño de menú inventarios<br>-Diseño de interfaz de usuario para ingreso de datos bienes e insumos.<br>-Inserción de datos y almacenamiento de los mismos |
| 28 | Introducción y almacenamiento de datos inventario agropecuario: El usuario podrá ingresar y almacenar los datos correspondientes al inventario agropecuario. Tareas realizadas<br>-Diseño de interfaz de usuario para ingreso de datos inventario agropecuario<br>-Inserción de datos y almacenamiento de los mismos. |

**Tabla 5.13: Historias de usuario Iteración 5**

### Cifras

Horas análisis y diseño: 14

Horas refactorización: 2

Horas codificación: 56

Horas pruebas: 39

Total horas hombre: 111

| **Pruebas de aceptación iteración 5** | | |
|---|---|---|
| **Prueba** | **Resultado Esperado** | **Evaluación de la Prueba** |
| Ingreso correcto de datos en fichas técnicas hembras de cría otras especies | Ficha técnica es guardada en base de datos y podrá ser visualizada. | Prueba satisfactoria |
| Ingreso erróneo de datos en ficha técnica hembras de cría otras especies. | Se produce un mensaje de error y no se guarda dicha ficha técnica | Prueba satisfactoria |
| Eliminación de ficha técnica hembras de cría otras especies | La ficha técnica seleccionada es eliminada de la base de datos | Prueba satisfactoria |
| Ingreso correcto de datos en fichas técnicas hembras de cría otras especies/labores semanales | Ficha técnica es guardada en base de datos y podrá ser visualizada. | Prueba satisfactoria |
| Ingreso incompleto de datos en ficha técnica hembras de cría otras especies, labores semanales. | Se muestra en pantalla un mensaje que sugiere al usuario introducir todos los datos. | Prueba satisfactoria |
| Ingreso correcto de datos en fichas técnicas crías otras especies. | Ficha técnica es guardada en base de datos y podrá ser visualizada. | Prueba satisfactoria |
| Ingreso erróneo de datos en ficha técnica crías otras especies | Se produce un mensaje de error y la ficha técnica no es guardada | Prueba satisfactoria |
| Eliminación de ficha técnica crías otras especies. | La ficha técnica seleccionada es eliminada de la base de datos. | Prueba satisfactoria |
| Ingreso correcto de datos en fichas técnicas crías otras especies/labores semanales. | Ficha técnica es guardada en base de datos y podrá ser visualizada | Prueba satisfactoria |
| Ingreso incompleto de datos en fichas técnicas crías otras especies/labores semanales | Se muestra en pantalla un mensaje que sugiere al usuario introducir todos los datos. | Prueba satisfactoria |
| Ingreso correcto de datos en fichas técnicas bienes e insumos | Ficha técnica es guardada en base de datos y podrá ser visualizada. | Prueba satisfactoria |
| Ingreso incompleto de datos en fichas técnicas bienes e insumos | Se muestra en pantalla un mensaje que sugiere al usuario introducir todos los datos. | Prueba satisfactoria |
| Ingreso erróneo de datos en fichas técnicas bienes e insumos | No se guarda dicha ficha técnica | Prueba satisfactoria |
| Eliminación de datos inventario bienes e insumos | Los datos del bien seleccionado son eliminados de la base de datos | Prueba satisfactoria |
| Ingreso correcto de datos en fichas técnicas inventario agropecuario | Ficha técnica es guardada en base de datos y podrá ser visualizada. | Prueba satisfactoria |
| Ingreso incompleto de datos en fichas técnicas inventario agropecuario | Se muestra en pantalla un mensaje que sugiere al usuario introducir todos los datos. | Prueba satisfactoria |
| Ingreso erróneo de datos en fichas técnicas inventario agropecuario. | No se guarda dicha ficha técnica | Prueba satisfactoria |
| Eliminación de inventario agropecuario | El elemento seleccionado del inventario agropecuario es eliminado de la base de datos | Prueba satisfactoria |

**Tabla 5.14: Pruebas de aceptación Iteración 5**

### Lecciones quinta iteración

A medida que transcurre el tiempo de utilización de la metodología y se avanza en el desarrollo de la aplicación se logra un mayor grado de conocimiento que hace más eficientes los grupos de trabajo, por ejemplo, se pudo observar que incorporar entregas intermedias

del entregable permite certificarse de que se está desarrollando exactamente lo que el cliente necesita, y en caso que no sea así, se descubre mucho más temprano que hay correcciones por hacer. Esto debido a que no se puede contar con el cliente en el sitio de trabajo, las organizaciones que puedan contar con el cliente en el sitio de trabajo cuentan con una gran ventaja en ese sentido.

El haber enfocado el esfuerzo de programación desde las primeras iteraciones en crear un producto flexible, ha permitido que la aplicación pueda adaptarse a los cambios finales que ha sugerido el cliente sin mayores traumatismos, para esta iteración se desarrolló una historia de usuario que no estaba prevista en el plan de lanzamientos, y era una ficha técnica para labores semanales en proyectos porcinos.

Aunque se dio cumplimiento a los requisitos planteados por el cliente en las historias de usuario, se generaron "bugs" que también deben ser corregidos, esta fue una labor realizada en la sexta iteración.

Para compensar la reducción del ritmo de trabajo en la iteración anterior se trabajó durante dos semanas alrededor de 50 horas semanales, algo que puede ser visto como una contravención a la práctica de cuarenta horas semanales. XP acepta un aumento de horas semanales trabajadas si esto no se hace por mas de dos semanas, en el presente caso de estudio fue necesario hacerlo así para dar cumplimiento al plan de lanzamientos, además no se originó una mayor cantidad de errores de programación que es lo que XP busca evitar, argumentando que después de ocho horas de programación diarias, se generaran errores producto del agotamiento de los desarrolladores. La estrategia empleada para minimizar el impacto de posible sobrecarga de trabajo fue distribuir las horas adicionales laboradas en los días de descanso.

#### 5.3.3.6 Sexta iteración

En el desarrollo de esta iteración se presentaron algunos eventos que afectaron el normal transcurso de la misma. Estos hechos fueron: el computador en el se desarrollaba la tesis pasó a manos de la granja agropecuaria, por tanto hubo que tramitar la asignación de otro, lo cual sólo fue posible cuando hubo una libre que cumpliera las características requeridas. Además, El Director del proyecto debió ausentarse para continuar estudios de doctorado

fuera del país, por tanto se hace necesaria la búsqueda de alguien idóneo que lo reemplace, por tanto, hubo cambios en la asignación de roles dentro del proyecto, como se muestra en la tabla 5.15.

| **Miembros y roles segunda iteración** | | | |
|---|---|---|---|
| **Miembro** | **Grupo** | **Rol XP** | **Metodología** |
| Paola Andrea Manquillo | A | Programador, Modelador | XP |
| Carlos Alberto Ardila | A | Director/entrenador | XP |
| Jesús Emilio Rivera | A | Tester, Modelador | XP |
| Carlos Quintín | A | Cliente | XP |
| Francisco Pino | A | Consultor | XP |

**Tabla 5.15: Miembros y roles segunda iteración**

**Resultados sexta iteración**

La tabla 5.16 presenta un resumen de las historias de usuario desarrolladas durante la sexta iteración, note que entre estas historias de usuario se encuentran dos que son correcciones que estaban pendientes desde la segunda iteración, las restantes son historias de usuario que se fueron agregando en la fase final del desarrollo. Esta iteración tuvo un desarrollo bastante lento, puesto que nuevamente uno de los desarrolladores entró a trabajar, y a eso se le suma que ya se habían tenido retrasos por falta de computador y por la ausencia de Director del proyecto.

**Cifras**

Horas análisis y diseño**:** 12
Horas refactorización: 18
Horas codificación: 68
Horas pruebas: 37
Total horas hombre: 135

| **Iteración 6** | |
|---|---|
| **Historia** | **Descripción** |
| 29 | El usuario podrá ingresar y almacenar los datos correspondientes a la ficha técnica de cada cerda de cría y su camada entre un par de fechas determinadas. Tareas:<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica porcinos, etapa lactancia por fechas.<br>-Introducción y almacenamiento de datos fichas técnicas porcinos, etapa lactancia por fechas.<br>-Inserción de datos y almacenamiento de los mismos. |
| 30 | Introducción y almacenamiento de datos fichas técnicas porcinos, otras etapas por fechas: El usuario podrá ingresar y almacenar los datos correspondientes a la ficha técnica de cada etapa entre un par de fechas determinadas. Tareas:<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica porcinos, otras etapas por fechas.<br>-Inserción de datos y almacenamiento de los mismos. |
| 31 | Agregar calendario: El usuario dispondrá de un calendario para ingreso de fechas, de modo que se reduce la posibilidad de errores en el ingreso de fechas en formato java, que era como se hacia inicialmente |
| 32 | Introducción de fichas técnicas agrícolas por fechas: El usuario podrá ingresar y almacenar los datos correspondientes a la ficha técnica de cada cultivo entre un par de fechas determinadas. Tareas:<br>-Diseño de interfaz de usuario para ingreso de datos ficha técnica agrícola por fechas.<br>-Inserción de datos y almacenamiento de los mismos. |
| 33 | Generación de reportes en formato Excel de costos individuales: El sistema debe estar en capacidad de presentar el registro de costos individuales y el detalle de los costos. Esto le permite al usuario imprimir dicho reporte con lo que podrá sustentar sus informes a la Vicerectoría Administrativa. |
| 34 | Generación de reportes en formato Excel de ventas diarias: El usuario tendrá acceso a un reporte en formato Excel, que muestra los productos vendidos, la fecha de venta y el valor de la venta. Una vez generado, el usuario imprimirlo con lo que soportará sus informes a la Vicerectoría Administrativa. |
| 35 | Generación de reportes en formato Excel de utilidades para un producto determinado: El informe generado mostrará las utilidades totales para todos los productos, a la fecha. También se mostrarán aquí los costos totales y los ingresos totales a la fecha |

**Tabla 5.16: Historias de usuario iteración 6**

La tabla 5.17, resume las pruebas de aceptación realizadas durante esta fase.

### Lecciones sexta iteración

Al iniciar esta iteración se le instaló al cliente un prototipo funcional de la aplicación de modo que él pudiera familiarizarse con el sistema y tuviera la oportunidad de habituarse a su manejo. Del proceso de interacción del cliente con el sistema surgieron sugerencias para mejorar la aplicación, pero debido a las limitaciones de tiempo no se estaba en capacidad de dar solución a todos los nuevos requerimientos, ya que algunos implicaban cambiar la lógica de programación, y eso era muy altamente inconveniente en esta fase del proyecto, por eso, estos nuevos requisitos fueron negociados con el cliente y aquellos a los que no se le da solución en esta versión de la aplicación, quedan como alternativa para trabajo futuro con la metodología.

| **Pruebas de aceptación iteración 6** | | |
|---|---|---|
| **Prueba** | **Resultado Esperado** | **Evaluación de la Prueba** |
| Ingreso de datos correctos en fichas técnicas porcinos etapa lactancia por fechas | La ficha técnica es guardada en la base de datos y el usuario podra visualizarla en la interfaz. | Prueba satisfactoria |
| Ingreso de datos erróneos en fichas técnicas porcinos etapa lactancia por fechas | Se despliega un mensaje en pantalla que solicita ingresar los datos correctos. | Prueba satisfactoria |
| Ingreso de datos incompletos en fichas técnicas porcinos etapa lactancia por fechas | Se despliega un mensaje en pantalla que solicita ingresar todos los datos | Prueba satisfactoria |
| Ingreso de datos correctos en fichas técnicas porcino otras etapas por fechas | La ficha técnica es guardada en la base de datos y el usuario podra visualizarla en la interfaz. | Prueba satisfactoria |
| Ingreso de datos erróneos en fichas técnicas porcinos otras etapas por fechas . | Se despliega un mensaje en pantalla que solicita ingresar los datos correctos. | Prueba satisfactoria |
| Ingreso de datos incompletos en fichas técnicas porcinos otras etapas por fechas. | Se despliega un mensaje en pantalla que solicita ingresar todos los datos. | Prueba satisfactoria |
| Introducción y almacenamiento de fechas a través del calendario. | La fecha introducida a través del calendario es guardada en la base de datos. | Prueba satisfactoria |
| Ingreso de datos correctos en fichas técnicas agrícolas por fechas | La ficha técnica es guardada en la base de datos y el usuario podra visualizarla | Prueba satisfactoria |
| Ingreso de datos incompletos en fichas técnicas | Se despliega un mensaje en pantalla que solicita ingresar los datos correctos. | Prueba satisfactoria |
| Generar reporte en formato Excel de cotos individuales | Se genera un informe con los costos que contiene el valor y el detalle de los gastos. | Prueba satisfactoria |
| Generar reporte en formato Excel de ventas diarias | Se genera un informe con el detalle, el valor y la fecha de la venta | Prueba satisfactoria |
| Generar reporte en formato Excel de utilidades | Se produce un documento que contiene las utilidades de cada producto, el total de costos e ingresos. | Prueba satisfactoria |

**Tabla 5.17: Pruebas de aceptación iteración 6**

Durante esta etapa se incrementó la refactorización, ya que se había dado cumplimiento a los requisitos del cliente acordados, entonces la prioridad pasó a ser eliminar duplicidad, refinar el código, mejorar algunos aspectos de la interfaz de usuario y eso fue algo en lo que se invirtió un tiempo considerable, pues el mejor momento de hacer estas labores es cuando ya se han cumplido los requisitos.

Es importante anotar que en este proyecto no hubo un estricto proceso de contratación, de allí la posibilidad de negociación tan abierta que es propia de XP y que se puso en práctica también en esta fase.

En los procesos de contratación tradicionales el cliente fija unos requisitos, los desarrolladores proporcionaban un estimativo de los costos y tanto requisitos como costos quedan estipulados en un contrato, dando escaso margen de cambio en requisitos. La fecha de entrega se acordaba entre las dos partes, con lo que quedaban fijos, costos, tiempo y alcance del proyecto; quedando como variable la calidad, que sería la afectada en caso que los desarrolladores se sientan presionados para dar cumplimiento a los ítems del contrato. Esta manera en que se hacían las cosas generaba problemas, pues un producto que no cumple estándares de calidad, genera errores costosos, ya que estos son detectados tardíamente, con consecuencias inmediatas: clientes insatisfechos y desarrolladores que adquieren una mala reputación. El problema básico es que al fijar precio, fecha y alcance en el contrato, generará ruptura de intereses de ambas partes (clientes y desarrolladores) en Optional scope contracts Kent Beck Software first class [21] se resumen estas posiciones egoístas de las partes: el cliente interpreta los requisitos tan ampliamente como sea posible, para conseguir tanto como sea posible a bajo costo; el desarrollador interpreta los requisitos tan estrechamente como sea posible, para reducir recursos. El cliente desea que el trabajo sea realizado a la mayor brevedad posible, el desarrollador desea finalizar el trabajo a la fecha, para tener tiempo de conseguir el próximo contrato. El cliente desea máxima calidad, el desarrollador invierte en calidad justo lo que el cliente pagará.

De ahí surge la necesidad de otro tipo de contrato que equilibre los intereses de las partes, XP con su modelo de contrato, da prioridad a la calidad del producto y deja como opción variable el alcance del proyecto, que es negociado con los desarrolladores, lo cual no debe ser controversial, si se tiene en cuenta que en la mayoría de los casos el cliente no tiene claro lo que busca desde el inicio, pero es inaceptable para el que sepa cuanto va a costar y cuando va estar terminada su aplicación.

### 5.3.4 Fase de Producción

En este momento se considera que ya se ha finalizado la programación de las historias de usuario pero se permiten sugerencias del cliente en la medida en que el tiempo lo permita, en este caso se llegó a un acuerdo con el cliente, el cual consistió en asignar las nuevas sugerencias que se presenten en esta etapa para un trabajo futuro que puede ser llevado a cabo por un nuevo grupo de estudiantes.

Se logró dar cumplimiento a las funcionalidades del cliente que fueron dadas a conocer hasta antes de finalizar la última iteración, como imprimir algunas fichas técnicas, pero no con algunas que se hicieron durante la fase de producción, estas funciones que quedan pendientes para una nueva versión son las siguientes.

Permitir el ingreso de valores numéricos con punto de mil y coma de millón.

Transferencia automática de datos en los siguientes casos:
- El dato correspondiente a Número de unidades ingresado en proyectos agrícolas, transferirse a unidades sembradas de fichas técnicas agrícolas.
- El dato correspondiente a unidades vendidas en productos agrícolas, transferirse a unidades cosechadas en fichas técnicas agrícolas.

### 5.3.5. Fase de mantenimiento y muerte

En esta etapa se capacitó al cliente en la instalación y manejo de la aplicación, esta fue una labor sencilla, pues el cliente había interactuado previamente con el software durante el proceso de pruebas, lográndose el objetivo de esta fase que era la capacitación del cliente.

De forma paralela a la labor de capacitación del cliente se realizó la documentación del proceso de adaptación de la metodología en entornos pequeños, por ser este un caso de estudio, se consideró la realización de una documentación completa del proceso, que incluye las pruebas y las historias de usuario.

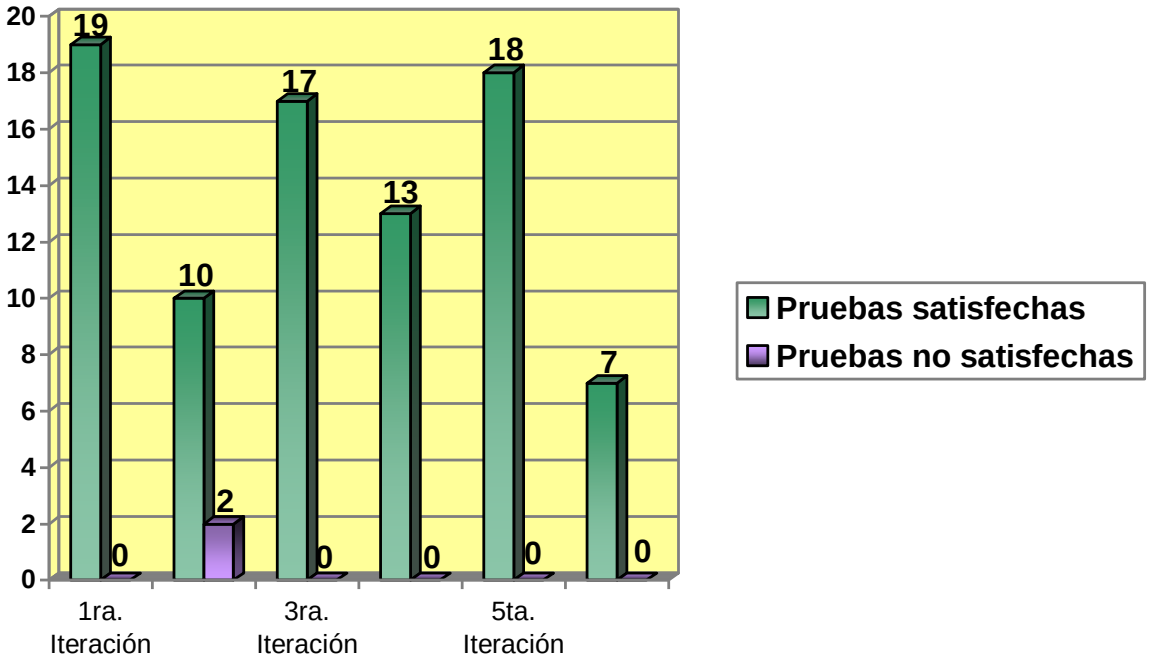
## 5.4 Resumen de pruebas

En el transcurso de la fase de iteraciones se realizaron pruebas cuyos resultados se muestran en la figura 5.1 y se resumen a continuación:

Iteración 1: Pruebas satisfechas 19, pruebas no satisfechas 0
Iteración 2: Pruebas satisfechas 10, pruebas no satisfechas 2
Iteración 3: Pruebas satisfechas 17, pruebas no satisfechas 0
Iteración 4: Pruebas satisfechas 13, pruebas no satisfechas 0

Iteración 5: Pruebas satisfechas 18 pruebas no satisfechas 0

Iteración 6: Pruebas satisfechas 7, pruebas no satisfechas 0




**F**

**Figura 5.2: Resumen de pruebas por iteración**

# 6. CONCLUSIONES Y TRABAJO FUTURO

## 6.1 Recomendaciones

A partir de la aplicación del proceso propuesto y considerando los aspectos de productividad, competitividad y calidad en el desarrollo de software en el Cauca se pueden resaltar:

**En cuanto a productividad:**

- La eficiencia con la cual se utilizan los factores de producción en los procesos de desarrollo es un aspecto crítico para incrementar la productividad en software. La eficiencia en las organizaciones regionales que se decidan por aplicar el proceso propuesto dependerá de varios factores:

  i. Conocimiento humano en el proceso y disposición a aplicarlo por parte de todos los miembros de la organización, por eso es recomendable una capacitación previa en el proceso, valores, prácticas y principios de programación extrema.

  ii. Calidad de los factores involucrados en el proceso de desarrollo de software: nivel de conocimiento de las herramientas de desarrollo por parte del equipo de programadores, entorno de trabajo que potencie la comunicación y realimentación abierta y fluida entre todos los miembros.

- Un factor determinante de la productividad en pequeñas organizaciones que no pueden asignar una persona diferente a cada uno de los roles, es una muy adecuada determinación de roles para cada miembro basada en el conocimiento y habilidades de cada integrante. Una superposición (más de un rol ejercido por una sola persona) de roles puede ser útil para que no queden trabajos sin hacer, pero esto produce una sobrecarga de trabajo en los miembros del equipo de desarrollo, que al ver incrementadas sus labores y responsabilidades, probablemente no puedan dar cumplimiento al cronograma previsto.

**En cuanto a calidad:**

- La aplicación del proceso propuesto incluye prácticas enfocadas a fomentar la calidad del software producido como la programación por pares, refactorización, pruebas y estándares de codificación.

**En cuanto a competitividad:**

- El proceso propuesto conjuga una serie de prácticas que al ser aplicadas adecuada y conjuntamente promueven agilidad al proceso de desarrollo (esto se refleja a través de entregas pequeñas y frecuentes, integración continua), adicionalmente. Al incentivar la reducción del tiempo de desarrollo y mejorar la calidad se está impulsando la competitividad.

- El efecto en la competitividad de las empresas regionales que se decidan en la aplicación del proceso de programación extrema dependerá básicamente de los logros alcanzados en aspectos críticos como la disminución de tiempos de desarrollo y el mejoramiento en los productos terminados, el avance en estos aspectos solo dependerá de la asimilación de las mejores prácticas del proceso de programación extrema.

## 6.2 Conclusiones

Se pueden diferenciar dos categorías de conclusiones, las que están relacionadas con todo el proceso y las que conciernen a sus prácticas específicas.

### 6.2.1 Conclusiones del proceso

- Se logró dar cumplimiento a los requisitos del cliente que proporcionan un software de gestión a la medida de sus necesidades, ofreciéndole flexibilidad ya que permite al usuario la utilización del software aunque se presenten cambios en la granja (puesta en marcha de nuevos proyectos o suspensión de algunos de los existentes), logrando así la satisfacción del cliente con el producto de un lado y de otro lado adquiriendo un

conocimiento-experiencia de la metodología implementada. El correcto funcionamiento del sistema se verificó durante el desarrollo de las iteraciones, mediante la aprobación de las pruebas de aceptación y al final, mediante las pruebas del sistema integrado, logrando así un alto grado de satisfacción de los requisitos inicialmente planteados.

- No existe una metodología capaz de hacer frente con éxito a todo tipo de proyecto de desarrollo de software. Las metodologías deben adaptarse al escenario del proyecto (recursos técnicos, recursos humanos, tiempo de desarrollo). A lo largo de los años, las metodologías tradicionales han intentado abarcar la mayor cantidad de situaciones y elementos del contexto del proyecto, demandando un enorme esfuerzo para ser aplicadas, en especial en proyectos que cuentan con pocas personas, ya que aplicar la rigurosa metodología exige numeroso personal. La programación extrema ofrece una alternativa para las organizaciones que tienen dificultad para adaptar metodologías tradicionales, a eso se debe su popularidad en los últimos años, pero deben tenerse en cuenta ciertos aspectos para su implementación exitosa.

- Aunque existen numerosos casos de adopción de programación extrema que han logrado reducción del tiempo de desarrollo y mejoramiento de calidad en los productos en empresas de Estados Unidos, lo cierto es que no se cuenta con información sobre el impacto real que la introducción de XP pueda tener en las empresas en los niveles de productividad, competitividad y calidad a nivel regional. Es necesario convertir la evidencia intuitiva que existe en indicadores claramente definidos, que permitan medir con claridad el impacto que se logre y a través de que elementos del proceso y prácticas se logra.

- Si bien la metodología es tolerante al cambio, los cambios rápidos en los requisitos generan dificultades, y es que hay una constante entrada de nuevas funcionalidades y es difícil saber cuando algo se puede dar por finalizado, esto además hace que los planes de entregas previstos tengan que ser modificados. Una forma de adaptarse al cambio es la realización de reuniones lo más frecuente posible con el cliente, pero esto no reduce el gran problema que es la cantidad de requerimientos que se ven incrementados asiduamente. Para mayor simplicidad en los informes, se decidió

documentar las historias de usuario y pruebas que corresponden al producto final y no cada uno de los diferentes prototipos que sufrieron modificaciones.

- Una limitación en este caso de estudio fue la división en tareas de las historias de usuario y su estimación en días ideales de programación, es una labor compleja con la que los desarrolladores no estaban habituados. Por un lado es necesario saber diferenciar y clasificar bien las tareas de programación, porque de lo contrario no es claro en cuál tarea se esta trabajando. Para aumentar más la inconveniencia de la labor, el cliente agregaba funcionalidades en el transcurso de la iteración, por lo que de haber existido algún estimativo previo de las tareas de programación, hubiese tenido que ser ajustado. Inicialmente se tuvo la intención de tomar como datos históricos los obtenidos en las tres primeras iteraciones, y así en las iteraciones siguientes realizar las estimaciones y calcular la velocidad del proyecto, pero se encontró que inicialmente, debido a no tener experiencia en seguir el proceso, el desarrollo fue lento, en la segunda iteración el ritmo aumentó pero en la tercera de nuevo fue lento debido al mayor esfuerzo que requerían las historias de usuario aquí implementadas. De acuerdo a lo observado, no hubo un patrón de comportamiento que marcara el ritmo de trabajo y por ello se decidió que así no convenía hacer estimativos en tareas de programación.

- Una gran dificultad para un proceso guiado por programación extrema es cuando el cliente no tiene una buena disposición para mantener comunicación con el equipo de desarrollo, es algo que se debe tener muy en cuenta en las organizaciones que estén interesadas en implementar procesos ágiles, ya que la poca comunicación del cliente es causante de males mayores, no informar cambios oportunos en los requisitos que se traduce en trabajo de desarrollo desperdiciado, no hacer presencia oportuna en las entregas de prototipo acordados que produce retrasos del plan de lanzamientos. Algunos autores aseguran que si el cliente no hace presencia en el sitio de trabajo, el desarrollador debe asumir lo que el cliente necesita, pero no es una buena idea porque entre las múltiples posibilidades a seguir en el curso del proyecto, solo una es la que satisface al cliente, eso quiere decir que existe la probabilidad de suponer aquello que el cliente no está deseando.

- Debido a las condiciones que impone XP para su correcta implementación, parece más adecuado para ser aplicada en entornos empresariales que en entornos académicos, pues es mucho más fácil convenir un horario de programación por pares, cuando el personal de desarrollo labora en una empresa, que cuando el personal corresponde a estudiantes que pueden tener compromisos académicos adicionales. Por otra parte, el grado de compromiso del cliente con el proyecto es mayor cuando existe un contrato que contempla los eventos a seguir en caso de incumplir los acuerdos de su presencia en determinadas etapas del proceso. Lo que se experimentó fue que el cliente subestimó el compromiso previo de una reunión semanal debido a la naturaleza de estudiantes (de los desarrolladores) que están comprometidos en su trabajo de grado.

- En términos generales el proceso propuesto cuenta con los elementos necesarios para que el proyecto de software sea exitoso y de calidad pero se debe pensar si el ámbito en el que se piensa implementar es el adecuado, previamente la organización desarrolladora de software debe asegurarse de la disponibilidad del cliente, convenir los horarios de las parejas de desarrolladores, y debe contar las instalaciones físicas en las que se pueda ubicar al equipo de trabajo en el mismo lugar. Si estas condiciones son favorables, se puede contemplar la posibilidad de adoptar programación extrema para el proceso de desarrollo en equipos no muy numerosos (hasta de veinte personas, según su autor Kent Beck). No es recomendable adoptar programación extrema si el cliente no muestra interés por la comunicación abierta y frecuente con el equipo de trabajo, en últimas lo que hace que el proceso sea exitoso, no es la aplicación de sus prácticas de forma aislada sino la aplicación de ellas en conjunto, ya que éstas en conjunto refuerzan los valores de las metodologías ágiles y se complementan entre si.

### 6.2.2 Conclusiones de las prácticas

**Programación por pares**

- Una de las prácticas de programación extrema que se aplicó en la primera iteración y luego debió replantearse fue la programación por pares, el experimentar con programación por pares y programación individual permitió descubrir algunas

desventajas de realizar dicha práctica en parejas dependiendo de la categoría de cada uno de los desarrolladores, aunque este es un caso particular y no es posible generalizar a partir de este trabajo, se observó que cuando uno de los desarrolladores es de nivel "experto" y el otro "novato", el segundo termina asumiendo una actitud pasiva en el desarrollo del código, mientras que el experto termina codificando solo. En "Will Pair Programming Really Improve Your Project" [22] reseñan el libro *Pair Programming Illuminated* de Laurie Williams y Robert Kessler, en donde clasifican las distintas categorías de la pareja de desarrolladores: experto-experto, experto-intermedio, experto-novato, novato-novato, extrovertido-extrovertido, extrovertido-introvertido, introvertido-introvertido. Estas combinaciones posibles debido a la rotación de personal, aunque en [22] se recomienda que la combinación experto-novato se puede aplicar durante el desarrollo, si el programador experto está dispuesto a dedicar parte del día a capacitar al novato. Para el desarrollo de este trabajo se decidió que lo mejor era que el novato realice otras labores del desarrollo que también consumen buena parte de la iteración, como son las pruebas, y con ello se logró disminuir las horas hombre empleadas en realizar una iteración. Algo que evidentemente no es recomendable es una pareja del tipo novato-novato, ya que es de esperarse que incurran en errores de programación y ese no es el objetivo. Sea cual fuere la categoría de la pareja de programadores conviene clasificar las labores de desarrollo, eso con el fin de optimizar los tiempos empleados, porque no todas las tareas ameritan hacerse en pares.

- En el artículo "On the productivity of agile software practices: an industrial case study" [23] se resumen los resultados de un caso de estudio empresarial sobre productividad en prácticas de software ágil, los resultados son contundentes, la productividad aumenta entre un 65% y un 302%, dependiendo de la métrica usada, aunque no se puede concluir que este aumento se deba solamente a la programación por pares o al uso de controladores de pruebas automatizados, que representan una gran disminución de tiempo en la corrección de errores, que se refleja en menos errores que se transfieran a fases posteriores del desarrollo y esto permite lograr una mayor calidad del producto final.

**Pruebas**

- La puesta en práctica de definir las pruebas antes de haber codificado la funcionalidad representó varias ventajas: la primera, escribir las pruebas, requiere de los desarrolladores un buen entendimiento de lo que se está implementando. Será necesario saber que tipos y valores de entradas tendrá una función y debe saberse lo suficiente sobre la función para determinar lo que se espera a la salida. En segundo lugar, el escribir las pruebas primero, permite dimensionar que cantidad de código será necesario escribir. Esto puede significar una más compacta implementación del problema que está siendo solucionado.

- Debido al trabajo adicional que representa la implementación de pruebas automatizadas no se contempló esta posibilidad en el presente caso de estudio, ya que el aprendizaje de nuevas herramientas tiene un alto costo en tiempo, más aún cuando se trata de herramientas con poco material de referencia disponible y esto hubiese significado un incumplimiento del acuerdo pactado con el cliente.

**Entregas frecuentes**

- Algo que contribuyó a mantener la confianza en que el software realmente trabaja fue las entregas frecuentes, ya que cada una de ellas ha pasado las pruebas de aceptación, lo cual generó satisfacción y seguridad no sólo para el equipo de desarrollo, sino también para el cliente, que ve reflejado cómo en cada prototipo se cuenta con más funcionalidades que en el anterior, esto minimiza el riesgo de pérdida de la inversión para el cliente y de incumplimiento del contrato para los desarrolladores, pues existen muchas fases intermedias en las que se puede decidir a tiempo los cambios que sean necesarios, a diferencia de los procesos en que el cliente sólo conoce el producto al final y al encontrar una serie de errores podría acarrearle como consecuencia la pérdida de la inversión.

- Las entregas frecuentes son una muy buena práctica de la programación extrema, y debería ser tenida en cuenta en la adopción de otras metodologías que quieren minimizar el riesgo de fracaso de sus proyectos, pues las entregas frecuentes permiten un

seguimiento al desarrollo, una gestión de requisitos y dinamiza la realimentación dentro del proceso.

**Cliente en el sitio de trabajo**

- Una de las prácticas difíciles de aplicar en programación extrema, es que el cliente debe permanecer en el sitio de trabajo, es de esperarse que el cliente sea una persona que por múltiples razones no podrá dedicar buena parte de su tiempo a estar presente donde se desarrolla el software, por ello cada organización acordará con el cliente unas reuniones periódicas, donde el cliente podrá actualizar los requisitos, añadir alguna funcionalidad, o sugerir cambios y a su vez el equipo de desarrollo tiene la oportunidad de plantear alguna duda con respecto al desarrollo en curso. Estas reuniones no necesitan ser largas, puesto que se supone que serán frecuentes, en caso que la presencia física del cliente por alguna razón no sea posible se debe intentar mantener la comunicación por otros medios, (mensajería instantánea o teléfono, por ejemplo).

- Otro aspecto a considerarse además de la presencia del cliente es su nivel como usuario de software, pues al ser parte activa del proyecto debe tener criterio y fundamento en la toma de decisiones como pruebas, aprobación de prototipos, claridad en los requisitos.

**Integración continua**

- Este equipo estuvo conformado por solo dos personas a cargo del desarrollo y pruebas, por lo tanto, el proceso de integración se vuelve más sencillo, y no se presentó ningún problema de integración, por ello es poco lo que se puede decir al respecto, salvo que se recomienda emplear estándares de codificación en equipos donde haya rotación de personal en distintas secciones del código, y que cada par de programadores actualice constantemente en el repositorio las funcionalidades terminadas ( probadas).

**Cuarenta horas por semana**

- Las cuarenta horas semanales, se proponen para evitar que haya sobrecarga de trabajo, puesto que a mayor cantidad de horas laboradas la acumulación de estrés lleva a que se produzcan errores. En el presente trabajo fue necesario romper la regla hacia el final del

proceso, sin que esto significara algo traumático ni un aumento de errores por encima del promedio, pues el aumento de horas se distribuyó en los días de descanso y las jornadas diarias de los días laborales no se excedieron de las ocho horas.

**Refactorización**

- La refactorización es otra de las buenas prácticas de XP, se debe tener en cuenta que a mayor cantidad de código producido, la refactorización se vuelve más necesaria para evitar recurrencia y para depurar el código producido, también es indispensable para que el código pueda ser más fácilmente entendido por otros, ya que le imprime orden y modularidad, teniendo en cuenta que en XP todo el equipo de desarrollo debe estar en capacidad de trabajar en cualquier sección del código, es una necesidad que el código sea refactorizado.

**Simplicidad**

- Poner en práctica la simplicidad es algo que depende mucho de la naturaleza del proyecto y de los requisitos del cliente, esta práctica es más pertinente aplicarla en las labores de modelamiento, de diseño y de documentación, de ahí que para el desarrollo del proyecto el equipo decidió documentar sólo las historias de usuario y las pruebas finales y no cada una de las versiones. Para el modelamiento, se seleccionaron las historias de usuario más representativas, lo cual ahorra el tiempo dedicado al diseño.

**Juego de la planificación**

- La colaboración con el cliente y la flexibilidad en el contrato son indispensables en una metodología que es abierta a los cambios de requisitos. Esta práctica permite una negociación del contrato que da prioridad a la calidad del proyecto, dejando como variable el alcance, lo cual es aceptable teniendo en cuenta que el cliente muchas veces no tiene claro lo que realmente necesita.

**Propiedad colectiva del código**

- Debido a la programación por pares y a la rotación del personal, se desprende esta práctica, que debe estar presente en todo proyecto guiado por programación extrema.

**Estándares de codificación**

- Cuando el grupo de desarrolladores es numeroso, los estándares de codificación cobran mayor vigencia e importancia y en equipos que sólo se componen de un par de programadores también son necesarios para facilitar entendimiento y comprensión. Debe preverse que el software desarrollado requiere mantenimiento y es susceptible de actualizaciones y mejoras, por lo tanto son indispensables estándares que permitan una comprensión y preservación completa del código, independiente de las posibles variaciones del personal a cargo de mantenerlo.

## 6.3 Trabajo futuro

A corto plazo, se pueden realizar versiones más avanzadas del prototipo propuesto, algo en lo que el cliente también está interesado. A mediano plazo, existen muchas posibilidades de trabajo futuro en el campo de las metodologías ágiles, este caso de estudio representa solo un pequeño paso en la adopción de las prácticas involucradas en la programación extrema y alimentada con modelamiento ágil. Sería muy interesante tratar de guiar el proceso sin la presión del tiempo, para que los participantes puedan apropiar bien las prácticas, en especial la práctica de la codificación de pruebas automatizadas, con lo que se puede estudiar la incidencia del empleo de estas herramientas en la productividad.

Algo que se debe pretender también en trabajos posteriores es vincular a una persona con experiencia previa en este tipo de procesos, que sirva de guía a los otras personas del equipo, y que éste sea conformado por varias parejas de desarrolladores ya que eso permite además de validar los resultados obtenidos en un mayor número de personas, analizar las prácticas de integración frecuente y estándares de codificación, porque el XP adaptado en el presente caso de estudio, no permitió contemplar todos los aspectos del proceso, debido a que se aplicó en una pareja de desarrollo.

Existen otras metodologías ágiles que también pueden ser objeto de estudio, tales como scrum, cristal, entre otras, las cuales tienen en común los principios ágiles y sobre los que existe también material de referencia.

# REFERENCIAS

[1] ColombiaCompite. Características de la industria del software. Retirado de: http://www.colombiacompite.gov.co/site/redes2.asp?idcatinfo=1037&idsub=322

[2] Valdés Cárdenas, Luis Eduardo. Situación actual de la informática en Colombia. Extraido de: http://www.agenda.gov.co/documents/files/InformeCATI_ABRIL2004.ppt

[3] Hurtado Alegría, Julio. Sistema integral para el mejoramiento de los procesos de desarrollo de software en Colombia (SIMEP-SW). Junio de 2003.

[4] Ardila Arenas, Carlos Hernando. Folleto informativo ALCOSA –Alianza colombiana de software. Retirado de http://www.acis.org.co/archivosAcis/alcosa/alcosa1.doc .Marzo de 2003.

[5] http://www.nasscom.org/artdisplay.asp?Art_id=4782

[6] Fowler, Martin. "The New Methodology". Traducido por Alejandro Sierra. Marzo/Abril de 2003.

[7] Canós, J., Letelier, P., Penadés, M. "Metodologías ágiles en el desarrollo de software". Actas de las VIII Jornadas de Ingeniería del software y Bases de Datos, JISBD 2003. Alicante, Noviembre de 2003.

[8] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunnimgham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas. "Principles behind the Agile Manifesto". http://agilemanifesto.org/principles.html, 2001.

[9] Fowler, Martin. "The New Methodology". Traducido por Alejandro Sierra. Marzo/Abril de 2003.

[10] Alcanzando CMMI a través de Métodos Ágiles. Julio Ariel Hurtado Alegría, Cecilia Bastarrica. Proyecto SIMEP-SW. Conciencias-Unicauca. 2005, pp.6-10

[11] Beck, K. "Extreme programming explained. Embrance Change". Reading. Addison Wesley Longman, 2000

[12] Spike solution. http://c2.com/cgi/wiki?SpikeSolution

[13] Programación Extrema - Experimentar para creer. http://www.publicaciones.spc.org.pe/SPCMagazine/editions/II-1/html/articulos/articulo1.htm

[14] Extreme programming (XP): Un nuevo método de desarrollo. http://www.ati.es/novatica/2002/156/156-8.pdf

[15] Scott Ambler. Agile Modeling: Effective practices for Extreme Programming and the Unified Process. John Wiley & Sons, 2002

[16] [Jeffries] Ron Jeffries "Essential XP: Unit Tests at 100", http://www.xprogramming.com/xpmag/expUnitTestsAt100.htm

[17] Agile Modeling Principles V.2. http://www.agilemodeling.com/principles.htm

[18] Agile Modeling Practives V.2. http://www.agilemodeling.com/practices.htm

[19] Pressman, Roger, 2004. Ingeniería del software. Un enfoque práctico. Sexta edición Prentice-Hall

[20] Una explicación de la programación extrema. V encuentro x Base 2003 Madrid. Manuel Calero Solis. http://www.willydev.net/descargas/prev/ExplicaXP.pdf.

[21] Optional scope contracts Kent Beck Software first class. Cleal, Dave. Pág 2 http://www.xprogramming.com/ftp/Optional+Scope+Contracts.pdf

[22] Will Pair Programming Really Improve Your Project? (A critical look at the book *Pair Programming Illuminated*) Matt Stephens and Doug Rosenberg.
http://www.methodsandtools.com/PDF/dmt0403.pdf (Pág. 23- 26)

[23] Maurer, F. y Martel, S. On the Productivity of Agile Software Practices: An Industrial Case Study.
http://ebe.cpsc.ucalgary.ca/ebe/attach/.Publications_2002/MaurerMartel2002c.pdf

# ACRÓNIMOS

**AM**: Agile Modeling

**CMMI** Capability Maturity Model Integrated

**CRC**: Class, Responsibility, Collaboration

**DIAN**: Dirección de impuestos y aduanas nacionales

**NASSCON** National Association of Software and Service Companies

**PYME**: pequeña y mediana empresa

**SIGA**: Sistema de Información para Granjas Agroindustriales

**SIMEP-SW:** Sistema integral de mejoramiento de procesos de software

**SEI**: Software Engineering Institute

**XP**: extreme Programming

**RUP** Rational Unified Process

# GLOSARIO

**CMMI** Capability Maturity Model Integrated. Método de definir y gestionar los procesos a realizar por una organización. Fue desarrollado inicialmente para los procesos relativos al software por la Universidad Carnegie-Mellon para el SEI (Software Engineering Institute).

**Cristal:** Familia de metodologías que incluye diversos métodos para seleccionar el más conveniente en cada proyecto individual. Además de las metodologías el enfoque cristal también incluye los principios para adoptar las metodologías según las circunstancias de los diversos proyectos.

**Día ideal de programación:** Día de programación de ocho horas, en que los programadores saben exactamente qué hacer y la codificación se realiza sin interrupciones.

**DIAN**: Dirección de impuestos y aduanas nacionales. Entidad destinada a prestar un servicio de facilitación y control a los agentes económicos, para el cumplimiento de las normas que integran el sistema tributario, aduanero y cambiario, obedeciendo los principios constitucionales de la función administrativa, con el fin de recaudar la cantidad correcta de tributos, agilizar las operaciones de comercio exterior, propiciar condiciones de competencia real, proveer información confiable y oportuna, y contribuir al bienestar social y económico de los colombianos

**NASSCOM:** Asociación nacional de industrias y servicios de software de la India, que busca posicionar a dicho país como la fuente global proveedora de servicios software.

**McKinsey:** Firma de consultaría privada centrada en resolver temáticas que conciernen a la gerencia en corporaciones y organizaciones grandes.

**MiPYME**: MiniPyMES, Empresas de reducido número de personal, y con bajo capital, en Colombia se consideran las de menos de veinte empleados.

**RUP**: El Proceso Racional Unificado o RUP (Rational Unified Process), es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado constituye la

metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

**SEI:** Instituto de ingeniería de software, fundado por el Congreso Americano en 1984 para desarrollar modelos de evaluación y mejora en el desarrollo de software, que dieran respuesta a los problemas que generaba al ejército americano la programación e integración de los sub-sistemas de software en la construcción de complejos sistemas militares. Financiado por el Departamento de Defensa americano y administrado por la Universidad de Carnegie Mellon.

**Scrum:** metodología desarrollada para administración de procesos de desarrollo en sistemas. Esta es una metodología empírica que aplica las ideas de la teoría de control del proceso industrial al desarrollo de sistemas dando por resultado un acercamiento que reintroduzca las ideas de flexibilidad, adaptabilidad y productividad.

**Spike**: Un spike es un experimento dinámico de código, realizado para determinar cómo podría resolverse un problema. Es la versión ágil de la idea de prototipo. Se llama así porque "va de punta a punta, pero es muy fino" y porque en el recorrido de un árbol de opciones implementaría una opción de búsqueda depth-first.

**Stakeholders:** Quienes se ven afectados de manera positiva o negativa por la realización del proyecto.

**Velocidad del proyecto:** La velocidad del proyecto es una medida tomada de iteraciones anteriores y con la cual se logra estimar el tiempo que tomará cada una de las entregas e iteraciones del proyecto a desarrollar y que historias se deben tener desarrolladas al final de cada iteración. La velocidad del proyecto es el dato sobre el que se basa un programador XP para determinar, tanto tiempo, como alcance del proyecto.