

UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE CUENCA

FACULTAD DE INGENIERÍAS

CARRERA DE INGENIERÍA DE SISTEMAS

Tesis previa a la obtención del Título de:

Ingeniero de Sistemas

TÍTULO DEL TEMA:

Estudio de las Metodologías de desarrollo de Software Libre y su
aplicación en un caso práctico

AUTORES:

Joaquín Andrés Argudo Vásquez
William Hermel Astudillo Quituisaca

DIRECTOR:

Ing. Germán Ernesto Parra González

Cuenca, Febrero del 2010

Dedicatoria

Queridos padres, eh aquí el fruto del esfuerzo constante y la entrega diaria durante seis años de vida académica. Estoy seguro que esta meta alcanzada no hubiese sido posible sin el continuo apoyo de ustedes dos, mil gracias por sus buenos consejos y su afán por siempre darme lo mejor, esta tesis es de ustedes, los amo!!

Andrés Argudo

Agradecimientos

A ti papá Dios que siempre has estado conmigo en las buenas y malas, solo tú sabes lo que he pasado para llegar aquí, gracias de verdad, espero no defraudarte nunca. Un especial agradecimiento a usted Ing. Germán Parra por todo el apoyo que nos brindó desde el comienzo del proyecto hasta la culminación del mismo. Gracias también a todos mis familiares y amigos que siempre están ahí pendientes de todo, y por supuesto no podía olvidarme de mis profesores, de verdad gracias por todos sus conocimientos impartidos, que Dios les bendiga siempre, gracias a todos!!

Andrés Argudo

Dedicatoria

Con mucho cariño y amor dedico este trabajo de tesis y mi carrera universitaria a los seres más queridos y especiales para mí:

A Dios, por ser mi compañero de toda la vida, y estar conmigo siempre cuidándome y protegiéndome.

A mi familia, que siempre me apoyo, y estuvo conmigo en las buenas y malas.

A mi madre Angelita, que con su gran esfuerzo nunca hizo que me falte nada.

A mi enamorada Bernardita, cuyo apoyo ha sido muy importante en todos estos años, siempre dándome mucha comprensión y amor.

A mis amigos y compañeros, que con sus amistad y consejos me ayudaron a conseguir esta meta.

William Astudillo

Agradecimientos

Agradezco En Primer lugar quiero agradecer a Dios por permitirme lograr una meta más en mi vida, porque siempre he sentido su presencia y apoyo en todos los momentos de mi vida.

A mi familia, que siempre se sacrifico por mí, para que yo pueda estudiar y alcanzar mi meta de convertirme en un profesional, a mi hermano, mi padre, mis tíos, pero en especial a mi madre, pues ella ha estado conmigo incondicionalmente apoyándome y guiándome durante toda mi vida.

A mis amigos y compañeros que siempre estuvieron conmigo, que me alentaron a continuar, y que me enseñaron muchas cosas, que me servirán para toda la vida, tales como el valor de un verdadero amigo.

A las innumerables personas que me dieron ánimo y aliento para continuar, a mis profesores que me enseñaron lecciones para toda la vida, en especial al Ing. Germán Parra, por el apoyo que nos brindo durante la elaboración de esta tesis.

William Astudillo

CERTIFICADO

El presente trabajo de tesis previo a la obtención del título de Ingeniero de Sistemas fue guiado satisfactoriamente por el Ing. Germán Ernesto Parra González, quien autoriza su presentación para continuar con los trámites correspondientes.

Cuenca, 12 de Febrero del 2010

Ing. Germán Ernesto Parra González.

DIRECTOR DE TESIS

DECLARACIÓN DE RESPONSABILIDAD

Los conceptos desarrollados en este trabajo, así como todo el estudio e implementación de este Proyecto, son de exclusiva responsabilidad de los Autores.

Cuenca, 12 de Febrero del 2010

Joaquín Andrés Argudo Vásquez

AUTOR

William Hermel Astudillo Quituisaca

AUTOR

ÍNDICE DE CONTENIDOS

CAPITULO I

INTRODUCCIÓN

1.1	¿Qué es el Software No Libre?	1
1.1.1	Derechos de Autor Informático	3
1.1.2	Licencias Propietarias	4
1.2	¿Qué es el Software Libre?	6
1.2.1	Principios del Software Libre	6
1.2.2	Antecedentes históricos del Software Libre	8
1.2.3	Comunidades de Software Libre	9
1.2.3.1	¿Qué es una Comunidad de Software Libre?	9
1.2.3.2	Comunidades de Software Libre en Ecuador	10
1.3	Derechos de Autor	11
1.3.1	Copyright	11
1.3.2	Copyleft	11
1.3.3	Dominio Público	12

CAPITULO II

SOFTWARE LIBRE vs SOFTWARE NO LIBRE

Introducción	14	
2.1	Ventajas del Software Libre y Software No Libre	15
2.1.1	Ventajas del Software Libre	15
2.1.2	Ventajas del Software No Libre	18
2.2	Desventajas del Software Libre y Software No Libre	20
2.2.1	Desventajas del Software Libre	20
2.2.2	Desventajas del Software No Libre	21
2.3	Cuadro comparativo de Ventajas y Desventajas de Software Libre y Software no Libre	24
2.4	Metodologías de Desarrollo de Software	25
2.4.1	Metodologías para el desarrollo de Software Libre	26
2.4.1.1	Metodologías Agiles	26

2.4.2 Metodología de desarrollo de Software No Libre	27
2.4.2.1 Metodologías de desarrollo tradicionales	28
2.4.3 Metodologías Agiles (Software Libre) vs Metodologías Tradicionales (Software no libre)	28

CAPITULO III

LICENCIAS DE SOFTWARE LIBRE

Introducción	30
3.1 Clasificación	31
3.1.1 Licencias Permisivas	31
3.1.1.1 Licencias BSD	32
3.1.1.2 Licencias tipo MPL	32
3.1.2 Licencias Copyleft	33
3.1.2.1 Ventajas del Copyleft	35
3.1.2.2 Desventajas del Copyleft	36
3.1.3 Listado de licencias de Software Libre	36
3.1.4 Cuadro comparativo de los tipos de Licencia de Software Libre.....	39
3.2 Validez Legal	39
3.2.1 ¿Que es validez legal?	39
3.2.2 Validez Legal del Software Libre	39
3.3 Licencias GNU GPL	42
3.3.1 GNU GPL V3	42
3.3.1.1 Características de la licencia GNU GPL	43
3.3.1.2 Licencias compatibles con GNU GPL	45
3.3.2 Licencia GNU LGPL (Lesser General Public License)	45
3.3.3 Licencia GNU FDL (Free Document License)	46

CAPITULO IV

ORGANIZACIONES Y FUNDACIONES QUE RESPALDAN AL SOFTWARE LIBRE

Introducción	47
--------------------	----

4.1 FSF (Free Software Foundation)	48
4.1.1 Actividades que realiza la FSF	48
4.1.2 Fundaciones para el Software Libre (FSFs)	50
4.1.2.1 FSFLA (Fundación para el Software Libre Latinoamericana).....	51
4.2 Proyecto GNU	53
4.2.1 Antecedentes	54
4.2.2 Aplicaciones desarrolladas por el proyecto GNU	55
4.2.3 La Catedral y el Bazar	55
4.3 OSI (Open Source Initiative)	57
4.3.1 Beneficios	59
4.3.2 OSI Certified Open Source Software	59
4.3.3 Criterios de Definición de Código Abierto	60
4.3.4 Estándar de Software Libre en Ecuador	62
4.3.4.1 Beneficios del Estándar de Software Libre en Ecuador.....	63

CAPITULO V

METODOLOGÍAS DE DESARROLLO DE SOFTWARE

Introducción	64
5.1 Antecedentes	65
5.2 Metodología Extreme Programming (Programación Extrema)	66
5.2.1 Fases de desarrollo de la Programación Extrema	69
5.2.1.1 Fase de Exploración	69
5.2.1.2 Fase de Planificación de la Entrega	69
5.2.1.3 Fase de Iteraciones	70
5.2.1.4 Fase de Producción	70
5.2.1.5 Fase de Mantenimiento	70
5.2.1.6 Muerte del proyecto	71
5.2.2 Características de la Metodología	71
5.2.2.1 Como controlar las situaciones imprevistas	72
5.2.2.2 Función de los usuarios	72
5.2.2.3 Orientación a las personas	73
5.2.2.4 Medir los avances del proyecto	74

5.2.2.5	Pruebas del proyecto con programación extrema	74
5.2.2.6	Cuando es recomendable esta metodología	75
5.2.3	Ventajas y Desventajas de la metodología Extreme Programming	75
5.3	Metodología Rational Unified Process (Proceso Unificado Racional)	76
5.3.1	Antecedentes	76
5.3.2	Estructura del RUP	77
5.3.2.1	Visión estática de la metodología RUP	78
5.3.2.2	Visión Dinámica de la metodología RUP	79
5.3.3	Ciclo de Vida de la metodología RUP	80
5.3.3.1	Fase de Inicio	80
5.3.3.2	Fase de Elaboración	80
5.3.3.3	Fase de Construcción	81
5.3.3.4	Fase de Transición	81
5.3.4	Las seis mejores prácticas	82
5.3.4.1	Desarrollar Iterativamente	82
5.3.4.2	Administrar Requerimientos	83
5.3.4.3	Arquitecturas basadas en componentes	83
5.3.4.4	Modelamiento Visual	84
5.3.4.5	Verificar la Calidad	84
5.3.4.6	Controlar los Cambios	84
5.3.5	Disciplinas de un proceso RUP	85
5.3.6	Ventajas y Desventajas de RUP	86
5.4	Metodología Watch (Método del Reloj)	86
5.4.1	Generalidades	86
5.4.2	Estructura del Método Watch	87
5.4.2.1	Fase de Análisis	87
5.4.2.2	Fase de Diseño	88
5.4.2.3	Fase de Construcción	88
5.4.2.4	Fase de Pruebas	89
5.4.2.5	Fase de Instalación	89
5.4.3	Componentes del Método Watch	90
5.4.3.1	Modelo de Productos	90
5.4.3.2	Modelo de Actores	91
5.4.3.3	Modelo de Procesos	92

5.4.4 Ventajas y Desventajas del Método Watch	94
5.5 Características del modelo de desarrollo de Eric S. Raymond	94
5.5.1 Lecciones enumeradas en el ensayo “La Catedral y el Bazar”	95
5.6 Cuadro Comparativo de las Metodologías analizadas	104

CAPITULO VI

METODOLOGÍA DE DESARROLLO APLICADA AL SOFTWARE LIBRE

Introducción	106
6.1 Estructura de la Metodología Propuesta	107
6.1.1 Dimensión Estática de la Metodología Propuesta	107
6.1.1.1 Roles	108
6.1.1.1.1 Roles Analistas	110
6.1.1.1.2 Roles Desarrolladores	111
6.1.1.1.3 Roles Gestores	111
6.1.1.1.4 Roles de Apoyo	112
6.1.1.1.5 Pruebas	113
6.1.1.2 Equipo de Trabajo	113
6.1.1.3 Actividades	116
6.1.1.4 Artefactos	118
6.1.1.5 Flujos de Trabajo	119
6.1.1.5.1 Modelo de negocios	120
6.1.1.5.2 Requerimientos	120
6.1.1.5.3 Análisis y Diseño	121
6.1.1.5.4 Implementación	122
6.1.1.5.5 Pruebas	122
6.1.1.5.6 Producción	123
6.1.1.5.7 Gestión del proyecto	124
6.1.1.5.8 Gestión y Configuración de Cambios	124
6.1.1.5.9 Gestión del Entorno	124
6.1.2 Dimensión Dinámica de la Metodología Propuesta	125
6.1.2.1 Fase de Análisis del Dominio de la Aplicación	125
6.1.2.2 Fase de Especificación de Requerimientos	126

6.1.2.3 Fase de Diseño	126
6.1.2.4 Fase de Construcción	127
6.1.2.5 Fase de Pruebas	127
6.1.2.6 Fase de Liberación	128
6.2 Como utilizar ésta Metodología en un Proyecto Pequeño	128
6.2.1 Equipos de Trabajo y Roles	128
6.2.1.1 Equipo de Gestión	129
6.2.1.2 Equipo de desarrollo	129
6.2.1.3 Equipo de Soporte	129
6.2.2 Flujos de Trabajo y Actividades	130
6.2.2.1 Modelo de Negocios	130
6.2.2.2 Requerimientos	131
6.2.2.3 Análisis y Diseño	132
6.2.2.4 Implementación	133
6.2.2.5 Pruebas	133
6.2.2.6 Gestión de Configuración y Cambios	134
6.2.2.7 Gestión del Proyecto	134
6.2.3 Artefactos	134
6.3 Plantillas para la Elaboración de los diferentes Artefactos del Sistema	136
6.3.1 Plantilla para Elaborar un Documento de Estudio del Negocio	136
6.3.2 Plantilla para Elaborar un Documento de Requerimientos	137
6.3.3 Plantilla para Elaborar un Documento de Análisis y Diseño	138
6.3.4 Plantilla para Elaborar un Documento de Implementación del Proyecto	139
6.3.5 Plantilla para Elaborar un Documento de Pruebas del Proyecto	140
6.3.6 Plantilla para Elaborar un Documento de Control de Cambios	141
6.3.7 Plantilla para Elaborar un Documento de Gestión del Proyecto	142

CAPITULO VII

ELABORACION DEL CASO PRÁCTICO

Introducción	144
7.1 Documento de Estudio del Negocio	144

7.2 Documento de Requerimientos	147
7.3 Documento de Análisis y Diseño	150
7.4 Documento de Implementación del Proyecto	164
7.5 Documento de Pruebas	168
7.6 Manual de Usuario	175
CONCLUSIONES	188
RECOMENDACIONES	190
ANEXOS	191
Anexo 1: Decreto presidencial 1014	191
Anexo 2: Enviar un email a opensource.org	194
Anexo 3: Fotografía con Richard Stallman, padre del Software Libre.....	195
BIBLIOGRAFIA	196

CAPITULO I

INTRODUCCIÓN

El software se refiere al soporte lógico de un computador, y comprende el conjunto de los componentes lógicos (programas) necesarios para poder realizar tareas específicas, complementadas por el hardware. Las aplicaciones informáticas pueden ser muy diversas, y van desde Sistemas Operativos, controladores de dispositivos, aplicaciones ofimáticas, bases de datos, etc.

El software en los últimos años ha pasado a ser una parte fundamental de las tareas de las personas, y empresas debido a que ayudan a ser más rápidos y eficientes en las diferentes tareas. El software es considerado como una obra intelectual, protegida con derechos de autor, lo que significa que su creador puede cobrar por su utilización y explotación, de allí que varias empresas lo han visto como un gran negocio, claro ejemplo de ello es Microsoft, una de las empresas más grandes del mundo. Este software es considerado como privativo, pues las empresas y personas restringen la distribución y modificación del mismo.

Esto motivo la creación del software libre, este software no tiene como motivación principal el lucro, sino más bien fines sociales y políticos, principalmente busca brindar a los usuarios la libertad de ejecutar, compartir y mejorar el software, brindando el código fuente del software, y mediante el trabajo en grupo de comunidades que apoyan al Software Libre, prácticamente existen todo tipo de aplicaciones informáticas libres.

1.1 ¿Qué es el Software No Libre?

Teniendo en cuenta que el software no engloba únicamente programas de ordenador sino que va más allá de un concepto netamente técnico y que éste se ve asociado con la documentación que describe como está formado el sistema además de la configuración de los datos necesarios para que estos programas funcionen como se espera; pasamos a definir que es el Software No Libre.

Muchos expertos en la materia catalogan al Software No Libre de muchas maneras, unos lo llaman Software Privado, otros Software con Propietario y otros Software Privativo. Pero todos ellos parten de un mismo concepto cuya característica principal se enfoca en que los usuarios se ven totalmente limitados a usar, modificar, redistribuir y redimensionar su estructura funcional sin permiso expreso del titular del software.

Esto ha generado mucha controversia en la industria del software ya que se han generado corrientes a favor y en contra de estos principios, unos por un lado defienden a capa y espada los derechos de autor que deben existir sobre dichos productos considerados como obras literarias y otros que desean usar el programa con cualquier propósito, de estudiar cómo funciona y adaptarlo a sus propias necesidades.

Bajo esta iniciativa de Software No Libre, cualquier persona física o jurídica sea ésta una compañía, empresa o fundación, posee los derechos absolutos sobre los productos de software que desarrolle, restringiendo al mismo tiempo la libertad de usar el código para modificarlo, mejorarlo e incluso distribuir unas copias.

Con respecto al Software Propietario mencionado en párrafos anteriores, su característica principal es que mantiene la iniciativa sobre los derechos reservados tanto en el uso, modificación y distribución del software. El término de Software Privativo fue introducido por Richard Stallman en el año 2003 al referirse al Software que limita al usuario de hacer uso de su libertad para ajustar dicho producto a sus necesidades. Mientras que el término de Software No Libre fue introducido por la FSF (Free Software Foundation).

Una de las compañías más representativas y conocidas a nivel mundial por todas las personas en la industria del Software no Libre es sin duda Microsoft con sus aplicaciones Microsoft Office, caracterizada por vender sus productos sin el código fuente, frustrando la libertad a los usuarios de adaptarlo a sus necesidades propias viéndose en la obligación de adquirir una licencia para su uso. Esto ha hecho que se forme un Monopolio informático alrededor del mundo ya que poseen un control absoluto del mercado incluso frente a sus principales competidores, mencionamos

como ejemplo Microsoft Office frente a OpenOffice.org de código abierto. Esta situación ha permitido establecer un tipo de economía de red en el que la empresa impone sus productos a nivel mundial llegando a formar como un tipo de estándar en el uso de sus herramientas obteniendo así una ventaja competitiva enorme.

Pero no todo ha sido color de rosa para la compañía ya que a través de los años ha enfrentado un sinnúmero de juicios antimonopolio en los EE.UU y en muchas ocasiones han sido cuestionadas sus prácticas al tener una visión anti ética del software.

1.1.1 Derechos de Autor Informático

Mucha gente confunde los derechos de autor con las patentes como un solo concepto, lo cual no es así ya que ambas ofrecen un distinto tipo de protección, la primera solo ofrece protección a las expresiones pero no a las ideas o procedimientos mientras que, según la WIPO (Organización Mundial de la Propiedad Intelectual) *“Una patente es un derecho exclusivo concedido a una invención, es decir, un producto o procedimiento que aporta, en general, una nueva manera de hacer algo o una nueva solución técnica a un problema. Para que sea patentable, la invención debe satisfacer determinados requisitos.”*¹

Para que una invención sea protegida por una patente debe cumplir con las siguientes condiciones: ser novedosa, tener un nivel inventivo y una aplicación industrial. Ser novedosa implica que la invención debe ser nueva y constituir un progreso para la comunidad. Tener un nivel inventivo significa que la invención debe tener originalidad sin que se derive o se incluya de forma evidente de otra invención. Por último una invención debe tener una aplicación industrial de modo que su producto pueda ser fabricado o utilizado en cualquier clase de industria.

En el ámbito informático todo software tiene dos componentes: un componente escrito que es el código fuente y un componente técnico formado por todos los

¹ Tomado de: OMPI Servicios, “Preguntas Frecuentes”,
http://www.wipo.int/patentscope/es/patents_faq.html#patent.
Fecha de Consulta 01 de Julio del 2009

algoritmos. Las patentes de software protegen a este solo a partir de los algoritmos, es decir, protegen las ideas, estableciéndose un monopolio frente a empresas pequeñas que necesitan de estas ideas o productos patentados para desarrollar sus aplicaciones.

Por el contrario, los derechos de autor protegen el cómo se expresan esas ideas, es decir la manera en que se escribe o se dice una obra, así se garantiza a los autores la originalidad de sus obras quedando exentas de copias ilegales y otras formas de plagio, así por el uso legal de la obra los autores reciben un beneficio económico. En la Figura 1.1 se presenta un resumen de los tipos de protección para el software.

TIPOS DE PROTECCION PARA EL SOFTWARE	
PATENTE	DERECHO DE AUTOR
Protege las ideas expresadas en los algoritmos	Protege el cómo se expresan esas ideas en los algoritmos

Figura 1.1 Tipos de Protección para el Software

1.1.2 Licencias Propietarias

Este tipo de licencias se aplica al Software Propietario y es usado por varias empresas en sus productos alrededor del mundo, entre ellas destacan ESRI, Autodesk, IBM, Oracle, Microsoft, entre otras. Para explicar este tema tomaremos como referencia a Microsoft quién en su página principal muestra una guía de licencias que detalla las distintas opciones de licencia disponibles para ofrecer al mercado.

“Una licencia de software otorga al usuario derecho legal a utilizar un software. Por cada programa de software de Microsoft que se utiliza, se otorga una licencia al usuario y ésta se documenta en el Contrato de Licencia de Usuario Final (CLUF).

Un usuario de software, necesita una licencia. El acuerdo de licencia da al usuario el derecho de utilizar el software.”²

Existen diferentes maneras de adquirir una licencia de Software Microsoft, la más usual es la Licencia por Volumen y consiste en comprar licencias de productos a un precio con un descuento según el volumen adquirido. Actualmente la compañía maneja varios programas de Licencia por Volumen para pequeñas y medianas empresas con Open Multilicencia (para clientes corporativos y académicos que compran un mínimo de 5 licencias de software) u Open Suscripción (OSL: suscripción de software, para organizaciones con más de 10 PCs donde el software no es propiedad de sus clientes en ningún momento, simplemente pagan por utilizarlo durante un período de 3 años) y programas para clientes de grandes empresas con Contrato Select (para clientes corporativos y académicos con más de 250 PCs.) y Enterprise Agreement EA (para grandes clientes corporativos con más de 250 PCs., y que tienen derechos a actualizar a las versiones nuevas de los productos para los que se adquiere una licencia) o Enterprise Agreement Suscripción EAS (que ofrece los mismos beneficios que con un EA más un ahorro del 15% del coste de un EA).

Para el sector educativo existen tipos de licencias como Campus y School Agreements (es un contrato de licencia de suscripción en el que la institución puede ejecutar cualquier software de una larga lista de productos para aplicaciones y sistemas operativos incluyendo todas las actualizaciones), Caja Académica (apropiado para colegios que compran cantidades muy pequeñas de software, contiene CDs, manuales y un acuerdo de licencia), Open Académico (es un programa que ofrece descuentos cuando compran al menos 5 licencias) y Select Académico (programa de gran volumen de licencias para colegios con más de 250 PCs).

² Tomado de: Microsoft Licencias, “Explicación de las Licencias de Microsoft”, <http://www.microsoft.com/spain/licencias/novedades/explicacion.mspix>.
Fecha de consulta 26 de Junio del 2009.

1.2 ¿Qué es el Software Libre?

Según el proyecto GNU: “*El software libre es una cuestión de la libertad de los usuarios de ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software.*”³

Según Richard Stallman, creador de los conceptos Free Software, y fundador de la FSF, El Software Libre brinda el control de la informática ya sea personal, empresarial o gubernamental, pues al conocer el código fuente y analizarlo, se puede saber exactamente que hace el software, y evitar que demás personas u organizaciones accedan a nuestros sistemas, o datos a través de medios como software espía. Además se lo puede modificar para necesidades concretas que no estén satisfechas con el software actual.

El Software Libre suele estar disponible gratuitamente, o al precio de coste de la distribución a través de otros medios; sin embargo el Software Libre no se refiere a software gratis, por lo que el asociar Software Libre a "software gratuito" (freeware), es incorrecto. El software libre puede ser comercialmente distribuido y no perder sus características de Software Libre.

También suele confundirse software libre con software de dominio público, que no requiere de licencia para su uso, debido a que sus derechos de explotación son para toda la humanidad, y pertenece a todos por igual. Cualquiera puede hacer uso de él, siempre con fines legales y señalando su autoría original. Este software es aquel cuyo autor lo dona a la humanidad o cuyos derechos de autor han expirado, tras un plazo contado desde la muerte de este, habitualmente 70 años. Si un autor condiciona su uso bajo una licencia, por muy débil que sea, ya no es del dominio público.

1.2.1 Principios del Software Libre

El Software Libre, es una forma ética de entender el software, para su desarrollo, comercialización y uso, el Software Libre está compuesto por valores como la ética, la colaboración, la competitividad, seguridad, privacidad, eficiencia, no discriminación y su valor más importante la libertad.

³ Tomado de GNU Project: Free Software Foundation, “La Definición de Software Libre”, <http://www.gnu.org/philosophy/free-sw.es.html>.
Fecha de Consulta 01 de Julio del 2009

El Software Libre es una filosofía que se rige por cuatro libertades que deberían ser cumplidas para que se les considere como tal.

El Software Libre específicamente brinda al usuario las siguientes libertades:

- **Libertad 0:** “La libertad de ejecutar el programa, para cualquier propósito”.
- **Libertad 1:** “La libertad de estudiar cómo trabaja el programa, y adaptarlo a sus necesidades”.
- **Libertad 2:** “La libertad de redistribuir copias para que pueda ayudar al prójimo”.
- **Libertad 3:** “La libertad de mejorar el programa y publicar sus mejoras y versiones modificadas en general, para que se beneficie toda la comunidad”.

Se dice que el software es libre siempre que el usuario disponga de todas las libertades anteriores, por lo que es necesario disponer del código fuente para el estudio y modificación del software, además cualquiera que redistribuya el software debe dar las mismas libertades.

El software libre ha tenido una gran evolución y desarrollo debido a que el Software Libre consigue involucrar a toda la comunidad para que trabaje unida para arreglar problemas. Los usuarios no sólo informan de errores, incluso los arreglan. Los usuarios trabajan juntos, para llegar al fondo del problema y hacer que el software funcione sin problemas.

Además el Software Libre se ha convertido en una cuestión filosófica y política muy fuerte, ya que busca que todos se puedan beneficiar del software quitándole las trabas de uso y distribución respecto al software no libre.

Las motivaciones para desarrollar Software Libre, es diferente de los objetivos del software no libre, mientras que el software no libre compite comercialmente. Los paquetes de software libre compiten por una buena reputación y un programa que satisfaga las necesidades de los usuarios no alcanzará la popularidad esperada. Además un autor que pone el código fuente al alcance de todos arriesga su reputación, y le conviene hacer el software limpio y claro, de lo contrario podría perder su reputación frente a la comunidad.

1.2.2 Antecedentes históricos del Software Libre

En 1980 las computadoras comenzaban a utilizar sistemas operativos privativos, forzando a los usuarios a aceptar condiciones restrictivas que impedían realizar modificaciones a dicho software. Si algún programador encontraba algún error en la aplicación, lo máximo que podía hacer era dar parte a la empresa desarrolladora para que ésta implemente una solución, aunque el programador sea capaz de solucionar el problema y lo quiera hacer sin pedir nada a cambio, el contrato le impedía que realice dicha mejora.

Este fue el caso de Richard Stallman que trabajó en un laboratorio que había recibido una impresora donada por una empresa externa y era utilizada en red por todos los trabajadores, pero en dicha impresora cada cierto tiempo el papel se atascaba y no existía ningún aviso que notificase a los usuarios de la situación, con lo que se generó una pérdida de tiempo y otros inconvenientes.

Richard Stallman decidió implementar el envío de un aviso por red cuando la impresora se bloqueara, esto implicaba tener acceso al código fuente de los controladores de la impresora. Pidió a la empresa propietaria acceso al código fuente, comentando lo que pretendía realizar sin pedir nada a cambio por lo que la empresa se negó a otorgarle el código fuente.

Con este antecedente, en 1984, Richard Stallman comenzó a trabajar en el proyecto GNU (GNU's Not Unix), y en 1985 fundó la FSF (Free Software Foundation), donde definió los conceptos de free software y el concepto de "copyleft" buscando brindar a los usuarios libertad y restringir las posibilidades de apropiación del software.

Desde entonces se han elaborado un sinnúmero de proyectos de software libre, siendo el más importante el proyecto del Sistema Operativo GNU, que es el caso más representativo del éxito del software libre.

Aunque Richard Stallman se ha enfocado casi plenamente en el aspecto político y filosófico del software libre, al fundar la FSF, nosotros consideramos que los aspectos técnicos también deben ser analizados y mejorados, para poder dar un crecimiento más ordenado, y poder continuar mejor con la mejora del software.

1.2.3 Comunidades de Software Libre

Las comunidades de Software Libre, han sido una de las principales fortalezas para que el software libre se desarrolle y se difunda, la comunidad se encarga de brindar apoyo en el desarrollo, difusión y soporte del software libre.

Para llevar adelante una comunidad se requiere un gran esfuerzo y trabajo en conjunto de todos sus miembros, un ejemplo de estas comunidades, es la comunidad que respalda al proyecto GNU, que tiene más de 20 años de existencia, y miles de miembros por todo el mundo, que han aportado para fortalecer y mejorar el Sistema Operativo GNU.

1.2.3.1 ¿Qué es una Comunidad de Software Libre?

Una Comunidad de Software Libre es un grupo de personas que cooperan entre sí en distintas áreas relacionadas con el Software Libre, como el desarrollo, distribución y soporte. Estas personas pueden ser usuarios, desarrolladores, distribuidores, traductores o partidarios del movimiento de Software Libre.

Sus objetivos se basan en las libertades negadas por el modelo de software privativo, mediante la cooperación, se busca compartir el conocimiento entre sus miembros, con el fin de poder adaptar libremente las aplicaciones a las necesidades de cada usuario. Fortaleciendo valores como Transparencia, Solidaridad, Profesionalismo, Ética, Renovación, Unión, Constancia, y Colaboración.

Las comunidades de Software Libre han tenido un gran crecimiento, en sus inicios estaban compuestas principalmente por programadores y académicos, y actualmente se han unido a ellas muchas más personas, empresas y fundaciones como por ejemplo Apache, PHP, Sun Microsystems. Lo que ha permitido que existan muchas aplicaciones libres de primer nivel como Open Office.

Las comunidades de Software Libre utilizan como medio principal de comunicación a la Internet, mediante el uso de páginas web de comunidades, correo electrónico, wikis, foros, conferencias virtuales, entre otras.

Dependiendo del área de la comunidad, sus objetivos pueden ser diferentes. Pero todas tienen en común el espíritu cooperativo, la búsqueda continúa del

mejoramiento y difusión del software libre y del conocimiento, y la libertad de los usuarios como su principal interés.

Entre las actividades comunes que realiza una Comunidad de Software Libre están conferencias, festivales de instalación, capacitaciones, entre otras.

1.2.3.2 Comunidades de Software Libre en Ecuador

En Ecuador el Software Libre se ha difundido mucho, y gracias al decreto 1014 firmado el 10 de abril del 2008, en el cual se define al uso de Software Libre como política de estado, la comunidad de Software Libre del Ecuador, ha dado un paso muy grande, y busca realizar una revolución tecnológica .

La comunidad de Software Libre del Ecuador buscó la necesidad de unirse y por esto surgió Asociación de Software Libre del Ecuador (ASLE), el 31 de marzo del 2007.

ASLE, busca ser el centro de los esfuerzos para difundir el Software Libre y sus beneficios en el Ecuador y ser el núcleo de agrupación de gente y empresas relacionadas con el Software Libre en el Ecuador. Según ASLE, estas son las principales comunidades de Software Libre en el Ecuador.

- Ecuallug: Foro de discusión y soporte para usuarios del sistema operativo GNU/Linux.
- Openecuador: Foro de discusión y soporte de Software Libre en general.
- Aprende Libre: Comunidad orientada al Software Libre y la educación.
- COPLEC: Comunidad de Programadores de Software Libre del Ecuador.
- SasLibre: Servicios Académicos y de Software Libre.
- CRICE: Comunidad orientada a la discusión y soporte de Software Libre en Cuenca.
- KOKOA: Comunidad de Software Libre de la ESPOL.
- UTPLINUX: Comunidad de la Universidad Técnica Particular de Loja.

1.3 Derechos de Autor

Todos los seres humanos tenemos la capacidad de crear. Todo creador de una obra intelectual, de cualquier tipo es un autor. Los derechos de autor se refieren a un conjunto de normas creadas para proteger estas obras y brindar reconocimiento a su autor, dándole la facultad de oponerse a cualquier modificación de su obra, sin su consentimiento, así como para el uso o explotación de su obra por terceros. El derecho de autor es un término jurídico que describe los derechos concedidos a los creadores por sus obras literarias y artísticas.

El tipo de obras que abarca el derecho de autor incluye: obras literarias, documentos de referencia, periódicos y programas informáticos; bases de datos; películas, composiciones musicales entre otros.

1.3.1 Copyright

El derecho de autor o Copyright se basa en la idea de un derecho personal del autor, sobre su obra, y busca brindar protección a las obras que sean concebidas por las personas o empresas.

El derecho moral está constituido como parte de la persona del autor: reconoce que la obra es expresión de la persona del autor y así se le protege. La protección del copyright se limita estrictamente a la obra, sin considerar atributos morales del autor en relación con su obra, excepto la paternidad; no lo considera como un autor propiamente tal, pero tiene derechos que determinan las características de utilización de una obra.

1.3.2 Copyleft

Es un grupo de derechos de autor definidos para eliminar las restricciones de distribución o modificación impuestas por el copyright, pero con la condición de que el trabajo resultante mantenga el mismo sistema de derechos de autor que el original. Bajo estos derechos se pueden proteger cualquier tipo de elaboración creativa.

En la Década de 1970 Richard Stallman elaboró un intérprete de Lisp (LIStProcessing), la compañía Symbolics se interesó en este intérprete por lo que Richard Stallman accedió a facilitarles una versión bajo licencia de dominio público. Luego dicha empresa amplió y mejoró el software original, Richard Stallman

pretendió acceder a dichas modificaciones pero esta compañía se negó a proporcionárselas.

Stallman decidió tratar de erradicar este tipo de comportamiento al que lo denominó “software hoarding” (acaparamiento del software), creando su propia licencia de derechos de autor dentro del marco legal existente denominada GNU GPL (Licencia Pública General).

“En el proyecto GNU, nuestro objetivo es el dar a todo usuario la libertad de redistribuir y cambiar software GNU. Si los intermediarios pudieran quitar esa libertad, nosotros tendríamos muchos usuarios, pero esos usuarios no tendrían libertad. Así en vez de poner software GNU en el dominio público, nosotros lo protegemos con Copyleft. Copyleft garantiza que cada usuario tiene libertad.”⁴

1.3.3 Dominio Público

Implica que todo el mundo tiene derecho sobre la obra, y pueden ser explotadas por cualquier persona o corporación. Las obras cubiertas por el derecho de autor pasan al dominio público pasados 50 años desde la muerte de su autor como mínimo, aunque el plazo también puede ser de 70, 80 o 100 años desde la muerte del autor.

En los países que subscribieron el Acuerdo de Aspectos de Propiedad Intelectual aplicado al Comercio (ADPIC o TRIPS), se comprometen a un mínimo de 70 años tras la muerte del autor. En algunos países, el autor puede voluntariamente ceder al dominio público una obra, es decir, renunciar a los derechos patrimoniales sobre su obra, manteniendo la paternidad sobre la misma.

Nuestro país también respeta y tiene su ley de propiedad intelectual en la cual se expresa que *“El Estado reconoce, regula y garantiza la propiedad intelectual adquirida de conformidad con la ley, las Decisiones de la Comisión de la*

⁴ Tomado de GNU Project: Free Software Foundation, “Copyleft”, <http://www.gnu.org/copyleft/copyleft.es.html>.
Fecha de Consulta 08 de Julio del 2009

*Comunidad Andina y los convenios internacionales vigentes en el Ecuador.”*⁵. La propiedad intelectual comprende:

1. Los derechos de autor y derechos conexos.
2. La propiedad industrial que abarca entre otros elementos los siguientes:
 - a. Las invenciones.
 - b. Los dibujos y modelos industriales.
 - c. Los esquemas de trazado (topografías) de circuitos integrados.
 - d. La información no divulgada y los secretos comerciales e industriales.
 - e. Las marcas de fábrica, de comercio, de servicios y los lemas comerciales.
 - f. Las apariencias distintivas de los negocios y establecimientos de comercio.
 - g. Los nombres comerciales.
 - h. Las indicaciones geográficas.
 - i. Cualquier otra creación intelectual que se destine a un uso agrícola, industrial o comercial.
3. Las obtenciones vegetales.

⁵ Tomado de: Ley de propiedad intelectual Ecuador,
<http://www.gocomputer.com.ec/faqdemo/pdf.php?cat=4&id=4&lang=es>.
Fecha de Consulta 16 de Julio del 2009

CAPITULO II

SOFTWARE LIBRE vs SOFTWARE NO LIBRE

Introducción

Cuando una empresa se encuentra en el punto de elegir por una solución informática o bien analizar la ya existente, debe tomar en cuenta muchos factores primordiales con respecto a su funcionalidad, verificar que dicha solución ya ha sido probada por otras empresas y que ha tenido el éxito esperado, y que además se adapta perfectamente a sus necesidades de negocio.

Para esto la persona o grupo de personas que tomen la decisión de adquirir el producto o herramienta deben considerar todas las ventajas y desventajas que la solución le ofrece y ser capaz de elegir la solución correcta que supere sus expectativas tanto en rendimiento como en costo.

Muchas de las veces el análisis detallado de estas soluciones puede resultar un proceso tedioso y aburrido, pero el tenerlos en cuenta ahorrará muchos problemas que se presenten en el futuro.

El presente capítulo pretende exponer los diferentes puntos de vista y directrices que tienen el Software Libre y Software No Libre, con ello no pretendemos inducir al lector a un tipo de software específico, sino más bien llamamos al estudio de los mismos con el fin de estar al tanto de más y nuevos conocimientos en esta nuestra industria, las tecnologías y sistemas de información.

2.1 Ventajas del Software Libre y Software No Libre

A partir de la analogía realizada en el capítulo anterior y desde una perspectiva de usuario final, podemos darnos cuenta que el Software Libre presenta muchas y mejores ventajas que el Software No Libre, tampoco queremos decir que este último es obsoleto o ineficaz ya que también presenta ventajas significativas en ciertas aplicaciones.

2.1.1 Ventajas del Software Libre

- Costos bajos en la adquisición.

Los bajos Costos del Software Libre permiten a los usuarios finales disponer de una mayor gama de productos y servicios basados en esta tecnología. Dichos costos están relacionados con la licencia de uso que se le dará al software y son un factor determinante al momento de elegir que productos utilizar, sobretodo en empresas públicas en donde la cuestión económica juega un papel muy importante, y si partimos de la base de que el Software Libre carece de costos de licencia, esta parte del presupuesto se puede invertir en otros fines.

Es así que varias administraciones públicas alrededor del mundo han mostrado su apoyo al uso y desarrollo de Software Libre por lo que han optado por migrar total o parcialmente sus servidores y sistemas de escritorio a ésta plataforma, tal es el caso de nuestro país en el que se decidió el uso de Software Libre como política de gobierno emitiéndose el decreto 1014 el 10 de abril de 2008 que establece como política pública para las Entidades de la Administración Pública Central la utilización de Software Libre en sus sistemas y equipamientos informáticos, creándose así un género tecnológico propulsor de riqueza nacional.

- Libertad de uso y Distribución

El usuario que utiliza licencias de Software Libre está en la libertad de instalar los productos de software tantas veces quiera y en las máquinas que desee sin ninguna

restricción. De esta manera, el usuario se libera de toda dependencia de un proveedor único y ya no queda atado a él para nuevas versiones.

- Mejora constante de la infraestructura tecnológica

Uno de los objetivos del Software Libre es compartir información a todos sus usuarios a través de las distintas distribuciones existentes en la red, esta iniciativa ha hecho que ellos tengan un papel importante al influir en la evolución del software mediante sus aportaciones, contribuyendo a la mejora constante de la infraestructura tecnológica y la depuración de errores.

- Libre manipulación de la infraestructura tecnológica

El usuario puede acceder al código fuente de la aplicación, estudiarlo, modificarlo e incluso utilizarlo para el desarrollo de nuevos productos sin la necesidad de comenzar desde cero.

- Reducción de los requisitos de Hardware

La mayoría de productos basados en Software Libre requieren de requisitos de hardware bastante menores que soluciones basadas en Software No libre, por ejemplo, servidores con sistemas Linux que no utilizan interfaz gráfica. Incluso en ocasiones, ciertos proyectos que estén utilizando Software No Libre pueden verse afectados en la compatibilidad con alguna plataforma antigua, situación que para el Software Libre no es problema ya que existe un soporte de toda la comunidad en plataformas de hardware de este tipo.

- Soporte y Compatibilidad a largo plazo

El interés de las empresas que producen Software No Libre es vender sus productos en todo el mercado y producir nuevos productos de software que emplee nuevas tecnologías y no dar soporte a la resolución de fallos producidos en los anteriores haciéndoles de ver que son obsoletos, algo que no sucede con Software Libre ya que

existe un proceso de revisión pública en el que los usuarios pueden detectar posibles errores así como las soluciones al mismo.

- Aplicaciones sin puertas traseras

La libertad de acceder al código, estudiarlo y modificarlo, permiten a los usuarios auditar los programas, por lo que la existencia de puertas traseras no viene al caso ya que cualquier código oculto se pondría de manifiesto e iría en contra de los intereses de toda una comunidad. En el Software No Libre jamás podremos saber si los programadores originales introdujeron puertas traseras que pongan en peligro la seguridad del sistema o la privacidad de los datos.

- Software Personalizado

La mayoría de productos basados en Software No Libre vienen en paquetes que muchas veces no se adaptan a las necesidades específicas de las empresas, por lo que surge la necesidad de poder personalizar el software de manera que cubran áreas específicas de la empresa a costos bajos. Esta solución viene dada con el software Libre ya que permite personalizar los distintos productos hasta que cubran nuestra necesidad gracias a la disposición del código fuente.

- Involucra a toda la comunidad para que trabaje unida en la solución de problemas

Una de las características de las comunidades de Software Libre es su trabajo en equipo, todas ellas se apoyan en la solución de problemas mediante una serie de foros en los que diferentes usuarios alrededor del mundo no sólo informan de errores y problemas, sino que incluso los arreglan, así toda la comunidad trabaja unida para llegar al fondo del problema y hacer que el software funcione sin problemas.

2.1.2 Ventajas del Software No Libre

- Control de Calidad en los Productos

Las empresas que producen software de este tipo se preocupan principalmente del control de calidad en sus aplicaciones, por ello realizan un sinnúmero de pruebas con el fin de detectar fallos garantizando así un producto óptimo para el usuario. Se preocupan además de la estética del programa, dando un mejor acabado a las aplicaciones.

- Fuerte relación con empresas fabricantes de Hardware

Esta ventaja es fácil de explicar cuando mencionamos por ejemplo el caso de Microsoft que tiene un dominio bastante grande del mercado de software y que tiene varias relaciones con empresas fabricantes de hardware a quienes incita a producir drivers que sean solo compatibles con Windows, y además desarrollan dispositivos que mejoran el rendimiento cuando se ejecutan aplicaciones y sistema operativo Microsoft.

- Protección del Software desarrollado

El desarrollo de software requiere de importantes inversiones en recursos de investigación para su desarrollo, por lo que ese esfuerzo en generar un producto final debe ser protegido contra apropiaciones ilegales.

- Mayor mercado laboral

La mayoría de empresas poseen productos de Software No Libre para la ejecución de sus procesos básicos, sean estos transaccionales, informativos, etc. por distintas razones: seguridad, soporte, mantenimiento, etc. Por ello el conocimiento de estas herramientas asegurará un mercado laboral mucho mayor que con otras.

- Mayor Cantidad de Personal altamente capacitado

Existen profesionales en el medio capacitados altamente en el uso de productos de Software No Libre, quienes están dispuestos a brindar asesoramiento sobre las distintas herramientas existentes en el mercado. En ciertos casos se establece de antemano un contrato por un período de tiempo dado para la utilización de sus servicios.

- Toma de decisiones centralizada en el desarrollo del Software

La toma de decisiones centralizada se hace en relación a una línea de productos haciendo que éstos sean funcionales y altamente compatibles. Lo que no sucede con el Software Libre ya que las decisiones son tomadas por diferentes grupos y organismos descentralizados que muchas veces no llegan a un mismo consenso. Esto produce incompatibilidades en los productos finales degradando la calidad del software.

- Publicaciones y documentaciones acerca del uso

Existe mucho material en publicaciones y documentaciones acerca del uso de todas las tecnologías de información que distribuyen empresas de Software No Libre, haciendo que el aprendizaje de sus herramientas sean en el menor tiempo y sobretodo que exista una comprensión total del producto.

- Soporte total por una sola compañía

En la mayoría de los casos los productos existentes en una organización son de un mismo fabricante, el caso más común es Microsoft quien ofrece un soporte de una línea de productos específica en un solo paquete, por otro lado con el Software Libre no existe un soporte de un único proveedor por lo que puede dar lugar a inconsistencias e incompatibilidades en los sistemas existentes al venir el soporte de varios lados.

2.2 Desventajas del Software Libre y Software No Libre

2.2.1 Desventajas del Software Libre

- Mayor esfuerzo en el aprendizaje

El diseño y estructura del Software Libre presenta un nivel mayor de complejidad a sus usuarios frente al ya conocido Software No Libre, ya que cuenta con una serie de comandos para su administración y configuración que dificultan de cierto modo el aprendizaje, por ejemplo, para la instalación de una herramienta en Software No Libre, solo es necesario seguir los pasos del Wizard, mientras que en Software Libre se requiere del conocimiento de comandos comunes para dicha instalación. Además la navegación resulta mucho más fácil en herramientas propietarias.

- No existen garantías del producto

Los productos de Software Libre al ser de código abierto no ofrecen ninguna garantía en su funcionamiento ya que no existe ninguna compañía que respalde la tecnología, el único soporte que existe se presentan en los foros de Internet.

- Riesgos en la migración a la plataforma

Muchas compañías por el temor de experimentar problemas y conflictos en sus sistemas como consecuencia de la migración a la plataforma libre, no se arriesgan al cambio y prefieren operar de la misma forma como han venido trabajando a pesar de que la nueva opción ofrece una reducción de costos en sus licencias. Todo se debe a la incertidumbre por lo desconocido.

- Menor compatibilidad con el Hardware

Debido al gran monopolio existente en compañías de Software No Libre relacionadas con algunas empresas fabricantes de Hardware, existe poco Hardware compatible con Software Libre, esto es debido a que no se producen los suficientes controladores

como para elevar el crecimiento de usuarios de esta tecnología, un ejemplo claro son los video juegos que en su mayoría se venden para plataformas Windows.

- Interfaces gráficas poco amigables

La mayoría de productos poseen una Interfaz Gráfica de Usuario (GUI) un poco inestable en relación a productos propietarios, aunque últimamente se ha producido una evolución tecnológica grande en interfaces gráficas de usuario comunes como KDE, GNOME y el manejador de ventanas WINDOWMAKER.

- Generación fácil de Troyanos

Debido a que el código es abierto para todo público, existe la posibilidad de que se introduzcan códigos maliciosos como troyanos y si estos logran confundirse con la versión original, pueden producir errores en el sistema.

- Mala gestión del código fuente

Debido a que toda la comunidad aporta con sus conocimientos a partes concretas del código pueden surgir problemas con la mala gestión del mismo ya que no se sigue un método formal para aportar con los cambios lo cual puede llevar a la falta de piezas clave que nadie ha contribuido.

2.2.2 Desventajas del Software No Libre

- Existencia de puertas traseras en el código

Dentro del código pueden existir fragmentos que cumplan otras funcionalidades distintas a las originales y que además no se encuentren documentadas, tal es el caso de puertas traseras diseñadas por el fabricante de software para satisfacer sus propios intereses, todas ellas transparentes al usuario final. Este problema se produce por el hecho de que el código fuente del Software Propietario es un total desconocimiento para el usuario y además los resultados son impredecibles.

- Cursos de entrenamiento costosos

La incorporación de cualquier herramienta nueva obliga a cualquier empresa a invertir en capacitación para sus empleados con el ánimo de obtener el máximo provecho de ella esperando mejorar sus procesos de negocio. Dichas capacitaciones en la mayoría de los casos resultan un valor extremadamente alto ya que el tutor viene de fuera de la ciudad o del país y hay que cancelar aparte de la capacitación gastos varios como alojamiento, comida, etc., valores que a veces son inalcanzables por empresas pequeñas.

- Deficiente soporte técnico

Cuando se produce un error en cualquier producto de Software no Libre, el tiempo que tarda en recibir asistencia es elevado ya que inicialmente la compañía fabricante no quiere reconocer el error ya que dañaría su imagen, pero una vez que lo reconocen tardan demasiado para corregirlo y ofrecer una solución al cliente. Mientras tanto tiene que esperar ya que no puede depender de otra entidad encargada de ofrecer soluciones de ese tipo.

- Altos costos en la adición de módulos

Cuando una empresa que está utilizando una línea de productos de Software No Libre crece y necesita de herramientas adicionales que cubran sus nuevos procesos de negocio, se ve obligada a adquirir e integrar nuevos módulos adicionales que cubran sus expectativas, los mismos que tienen un costo elevado y que hacen difícil su adopción en la mayoría de los casos. Este problema se presenta en mayor parte en empresas pequeñas y medianas que tienen que pagar un costo elevado para beneficiarse del producto.

- Ilegalidad de copias del producto

Cualquier producto de Software No Libre posee licencias de uso que impiden la libre distribución de sus productos considerándose ilegal cualquier copia del mismo.

Cualquier innovación que se le quiera hacer al software es derecho exclusivo de la compañía fabricante, lo único que se podría hacer es vender la idea a la compañía propietaria.

- Poca seguridad en sus sistemas operativos

Existen muchas falencias de seguridad en sus sistemas operativos, tal es el caso de Microsoft con su producto Windows Vista que al parecer tiene varias vulnerabilidades según investigadores de IBM y VMWare quienes revelaron que existe una técnica que permite obtener el control total de Windows Vista a través de la forma en que se cargan las librerías dinámicas (DLLs) en memoria de la máquina cuando se ejecutan algunos programas como el navegador Internet Explorer, además de la ineficacia para impedir el paso de amenazas de spyware, adware y actualizaciones poco frecuentes y débil protección de antivirus y antispysware.

- Costo elevado de licencias

Las licencias de Software No Libre involucran un alto costo en su adquisición, mucha de las veces este costo supera a los beneficios esperados por la herramienta ya que no se ajusta de manera plena a las necesidades de la empresa.

- Monopolio de Servicios de Soporte

En la industria del software es muy común encontrar proveedores que no solo venden un producto a sus clientes, sino que ofrecen una gama completa de servicios que pretenden cubrir como soluciones en dichas empresas, es así que estas en la mayoría de los casos adquieren una gama de productos de software de un mismo proveedor formándose así un monopolio de Servicios de Soporte en el que el proveedor decide la calidad de soporte que pretende brindar a sus clientes.

2.3 Cuadro comparativo de Ventajas y Desventajas de Software Libre y Software no Libre.

CUADRO COMPARATIVO DE VENTAJAS Y DESVENTAJAS DEL SOFTWARE LIBRE Y SOFTWARE NO LIBRE			
Ventajas del Software Libre	Desventajas del Software Libre	Ventajas del Software No Libre	Desventajas del Software Libre
Costos bajos en la adquisición	Altos costos en la adición de módulos	Control de Calidad en los Productos	No existen garantías del producto
Libertad de uso y Distribución	Costo elevado de licencias	Fuerte relación con empresas fabricantes de Hardware	Menor compatibilidad con el Hardware
Mejora constante de la infraestructura tecnológica	Cursos de entrenamiento costosos	Protección del Software desarrollado	No existe protección del Software desarrollado
Libre manipulación de la infraestructura tecnológica	Ilegalidad de copias del producto	Plataforma comúnmente usada en todas las empresas	Riesgos en la migración a la plataforma
Reducción de los requisitos de Hardware	Amplios requisitos de Hardware	Interfaz gráfica bastante amigable	Interfaz gráfica poco amigable
Suporte y Compatibilidad a largo plazo	Monopolio de Servicios de Soporte	Toma de decisiones centralizada en el desarrollo del Software	Mala gestión del código fuente por decisiones descentralizadas
Aplicaciones sin puertas traseras	Existencia de puertas traseras en el código	Publicaciones y documentaciones acerca del uso	Mayor esfuerzo en el aprendizaje

Tabla 2.1 Comparación entre Ventajas y Desventajas de Software Libre y Software No Libre

2.4 Metodologías de Desarrollo de Software

Las metodologías de desarrollo de software se refieren a una estructura de soporte definida mediante el cual un proyecto de software puede ser organizado y desarrollado y están orientadas a estructurar, planear y controlar el proceso de desarrollo en sistemas de información. Una metodología de desarrollo consta generalmente de:

- La filosofía de desarrollo de software, esta depende de el enfoque del proceso de desarrollo de software.
- Herramientas, modelos y métodos para asistir al proceso de desarrollo de software.

Dependiendo de la metodología de desarrollo de software, se puede establecer diferentes enfoques de desarrollo, esta analogía se describe en la tabla 2.2.

ENFOQUE DE DESARROLLO	TIPO DE DESARROLLO
Modelo en Cascada	Desarrollo lineal
Prototipado	Desarrollo iterativo
Incremental	Desarrollo lineal & iterativo
Espiral	Desarrollo lineal & iterativo
RAD (Rapid Application Development)	Desarrollo iterativo

Tabla 2.2 Enfoques y Tipos de Desarrollo

Estos enfoques serán analizados en el capítulo 5 en conjunto con las metodologías de desarrollo de software. El desarrollo de software contrastan la forma de desarrollar software libre, respecto a desarrollar software no libre, debido principalmente a que el software no libre busca “proteger” su obra, entonces trata de ocultar a las demás personas características sobre su funcionamiento y la implementación de sus componentes, mientras que en el desarrollo de software libre, el código debe estar

disponible para brindar la libertad de entender y estudiar la implementación del software, además pensamos nosotros que además de incluir el código fuente, si el software se desarrolla basándose en una metodología concreta y generando la documentación correspondiente, sería mucho más eficiente su estudio, comprensión y modificación sus usuarios.

2.4.1 Metodologías para el desarrollo de Software Libre

Las metodologías de desarrollo de software libre deben enfocarse en la colaboración de sus programadores, además de facilitar su interacción con los usuarios finales. Prácticamente, en el desarrollo de software libre predominan las metodologías de desarrollo iterativas, especialmente las de desarrollo ágil y programación extrema. Pues estas son las que mejor se adaptan a los principios del software libre.

2.4.1.1 Metodologías Agiles

Las metodologías ágiles (AMS, Agile Methods) han tenido una gran acogida y desarrollo en la construcción del software libre, prácticamente creciendo a la par, debido principalmente a que las AMs y el Software Libre orientan el desarrollo de software hacia una organización menos formal y jerárquica. Dándole mayor prioridad a la persona, y centrándose en el objetivo principal del desarrollo, que es producir un sistema de gestión con la cantidad correcta de funcionalidades.

Esto significa que el sistema final tiene que incluir sólo el mínimo número de características necesarias para satisfacer por completo al cliente real. Las AMs y el software libre, rompen los valores establecidos para la producción de software de una manera muy exitosa respecto a los enfoques tradicionales que últimamente han fallado.

Los partidarios de las metodologías ágiles tienen varios principios que concuerdan con los principios del software libre. Los cuales se describen a continuación, y están publicadas en [http:// www.agilemanifesto.org](http://www.agilemanifesto.org):

- La máxima prioridad es satisfacer al cliente, a través de la entrega oportuna y continua de software de valor.
- Bienvenidos los requisitos cambiantes, incluso a finales del desarrollo. Los procesos ágiles aprovechan la ventaja competitiva del cliente.
- Trabajar en la entrega de software con frecuencia desde un par de semanas hasta un par de meses, tratando de que sean tiempos cortos.
- La gente de negocios y desarrolladores deben trabajar juntos a diario durante todo el proyecto.
- Construir proyectos en torno a individuos motivados. Brindándoles el medio ambiente y el apoyo que necesitan, y confiar en ellos para hacer el trabajo.
- El método más eficiente y eficaz de transmisión de información hacia y dentro de un equipo de desarrollo es la conversación cara a cara.
- Simplicidad, el arte de maximizar la cantidad de trabajo no realizado es esencial.
- Las mejores arquitecturas, requisitos y diseños salen de la autoorganización de los equipos.

Estas características buscan orientar el software hacia la persona y cubrir sus necesidades específicas, no se orientan en generar gran cantidad de documentación, sino solo la necesaria, considerando como principal documentación el correcto funcionamiento del programa y sus salidas, mediante captura de pantallas, además que al desarrollar en conjunto con el cliente, el resultado final es más fácil de comprender y utilizar para el cliente.

2.4.2 Metodología de desarrollo de Software No Libre

En el desarrollo de software no libre, por lo general se trata de mantener el conocimiento dentro de la organización que desarrolla el software, además de tener una estructura más jerárquica, y formal, también genera mucha documentación, además hay que tomar en cuenta que la metodología por si sola solo representa un documento muerto sino está orientada hacia las personas.

2.4.2.1 Metodologías de desarrollo tradicionales

Estas metodologías buscan seguir una secuencia en etapas válidas con tecnologías informáticas o manuales. Tienen un ciclo de vida, prácticamente fijo, y su desarrollo casi siempre es lineal, constan de las siguientes etapas:

- Diagnóstico: Descubrir el problema.
- Estudios de Factibilidad Económico, técnico y operacional.
- Diseño Lógico: Informe sobre qué hacer, qué cambiar, eliminar o resolver.
- Diseño Físico: Informe sobre cómo hacerlo, primer contacto con el Hardware.
- Construcción y Pruebas: Se hacen los programas y estructura de datos, y se prueban.
- Implementación: Poder a funcionar al software.

Y luego de su desarrollo, las fases de:

- Operación.
- Revisión post-implementación.
- Mantenimiento.

Cada etapa tiene un tipo de control el cual se encuentra al final de la misma y es donde se decide seguir con el proyecto o abortarlo. Estas metodologías buscar generar software de mayor calidad, respecto a las metodologías ágiles.

2.4.3 Metodologías Ágiles (Software Libre) vs Metodologías Tradicionales (Software no libre)

Metodologías Ágiles	Metodologías Tradicionales
Basada en la capacidad de realizar de forma inmediata innovaciones positivas, que provienen de producir software.	Basada en normas de estándares seguidos por el entorno de desarrollo.
Se adapta mejor a cambios durante el	Ofrece resistencia a los cambios.

Metodologías Ágiles	Metodologías Tradicionales
desarrollo de software.	
Procesos menos controlados, prácticamente sin normas.	Procesos más estrictos y controlados según políticas y normas.
No se basan en contratos, o de haberlos son muy flexibles.	Se basan en contratos, por lo general buscan generar valor monetario.
El cliente, o usuario es parte del equipo de desarrollo de software.	El cliente no participa en el desarrollo del software, a menudo solo asiste a reuniones.
Por lo general son grupos pequeños, trabajando dentro del mismo espacio físico.	Grupos grandes y generalmente distribuidos en diferentes ubicaciones.
Se utilizan pocos recursos.	Se utilizan muchos recursos.
Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.
Se orienta a satisfacer la necesidad específica del usuario.	Se orienta a generar un valor monetario, con el fin de obtener ganancias.
Enfoque de desarrollo iterativo, y flexible.	Enfoque de desarrollo lineal y estricto.
Ha tenido un gran desarrollo y evolución.	No se han desarrollado de tanto como las metodologías ágiles.
Buscan compartir sus conocimientos.	Buscan proteger y no compartir sus conocimientos.

Tabla 2.3. Comparación entre Metodologías Ágiles y Metodologías Tradicionales

CAPITULO III

LICENCIAS DE SOFTWARE LIBRE

Introducción

Como ya hemos comentado, el usuario de Software Libre, es libre de poder acceder, usar y modificar el código fuente del software, sin embargo es muy importante tener en cuenta algunas implicaciones legales que estas prácticas conllevan. El hecho de que el Software libre sea “libre”, no significa que sea gratis ni que el usuario pueda hacer lo que quiera con él, al contrario, tiene que seguir ciertas normas y estamentos. Esta confusión es común en muchas personas ya que asocian el término inglés “free” con gratuidad y piensan que el código se puede adquirir gratis, lo cual no es así.

En términos legales, tanto el Software Libre como Software Propietario se distribuyen bajo licencia. Una licencia es un contrato entre el propietario del software y el usuario final, es decir, es una autorización formal que el autor de un software otorga a un usuario para que pueda usar un producto, esta documentación contiene todas las normas y estamentos distribuidos en una serie de apartados bastante extensos en los que se incluyen por ejemplo los derechos del comprador, los límites en la responsabilidad por fallos en los productos, limitaciones en la reinstalación del producto en otros equipos, entre otros.

El presente capítulo pretende describir algunas de las licencias existentes en el Software Libre, clasificándolas según las condiciones que imponen al momento de redistribuir un producto, así como las posibles consecuencias de su uso; se detallan también algunas de las implicaciones legales que se deben tomar en cuenta al momento de hacer uso de una aplicación de Software Libre teniendo en cuenta las condiciones de uso, redistribución, modificación, uso y copias.

3.1 Clasificación

Existe una gran variedad de licencias de Software Libre, cada una de las cuales determinan si un usuario está en la capacidad de redistribuir un programa libre o si tiene que ajustarse a condiciones especiales al momento de su distribución. Como se mencionó en párrafos anteriores, la clasificación de las distintas licencias de Software Libre se las realiza teniendo en cuenta las condiciones del contrato o cláusulas del mismo, cada una de ellas con consecuencias diferentes en su uso.

A pesar de que no podríamos enumerarlas todas, mencionaremos las licencias más importantes clasificándolas en dos grandes grupos: licencias permisivas y licencias Copyleft, la tabla 3.1 lista diferentes licencias pertenecientes a estos grupos. Considerando que ambas siguen diferentes filosofías de desarrollo y distribución, la primera no impone condiciones al usuario, es decir, permite el uso, redistribución y modificación de un producto de software sin ninguna restricción; mientras que la última impone ciertas condiciones en el caso de la redistribución.

3.1.1 Licencias Permisivas

Este tipo de licencias no obligan necesariamente al usuario a que sus cambios realizados en la estructura del código siga siendo Software Libre, es decir, el usuario tiene libertad ilimitada con respecto al software producto de su modificación, además de ello puede redistribuirlo bajo términos diferentes a los del Software Libre, incluso sin haber realizado ninguna modificación.

Entre sus cualidades destacan la flexibilidad y fácil uso, además de que permite unir y compatibilizar código abierto y cerrado, generando una gran ventaja de desarrollo en aplicaciones empresariales que luego se podrán licenciar bajo otros términos distintos a los de este tipo de licencia. Entre las licencias permisivas más conocidas destacan las licencias BSD (Barkeley Software Distribution) y las licencias MPL (Mozilla Public License).

3.1.1.1 Licencias BSD

Empezó a usarse en la década de 1980 donde se dio a conocer las distintas modificaciones de Unix, entre sus estamentos, este tipo de licencia obliga a reconocer la autoría general y a darle crédito, sin que esto implique una obligación en la redistribución de fuentes o binarios.

La licencia de este tipo obligaba a sus usuarios a incluir una cláusula de publicidad de la universidad de Berkeley quién fuera la impulsora y creadora de esta licencia, a partir del año 1999 se omitió dicha cláusula, la misma que decía: “*All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the University of California, Berkeley and its contributors.*”⁶. Como ejemplo de software en el que se ha utilizado licencias BSD mencionamos al juego de iconos empleado en el J-Lan Communicator, un software diseñado para la comunicación entre diferentes hosts de una LAN usando el protocolo UDP.

3.1.1.2 Licencias tipo MPL

Mozilla Public Licence es la licencia propia de Mozilla.org y Netscape usada en varias de sus aplicaciones, las cuales se rigen bajo las cuatro libertades del Software Libre enunciadas por la FSF (Free Software Foundation). Algunas de sus restricciones hacen que la licencia sea incompatible con la GPL (General Public Licence) ya que no permite redistribuir trabajos derivados bajo la licencia original y cualquier cambio en el código fuente debe ser incluido en el resto de versiones en un plazo de 12 meses o 6 meses si se ha realizado un ejecutable del programa con dichos cambios, además hasta la versión 1.1 no se podían enlazar módulos de GPL con MPL.

El contrato de la licencia se encuentra en la página principal de Mozilla.org y según el sitio web: “*The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the*

⁶ Tomado de: GNU Operating System, “El problema de la licencia BSD”,
<http://www.gnu.org/philosophy/bsd.es.html>,
Fecha de Consulta 08 de Septiembre del 2009

*License. You may obtain a copy of the License at <http://www.mozilla.org/MPL/>”⁷. Actualmente *Google Code* está utilizando licencias MPL para compartir con todos sus usuarios parte del código de programación que se utiliza dentro de la compañía.*

3.1.2 Licencias Copyleft

El término Copyleft proviene de un juego de palabras en inglés que invierte el sentido y la jurisdicción del Copyright, fue creada por Richard Stallman del movimiento de Software Libre con el fin de hacer del software una herramienta libre y al alcance de todos.

Este tipo de licencia se encarga de las modificaciones relevantes que afectan de manera directa a los términos de explotación de un programa, transmitiendo así los efectos de la licencia del programa originario a las licencias de los programas derivados, es decir, prohíbe que de un programa libre se obtenga un derivado con fines comerciales o que se redistribuya con restricciones adicionales a las libertades de los usuarios finales.

Según la FSF, *“The simplest way to make a program free software is to put it in the public domain, uncopyrighted. This allows people to share the program and their improvements, if they are so minded.”*⁸

Copyleft no afecta directamente a los derechos de autor del programa originario, pero sí a los del autor del programa derivado en el momento en que redistribuye la obra, ya que este último aceptó las restricciones impuestas por el primero y está sujeto a infracciones de licencia por incumplimiento según las cláusulas Copyleft que haya impuesto el autor de la obra original.

Dichas infracciones se refieren al cese del amparo que brinda la licencia a él y al programa derivado que redistribuya convirtiéndose automáticamente en un producto

⁷ Tomado de: Mozilla Code Licensing, “Mozilla Public License Version 1.1”, <http://www.mozilla.org/MPL/MPL-1.1.html>, Fecha de consulta 09 de Septiembre del 2009

⁸ Tomado de: FSF Free Software Foundation, “What is Copyleft”, <http://www.fsf.org/licensing/essays/copyleft.html/view?searchterm=copyleft>, Fecha de Consulta 10 de Septiembre del 2009

ilegítimo. La estructura de las cláusulas Copyleft basan su contenido en los siguientes elementos:

1. Cualquier explotación en términos diferentes a los términos Copyleft no queda amparada por la licencia.
2. Cualquier persona que redistribuya un programa Copyleft u otros derivados de él, ha de poner a disposición del receptor una licencia Copyleft de iguales características, sin restricciones adicionales.
3. Si se desea incorporar partes del programa Copyleft a otros programas libres que tengan condiciones de distribución distintas, se debe obtener un permiso del autor original.
4. No puede redistribuirse un programa Copyleft si por impedimento forzoso, la redistribución no va a poder ser Copyleft.

Con la finalidad de que un trabajo no fuera estéril, fácilmente privatizable y bajo los preceptos del Copyleft, se lanza en 1989 la primera versión de la GPL o Licencia Pública General que vendría a ser el logro más grande de la FSF no solo en el ámbito informático sino también jurídico. GNU GPL es una licencia creada por la FSF a mediados de la década de 1980, y su objetivo principal es proteger la libre distribución, modificación y uso de software al declarar que el software bajo esta licencia es software libre, y protegerlo de intentos de apropiación que limiten la libertad a los usuarios.

La licencia GPL tiene la forma de un contrato por lo que usualmente se la denomina contrato de licencia o acuerdo de licencia. Para ser válida la GPL debe cumplir los requisitos legales de formación contractual en cada jurisdicción, un ejemplo común de licencia MPL es la tan conocida enciclopedia *Wikipedia*. En el siguiente apartado se tratará más a fondo el tema de las licencias GNU.

3.1.2.1 Ventajas del Copyleft

- Libertad para escoger que licencia se adapta mejor a las necesidades.

Cuando se elige una licencia Copyleft, cualquier desarrollador es libre de escoger el tipo de licencia y nivel de protección que desea dar a su programa, manteniendo así un control total sobre la obra sin que nadie ejerza poder sobre ella, además no se depende de ninguna entidad que gestione los derechos de forma autónoma.

- Garantía de que los trabajos generados disponen de una licencia segura y legal.

Todo producto de software generado bajo las normativas de Copyleft dispondrá de una cobertura legal que asegura al mismo de un uso diferente a los planteados por el autor original, los mismos que quedan protegidos totalmente al igual que una licencia tipo Copyright.

- Autoría y propiedad de los trabajos garantizada

Una vez que un autor registra su producto de software bajo licencia Copyleft, esta deja claramente señalada la autoría del mismo y que siempre debe quedar reconocida en reproducciones o modificaciones posteriores.

- Duración de la licencia a lo largo del tiempo

Una de las características de una licencia Copyleft es que esta se transmitirá de un trabajo a otro, asegurando así que nadie se adueñe de un trabajo ajeno a lo largo del tiempo aunque este se reproduzca, se copie o se hagan obras derivadas de la original.

- Gratuidad de la obtención de la licencia Copyleft

Todas las licencias Copyleft se las puede adquirir de forma gratuita en la red, lo cual incentiva a la sociedad a licenciar sus obras bajo los términos antes estudiados y hacer llegar su trabajo a un número amplio de personas y de cierta forma conseguir un prestigio profesional y reconocimiento público.

3.1.2.2 Desventajas del Copyleft

- Hace referencia a las licencias que no se heredan a todos los trabajos derivados, según como estos se hayan derivado, es decir, se mencionan todo el grupo de licencias Copyleft según como la obra se vaya redistribuyendo.
- Al momento de redistribuir un programa, cualquier cambio efectuado sobre el mismo debe difundirse con Copyleft a excepción de cambios sobre el software que se relaciona con él, lo que permite a programas con cualquier licencia vincularse con programas con Copyleft.
- Algunos términos de Copyleft permiten que las creaciones no estén completamente sujetas a todos los principios del Copyleft, esto se conoce como Copyleft parcial.

3.1.3 Listado de licencias de Software Libre

Licencia	Autor	Ultima Versión	Fecha de Publicación	Tipo de Licencia	Posibilidad de diferentes licencias	Cambios al software bajo otro tipo de Licencia
Academic Free License	Lawrence E. Rosen	3	2002	Permisiva	Si	Si
Apache License	Apache Software Foundation	2.0	2004	Permisiva	Si	Si
Apple Public Source License	Apple Computer	2.0	2003	Copyleft	Si	No
Artistic License	Larry Wall	2.0	2000	Permisiva	Si	Parcial
Berkeley Database License	Oracle Corporation		2008	Permisiva	No	No
BSD license	Universidad de California			Permisiva	Si	Si
Boost Software License		1.0	2003	Copyleft	Si	Si

Licencia	Autor	Ultima Versión	Fecha de Publicación	Tipo de Licencia	Posibilidad de diferentes licencias	Cambios al software bajo otro tipo de Licencia
Common Development and Distribution License	Sun Microsystems	1.0	2004	Copyleft	Si	Si
Common Public License	IBM	1.0	2001	Copyleft	Si	No
Creative Commons Licenses	Creative Commons	3.0	2002	Copyleft	No	Parcial
Cryptix General License	Cryptix Foundation		1995	Permisiva	Si	Si
Eclipse Public License	Eclipse Foundation	1.0		Copyleft	Si	No
Educational Community License		1.0		Copyleft	Si	Si
Eiffel Forum License	NICE	2	2002	Copyleft	Si	Si
GNU General Public License	Free Software Foundation	3.0	2007	Copyleft	No	No
GNU Lesser General Public License	Free Software Foundation	3.0	2007	Copyleft	Si	No
Hacktivismo Enhanced-Source Software License Agreement	Hacktivismo/Cult of the Dead Cow		2002	Copyleft	No aplica	No aplica
IBM Public License	IBM	1.0	1999	Copyleft	Si	Si
Intel Open Source License	Intel Corporation			Copyleft	Si	Si
ISC license	Internet Systems Consortium			Permisiva	Si	Si
LaTeX Project Public License	LaTeX project	1.3c		Permisiva	Si	Si

Licencia	Autor	Ultima Versión	Fecha de Publicación	Tipo de Licencia	Posibilidad de diferentes licencias	Cambios al software bajo otro tipo de Licencia
MIT license / X11 license	MIT		1988	Permisiva	Si	Si
Mozilla Public License	Mozilla Foundation	1.1		Copyleft	Si	limitada
Netscape Public License	Netscape	1.1		Copyleft	No aplica	No aplica
Open Software License	Lawrence Rosen	3.0	2005	Copyleft	Si	No
OpenSSL license	OpenSSL Project			Copyleft	No aplica	No aplica
PHP License	PHP Group	3.01		Permisiva	Si	Si
Poetic License	Alex Genaud	1.0	2005	Permisiva	Si	Si
Python Software Foundation License	Python Software Foundation	2		Permisiva	Si	Si
Q Public License	Trolltech			Permisiva	No	No
Sun Industry Standards Source License	Sun Microsystems			Permisiva	Si	No
Sun Public License	Sun Microsystems			Copyleft	Si	No
Sybase Open Watcom Public License		2.0	2004	Copyleft	Si	No
W3C Software Notice and License				Permisiva	Si	Si
XFree86 1.1 License				Permisiva	Si	Si
Licencia zlib/libpng				Permisiva	Si	Si
Zope Public License				Permisiva	Si	Si

Tabla 3.1. Listado de Licencias de Software Libre

3.1.4 Cuadro comparativo de los tipos de Licencia de Software Libre

En la tabla 3.2 se comparan las diferentes licencias de Software Libre analizadas en el capítulo.

Licencias Permisivas	Licencias Copyleft
Licencias BSD Licencias MPL	Licencias GNU

Tabla 3.2 Cuadro comparativo de los tipos de licencia de Software Libre

3.2 Validez Legal

3.2.1 ¿Que es validez legal?

Una norma es válida cuando cumple con los requisitos formales y materiales necesarios para su producción. Es decir una norma es válida cuando existe de acuerdo con el Derecho.

“El término validez alude a una propiedad de los actos o de las normas y significa "existencia jurídica". Con la existencia jurídica o validez, se quiere aludir a que los actos y las normas que se derivan de esos actos, son actos humanos y normas que serán considerados actos jurídicos y normas jurídicas. Para cumplir con los requisitos establecidos en el ordenamiento jurídico. Una norma es válida cuando la podemos identificar como perteneciente a un sistema jurídico, cuando existe un acuerdo con el Derecho.”⁹

3.2.2 Validez Legal del Software Libre

Desde sus inicios el Software Libre ha tenido una muy buena acogida tanto en su desarrollo, uso y distribución por parte de sus usuarios, debido principalmente a que presentan otros beneficios tales como la libertad de elección, la protección de la

⁹ Tomado de: Wikipedia.org, “Validez Jurídica”,
http://es.wikipedia.org/wiki/Validez_jur%C3%ADdica.html,
Fecha de Consulta 06 de Septiembre del 2009

inversión, buena comunicación e interoperabilidad, entre otros. Pero siempre teniendo que lidiar con aspectos jurídicos que no le permitían garantizar la libertad de sus usuarios o evitar las apropiaciones de software.

Al inicio el Software Libre se publicó bajo la licencia GPL (dominio público), pero esta licencia, permite que las modificaciones que se realicen al software se registren como software privativo, de esa forma varias personas y empresas se adueñaron de varias obras, esto motivó a los partidarios del Software Libre como la FSF, y el proyecto GNU a crear un marco legal que permitiera cumplir con sus principios, buscando que el Software Libre este sometido a las leyes de propiedad intelectual y derechos de autor, pero con la gran diferencia que estos promuevan las libertades del usuario, en lugar de limitar su uso, estudio, modificación y distribución como es en el caso de software no libre. De lo que nació el concepto de Copyleft, un grupo de derechos de autor definidos para eliminar las restricciones de distribución o modificación impuestas por el copyright, pero con la condición de que el trabajo resultante mantenga el mismo sistema de derechos de autor que el original. Además de las licencias GNU en sus diferentes versiones y otras licencias compatibles con las licencias GNU, para evitar estas apropiaciones y que favorezcan a los principios del Software Libre.

“En el proyecto GNU, nuestro objetivo es el dar a todo usuario la libertad de redistribuir y cambiar software GNU. Si los intermediarios pudieran quitar esa libertad, nosotros tendríamos muchos usuarios, pero esos usuarios no tendrían libertad. Así en vez de poner software GNU en el dominio público, nosotros lo protegemos con Copyleft. Copyleft garantiza que cada usuario tiene libertad.”¹⁰

Para garantizar la validez legal del Software Libre se ha buscado que sus licencias sean reconocidas mundialmente, mediante tratados internacionales, para ello la Organización Mundial de la Propiedad Intelectual o World Intellectual Property Organization (OMPI o WIPO) estableció el Convenio de Berna para la Protección de las Obras Literarias y Artísticas, y también el tratado de la OMPI sobre Derecho de Autor, los cuales han sido aceptados por la mayoría de países, los que reconocen que el Software Libre y de fuentes abiertas está sometido a propiedad intelectual y

¹⁰ Tomado de GNU Project: Free Software Foundation, “Copyleft”, <http://www.gnu.org/copyleft/copyleft.es.html>, Fecha de Consulta 07 de Septiembre del 2009

derechos de autor, igual que el software no libre, pero con licencias que permiten cumplir su objetivo de brindar al usuario libertad de uso, estudio, modificación y distribución.

La FSF siempre ha estado tratando de fortalecer el marco jurídico para el desarrollo y distribución de Software Libre, buscando mejorar el Copyleft y las licencias GNU. Aunque ha resultado una tarea muy ardua y difícil debido a las diferentes legislaciones existentes en varios países y a las varias trabas a las que se ha expuesto al Software Libre, la FSF trabaja con varios gobiernos tratando de crear un marco legal que garantice las libertades que el Software Libre ofrece a sus usuarios. Teniendo muy buenos resultados, tanto así que muchos países incluido Ecuador han declarado como política de estado el uso de Software Libre.

La licencia GPL al ser un documento que cede ciertos derechos al usuario, asume la forma de un contrato, por lo que usualmente se la denomina contrato de licencia o acuerdo de licencia. Esta es reconocida por la OMPI (Organización Mundial de Propiedad Intelectual), en lo relacionado a *Derechos de Autor y Derechos Conexos*, nuestro país es parte del comité Permanente de Derechos de Autor y Derechos Conexos de la OMPI y participa de el activamente, además de la política de uso de Software Libre en instituciones de administración pública, lo que ha impulsado mucho el desarrollo y uso de Software Libre, por lo que las licencias de Software Libre son reconocidas y válidas en nuestro país.

Cabe destacar que el Instituto Ecuatoriano de propiedad intelectual (IEPI) es el responsable de hacer cumplir con lo relacionado a derechos de autor en nuestro país, además en el IEPI podemos registrar las aplicaciones de software que sean desarrolladas, aunque por definición de derechos de autor, Los derechos nacen con la creación de la obra.

“No es preciso por lo tanto ninguna inscripción registral para el nacimiento de los derechos de autor y afines. Pero la inscripción se presume cierta, quien alegue que no lo es debe probarlo, y quien tiene algo inscrito a su nombre no necesita probar que es suyo”¹¹.

¹¹ NONIUS, Jorge: Introducción a la propiedad intelectual, <http://www.laespinal.org/articulos/propintlect/propintlect.pdf>, página 43, Fecha de Consulta 09 de Septiembre del 2009

3.3 Licencias GNU GPL

GNU GPL es una licencia creada por la FSF a mediados de la década de 1980 y su objetivo principal es proteger la libre distribución, modificación y uso de Software Libre y protegerlo de intentos de apropiación que limiten la libertad a los usuarios.

Además la GNU GPL cuenta con otras licencias complementarias, como la licencia de documentación libre de GNU (GFDL), la Open Audio License para trabajos musicales, la LGPL y la LGPL (Lesser General Public License) Licencia Publica General Reducida que facilitan el enlace dinámico de aplicaciones libres a aplicaciones no libres.

La licencia GPL tiene la forma de un contrato por lo que usualmente se la denomina contrato de licencia o acuerdo de licencia. Para ser válida La GNU GPL debe cumplir los requisitos legales de formación contractual en cada jurisdicción. Para lo que debemos registrar el software en el IEPI.

3.3.1 GNU GPL V3

La versión oficial fue publicada el día 29 de junio de 2007 y está disponible en el sitio web del proyecto GNU. En la versión actual de GNU GPLv3 se pretende mejorar los siguientes aspectos:



- Formas en que algunas personas podrían quitar libertades a los usuarios.
- Prohibir el uso de software libre en sistemas DRM (Gestión Digital de Restricciones).
- Aumentar la compatibilidad con otras licencias.
- Facilitar su adaptación a otros países.
- Incluir cláusulas que defiendan a la comunidad de Software Libre del uso indebido de las patentes de software.

Además busca evitar que Copyleft sea desmejorado por los avances tecnológicos o legales. En la GNU GPL V3 se protege a los usuarios de las siguientes amenazas:

- Tivoization: Algunas empresas han creado diferentes tipos de dispositivos que ejecutan software GPL, pero de forma tal que el hardware pueda modificar el software que se ejecuta, mientras que el usuario no puede. Cuando un dispositivo impide al usuario realizar cambios al Software Libre, se lo denomina tivoization.
- Leyes que prohíben el software libre: En legislaciones como la Digital Millennium Copyright y la Directiva Copyright de la Unión Europea se establece que se convierte en un delito escribir o compartir programas informáticos que pueden no estar de acuerdo con los DRM (Restricciones de Mala Gestión Digital). Estas leyes no deben interferir con los derechos que la GPL otorga a sus usuarios.
- Ofertas discriminatorias de Patentes: Microsoft está tratando de reunir las regalías por el uso de Software Libre, esto interfiere con la libertad de los usuarios. Ninguna empresa debería ser capaz de hacer esto.

La versión 3 busca mejorar la compatibilidad entre software GPL y software no GPL, cuenta con mejoras para hacer más fácil su uso y comprender así el porqué de la licencia para todos.

3.3.1.1 Características de la licencia GNU GPL

La Licencia GNU GPL la usan la mayoría de los programas de GNU y más de la mitad de las aplicaciones de Software Libre. Aunque cabe recalcar que un Software Libre no puede estar protegido solo por esta licencia ya que existen otras licencias que brindan a los usuarios las libertades necesarias para considerar al software como libre.

Médiante la utilización de la licencia GNU GPL se puede exigir que todas las versiones posteriores (mejoradas) del software sean Software Libre, esto busca evitar el riesgo de la apropiación de software.

La licencia GNU GPL busca también distinguir entre diferentes versiones del mismo programa, las cuales pueden ser modificadas por diferentes usuarios al exigir al autor

de una versión que ponga su nombre en ella, y también busca proteger la reputación de los usuarios que han aportado a la construcción y mantenimiento del software.

Otra característica importante de la licencia GNU GPL es que si el usuario modifica el software para su uso privado, esta licencia no obliga al usuario a publicar una versión de software modificada, el usuario es libre de modificar software bajo esta licencia y utilizarlo en privado, esto permite adaptar el software a necesidades específicas, es decir, si el software no es público, el usuario no está obligado a publicar el código fuente de la versión modificada, pero si el software se publica de alguna manera, el usuario tiene la obligación de publicarlo con su código fuente bajo la licencia GNU GPL, es decir, el usuario está en libertad de publicar las versiones modificadas del software o de no hacerlo.

Si se publican versiones modificadas de software bajo esta licencia, estas deben ser licenciadas para todas las “terceras partes”, lo que quiere decir que todos los usuarios que accedan al software tienen la licencia del desarrollador bajo la licencia GNU GPL, sin que ello signifique que el desarrollador del software deba hacer algo físicamente para estos usuarios.

El software bajo licencia GPL, puede ser vendido, ya que el derecho de vender copias del programa es parte de la definición del Software Libre, no existe límites al precio del software siempre y cuando el programa sea acompañado de su código fuente o de la oferta escrita de proporcionar el código fuente si este es distribuido solo con sus archivos binarios.

La licencia GNU GPL no autoriza al desarrollador de software a exigir dinero o notificaciones de las personas que obtengan copias del programa, ya que usar y distribuir el programa es parte del concepto del Software Libre, la licencia GNU GPL es una licencia de Software Libre, por lo tanto busca garantizar su libre uso y distribución.

Si se distribuye software GNU GPL sin el código fuente, la persona que lo distribuye debe hacer por escrito una oferta de proporcionarlo posteriormente, esta oferta es válida para “cualquier tercera parte”, lo que quiere decir que si un usuario obtiene los archivos binarios de un programa por cualquier otro método, puede solicitar a sus autores que le proporcionen el código fuente.

La licencia GNU GPL permite desarrollar una versión modificada del programa bajo acuerdos de no divulgación, mientras que el programa no esté terminado o el cliente no de su aprobación ya que en esta situación no habría código bajo licencia GNU GPL que este siendo distribuido bajo un acuerdo de no divulgación, esta licencia concede al usuario el derecho de distribuir el software, pero este puede elegir si lo hace o no.

La licencia GNU GPL exige que se coloque una copia de esta primero en todo programa que sea distribuido, para que cualquiera que obtenga una copia de estos programas sepa cuáles son sus derechos, esta licencia debe ser íntegra con todos sus componentes incluyendo el preámbulo de la GPL y las instrucciones acerca de su uso en los propios programas, cabe recordar que la licencia GNU GPL tiene Copyright por lo que no se puede modificar.

3.3.1.2 Licencias compatibles con GNU GPL

Para que dos licencias sean consideradas compatibles, se debe poder combinar 2 o más programas o partes significativas de los mismos dentro de una aplicación más amplia teniendo permiso para poder realizarlo, si las 2 o más licencias de los programas permiten realizar esto se dice que las licencias son compatibles. Si no es posible respetar las cláusulas de las licencias a la vez se dice que estas licencias no son compatibles, la compatibilidad de las licencias muchas veces depende del tipo de combinación que se realice, por ejemplo algunas licencias permiten vincular módulos de programas pero no combinarlos en uno solo.

Cuando una licencia es compatible con GNU GPL, quiere decir que el otro programa que tiene una licencia distinta se puede combinar con otro bajo licencia GNU GPL dentro de uno más extenso, siempre que se publique bajo este tipo de licencia.

3.3.2 Licencia GNU LGPL (Lesser General Public License)

La Licencia GNU LGPL antes se la denominaba GNU Library General Public License (Licencia Pública General para Bibliotecas de GNU), es una licencia de software creada por la FSF.



Esta licencia contiene un Copyleft que no es tan fuerte ya que permite que el software se enlace con módulos no libres, es utilizada principalmente en la elaboración de módulos de programas.

Una característica importante de la LGPL es que se puede convertir cualquier código LGPL en código GPL. Esta característica es útil para la reutilización directa de código LGPL en código GPL de bibliotecas y aplicaciones o si se quisiera crear una versión del código que no pueda utilizarse en software privativo.

3.3.3 Licencia GNU FDL (Free Document License)

La licencia de Documentación Libre de GNU es una forma de Copyleft para ser usada en un manual, libro de texto u otro documento que asegure que todo el mundo tiene la libertad de copiarlo y redistribuirlo con o sin modificaciones, de modo comercial o no comercial. Esta licencia complementa a las licencias de GNU GPL, y LGPL debido a que la documentación es parte imprescindible del programa y no sería aceptable que el Software Libre no tenga documentación libre.

Esta licencia no se limita a manuales o documentación de software, puede usarse para cualquier texto, sin importar su temática o si se publica como libro impreso o no. Por lo general se usa en trabajos cuyo fin es ser instructivo o de referencia.

CAPITULO IV

ORGANIZACIONES Y FUNDACIONES QUE RESPALDAN AL SOFTWARE LIBRE

Introducción

El Software Libre desde sus inicios ha tenido una gran acogida y evolución, paso de ser una iniciativa de pocas personas a una filosofía compartida por millones, pero ha tenido que enfrentar muchos inconvenientes sobre todo en lo que respecta a la parte legal, debido a que sus desarrolladores no podían hacer cumplir su filosofía en sus programas, por ello surgieron varias organizaciones que le brindan soporte y están continuamente promoviendo su uso, desarrollo y defendiendo los principios del mismo.

Estas Organizaciones han crecido significativamente hasta llegar a convertirse en Organizaciones a nivel mundial, con millones de participantes. Entre las organizaciones que dan soporte al software libre y de fuentes abiertas se destacan las siguientes:

- El proyecto GNU que proporciona a los usuarios varias aplicaciones libres siendo su proyecto más representativo el sistema operativo GNU el cual ha tenido una acogida excelente, sobre todo desde que se unió al kernel Linux formando el S.O. GNU con Linux.
- La FSF (Free Software Foundation) que busca brindar soporte al proyecto GNU sobretodo en aspectos de cobertura legal, económica y logística, además de promover y defender el uso y desarrollo del Software Libre.
- Y por último la OSI (Open Source Initiative) que busca educar sobre los beneficios del software de fuentes abiertas y desarrollar los aspectos técnicos del desarrollo de software mediante la emisión de certificaciones de software de código abierto.

4.1 FSF (Free Software Foundation)

La FSF es una fundación para el Software Libre que nació a través de la iniciativa de



Richard Stallman y otros partidarios de la filosofía del Software Libre, fue creada en Octubre de 1985 principalmente para garantizar la difusión, el uso y el conocimiento del Software Libre en diversas áreas de la informática y el sistema operativo GNU.

En sus inicios la FSF estaba orientada a contratar programadores para crear Software Libre y de esta manera promover su desarrollo y uso, pero con el pasar del tiempo ha tenido una muy buena aceptación y como consecuencia de ello surgen varias empresas y autores que se dedicaron a desarrollar Software Libre, por lo que la FSF cambió su orientación hacia asuntos legales, organizativos y promocionales en beneficio de la comunidad de usuarios.

4.1.1 Actividades que realiza la FSF

La FSF siempre busca diferentes maneras de promover y defender al Software Libre y a sus usuarios, para ello entre sus actividades principales consta:

- Brindar soporte al proyecto GNU sobretodo en aspectos de cobertura legal, económica y logística.
- Elaborar, mantener y defender las licencias GNU GPL que es la licencia más utilizada en el Software Libre, además también debe elaborar mantener y defender otras licencias tales como:
 - La licencia de documentación libre GNU FDL.
 - La licencia pública general reducida (GNU LGPL).
- Hacer cumplir las licencias de Software Libre que están a su cargo mediante la recepción de denuncias de violaciones a las cláusulas de las licencias utilizando como instrumento legal la presentación de demandas. Pero esto se puede hacer únicamente con los programas sobre los que se tiene derechos de autor.

- Aloja proyectos de Software Libre ofreciendo una interfaz Web para hosting y el mantenimiento de las páginas Web de los proyectos, además de seguimiento de errores, listas de correo y transferencia de archivos, se estima que a la fecha hospeda más de 2800 proyectos de Software Libre en su portal GNU Savannah.
- La FSF busca defender los derechos de los usuarios y desarrolladores de Software Libre, para ello organiza seminarios para brindar formación legal y educación a los usuarios sobre qué aspectos legales son importantes de conocer cuando se utiliza software con licencia GNU GPL.
- Administra el proyecto Free Software Directory en el que almacena los programas que comprueba que son Software Libre, la Unesco ayuda a la FSF con el financiamiento de este proyecto. Se estima que actualmente existen más de 5000 programas dentro de este directorio.
- Busca premiar a las personas u organizaciones que hagan grandes aportes al avance del Software Libre mediante la entrega de los premios “FSF Award for the Advancement of Free Software” y “Free Software Award for Projects of Social Benefit”.
- Busca que la información sea asequible para los usuarios de Software Libre mediante la publicación de libros sobre todo de informática utilizando licencias de libre distribución mediante GNU Press.
- Pone en marcha campañas para incentivar el uso de software libre en lugar de software no-libre. Entre las más destacadas están las siguientes:
 - BadVista.org (Mala vista) que busca promover alternativas libres en lugar del sistema operativo Windows Vista.
 - PlayOgg.org para promocionar los archivos Ogg en lugar los MP3 y los AAC.
 - Free BIOS que busca apoyar la creación de una BIOS libre.
 - DefectiveByDesign.org (Defectuoso por diseño) que lucha para eliminar el DRM (Gestión Digital de Restricciones).

La FSF ha tenido mucho éxito en las actividades que realiza por lo que Charity Navigator, empresa que supervisa a la FSF, la calificó con 4 estrellas lo que quiere decir: "*Excepcional. Supera las normas del sector y lo hace mejor que la mayoría de las organizaciones benéficas*".¹²

Como consecuencia de su apoyo al Software Libre, la FSF goza de un gran respaldo de los partidarios del mismo, pero esto también le ha significado hacerse de varios enemigos sobre todo los que pertenecen a la industria del software no libre, además de varias personas que no comparten en su totalidad la filosofía del Software Libre que propone la FSF han creado líneas alternativas tales como la OSI.

4.1.2 Fundaciones para el Software Libre (FSFs)

La FSF ha buscado maneras más eficientes para poder realizar actividades alrededor de todo el mundo para promover y defender el Software Libre, por ello la FSF ha creado la red internacional de Fundaciones para el Software Libre (FSFs,). "*La creciente popularización del uso de Software Libre en el mundo, torna importante la consolidación de una red de FSFs regionales que trabajen articuladas, sosteniendo y fortaleciendo la filosofía, el marco jurídico y los ideales del Software Libre, de acuerdo a la definición de la FSF*"¹³, en las que constan:

- FSFLA, Fundación para el software libre Latinoamericana.
- FSFE, Fundación para el software libre Europea.
- FSFI, Fundación para el software libre para la India.
- FSFEUA, Fundación para el software libre de Estados Unidos de América.

Las mismas buscan localizar los esfuerzos en sus respectivas localidades para promover el Software Libre y dar respaldo a sus usuarios, todas estas organizaciones se consideran como hermanas debido a que buscan practicar los mismos valores,

¹² Tomado de: Charity Navigator, "Calificación de la FSF", <http://charitynavigator.org/index.cfm?bay=search.summary&orgid=8557>, Fecha de Consulta 18 de Septiembre del 2009.

¹³ Tomado de: Fundación del software libre de América Latina, "Declaración de intensiones", <http://www.fsfla.org/svnwiki/about/declaration-of-intent.es.html>, Fecha de Consulta 20 de Septiembre del 2009.

filosofía y compartir sus objetivos, siendo el trabajo coordinado muy importante para evitar cualquier tipo de sectorización o regionalización de la FSF.

4.1.2.1 FSFLA (Fundación para el Software Libre Latinoamericana)

La FSFLA es una organización que surge a partir de la FSF de Norte América, de la FSFE (Free Software Foundation Europa) y de la FSFI (Free Software Foundation India), buscando ser parte del fortalecimiento de esta red internacional de FSFs, tiene como objetivos:

- Actuar en conjunto con las demás FSFs para promover y defender el Software Libre.
- Alentar a los gobiernos de la región de aportar el Software Libre como política pública, ayudándoles a crear un marco jurídico adecuado.
- Alentar a las instituciones educativas a usar exclusivamente Software Libre para el uso de sus alumnos.

Estos objetivos se ven complementados con su misión que es defender los derechos y las libertades de usuarios y desarrolladores de software, luchar por la libertad de ejecutar el software que se use para cualquier propósito, de estudiar su código fuente y poder adaptarlo para que cumpla necesidades específicas, de copiarlo, distribuirlo y publicarlo cuando se desee, con o sin mejoras, de manera que todos puedan usar computadoras en libertad.

Actualmente se está trabajando en los objetivos específicos de la organización y los fundamentos políticos de su funcionamiento. Se puede contribuir directa o indirectamente con la FSFLA. La forma más directa de contribuir es involucrarse en alguno de los espacios abiertos y equipos activos de la organización tales como:

- Anuncios: Información regular sobre el trabajo de la FSFLA como boletines mensuales y publicación de anuncios para la prensa.
- Discusiones: Lista de discusiones generales para tratar asuntos relacionados a Software Libre, su uso, distribución y desarrollo en América Latina.

- Traducciones: Traducción de documentos, páginas, boletines y toda otra información necesaria para el trabajo de FSFLA en alguno de los idiomas de trabajo de la organización.
- Campaña contra DRM: Participar de la lucha contra los DRM (Gestión Digital de Restricciones) en América latina.
- Free Open Standards: Acciones de divulgación y incentivo al uso de estándares que promuevan interoperabilidad y prevengan monopolios y dependencias exclusivas y de denuncia y oposición a estándares propietarios.
- Iniciativa (GNU)²: Busca aumentar el contacto de FSFLA con comunidades y activistas comprometidos con el Software Libre en América Latina.
- Asuntos legales: El grupo de asuntos legales de FSFLA se dedica al análisis de todo lo que tiene que ver con aspectos jurídicos del Software Libre y el impacto de las legislaciones, normas vigentes, tratados de libre comercio y todo otro proceso legal sobre el uso, distribución y desarrollo de Software Libre.
- Eventos, stand y promociones: Discusiones sobre la presencia de la FSFLA en eventos, preparación de stands y productos promocionales.
- Sitio Web: Información relevante sobre Software Libre en la zona de actuación de la FSFLA, novedades, eventos, legislación, política, etc.
- Educación: Para cumplir con uno de los objetivos prioritarios de FSFLA, habilitamos la lista de correos para Educación y Software Libre a la cual invitamos académicos, educadores y activistas interesados.
- Modelos de negocios: Busca trabajar en modelos de negocios y asistencia a proyectos lucrativos ya sean individuales o colectivos, empresariales, cooperativos o de profesionales independientes que deseen trabajar en emprendimientos basados en Software Libre.
- Administración Pública: Este equipo tiene como su objetivo apoyar a las administraciones públicas y entes gubernamentales que ya trabajan en Software Libre o están evaluando la decisión política.

- Prensa: Colaborar con el trabajo de comunicaciones de FSFLA.

4.2 Proyecto GNU

El proyecto GNU fue fundado en el año de 1984 por Richard Stallman quien es considerado el padre del software libre, con el afán de eliminar las restricciones sobre la copia, redistribución y modificación de programas de software existentes en licencias privativas y así desarrollar un sistema operativo totalmente libre capaz de eliminar la necesidad de uso del software privativo o no libre.

En la Década de 1970 Richard Stallman elaboró un intérprete de Lisp (LIStProcessing), la compañía Symbolics se interesó en este intérprete por lo que Richard Stallman accedió a facilitarles una versión bajo licencia de dominio público. Luego dicha empresa amplió y mejoró el software original, Richard Stallman pretendió acceder a dichas modificaciones pero esta compañía se negó a proporcionárselas. Stallman decidió erradicar este tipo de comportamiento al que lo denominó “software hoarding” (acaparamiento del software), creando su propio sistema operativo libre completo que garantice las libertades al usuario en todos los aspectos.

Con esta iniciativa se lanza el proyecto GNU, acrónimo que significa GNU’s Not Unix, creando de igual manera la FSF (Free Software Foundation) para dar soporte institucional, logístico, legal y financiero al proyecto, quién a través de una visión estratégica y presidida por el deseo de preservar las libertades de los usuarios para modificar el software se opta por empezar a recorrer un largo y duro camino en defensa de los derechos de cientos de programadores y usuarios finales. Stallman tuvo que empezar su proyecto en condiciones deplorables ya que no se podía acceder a la red Internet como hoy en día ni tampoco se disponía de un compilador libre para empezar el trabajo, es así que inició su sistema operativo con la construcción de herramientas básicas para la programación: un editor, un compilador y un depurador.

4.2.1 Antecedentes

En 1990 el proyecto GNU contaba ya con un editor de textos llamado Emacs, un compilador GCC pero le hacía falta un núcleo o Kernel para sus aplicaciones, en sus inicios trabajaron con el núcleo **TRIX**, un núcleo de llamadas remotas a procedimientos desarrollado por el MIT (Massachussets Institute of Technology) el cual fue distribuido libremente.

Muchos programadores trabajaron arduamente en la modificación de este núcleo pero concluyeron que no era utilizable ya que funcionaba solo en equipos caros y difíciles de manejar, razón por la cual decidieron utilizar el núcleo **Mach** desarrollado por la CMU (Carnegie Mellon University), su función principal era realizar labores de administración sobre el Hardware para que el sistema operativo sea operado desde el espacio del usuario. Más tarde en el año de 1991 el núcleo Mach fue reemplazado por **Hurd**, una colección de servidores que se ejecutan sobre Mach para implementar archivos de sistema, protocolos de red y control de acceso a archivos. Desde entonces sería el núcleo oficial de GNU, pero debido a razones técnicas y conflictos internos en la organización, el desarrollo de Hurd terminó estancado.

A raíz de estos acontecimientos, en el mismo año adoptaron al sistema operativo GNU un núcleo llamado Linux utilizado ampliamente en la actualidad. Linux es el núcleo o kernel del sistema operativo libre denominado GNU/Linux que hoy en día es una gran alternativa frente a sistemas operativos no libres como Unix y Windows. Una distribución GNU/Linux es un conjunto de aplicaciones reunidas que permiten configurar e instalar paquetes de software y constan de una gran variedad de aplicaciones como entornos gráficos, servidores Web, correo, FTP, etc.

A medida que el proyecto GNU ganaba nombre y posición en el mercado, aparecieron muchos interesados en contribuir con el mismo, tal es el caso de Cygnus Solutions¹⁴ que ahora forma parte de Red Hat con sus valiosos aportes en el desarrollo y comercialización de los productos además del soporte técnico respectivo.

¹⁴ Sitio Oficial: <http://www.cygnus-solutions.com>

4.2.2 Aplicaciones desarrolladas por el proyecto GNU

El proyecto GNU se ha caracterizado por desarrollar aplicaciones de toda índole a través de sus valiosas aportaciones en el desarrollo de paquetes como CVS para el control de versiones de código fuente y herramientas gráficas para la depuración de errores como DDD. Entre sus trabajos más importantes mencionamos:

- GCC.- Compilador de varios lenguajes, específicamente C.
- GNOME.- Entorno gráfico para GNU.
- GZIP.- Aplicaciones para comprensión de datos.
- BISON.- Generador *parser* diseñado para substituir a *yacc*.
- BASH.- Intérprete de comandos.
- BFD.- Archivos de bibliotecas.
- CLASSPATH.- Bibliotecas diseñadas para Java.
- EMACS.- Editor de texto extensible.
- GSL.- Biblioteca científica para GNU.
- LILYPOND.- Editor de partituras musicales
- TEXINFO.- Sistema de documentación.
- OCTAVE.- Software numérico similar a MATLAB.
- GNU MDK.- Herramientas de programación en MIX.

4.2.3 La Catedral y el Bazar

La Catedral y el Bazar es un ensayo escrito por el hacker Eric S. Raymond en el año de 1997 en el que destaca la importancia del Software Libre a través de un análisis del surgimiento de GNU/Linux gracias al proyecto GNU, el documento compara en 19 lecciones las diferencias teóricas entre dos modelos de desarrollo, por un lado la catedral que representa al Software No Libre y por otro el Bazar que representa al Software Libre.

1. Todo buen trabajo de software comienza a partir de las necesidades personales del programador. (Todo buen trabajo empieza cuando uno tiene que rascarse su propia comezón).
2. Los buenos programadores saben qué escribir. Los mejores, qué reescribir (y reutilizar).

3. "Considere desecharlo; de todos modos tendrá que hacerlo." (Fred Brooks, The Mythical Man-Month, Capítulo 11).
5. Si tienes la actitud adecuada, encontrarás problemas interesantes.
5. Cuando se pierde el interés en un programa, el último deber es heredarlo a un sucesor competente.
6. Tratar a los usuarios como colaboradores es la forma más apropiada de mejorar el código, y la más efectiva de depurarlo.
7. Libere rápido y a menudo, y escuche a sus clientes.
8. Dada una base suficiente de desarrolladores asistentes y beta-testers, casi cualquier problema puede ser caracterizado rápidamente, y su solución ser obvia al menos para alguien. O, dicho de manera menos formal, "con muchas miradas, todos los errores saltarán a la vista". A esto lo he bautizado como la Ley de Linus.
9. Las estructuras de datos inteligentes y el código burdo funcionan mucho mejor que en el caso inverso.
10. Si usted trata a sus analistas (beta-testers) como si fueran su recurso más valioso, ellos le responderán convirtiéndose en su recurso más valioso.
11. Lo mejor después de tener buenas ideas es reconocer las buenas ideas de sus usuarios. Esto último es a veces lo mejor.
12. Frecuentemente, las soluciones más innovadoras y espectaculares provienen de comprender que la concepción del problema era errónea.
13. "La perfección (en diseño) se alcanza no cuando ya no hay nada que agregar, sino cuando ya no hay nada que quitar."
14. Toda herramienta es útil empleándose de la forma prevista, pero una **gran** herramienta es la que se presta a ser utilizada de la manera menos esperada.
15. Cuando se escribe software para una puerta de enlace de cualquier tipo, hay que tomar la precaución de alterar el flujo de datos lo menos posible, y ¡**nunca** eliminar información a menos que los receptores obliguen a hacerlo!
16. Cuando su lenguaje está lejos de un Turing completo, entonces el azúcar sintáctico puede ser su amigo.

17. Un sistema de seguridad es tan seguro como secreto. Cuídese de los secretos a medias.

Para resolver un problema interesante, comience por encontrar un problema que le resulte interesante.

19. Si el coordinador de desarrollo tiene un medio al menos tan bueno como lo es Internet, y sabe dirigir sin coerción, muchas cabezas serán, inevitablemente, mejor que una.

Este ensayo destaca la importancia del Software Libre a través de un análisis de un exitoso proyecto denominado *fetchmail*, el mismo que fue realizado para probar ideas acerca de la ingeniería de software sugeridas por la historia del sistema operativo GNU con Linux, el contraste de las 19 lecciones explica el porqué del éxito de las aplicaciones libres respecto a las comerciales.

Cada lección se basa en experiencias que surgieron a lo largo del desarrollo de *fetchmail*, y es un consejo valioso a seguir cuando se desarrolla Software Libre sobretodo en aplicaciones y proyectos grandes.

4.3 OSI (Open Source Initiative)

La OSI es una entidad sin fines de lucro dedicada a manejar y promover la definición de código abierto. Cuando los usuarios finales de un software pueden leer, redistribuir y modificar el código fuente, éste con el tiempo evoluciona y se vuelve más solvente, este hecho se produce por la aportación constante de conocimientos de una comunidad entera que quiere mejorar y corregir errores existentes. Este proceso de evolución produce un software de calidad en comparación al modelo común de software privativo donde solo unos pocos programadores tienen acceso al código fuente, mientras que el resto debe conformarse con solo tener la aplicación.

El objetivo de la OSI es llevar este modelo de desarrollo al mundo comercial impulsando cientos de proyectos tecnológicos de comunidades del conocimiento como por ejemplo tecnologías de Internet, plataforma Linux y varias aplicaciones que han dado una nueva perspectiva al mercado tecnológico de la industria del software.

Cada vez son más los usuarios que se unen a esta iniciativa de código abierto no solo por la gran cantidad de herramientas disponibles en el medio sino por los beneficios que aportan a la sociedad en términos éticos, sociales, económicos, etc. Existe una gran cantidad de instituciones educativas, gubernamentales y empresariales que han decidido migrar sus aplicaciones a esta plataforma, aunque todavía existe mucha polémica acerca de la conveniencia de este tipo de aplicaciones e implementaciones.

En nuestro país se decidió el uso de código abierto como política de gobierno emitiéndose el decreto 1014 el 10 de abril de 2008¹⁵ firmado por el Presidente Constitucional de la República, Econ. Rafael Correa Delgado, dicho documento se encuentra en el Anexo 1, y establece como política pública para las Entidades de la Administración Pública Central la utilización de Software Libre en sus sistemas y equipamientos informáticos, creándose así un género tecnológico propulsor de riqueza nacional.

Varias empresas del mercado informático han realizado comparaciones entre sistemas de código abierto y cerrado en temas de estabilidad, rendimiento, seguridad, entre otras, y varias de ellas afirman que los sistemas de código abierto representan una excelente alternativa como infraestructura tecnológica, tal es el caso de una comparación realizada entre las bases de datos MySQL y Oracle, para cada una se conoció el nivel de aceptación de los usuarios en términos de costos, rendimiento, funcionalidad, seguridad, disponibilidad, entre otras, la misma que se muestra en la tabla 4.1. Además de ello muchas organizaciones importantes utilizan código abierto en sus aplicaciones, entre ellas Yahoo, Google y Sony.

		Sistemas de Código Abierto	Sistemas de Código Cerrado
Parámetro de Medida	Puntaje Máximo	Puntaje MySQL	Puntaje Oracle
Costo y Licenciamiento	20	20	5
Rendimiento	10	10	9
Funcionalidad	150	117	150
Seguridad	100	80	100
Soporte	10	10	10
Almacenamiento	50	50	30
Tipos de Datos	140	116	126

Tabla 4.1 Comparativa de Bases de Datos de sistemas de código abierto y cerrado

¹⁵ Documento ubicado en:
http://www.opensourceecuador.com/wp-content/decreto_1014_software_libre_ecuador.pdf

4.3.1 Beneficios

Los beneficios de este modelo son varios, empezando por los costos cero en la adquisición de sus licencias para las aplicaciones, además se puede disponer del código fuente, así el usuario está en total libertad para utilizar el programa con cualquier propósito de uso común, el mismo que puede distribuir las copias que desee, además de estudiarlo y modificarlo según sus necesidades y la flexibilidad del caso, siempre y cuando se rija bajo las cuatro libertades que estipula el Software Libre.

De igual manera el acceso a la información es abierta a cualquier persona interesada en involucrarse en el área del conocimiento. Cualquier problema existente será tratado y solucionado gracias a cientos de comunidades a nivel mundial que constantemente están haciendo mejoras a los productos manteniéndolos siempre en constante crecimiento. Para determinar que un software se rige bajo los preceptos de código abierto, la OSI utiliza un certificado llamado ***OSI Certified Open Source Software***, en el determina si el código fuente de un software es publicado abiertamente y si la licencia está disponible sin cargo. Algunos ejemplos de licencias open source figuran: MPL (Mozilla Public License), BSD (Barkeley Software Distribution), GNU GPL (General Public License), GNU LGPL (Lesser General Public License) y MIT (Massachussets Institute of Technology), cualquier producto de software distribuido bajo una de estas licencias se considerará como software de código abierto.

5.3.2 OSI Certified Open Source Software

OSI Certified es un certificado que emite la entidad para identificar a un software que se distribuye bajo una licencia que se ajusta a la definición de código abierto, teniendo en cuenta que dicha licencia consta en la lista de licencias aprobadas por la OSI. Si la marca de OSI Certified se distribuye bajo una licencia que no esté aprobada por la OSI será considerado como una infracción de marcas de certificación de la OSI y es contra la ley.



Para distribuir un producto de software como OSI Certified, se debe adjuntar las siguientes líneas sin modificaciones:

*“This software is OSI Certified Open Source Software. OSI Certified is a certification mark of the Open Source Initiative”*¹⁶. Si la distribución se la hace de forma electrónica, se debe colocar en un archivo de texto “readme” la marca de OSI Certified, por el contrario si se hace de forma tangible, debe colocarse la marca de OSI Certified en las hojas impresas o medios ópticos como Disquetes, CD-ROM, etc.

Si un usuario tiene a disposición una licencia y quiere aprobarla bajo los términos de la OSI, tiene que contar con los siguientes requisitos, la licencia debe tener un nombre único frente a todas las demás, debe difundirse en dos formatos: Html como página web y en texto plano, el usuario debe hacer un análisis legal de cómo la licencia cumplirá con los términos de la Open Source Definition y se debe enviar un correo a la entidad indicando que licencia existente y aprobada por la OSI se asemeja más a la del usuario explicando porque ésta no cumple sus necesidades, además se debe explicar cómo el software distribuido bajo la licencia del usuario puede ser usado con software distribuido bajo otras licencias de código abierto y por último adjuntar la versión en texto plano de la licencia. En el Anexo 2 se encuentra un ejemplo de cómo enviar un correo a opensource.org para que se dé trámite a la aprobación de una licencia nueva.

4.3.3 Criterios de Definición de Código Abierto

Para que un producto de software se distribuya en términos de código abierto debe cumplir con los siguientes criterios:

1. Libre Distribución

Siempre y cuando la licencia no restrinja a un tercero a vender o entregar el software como componente de una distribución de software que tenga programas de fuentes distintas.

¹⁶ Tomado de: Open Source Initiative, “OSI Certification Mark and Program”, http://opensource.linux-mirror.org/docs/certification_mark.php,
Fecha de Consulta: 25 de Septiembre del 2009.

2. Código Fuente

El programa que se vaya a distribuir debe incluir un código fuente y el compilado. Si un producto no se distribuye con el código fuente, debe explicar de forma clara cómo obtenerlo sin ningún costo desde internet, además este debe estar en un formato en el que cualquier programador pueda modificarlo.

3. Trabajos Derivados

La licencia debe considerar las modificaciones y trabajos derivados del caso y debe permitir que sean distribuidos bajo los mismos términos de la licencia del programa original.

4. Integridad de la fuente del autor del código

La licencia puede restringir el código fuente desde que ha sido distribuido solo si la licencia permite la distribución de parches en el código fuente para propósitos de modificación del programa en tiempo de compilación.

5. No debe existir discriminación de personas o grupos

La licencia como tal, no debe discriminar por ningún motivo a personas o grupos de personas por sus ideologías o formas de pensar.

6. No debe existir restricciones de aplicación en campos específicos

La licencia no debe restringir el uso del programa en un campo específico como negocios, educación, finanzas, etc.

7. Distribución de la Licencia

Los derechos de distribución que se agregan al programa debe aplicarse para quienes se distribuye el programa, sin la necesidad de contar con una licencia adicional para ellos.

8. La licencia no debe ser específica a un producto

Si un programa es extraído de una distribución y usado o distribuido dentro de los términos de la licencia del programa, todas las partes redistribuidas deben tener los mismos derechos que la distribución del programa original.

9. La licencia no debe restringir otro software

Es decir, la licencia no debe imponer condiciones sobre otro software para que se distribuya en el mismo medio como software de código abierto.

10. La licencia debe ser tecnológicamente neutral

Lo que significa que ninguno de los términos de la licencia puede basarse en tecnologías individuales.

4.3.4 Estándar de Software Libre en Ecuador

El 27 de marzo del año 2009 fue una fecha significativa para la ASLE (Asociación de Software Libre del Ecuador) ya que se aprobó la norma técnica ecuatoriana NTE ISO/IEC 26300:2009¹⁷ según acuerdo ministerial 031-2009 en el que se describe un esquema XML para aplicaciones ofimáticas y tecnologías de información. La norma fue analizada por personal de varias instituciones gubernamentales como el SRI (Servicio de Rentas Internas) y la subsecretaria de informática de la Presidencia de la República y fue impulsada por la visita al país del padre del software libre Richard Stallman quien defendió la libertad de los usuarios al acceso del código fuente de las aplicaciones.

Entre los capítulos del documento se destaca la especificación del formato OpenDocument con sus contenidos gráficos, diagramáticos y el trabajo con formularios, además del tipo de datos usados por el esquema OpenDocument.

¹⁷ Tomado de: INEN Instituto Ecuatoriano de Normalización, “NORMA TECNICA ECUATORIANA NTE ISO/IEC 26300:2009”, http://www.inen.gov.ec/normas/norma.php?COD_NORMA=2844, Fecha de Consulta 15 de Septiembre del 2009.

4.3.4.1 Beneficios del Estándar de Software Libre en Ecuador

La aprobación de la norma técnica ecuatoriana NTE ISO/IEC 26300:2009 sin lugar a dudas impulsa el uso y desarrollo de aplicaciones de Software Libre en nuestro país, no solo en empresas gubernamentales que utilizan documentos públicos sino que también anima a empresas privadas a usar aplicaciones ofimáticas y tecnologías de información de código abierto y así formar una sociedad tecnológica libre sin ataduras a formatos de documentos privativos que muchas de las veces conllevan a dependencias de un único fabricante.

El estándar OpenDocument permite a las distintas empresas de nuestro país almacenar documentos como hojas de cálculo, memorandos, presentaciones y gráficas utilizando el esquema de OpenOffice.org sin que esto implique un costo por licencia para su uso, lo cual evita las relaciones comunes con proveedores únicos de software y por ende ahorros económicos significativos para la empresa.

La adopción de este estándar abierto, permite que muchos proveedores desarrollen aplicaciones sin que esto implique un costo de licencias por las mismas, mejorando continuamente la calidad tecnológica de sus productos e incluso se puede escoger cualquier sistema operativo para leer y editar los documentos.

CAPITULO V

METODOLOGÍAS DE DESARROLLO DE SOFTWARE

Introducción

La metodología dentro de la ingeniería de software cumple con la función de elaborar estrategias de desarrollo de software que disminuyan las prácticas predictivas que frecuentemente requieren demasiado esfuerzo para poder culminar de manera correcta el software además de desfases de costos, tiempos, y sobre todo gran desgaste del equipo de desarrollo. Para poder convertir el desarrollo de software en un proceso formal, pudiendo anticipar sus resultados, es necesario conseguir un fortalecimiento de prácticas adaptativas, las que están centradas en las personas y equipos de desarrollo, además de ser orientadas hacia la funcionalidad y entrega permitiendo obtener un software de alta calidad, que cumpla las necesidades y expectativas del usuario o cliente.

El desarrollo de software además de su dificultad intrínseca, tiene un gran impacto en los negocios los cuales requieren sistemas eficientes que cumplan satisfactoriamente sus necesidades, esto convierte en evidentes las ventajas que se pueden obtener de aplicar una metodología de desarrollo formal en proyectos de software sobre todo respecto a la calidad, tiempos y costos.

Hoy en día existen numerosos estándares y modelos en los cuales podemos basar un proyecto de software independientemente de los elegidos como referencia, la implantación de una metodología de desarrollo de software plantea diversos retos cuya resolución está más cerca de lo humano que de lo técnico. Por ejemplo el Software Libre tiene métodos de desarrollo que los desarrolladores de software no libre no los consideraban válidos y pensaban que llevarían las aplicaciones a un caos total, pero dieron grandes resultados, generando aplicaciones muy buenas tales como el sistema operativo GNU con Linux.

5.1 Antecedentes

En sus inicios el software era bastante sencillo, cumpliendo solo funciones específicas y reducidas, por lo que desarrollarlo no requería de un gran equipo, siendo suficiente muchas veces solo un programador, que imponía su ritmo de trabajo y en un plazo razonable entregaba programas útiles aunque bastante sencillos, para este software el programador era como un mago debido a que solo él conocía como se desarrollo el software, y en muchos caso solo él sabía cómo funcionaba. Tampoco existían los estándares actuales para poder realizar la documentación del software como el UML, a lo mucho lo que se elaboraba era un flujograma, el cual al momento de intentar entender el funcionamiento del software no era de gran ayuda.

Con el tiempo las aplicaciones sencillas no satisfacían las necesidades de los usuarios que eran cada vez más exigentes, además de que la evolución de la tecnología era notable y los ordenadores disponían de muchos más recursos de hardware, lo que les daba más capacidad de procesamiento y con ello el surgimiento de interfaces gráficas más vistosas como las ventanas, además de impresoras más potentes que permitían imprimir gráficos, diferentes fuentes y nuevos dispositivos como el ratón, lectores de código de barras, etc. Por lo que el software dejo de ser una tarea sencilla, el adaptarse a estos nuevos dispositivos y entornos requería de mayor esfuerzo, el software fue cada vez más grande y mucho más complicado con lo que surgió la necesidad de varios desarrolladores y tiempos de desarrollo más largos.

Todo esto trajo consigo la necesidad de organización, surgiendo las metodologías de desarrollo, las cuales requerían mayor esfuerzo, pues se basaban en escribir mucha documentación antes de iniciar el desarrollo del software, es decir primero era necesario escribir todo lo que el software tenía que hacer para poder llegar a un acuerdo entre los usuarios y desarrolladores. Luego había que escribir qué se iba a hacer, de esta forma el equipo organizado trabajaba por un objetivo común, equilibrando la carga de trabajo y pudiendo calcular de mejor manera los tiempos y costos del desarrollo, así como aumentando la calidad del software.

Pero este aspecto de organización y papeleo diezmó el encanto de los programadores hacia el desarrollo de software, debido a que en lugar de dedicarse a los que les gustaba, es decir programar según sus gustos y costumbres, debían seguir diseños de

otras personas del equipos y dedicar gran parte de su tiempo a leer y a hacer documentos, organizar reuniones, planificar trabajo, etc.

Entonces para tratar de recuperar el encanto de la programación y disminuir el tedio del papeleo para los programadores surgieron nuevas metodologías de desarrollo en las que el papeleo se reducía considerablemente, las cosas eran más sencillas y nuevamente los programadores se dedicaban a lo que les gusta, programar.

Estas además dan más valor a las personas y la comunicación entre ellas un ejemplo de ello son las denominadas metodologías ágiles en las que se busca una organización menos formal y jerárquica, sobretodo eliminar el exceso de papeleo, se basa en desarrollo iterativo y los usuarios finales tienen mucha importancia, pues ellos definen las funcionalidades con las que se debe desarrollar el software y la comunicación es parte fundamental de esta metodología.

Cabe destacar que todas las metodologías tienen puntos a favor como en contra, por lo que es muy importante conocerlas y ver cuál es la que brinda mejor soporte al desarrollo de la aplicación requerida.

5.2 Metodología Extreme Programming (Programación Extrema)

La metodología de programación extrema, eXtreme Programming (XP) es un enfoque de la ingeniería de software formulado por Kent Beck, la misma que ha tenido gran aceptación y se ha desarrollado mucho en los últimos años, debido principalmente a que busca una organización menos formal y jerárquica en el desarrollo de software planteando como objetivo central del desarrollo producir un sistema con la cantidad correcta de funcionalidades con las que se cubrirán el número mínimo de características y necesidades para satisfacer por completo al usuario real.

Esta metodología busca eliminar actividades relacionadas con la elaboración de algunos documentos de especificaciones que no tienen relación directa con el resultado final del software. Se basa en los siguientes valores principales:

1. *Comunicación:* La necesidad de los desarrolladores de intercambiar ideas e información sobre el proyecto ya sea con los directores del proyecto o los usuarios de forma confiable, y fácil. La comunicación debe ser continua y rápida.

2. *Sencillez*: Cuando se tenga que elegir entre varias opciones, en lo posible elegir soluciones simples, sin que esto signifique aplicar enfoques simplistas; la programación extrema define un diseño simple en el que se realice el menor número posible de clases, métodos y que no tenga código duplicado.
3. *Retroalimentación*: Debe ser rápida en todos los niveles, principalmente se consigue ejecutando y probando el código, por lo que las entregas tempranas y frecuentes son muy importantes.
4. *Valor*: Todas las personas que participen en el proyecto deben tener la capacidad de expresar su valoración sobre el proyecto. Deberían ser abiertos y dejar que todos revisen e incluso modificasen su trabajo. Los cambios no deberían ser vistos con terror y los desarrolladores deberían tener el valor de encontrar mejores soluciones y modificar el código siempre que sea necesario y factible.
5. *Respeto*: Debe manifestarse en diversas formas y situaciones, son la base para una buena relación y cooperación entre todos los componentes del equipo de trabajo.

La Programación Extrema se basa en:

- *Desarrollo en iteraciones*: En cada iteración se agregan nuevas funcionalidades, o se corrigen errores generando distintas versiones.
- *Pruebas unitarias continuas*: Estas pruebas están orientadas a comprobar que la aplicación mantenga sus funcionalidades.
- *Programación en parejas*: Se hace esto con el fin de que el código se discuta y revise mientras se desarrolla el programa, basado en que los dos programadores pueden complementarse, generando código de mejor calidad con menos errores.
- *Interacción entre los desarrolladores y el usuario*: Mientras mejor sea esta, se minimizará el esfuerzo de ambas partes, pues se podrá tener una mejor comprensión de los problemas o necesidades de los usuarios y las soluciones que puedan brindar los desarrolladores.

- *Corrección de los errores antes de cada iteración:* Es importante que los resultados de cada iteración sean versiones estables. Es decir que hayan superado exitosamente la fase de depuración del programa.
- *Refactorización del código:* Busca hacer el código más legible y mantenible, pero debe garantizar su correcto funcionamiento manteniendo las pruebas unitarias.
- *Propiedad del código compartida:* Busca que cualquier integrante del proyecto pueda colaborar modificando código hecho por otro. La existencia de errores se comprueba mediante las pruebas.
- *Simplicidad del diseño:* Los diseños simples pero funcionales permiten que posteriores funcionalidades se puedan agregar de manera fácil y rápida.

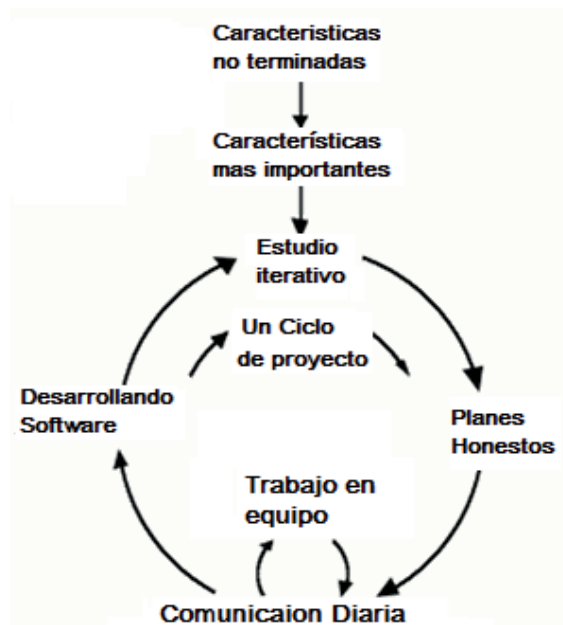


Figura 5.2.1 Flujo de características de la Programación Extrema

5.2.1 Fases de desarrollo de la Programación Extrema

5.2.1.1 Fase de Exploración

Los clientes mediante las historias de usuario (breves descripciones del trabajo de un sistema) plantean a grandes rasgos sus necesidades, las que permiten conceptualizar el software a desarrollar y la primera entrega del programa. En esta fase también los desarrolladores deben familiarizarse con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Estas son probadas construyendo un prototipo del sistema. Toma de pocas semanas a pocos meses, dependiendo del tamaño del proyecto y la destreza de los programadores en el uso de las herramientas de desarrollo.

5.2.1.2 Fase de Planificación de la Entrega

Se asignan prioridades a las historias de usuario, luego los programadores estiman el esfuerzo necesario para desarrollarlas, se llega a un acuerdo sobre el contenido de la primera entrega del sistema, y se determina un cronograma con el usuario, por lo general una entrega no puede tardar más de 3 meses.

La planificación de las entregas se hace mediante la utilización de puntos, donde 1 punto equivale a una semana ideal de trabajo. Una historia generalmente se la puede desarrollar entre 1 y 3 puntos. La utilización de puntos permite llevar un historial de la velocidad de desarrollo, basándose en puntos necesarios para cumplir una iteración, que suele ser la suma de los puntos correspondientes a las historias de usuario que se terminaron en la última iteración. Mediante esta medida de velocidad se puede estimar cuantas historias de usuario se pueden terminar antes de una fecha, o cuánto tiempo se necesita para completar un conjunto de historias de usuario.

La planificación se puede realizar basándose en el tiempo o el alcance. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, obteniendo cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

5.2.1.3 Fase de Iteraciones

En esta fase se realizan varias iteraciones antes de que el software sea entregado al usuario, estas iteraciones no suelen durar más de 3 semanas. En la primera iteración se busca establecer una arquitectura para el sistema para poder utilizarla en el resto del proyecto, por lo que se trata de implementar las historias de usuario que favorezcan a la arquitectura del sistema en la primera iteración, pero el cliente es el que decide que historias se implementaran en cada iteración. Al final de la última iteración el sistema debe estar listo para entrar en producción. Para elaborar el plan de iteración se debe tomar en cuenta los aspectos siguientes:

- Historias de usuario no tratadas.
- Velocidad del proyecto.
- Pruebas de aceptación no superadas de la iteración anterior.
- Tareas no terminadas en la iteración anterior.

Todo el trabajo de las iteraciones es expresado en tareas de programación, las cuales tienen un programador responsable, pero se desarrollan por parejas de programadores.

5.2.1.4 Fase de Producción

En esta fase se realizan pruebas adicionales y se revisa el rendimiento del software antes de ser llevado al cliente. Si en esta fase surgen cambios, se debe decidir si incluir esos cambios en la versión actual, de ser así estos cambios son relativamente cortos, por lo general de iteraciones que duran una semana, o si no se genera documentación para realizarlas en la etapa de mantenimiento.

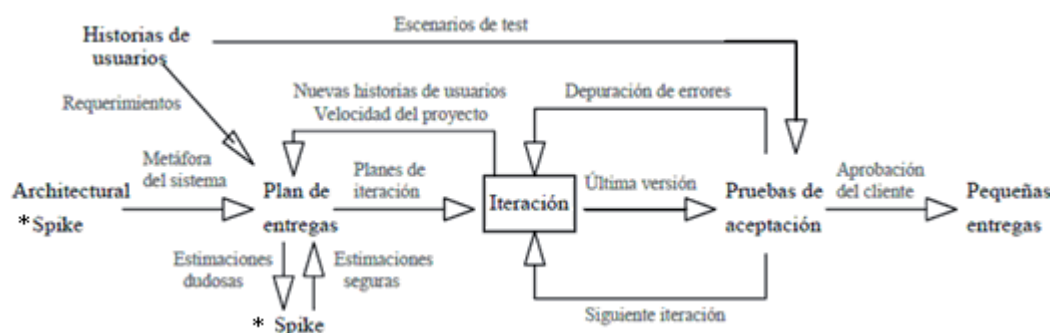
5.2.1.5 Fase de Mantenimiento

Se la ejecuta mientras la versión anterior sigue en producción, por lo que se deben dar tareas de soporte al cliente, se desarrollan nuevas iteraciones, aunque estas pueden ser más lentas, también puede cambiar el equipo de desarrollo, o su estructura.

5.2.1.6 Muerte del proyecto

Se da cuando el usuario no tiene nuevas historias que implementar en el proyecto, y están satisfechas todas las necesidades del cliente, incluidas el desempeño y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan cambios en la arquitectura del software.

Esta fase también se da si el proyecto no genera los beneficios que esperaba el cliente, o si el cliente decide no mantenerlo, o si no se cuenta con los fondos suficientes para continuar el proyecto.



* Spike = Pequeño programa que explora posibles soluciones potenciales

Figura 5.2.2 Fases de desarrollo con metodología de Programación Extrema¹⁸

5.2.2 Características de la Metodología

Esta metodología busca una separación entre el diseño y la construcción debido a que son actividades diferentes, el diseño es muy difícil de predecir por lo que requiere personal talentoso y creativo, esto significa más dinero en proyectos de software comercial, mientras que la construcción es más fácil de predecir, entonces se puede empezar a construir a partir de un buen diseño con personas que no tienen gran talento en el diseño pero si en la programación.

Cabe destacar que los procesos no se pueden planear fácilmente, por lo que es difícil que por no decir imposible que la planificación pueda proveer todos los aspectos de

¹⁸ Tomado de: Ingeniería del Software II, “Introducción a Extreme Programming”, Gerardo Fernández Escribano, <http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf>, Fecha de consulta: 15 de noviembre de 2009.

la elaboración del software. La planificación debería ser un proceso de mucho tiempo, un equipo amplio y requisitos estables para pretender esto.

Casi todo en el desarrollo de software depende de los requisitos, si no se pueden obtener requisitos estables no se puede obtener un plan predecible y por lo tanto sería muy difícil implementar una metodología predictiva y poder saber cuáles serían los resultados que se obtendrían. Pero eso no significa que un problema imprevisible no pueda ser tratado con una metodología, eso es un punto fuerte de esta metodología debido a que trata de controlar las situaciones imprevisibles mediante iteraciones.

5.2.2.1 Como controlar las situaciones imprevistas

Lo más importante y difícil es saber con precisión dónde está el programa, debido a que se necesita un mecanismo honesto de retroalimentación que informe con precisión cuál es la situación a intervalos frecuentes. Por ello es muy importante el desarrollo iterativo, la clave de este desarrollo es producir frecuentemente versiones del programa que funcionen, las cuales deben tener subconjuntos de las funcionalidades requeridos, por lo general son cortos en funcionalidad, pero deben satisfacer las demandas del sistema final, los que deben ser totalmente integra y probados muy cuidadosamente como una entrega final. Estas iteraciones deben darse en periodos tan cortos como sea posible, por lo general entre 1 y 3 semanas, y manejarlas adecuadamente con el fin de generar retroalimentación más rápida.

Una ventaja del desarrollo iterativo es que las situaciones cortas son más fáciles de controlar y planificar para una sola iteración, sobre cada iteración puede darse la base para planes posteriores. Mediante esta metodología se puede tratar un problema imprevisible, como varios previsible.

5.2.2.2 Función de los usuarios

En estos procesos adaptables, el cliente tiene control sobre el proceso de desarrollo de software, debido a que por cada iteración puede verificar el proceso y también cambiar la dirección del desarrollo del programa, por lo que es necesario que los usuarios tengan una relación más cercana con los desarrolladores, formando una sociedad de trabajo la que es esencial para que funcione esta metodología.

El cliente obtiene como resultado un software más usable y sensible a sus necesidades, como consecuencia de esta metodología hay que volver a rehacer el plan de desarrollo para cada iteración, pues es imposible establecer de manera exacta el tiempo, precio y alcance del programa en esta metodología, pero la ventaja es que se pueden controlar de mejor manera los riesgos que se presentan en el desarrollo del software e incluso se ven las variaciones como oportunidades.

Si el usuario consigue un programa más valioso que el costo que invirtió en él, se le considera un proyecto exitoso. Un buen proyecto con esta metodología construirá algo diferente y mejor que lo que se esperaba en el plan original.

No es fácil ejecutar un proceso adaptable, requiere un equipo eficaz de desarrolladores tanto individualmente como en equipo, además se debe tomar en cuenta a las personas como el factor de primer orden en el éxito del proyecto, debido a que esto crea un fuerte efecto de retroalimentación positiva, incluso si las personas trabajan motivadas y a gusto, lo hacen de mejor manera.

5.2.2.3 Orientación a las personas

Si las personas involucradas en el proyecto son profesionales hay que reconocer su competencia, por lo tanto saben cómo dirigir su trabajo técnico, es decir no es necesario o hasta resulta negativo tener un departamento de planificación separado que imponga como se debe desarrollar el proyecto, sobretodo porque esto puede repercutir en la motivación para el trabajo de los desarrolladores.

Uno de los elementos clave es la aceptación de un proceso en lugar de la imposición de un proceso, los procesos impuestos son muy resistidos, sobre todo si los que los imponen no participaron activamente del mismo. Aceptar estos procesos requiere de un gran compromiso de todo el equipo, los desarrolladores deben poder tomar las decisiones técnicas que crean correspondientes, por lo tanto dentro de la planeación solo ellos pueden estimar el tiempo que les tomara hacer su trabajo. Esto hace que tome importancia la confianza de los líderes del proyecto en los desarrolladores.

Los técnicos no pueden por si solos realizar todo el proceso de desarrollo, necesitan una guía a las necesidades que buscan satisfacer, por ejemplo si es un software comercial el que se va a desarrollar necesitan una buena comunicación con expertos en el negocio, esta comunicación no puede ser ocasional, sino continúa.

Si se desea obtener un proyecto excelente, se requiere de personas capaces en todas las áreas, un buen conocimiento del negocio permite que el programa cumpla las funciones requeridas.

5.2.2.4 Medir los avances del proyecto

Es muy complicado determinar la eficacia de los desarrolladores, sobre todo por la dificultad de aplicar medidas al software, las metodologías tradicionales imponían unidades de medida a cada factor, pero esto por las características de esta metodología puede llevar al trastorno de la medida a valores muy altos, por lo que se utiliza la gestión delegatoria que permite a los desarrolladores decidir cómo hacen su trabajo.

5.2.2.5 Pruebas del proyecto con programación extrema

Para los proyectos desarrollados bajo esta metodología existen frameworks de pruebas de unidades, están disponibles en varios lenguajes de programación, por ejemplo en java se utiliza JUNIT, en C y C++ se utiliza CPPUnit, en .NET se utiliza NUnit. Esta herramienta no es útil solo en la etapa de pruebas, se la usa desde el mismo desarrollo y ayuda a formalizar los requisitos, aclarar la arquitectura y el código del programa, la depuración de código, integración del código, y sobre todo la etapa de pruebas. Estas unidades de prueba se pueden personalizar según las necesidades específicas del proyecto.

Estos frameworks de pruebas permiten realizar ejecuciones del sistema de manera controlada, para poder evaluar el funcionamiento de cada uno de los métodos y funciones del programa, es decir en según los valores de entrada se evalúan los valores de retorno esperados, si este cumple con la especificación la unidad de prueba devolverá que paso la prueba, caso contrario no.

En la actualidad las herramientas de desarrollo de software como Eclipse y netbeans cuentan con plug-ins que permiten la generación de plantillas necesarias para la

creación de pruebas de una clase Java, las cuales se realizan de manera automática, facilitando al programador enfocarse en la prueba y el resultado esperado.

5.2.2.6 Cuando es recomendable esta metodología

Esta metodología tiene la ventaja de ser ligera lo que simplifica los procesos y es más probable darles un mejor seguimiento. Cabe recordar que es muy importante la confianza en el equipo de desarrollo, pues si se los considera como de baja calidad, este enfoque no es válido. Esta metodología es óptima para cuando se dan los siguientes factores:

- Requerimientos inciertos o cambiantes.
- Equipo de desarrollo motivado y responsable.
- Clientes que entienden, y participan activamente.

5.2.3 Ventajas y Desventajas de la metodología Extreme Programming

Ventajas	Desventajas
Se adapta muy bien a los requisitos cambiantes, debido a que divide el desarrollo en iteraciones que simplifican la elaboración del proyecto.	Resulta muy complicado planear el proyecto y establecer el costo y duración del mismo.
Está orientada al trabajo en parejas de modo que los programadores se complementen entre si, por lo que su trabajo suele ser más eficiente y con menos errores.	No se puede aplicar a proyectos de gran escala, que requerirán mucho personal, a menos que se las subdivide en proyectos más pequeños.
Hace que el usuario le dé más valor al proyecto al hacerle sentir parte activa del desarrollo del mismo. Evitando así los requisitos mal comprendidos.	Es más complicado medir los avances del proyecto, pues es muy complicado el uso de una medida estándar.

Ventajas	Desventajas
Permite superar con mayor facilidad circunstancias imprevistas en el desarrollo de software.	
Permite tener un proceso continuo con integraciones continuas, al tener entregas rápidas y frecuentes, garantiza que el software funcione como el usuario lo requiere.	
Permite una mejor y más eficiente comunicación entre todos los integrantes del proyecto.	

Tabla 5.2.1. Ventajas y Desventajas de la metodología Extreme Programming

5.3 Metodología Rational Unified Process (Proceso Unificado Racional)

5.3.1 Antecedentes

En el año de 1967 nació una metodología denominada *Ericsson Approach* diseñada por Ivar Jacobson como guía para el desarrollo de software basado en componentes quien a su vez introdujo el concepto de los Casos de Uso, diez años más tarde Jacobson fundó la compañía Objectory AB y lanza el proceso de desarrollo *Objectory*. Posteriormente en el año de 1995 la corporación Rational Software adquiere *Objectory* y lo fusiona con *Rational Approach* dando como resultado el desarrollo del *Rational Objectory Process (ROP)* adoptando el UML (Unified Modeling Language) como lenguaje de modelado.

Más adelante la corporación Rational Software implementó varios elementos como modelados de negocio y soluciones empresariales, todo con el fin de expandir ROP. Con estos antecedentes nace en el año de 1998 el Rational Unified Process (RUP) como una metodología de desarrollo de software compuesta por un conjunto de procesos prácticos que brindan una guía para el desarrollo de actividades en torno



al equipo de desarrollo, permitiendo seleccionar un conjunto de componentes de proceso que se ajustan fácilmente a las necesidades del proyecto, así este equipo de desarrollo de software se unifica optimizando la comunicación de cada miembro y por ende la asignación de recursos se hace más eficiente.

RUP junto a UML constituyen actualmente la metodología estándar más utilizada para el análisis, diseño, implementación y documentación de sistemas orientados a objetos ya que se utilizan en diferentes tipos de software, áreas de aplicación, niveles de competencia, tamaños de proyecto y se adaptan fácilmente a las necesidades de cada organización.

5.3.2 Estructura del RUP

RUP es una metodología en la que se describe quién, cómo, qué, en que tiempo y que actividades se van a desarrollar en el proyecto, el quién representa los distintos roles que puede desempeñar un individuo en la organización, el cómo se refiere a la unidad de trabajo que se asigna a un trabajador y el qué es la pieza de información utilizada por un proceso. Para el desarrollo satisfactorio de este proceso, es necesario cumplir con ciertas etapas, las mismas que en conjunto determinarán el ciclo de vida y el éxito de la aplicación. RUP hace uso de cuatro etapas en su metodología descritas de la siguiente manera: una fase de concepción, una fase de elaboración, una fase de construcción y una última de transición. Dentro de cada una de ellas se realizan varias iteraciones dependiendo de la dimensión del proyecto.

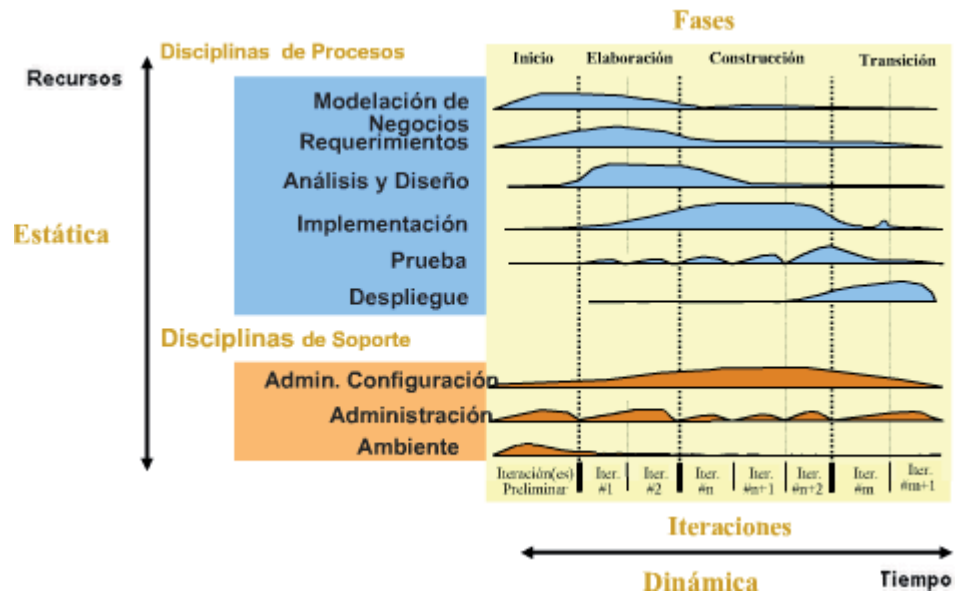


Figura 5.3.1 Estructura del RUP

La figura 5.3.1 describe la estructura del RUP en dos dimensiones, el eje horizontal representa el tiempo y proyecta una visión dinámica del proceso ya que muestra aspectos como las fases del ciclo de vida de un software RUP, metas e iteraciones; mientras que el eje vertical representa los diferentes recursos proyectados desde una vista estática, en estos se detallan las diferentes actividades, el personal encargado de las mismas, los roles, artefactos, disciplinas y los flujos de trabajo.

5.3.2.1 Visión estática de la metodología RUP

La visión estática de un proceso RUP permite describir los siguientes elementos:

1. Roles.- Entre los más comunes se encuentran los roles de gestión, de administración, de analista de sistemas y diseñador de pruebas.
2. Actividades.- Representa al trabajo que debe desempeñar cada rol.
3. Artefactos.- Son elementos tangibles de entrada y salida de las actividades, es decir, cosas que el proyecto produce o usa para componer un producto final. Los artefactos en la fase de Inicio están representados por una especificación de requerimientos, en la fase de Elaboración por los diagramas de casos de uso y en la fase de Construcción por los diagramas de clases, de secuencia, de estados y Modelo E-R.

4. Flujos de Trabajo.- constituyen la secuencia de actividades que producen resultados visibles.
5. Disciplinas.- Sirven para organizar las actividades del proceso, comprende 6 disciplinas de proceso y 3 de soporte.
 - En las de proceso tenemos: modelación de negocios, requerimientos, análisis y diseño, implementación, pruebas y desarrollo.
 - En las de soporte tenemos: gestión de configuración y cambio, gestión de proyecto y entorno.

5.3.2.2 Visión Dinámica de la metodología RUP

La visión dinámica de un proceso RUP permite apreciar su ciclo de vida mediante una descripción de procesos que son iterativos e incrementales, siempre respaldados por hitos que determinan su avance y permiten planear su continuidad en base a posibles cambios en su estructura.

Las cuatro fases que describen un proceso RUP son las siguientes:

1. Inicio.- En esta fase se obtiene una visión inicial del producto así como su alcance, además se identifican los principales casos de uso.
2. Elaboración.- En esta fase se hace una planificación de las actividades y del equipo de trabajo del proyecto, también se identifican las necesidades y el diseño de la arquitectura.
3. Construcción.- Comprende el desarrollo mismo del producto hasta la entrega al usuario final.
4. Transición.- Esta fase comprende la instalación del producto a los usuarios y la formación de los mismos, en ocasiones suelen surgir nuevos requisitos para el desarrollo.

5.3.3 Ciclo de Vida de la metodología RUP

5.3.3.1 Fase de Inicio

Es la primera fase dentro del ciclo de vida de un proceso RUP y en ella se tiene una primera apreciación de lo que será el producto final. Entre los aspectos que se identifican figuran el modelo de negocio, las principales funciones del sistema, la planificación del proyecto, los costes del proyecto, los actores y casos de uso críticos del sistema, la recuperación de la inversión y la priorización de los riesgos más importantes; todo con el fin de poner el proyecto en marcha siempre y cuando un análisis de negocio lo justifique, este análisis consiste en entender la estructura y funcionamiento de la organización para la cual el sistema va a ser desarrollado, la problemática existente con el fin de identificar mejoras y mantener un acuerdo con clientes sobre lo que el sistema podrá hacer.

Como resultado de esta fase se obtendrá un documento en el que se aprecian los requerimientos, las características y restricciones del proyecto, un glosario de terminología clave, una lista de riesgos y un plan de contingencia y un plan del proyecto; todos ellos logrados a partir de un entendimiento total de los requisitos y una comprensión de la arquitectura de desarrollo así como los gastos que estos implican.

5.3.3.2 Fase de Elaboración

En esta fase se construye un prototipo de la arquitectura que evoluciona en iteraciones sucesivas hasta convertirse en el sistema final, este prototipo contiene en detalle los casos de uso identificados en la fase anterior en base al dominio del problema, se desarrolla el plan del proyecto para la fase de construcción y se eliminan los riesgos potenciales.

También se planifican las distintas actividades del proyecto así como el personal para llevarlas a cabo considerando la comprensión global del sistema, entre ellos requisitos funcionales y no funcionales del proyecto. Para asegurar el éxito de esta fase, la visión de la arquitectura y del producto final deben ser estables, se deben ejecutar pruebas que demuestren que los riesgos han sido mitigados en un porcentaje bastante alto y se debe asegurar en lo posible que todos los usuarios involucrados hasta el momento estén de acuerdo con todas las propuestas del plan.

5.3.3.3 Fase de Construcción

Esta fase comprende el desarrollo del producto hasta la realización del mismo, se implementan las clases, los objetos, binarios, ejecutables y debe decidirse si puede ponerse en ejecución sin mayores riesgos, para ello debe existir una correcta planificación de qué subsistemas deben ser implementados y cómo estos van a ser integrados de tal forma que en conjunto cumplan los requerimientos de los usuarios finales. El equipo debe estar coordinado para el desarrollo de los módulos según los roles que estos desempeñen y de la experiencia adquirida; en esta fase suelen aparecer errores de diseño ya sea porque no se los había tenido en cuenta o se planificó alguna actividad de manera errónea.

La capacidad operacional del producto se obtiene de forma incremental a través de sucesivas iteraciones, en las que cada una comprende un ciclo de desarrollo completo dando como resultado una entrega de un producto ejecutable, siempre y cuando los componentes, las características y demás requisitos han sido integrados y probados en su totalidad; este ciclo de desarrollo está basado en las seis mejores prácticas con lo cual se logra reducir los riesgos del proyecto y así tener un subsistema ejecutable en el menor tiempo descritos en una o más versiones funcionales, estas mejores prácticas se detallan en el siguiente apartado. Es recomendable realizar un manual de operaciones para el usuario en el que se detalle a fondo la manera de interactuar con el sistema y así este se vea respaldado por una documentación básica de soporte.

5.3.3.4 Fase de Transición

Esta fase consiste en entregar el producto funcional al cliente, por lo que se requiere desarrollar nuevas versiones del producto, completar los manuales de usuario y entrenarlos en el manejo del sistema, para ello se realizan ajustes, configuraciones y la respectiva instalación. Se debe proveer asistencia y ayuda a los usuarios y si es que es el caso, realizar las respectivas migraciones de datos con el fin de conseguir la interoperabilidad de los sistemas. Como resultado de esta fase se obtendrá un prototipo operacional, documentos legales, una línea de base del producto que incluya todos los modelos del sistema y una descripción de la arquitectura utilizada. El éxito de esta fase se ve reflejado en el porcentaje en el que se han alcanzado los objetivos fijados en la fase de inicio y sobre todo si se ha cumplido con las expectativas del usuario final.

5.3.4 Las seis mejores prácticas

Las mejores prácticas fueron creadas para lograr la eficiencia en la producción de software, asegurando así un producto de alta calidad que cumpla las necesidades de los usuarios, estas se describen en seis recomendaciones de la siguiente manera:

5.3.4.1 Desarrollar Iterativamente

Hoy en día las exigencias de desarrollo de software son cada vez mayores por lo que debe haber un modelo flexible que se adapte a las necesidades cambiantes, sin que este afecte los resultados finales del producto, esto ha hecho que modelos antiguos de desarrollo como por ejemplo el de cascada que sigue una estructura lineal sean poco utilizados, por ello los profesionales del desarrollo de sistemas han optado por utilizar un modelo de desarrollo iterativo en sus aplicaciones, ya que permite una comprensión creciente de los requerimientos a la vez que se van agregando más funcionalidades al sistema, así, si se descubren fallas en fases posteriores de diseño, los costos serán controlados de mejor manera, a diferencia de seguir un modelo secuencial.

RUP utiliza un modelo iterativo en sus procesos en el que aborda las tareas más riesgosas del proyecto con el fin de conseguir un sistema ejecutable en el menor tiempo posible. La figura 5.3.2 muestra el proceso de una iteración en la que se debe realizar un análisis de las necesidades y requerimientos para luego seguir con la implementación y las pruebas respectivas, como resultado de esto se obtendrá un producto ejecutable.



Figura 5.3.2. Desarrollo Iterativo de un proceso RUP

5.3.4.2 Administrar Requerimientos

Teniendo en cuenta que los requerimientos reflejan las necesidades de los clientes de un sistema y que es necesario descubrirlos, analizarlos y documentarlos; la administración de requerimientos consiste en organizarlos de tal forma que queden documentados según su funcionalidad, riesgo, costo y prioridad; esto permite visualizar con precisión lo que hará el sistema, el progreso del proyecto y cómo este ayudará a los procesos de negocio, una buena forma de captar los requerimientos es mediante los casos de uso. La tabla 5.3.1 muestra un ejemplo básico de cómo administrar los requerimientos.

ADMINISTRACIÓN DE REQUERIMIENTOS DEL SISTEMA ABC				
Nro. De Requerimiento	Status	Riesgo	Prioridad	Costo
Req.1	Pendiente	Alto	Alto	\$ 100.33
Req.2	Aprobado	Bajo	Bajo	\$ 150.23
Req.3	Obligatorio	Medio	Alto	\$ 99.51
Req.4	Propuesto	Medio	Bajo	\$ 80.45
Req.5	Aprobado	Alto	Alto	\$ 105.78

Tabla 5.3.1. Ejemplo de cómo administrar requerimientos de un sistema

5.3.4.3 Arquitecturas basadas en componentes

Utilizar este tipo de arquitecturas nos permite identificar tempranamente los diferentes componentes del software que vamos a utilizar para el desarrollo del mismo, es decir, todo lo que necesitamos para que la aplicación funcione correctamente, entre los más conocidos están las librerías, los compiladores, archivos auxiliares como ejecutables, los compiladores y módulos del sistema. Este tipo de arquitecturas son bastante flexibles ya que promueven la reutilización de componentes y son fáciles de modificar, un aspecto a tener muy en cuenta es que no debemos confundir los componentes con sistemas operativos, bases de datos ni lenguajes de programación.

5.4.4.4 Modelamiento Visual

El modelamiento visual de una aplicación consiste en generarnos un nivel de abstracción de lo que será la estructura final del mismo, así como el comportamiento de la arquitectura y de los componentes; con esto lograremos analizar la consistencia entre el diseño y la implementación verificando la relación entre los diferentes componentes que se integrarán a la aplicación.

Una técnica muy útil dentro del modelamiento visual es la utilización del UML a través de diagramas de Casos de Uso, diagramas de Clases, diagramas de Estados, diagramas de Componentes y diagramas de Implementación.

5.4.4.5 Verificar la Calidad

Consiste en evaluar constantemente la calidad de un sistema con respecto a su funcionalidad, desempeño y confiabilidad.

Estas prácticas hoy en día forman parte de un proceso completo de desarrollo y no pueden estar aisladas o ser solo responsabilidad de un pequeño grupo de trabajo, para ello, es necesario contar con un plan de actividades de aseguramiento de la calidad que nos de la confianza de que el producto que estamos desarrollando cumple con todos los requerimientos propuestos y que el proceso es efectivo, esto nos asegurará la aceptación del cliente.

5.3.4.6 Controlar los Cambios

“Todo en el software cambia. Los requisitos cambian. El diseño cambia. El negocio cambia. La tecnología cambia. El equipo cambia. Los miembros del equipo cambian. El problema no es el cambio en sí mismo, puesto que sabemos que el cambio va a suceder; el problema es la incapacidad de adaptarnos a dicho cambio cuando éste tiene lugar”¹⁹.

Es necesario controlar y monitorear los cambios que se producen en el transcurso de nuestro proyecto mediante un proceso iterativo de desarrollo en el cual la comunicación es importante ya que se realizan solicitudes formales de cambios y para ello se necesita la claridad en la misma, caso contrario no existirá un entendimiento absoluto de los cambios.

¹⁹ Kent Allan Beck, autor de la Programación Extrema.

5.4.5 Disciplinas de un proceso RUP

Si visualizamos la figura 5.3.1 de la estructura del RUP, claramente podemos distinguir dos tipos de disciplinas, las de proceso y las de soporte. Una disciplina es un conjunto de actividades vinculadas a un área específica del proyecto y RUP las utiliza para facilitar la comprensión de todo proyecto de desarrollo. Un flujo de trabajo describe la secuencia en que se realizan las actividades en una disciplina, quienes la realizan y que artefactos se producen. Las disciplinas de proceso son las siguientes:

- Modelado del Negocio: Describe la estructura y el modelo de negocio de la organización.
- Requisitos: Refleja las necesidades del cliente expresado en casos de uso.
- Análisis y Diseño: Describe las diferentes vistas arquitectónicas del proyecto.
- Implementación: Comprende el desarrollo del software, prueba de unidades e integración.
- Pruebas: Son las diferentes métricas de evaluación del proyecto.
- Despliegue: Configuración del sistema que se va a entregar.

Mientras que las disciplinas de soporte son:

- Gestión de Configuraciones: Ayuda en el control de cambios y la integridad de los artefactos del proyecto.
- Gestión del Proyecto: Son las diferentes estrategias de trabajo en un proceso iterativo.
- Entorno: Es la infraestructura necesaria para desarrollar un sistema.

5.4.6 Ventajas y Desventajas de RUP

Ventajas	Desventajas
RUP al ser un marco de trabajo flexible, puede utilizarse en una variedad de tipos de sistemas, diferentes áreas de aplicación, tipos de organizaciones y dimensiones de proyectos.	RUP no menciona nada con respecto a la gestión y acuerdo de suministros.
Divide el esfuerzo del desarrollo de un proyecto de software en iteraciones que se ejecutan de forma planificada.	La gestión y evaluación de la calidad no se la realiza de forma muy detallada en la fase de transición.
Utiliza las seis mejores prácticas de desarrollo para lograr la eficiencia en la producción de software.	La medición y análisis de los procesos en cada fase no están contemplados de forma detallada.
Permite definir de manera clara quién debe hacer las cosas, qué debe hacerse, cuándo y cómo.	Se necesita mayor esfuerzo en la construcción de los modelos.
Su enfoque le permite utilizar modelos en lugar de gran cantidad de documentación.	
RUP es abierto y permite la incorporación de enfoques y artefactos complementarios como patrones de diseño y patrones de implementación.	

Tabla 5.3.2. Ventajas y Desventajas de RUP

5.5 Metodología Watch (Método del Reloj)

5.4.1 Generalidades

El método Watch es una metodología de desarrollo de software que permite describir de mejor manera los procesos técnicos, gerenciales y de soporte que utilizan actualmente los grupos de desarrollo para aplicaciones empresariales. Este método está fundamentado en tres conceptos básicos que llevan a cabo prácticas específicas para el desarrollo de proyectos de software, el primero CMMI (Capability Maturity Model Integration) debido a la mejora y evolución de los procesos de desarrollo y mantenimiento de productos de software, RUP debido al enfoque que tiene en la utilización de modelos en lugar de gran cantidad de documentación utilizando el UML que es un lenguaje concreto y bien definido, y finalmente se fundamenta en el PMBOK, un proyecto basado en la administración del cuerpo del conocimiento.

El método Watch utiliza un procedimiento de desarrollo incremental e iterativo en el que se van agregando más características al sistema conforme este avanza y es uno de los métodos más flexibles en la asistencia de desarrollo de aplicaciones ya que integra procesos de gestión con los procesos técnicos y de soporte usando las mejores prácticas de la ingeniería de software y gestión de proyectos.

5.4.2 Estructura del Método Watch

El método Watch basa su estructura en el concepto de las manecillas del reloj en sentido horario, en el que cada etapa del ciclo de vida se organiza en forma de un reloj. Este método está constituido por los siguientes tres procesos: procesos Técnicos que definen el ciclo de vida de una aplicación, procesos de Gestión que garantizan la calidad de la aplicación y procesos de Soporte que ayudan al cumplimiento de todas las actividades y planes definidos en el proyecto, este concepto se aprecia de mejor manera en la figura 5.4.1 de la estructura del método Watch.



Figura 5.4.1. Estructura del Método Watch

5.4.2.1 Fase de Análisis

Es la primera fase del ciclo de vida del método Watch y se basa en evaluar la función, rendimiento, restricciones, fiabilidad e interfaces que tendrá el sistema. Las restricciones de rendimiento abarcan los requisitos de tiempo de respuesta y procesamiento, se analizan las expectativas del proyecto y los costos del mismo, un error en la estimación del costo puede acarrear pérdidas grandes y poca recompensa al esfuerzo aplicado en el desarrollo.

Se definen también las metas del sistema a partir de entrevistas con quienes serán los usuarios del sistema, para ello se debe hacer un análisis de la problemática actual y la manera en que se llevan los procesos empresariales.

5.4.2.2 Fase de Diseño

La etapa de diseño implica la construcción de modelos del sistema con diferentes niveles de abstracción, en los cuales se vea reflejada la estructura del software que se va a implementar en la siguiente etapa, se diseñan también las diferentes interfaces y se detallan algunos algoritmos a utilizar. En esta fase por lo general empiezan a descubrirse errores y falencias que fueron pasados por alto en la etapa anterior, estos se corrigen y se mejoran los modelos de diseños.

Es importante identificar los diferentes subsistemas que forman el sistema y las relaciones que existen entre ellos, se debe diseñar una interfaz por cada uno de ellos y una estructura de datos a utilizar en la implementación del sistema, todos estos procesos deben estar documentados.

5.4.2.3 Fase de Construcción

La fase de construcción implica convertir una especificación del sistema en un producto ejecutable y funcional, se integran conceptos avanzados de programación de software en la implementación de las diferentes clases y modelos desarrollados en la etapa anterior. Es importante que el equipo de trabajo este coordinado de tal forma que el desarrollo de los diferentes módulos se los realice en el menor tiempo posible, esta coordinación está a cargo del jefe del proyecto quien debe organizar su equipo según las capacidades intelectuales de sus miembros, así habrán grupos expertos en programación, otros en base de datos, etc.

Es importante tener en cuenta que el software forma parte de casi todas las operaciones de negocio, por lo que es fundamental que el software nuevo se construya rápidamente para así aprovechar nuevas oportunidades del mercado sin dejar de lado la calidad del mismo.

5.4.2.4 Fase de Pruebas

Esta fase consiste en verificar todos los elementos del sistema a través de pruebas individuales y de integración, esto nos permite garantizar el correcto funcionamiento del proyecto en conjunto y que este satisface las especificaciones del cliente. El realizar pruebas al sistema en etapas tempranas del desarrollo nos permite mejorar la calidad del sistema detectando errores, falencias y demás inconsistencias presentes en los requisitos, para ello se debe contar con datos de prueba, con resultados esperados y los actores participantes.

Entre algunos de los factores que inciden en las fallas de un proyecto de software se encuentran los requisitos, en la mayoría de los casos estos son incompletos e inadecuados, por ello la necesidad de adelantar la generación de pruebas del sistema para realizar una verificación adicional sobre los requisitos y así corregir errores y mejorar la calidad. Una vez detectados los errores, se debe redactar un informe informando de ello a los encargados del levantamiento de requisitos del proyecto a que procedan a realizar las correcciones del caso.

5.4.2.5 Fase de Instalación

Esta fase comprende la entrega del producto al usuario y la instalación del mismo en su lugar de trabajo, el sistema pasa a producción y empieza a ser evaluado con información de entrada real del negocio. Se debe entregar al usuario la documentación respectiva del sistema en el que se indique el funcionamiento del programa, indicaciones y recomendaciones del mismo, así el se sentirá respaldado de contar con una ayuda documentada; aparte de ello se debe entrenar a los usuarios en el manejo del software para que así se familiaricen rápidamente con la aplicación y hagan de mejor manera los procesos que antes les costaba más trabajo realizarlos. El proceso no termina aquí, ya que siempre es necesario brindar asistencia técnica en caso de que ocurran errores en el software y corregirlos de manera que el cliente se sienta satisfecho de la inversión tecnológica realizada.

5.4.3 Componentes del Método Watch

Watch se compone de tres modelos básicos, un modelo de productos, un modelo de actores y un modelo de procesos, el primero describe las características generales que tienen las aplicaciones empresariales identificando los productos que se deben producir durante el desarrollo de una aplicación, el segundo se basa en la organización de los grupos de trabajo que desarrollan las aplicaciones, de igual manera los roles y responsabilidades de los diferentes actores que forman parte de los equipos; y por último el modelo de procesos que describe los procesos técnicos, gerenciales y de soporte que emplean los grupos de trabajo para desarrollar las aplicaciones. La figura 5.4.2 muestra los componentes del método Watch.

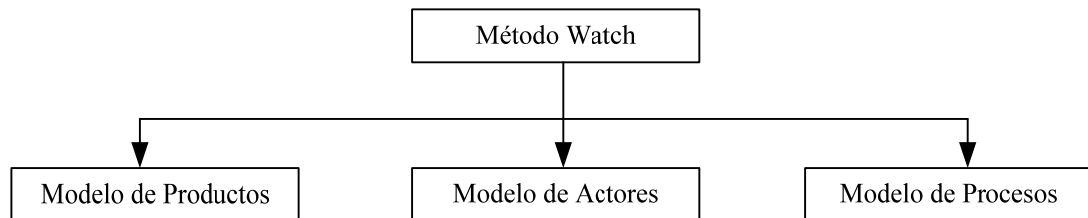


Figura 5.4.2. Componentes del Método Watch

5.4.3.1 Modelo de Productos

En cada actividad de los procesos técnicos, de gestión y de soporte se obtienen productos, de los dos primeros mencionamos algunos:

- Documento del dominio de la aplicación.- En este documento se describen los objetivos, costos, tiempo y recursos del proyecto.
- Documento del plan de tareas.- Contiene un cronograma de tareas, holguras y actividades críticas.
- Documento de la estructura del grupo de trabajo.- Se identifican los roles de los miembros del grupo a través de un diagrama de jerarquías.
- Documento de pruebas.- Contiene los planes de verificación y validación y el detalle de las pruebas a ejecutar para probar el correcto funcionamiento de la aplicación.

- Documento de riesgos del proyecto.- Contiene un plan de riesgos en el que se descubren las posibles vulnerabilidades, la probabilidad de ocurrencia de riesgos y actividades a llevarse a cabo para mitigarlos.

Entre los productos finales o entregables como aplicación empresarial se encuentran los diferentes programas, bases de datos y manuales que se utilizarán para el entrenamiento del personal en cuanto al uso de la aplicación. En el apartado del modelo de procesos (5.4.3.3) se describen exactamente los productos finales que se obtienen de los procesos técnicos, de gestión y de soporte, esenciales dentro de la metodología Watch.

5.4.3.2 Modelo de Actores

Un actor es una persona o una unidad organizacional que forma parte de un proyecto de software, entre los principales se encuentra el actor responsable de los procesos gerenciales que es el líder del proyecto, su rol es manejar todas las decisiones con respecto a los resultados obtenidos de los procesos de validación y verificación en cada etapa del ciclo de vida de la aplicación.

El modelo de actores identifica los diferentes actores del proyecto, así como los roles que desempeñan a favor del desarrollo de la aplicación, la figura 5.4.3 muestra un ejemplo de un modelo de actores.

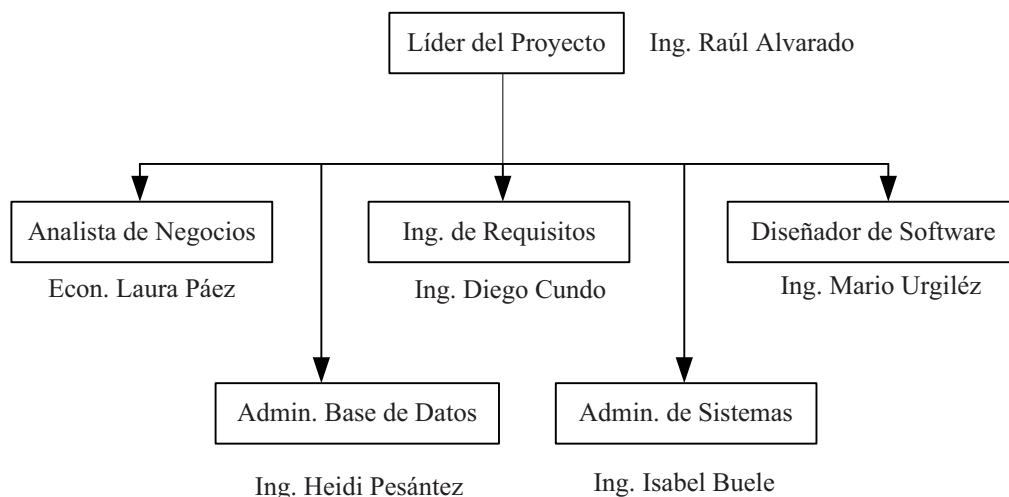


Figura 5.4.3. Ejemplo de un Modelo de Actores

5.4.3.3 Modelo de Procesos

Un proceso es un conjunto de actividades con un mismo fin, el modelo de procesos corresponde a los procesos que definen la trayectoria del proyecto y como se administran los recursos del equipo sean estos materiales o humanos, este se divide en procesos técnicos, procesos de gestión y procesos de soporte.

Los procesos técnicos se ejecutan en el orden de las manecillas del reloj, con la libertad de poder avanzar al próximo proceso o retroceder al anterior según los resultados obtenidos en el proceso de validación y verificación y la decisión del jefe del proyecto. Los procesos técnicos de análisis permiten modelar el dominio de la aplicación empresarial e identificar los diferentes requisitos que la aplicación debe satisfacer, los procesos de diseño establecen la arquitectura de la aplicación, los de construcción producen la aplicación de acuerdo a las especificaciones del diseño arquitectónico y los de pruebas aseguran el correcto funcionamiento operacional de la aplicación, los procesos técnicos se describen en la figura 5.4.4 de los componentes del modelo de procesos.

Como resultado de este proceso se obtienen los siguientes productos: un documento del dominio de la aplicación, documento de requisitos, documento de diseño, documento de implementación y un documento de pruebas.

Los procesos de gestión definen las tareas del líder en todas las etapas del proceso y asegura que la construcción de la aplicación sea realizada de forma organizada y eficiente y sobretodo que cumpla con el cronograma planteado, bajo los presupuestos asignados y procedimientos establecidos, todo con el fin de asegurar la calidad del producto al usuario final, los procesos de gestión se describen en la figura 5.4.4 de los componentes del modelo de procesos.

Como resultado de este proceso se obtienen los siguientes productos: plan del proyecto, informes de gestión, informe de procesos de desarrollo y notas del proyecto.

Los procesos de soporte son un conjunto actividades que brindan soporte a las tareas del líder del proyecto, aseguran la calidad de los productos, controla que el proceso de desarrollo definido para cada proyecto se cumpla, optimiza la configuración de las aplicaciones empresariales, maneja los riesgos que se puedan presentar en el

proyecto con el fin de mitigarlos, garantiza el uso apropiado de las aplicaciones mediante un entrenamiento a los usuarios finales sobre el funcionamiento del sistema y además asegura que el personal que conforma el equipo de trabajo es el idóneo y cuenta con los conocimientos necesarios para realizar las tareas requeridas de forma eficaz y eficiente, en las actividades de validación y verificación, el líder del proyecto es quien decide según los resultados obtenidos si continuar con la próxima fase o regresar a una fase anterior para corregir los errores del producto. Los procesos de soporte se describen en la figura 5.4.4 de los componentes del modelo de procesos.

Como resultado de este proceso se obtienen los siguientes productos: documento del plan de gestión de la configuración, documento del plan de gestión de la calidad, documento del plan de gestión de los riesgos, documento del plan de validación y verificación, documento del plan de pruebas y un documento del plan de formación a los usuarios finales sobre el uso y funcionamiento de la aplicación.

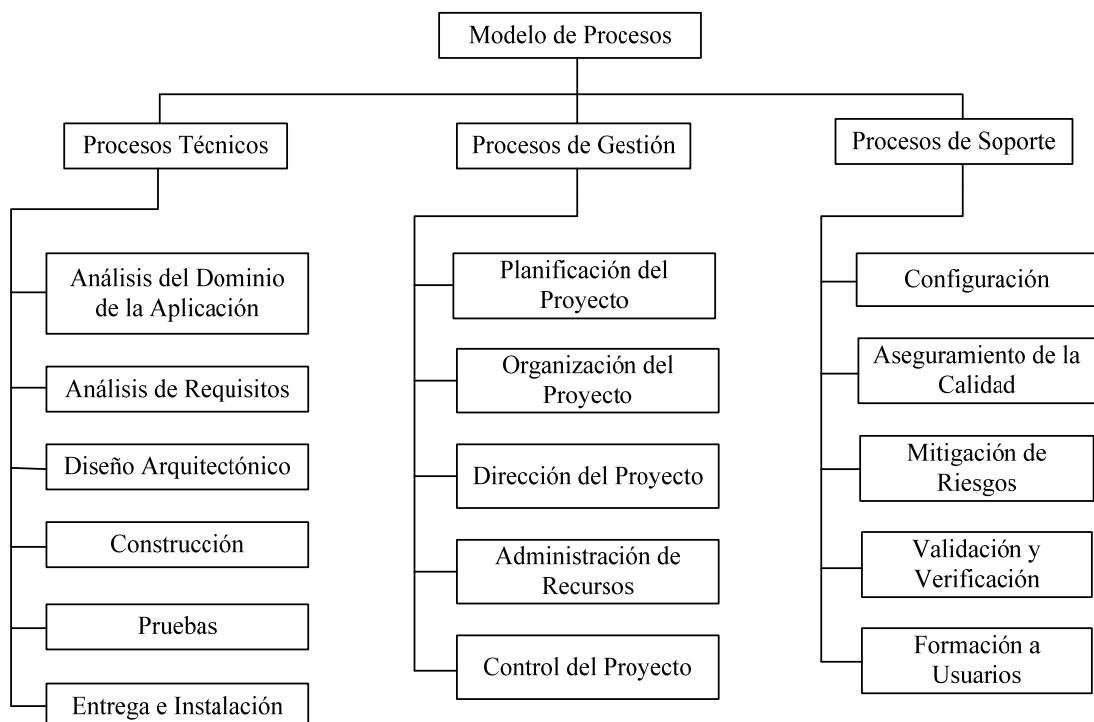


Figura 5.4.4. Componentes del Modelo de Procesos

5.5.4 Ventajas y Desventajas del Método Watch

Ventajas	Desventajas
La especificación del método de desarrollo de software es completamente útil ya que indica las actividades a realizar (Proceso), las actividades de los miembros del equipo (Actores) y los productos que cada actividad produce (Producto).	Es un método de desarrollo poco aplicable debido al nivel de detalle que exige para su implementación, esto hace que sea poco factible un desarrollo real con todas las características de este método.
Se especifican los procesos de gestión con los procesos técnicos y de soporte de forma detallada.	
Utiliza un procedimiento de desarrollo incremental e iterativo en el que se van agregando más funcionalidades al sistema.	
Integra aspectos de desarrollo del modelo espiral y desarrollo incremental.	
La estructura de procesos del modelo fue creado a través del estándar IEEE 1074 ²⁰ , para determinar el conjunto de actividades que deben ser incorporadas en el desarrollo de un producto.	

Tabla 5.4.1. Ventajas y Desventajas del Método Watch

5.5 Características del modelo de desarrollo de Eric S. Raymond

Eric S. Raymond ha sido un colaborador del Software Libre realizando varias contribuciones con el proyecto GNU, escribió varios ensayos a favor del software libre, pero sin duda el más conocido es “La catedral y el bazar”, en esta publicación se dictan ejemplos guías a tomar en cuenta para el desarrollo de Software Libre. Estas lecciones son las experiencias obtenidas de la realización del proyecto de Software Libre llamado *fetchmail*.

En esta publicación Raymond cuenta que era un usuario del sistema operativo Unix y no se imaginaba que el Sistema GNU iba a tener el éxito que obtuvo, pensaba que aplicaciones complejas tales como un sistema operativo requerían un enfoque más planeado y centralizado, así como la construcción de las catedrales, es decir, que

²⁰ Estándar para desarrollar modelos de procesos y métodos de desarrollo de software, el cual proporciona un marco metodológico para el diseño de los mismos, documento ubicado en: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?tp=&isnumber=16018&arnumber=741936&punumber=5984

debía ser cuidadosamente elaborado por genios, sin liberar versiones beta antes de tiempo.

A raíz de la liberación del Kernel Linux y su éxito, le llamó la atención el estilo de desarrollo de Linus Torvalds en el que consejos como “libere rápido y a menudo, delegue todo lo que pueda, sea abierto hasta el punto de la promiscuidad”. No eran una forma reverente de construir una catedral, más bien la comunidad Linux se asemejaba más a un bullicioso bazar de Babel, colmado de individuos con propósitos y enfoques dispares en el que se podían aceptar contribuciones de cualquiera. Aunque parecía muy difícil poder obtener un sistema coherente y estable, este estilo de desarrollo pudo conseguir un muy buen sistema operativo.

Raymond se adentró más en este tipo de desarrollo participando cada vez con proyectos más complejos, siguiendo de cerca el estilo de desarrollo del “bazar”, buscó demostrar que el estilo de desarrollo del “bazar” era mucho mejor que el de la “catedral”, escribiendo este ensayo en el que comparaba sus estilos de desarrollo.

5.5.1 Lecciones enumeradas en el ensayo “La Catedral y el Bazar”²¹

1. “Todo buen trabajo de software comienza a partir de las necesidades personales del programador (todo buen trabajo empieza cuando uno tiene que rascarse su propia comezón).”

Quiere decir que cuando el software surge de necesidades propias, los programadores trabajan con más gusto y énfasis, a que cuando se trabaja a cambio de un salario, en programas que no los necesitan, o quieren.

2. “Los buenos programadores saben qué escribir. Los mejores, qué reescribir (y reutilizar).”

Raymond dice que para ser un buen programador ha tratado siempre de imitar a uno de ellos. Debido a que una característica sobresaliente de los grandes programadores es la meticulosidad con la que construyen software. Debido a que obtener reconocimiento no se da en base al esfuerzo, sino por los resultados obtenidos por ellos. Además casi siempre será más fácil partir de una buena solución parcial que

²¹ Lecciones obtenidas del ensayo “La catedral y el Bazar” de Erick S. Raymond 1997, traducción de José Soto Pérez.

empezar todo de cero. Parte fundamental del éxito del bazar era compartir el código fuente generado, con lo que crecía el conocimiento de la comunidad.

3. "Considere desecharlo; de todos modos tendrá que hacerlo." (Fred Brooks, *The Mythical Man-Month*, Capítulo 11)

Quiere decir que no se entiende cabalmente un problema hasta que se implementa la primera solución. Y muchas veces en el proceso de desarrollo se topa con problemas muy difíciles de superar con la versión que se está desarrollando, siendo necesario desechar el trabajo realizado y partir de otra base que permita llegar a la resolución del problema de mejor manera. Raymond afirma que se debe estar dispuesto a empezar de nuevo al menos una vez.

4. Si tienes la actitud adecuada, encontrarás problemas interesantes.

Si se participa activamente, y con entusiasmo en un proyecto, este puede alcanzar otras dimensiones. Pudiendo pasar hacerle unos cuantos cambios menores a un software, a realizar cambios mayores, e incluso a hacerse responsable del proyecto de software, lo cual probablemente motivara más al desarrollador del software a implementar características más complejas al software.

5. "Cuando se pierde el interés en un programa, el último deber es darlo en herencia a un sucesor competente."

El interés en el proyecto hace que se trabaje de mejor manera en el mismo, cuando se pierde el interés se descuida el proyecto, aparecen errores, y si no se cambian las cabezas del proyecto corre el riesgo de fracasar, por lo que si se está en esta situación lo mejor es entregar el proyecto a alguien competente que comparta el mismo objetivo con el que surgió el proyecto y que demuestre estar comprometido con el mismo, aunque esto no suele ser nada fácil.

6. "Tratar a los usuarios como colaboradores es la forma más apropiada de mejorar el código, y la más efectiva de depurarlo."

Muchos programadores caen en el error frecuente de no darles a los usuarios la importancia suficiente, y muchas veces ni siquiera el tomarles en cuenta para el desarrollo del software, pero los usuarios son muy importantes, pues ellos además de probar que se está satisfaciendo su necesidad y que se están haciendo bien las cosas,

pueden convertirse en excelentes asociados, sobre todo para reducir tiempo en la depuración del programa, la detección de errores, y con un correcto estímulo en la resolución de los mismos ya sea aportando ideas de cómo resolver problemas, o resolviéndolo ellos mismos y colaborando con el proyecto.

7. “Libere rápido y a menudo, y escuche a sus clientes.”

Raymond afirma que los lanzamientos rápidos y frecuentes de versiones de prueba es parte importante del éxito del núcleo Linux, y muchas otras aplicaciones libres debido a que con la colaboración de los usuarios se podía detectar de manera muy eficaz los errores que tiene el programa. Esto es todo lo contrario a las versiones de programas elaboradas por el modelo de la catedral, pues el estilo propietario busca exponer al usuario al menor número de errores posibles, por lo que los lanzamientos se hacían en tiempos prolongados, mientras se requería demasiado esfuerzo para poder depurar los errores del software entre lanzamientos, mientras que en el estilo del bazar, esto se lo hacía muy rápida y eficiente, de tal forma que una versión estable podía conseguirse con 3 o 4 lanzamientos.

8. “Dada una base suficiente de desarrolladores asistentes y probadores de versiones beta, casi cualquier problema puede ser caracterizado rápidamente, y su solución ser obvia al menos para alguien”.

Raymond bautizo como ley de Linus a la siguiente afirmación: "Con muchas miradas, todos los errores saltarán a la vista".

Afirma que la diferencia fundamental en los estilos de la catedral y el bazar es la forma de realizar las pruebas del programa, afirma que el modelo de la catedral hay pocas personas en el desarrollo del software y solo ellas saben cómo este fue implementado y por ello los errores que surgen en este tipo de software son por lo general dañinos con apariencias inofensivas, profundos, confusos y de difícil comprensión. Hace falta meses de exámenes minuciosos por dedicada por completo para poder confiar que estos han sido superados. Debido a esto es que los existe una gran cantidad de tiempo entre los lanzamientos de prueba y las versiones estables, las que no resultan perfectas de todas formas, debido a que muchas veces no se pueden detectar todos los errores del software.

Mientras que en el modelo del bazar existen una gran cantidad de desarrolladores, los cuales han examinado el programa antes de lanzar una versión de prueba, detectando y corrigiendo los errores incluso antes de las versiones de prueba, por lo que los errores son normalmente leves, y de no serlo se convierten en leves con rapidez debido a que existen muchos colaboradores ansiosos de poner al derecho y al revés el programa, cada uno de los cuales tiene diferentes puntos de vista, herramientas y métodos para realizar las pruebas del programa, es decir a más usuarios más formas de probar el programa, lo que implica que se encuentren más errores, y si los usuarios colaboran en el desarrollo, se encontrarán más errores aun.

Esta característica hace muchísimo más ágil la fase de pruebas y depuración, incluso se puede afirmar que la depuración de un programa puede hacerse en paralelo, pues no necesariamente requiere una gran coordinación entre los diferentes depuradores y la coordinación del proyecto. Aunque podría darse un problema con la duplicación de trabajo por parte de los depuradores, pero este riesgo no es significativo debido a que se soluciona mediante los lanzamientos frecuentes de versiones y la difusión rápida de correcciones ya realizadas.

Además de que en el modelo del bazar se puede ofrecer a los usuarios diferentes versiones, como se lo hizo con el núcleo de Linux enumerándolas de manera que los usuarios pueden elegir entre versiones estables, o arriesgarse a encontrar errores con versiones inestables a cambio de nuevas funcionalidades.

9. “Las estructuras de datos inteligentes y el código torpe funcionan mucho mejor que en el caso inverso.”

En la actualidad el software está en función de la información que este gestiona, ya que la información resulta mucho más valiosa. Si se tiene la información se puede construir software que la manipule, mientras que si se tiene el software ¿Dónde este obtiene su información?

Antes los programadores ponían mucha más atención a el código, y las estructuras de datos las utilizaban solo como un soporte. Cuenta Raymond que para él era muy difícil de comprender el “Pop-Client”, debido a que su autor se había centrado demasiado en el código, descuidando las estructuras de datos, por lo que decidió

cambiar las estructuras de datos y reescribir el código fuente con el fin de poder entenderlo.

Si no se sabe cuál es la estructura de datos del programa el código fuente resulta mucho más complejo de comprender, mientras que en el caso contrario, partiendo desde las estructuras de datos se puede incluso intuir el código.

10. “Si usted trata a la gente que te ayuda a depurar un programa como si fueran su recurso más valioso, ellos le responderán convirtiéndose en su recurso más valioso.”

Raymond en la construcción de fetchmail, siguió el modelo de desarrollo del núcleo Linux, las cuales se asemejan mucho a las características de desarrollo de software ágil, tal como la programación extrema realizando las siguientes acciones:

- Lanzar versiones tempranas y con frecuencia
- Hizo crecer la lista de probadores de versiones beta, incluyendo a todos los que se ponían en contacto con él, demostrando interés en el proyecto.
- Envío anuncios espectaculares a la lista de probadores de versiones beta cada vez que publicaba una nueva versión, animando a la gente que continúe participando.
- Escuchaba lo que le decían sus colaboradores, les informaba y consultaba sobre las decisiones que tomaba respecto al diseño del software, tomaba en cuenta sus mejoras y agradecía su colaboración.

Obteniendo como recompensa de estas acciones informes de errores de alta calidad, muchas veces estos incluían soluciones propuestas.

Una forma de medir el éxito de la fase de depuración de las versiones beta es el tamaño de la lista de lista de colaboradores, fetchmail tenía alrededor de 249 miembros, muchos de los cuales al ser cubierta sus necesidades, pedían que se les retire de la lista porque consideraban que no era necesario seguir participando en la depuración.

11. “Lo mejor después de tener buenas ideas es reconocer las buenas ideas de sus usuarios. Esto último es a veces lo mejor.”

Cuando se está desarrollando software en conjunto, van a existir varias ideas para mejorar el programa, incluso puede seguir alguna nueva funcionalidad que puede ser tan buena que incluso pueden hacer parecer obsoletas a funciones existentes, y los programadores encargados del software deben estar en capacidad de reconocer estas ideas aunque su trabajo anterior se vea afectado, al haber un grupo grande de colaboradores es muy probable que surjan ideas que serán mejores que las ideas propias de la dirección del proyecto.

12. “Frecuentemente, las soluciones más innovadoras y espectaculares provienen de comprender que la concepción del problema era errónea.”

Cuando se emprende un proyecto de software, no es seguro que siempre se tomara el camino correcto para su implementación, si se encuentran con problemas muy difíciles de resolver, o sin tener ideas de cómo continuar el proyecto, puede ser que no se está desarrollando la respuesta correcta, y el problema debe volver a ser planteado.

13. "La perfección (en diseño) se alcanza no cuando ya no hay nada que agregar, sino cuando ya no hay nada que quitar."

No hay que dudar en quitar al programa características que no son necesarias, sobre todo si se puede hacer sin pérdidas de eficiencia y calidad. Al hacer esto se obtienen muy buenos beneficios, tales como incrementar el rendimiento del programa, la reducción de tareas correspondientes a las características retiradas del programa, por ejemplo la elaboración de manuales.

Cuando el código de un programa se hace mejor y más sencillo, se puede notar que este es correcto, por lo tanto la depuración que tenga el código de un programa influye en la calidad del mismo.

14. Toda herramienta es útil empleándose de la forma prevista, pero una *gran* herramienta es la que se presta a ser utilizada de la manera menos esperada.

Muchas veces en la marcha del desarrollo del programa se puede optar por realizar mejoras que al inicio no estaban contempladas, Cuenta Raymond que él no imaginaba ni planeaba el resultado que tuvo fechmail, por lo que tuvo que subir sus propios estándares en su desarrollo, para que este siga cumpliendo con las

expectativas de otros usuarios, pero siempre tratando de mantener el programa sencillo y robusto agregando mejoras que no las planeo al inicio del proyecto, con lo que su útil herramienta se convirtió en una gran herramienta.

El éxito de un programa puede darse cuando se lleva las buenas ideas de las personas a niveles que ellos no imaginaban, mediante una buena visión de ingeniería, tal como lo hizo Linus Torvalds al llevar una buena idea original de Andrew Tanenbaum de la construcción de un UNIX nativo simple, en el núcleo Linux.

15. “Cuándo se escribe software para una puerta de enlace de cualquier tipo ('gateway software'), hay que tomar la precaución de alterar el flujo de datos lo menos posible, y ¡*nunca* eliminar información a menos que los receptores obliguen a hacerlo!”.

Dar otro tipo de codificaciones a datos, pueden llevar a errores y ser muy difícil de implementar si no se sigue esta regla, mejor resulta entender mejor cómo funciona el envío de información y modificar su lógica, para adaptarla a las necesidades específicas del programa.

16. Cuando su lenguaje está lejos de un Turing completo, entonces el azúcar sintáctico puede ser su amigo.

Esto se refiere a que muchos software utiliza archivos de configuración, (por ejemplo los .rc) estos incluyen ordenes adicionales para el programa, los cuales tienen una serie de órdenes, las cuales antiguamente se formaban por pares de dato – orden, pero en la actualidad con los recursos que disponen los computadores, es factible pensar en un lenguaje que favorezca más a la comprensión de los usuarios que a la eficacia del computador, sin embargo se debe ser prudente, este lenguaje no debe ser causa de errores o confusiones de los usuarios, además si se quiere asemejar este lenguaje a otro por ejemplo del tipo inglés, se debe restringir el dominio del lenguaje haciendo que no se asemeje a un lenguaje de propósito general, debido a que si las cosas que dice este lenguaje no son muy complicadas será más fácil de comprender para los usuarios y se evitara errores.

17. “Un sistema de seguridad es tan seguro como un secreto. Cuídese de los secretos a medias.”

Raymond ponía por ejemplo que los usuarios de fechmail le pedían que codificara las contraseñas en los archivos .rc, pero él no lo hizo debido a que brindaría una falsa sensación de seguridad a los usuarios, sobre todo a los que no estén muy involucrados en el proyecto, debido a que si alguien lograba tener acceso a estos archivos, con encontrar un decodificador podría descifrar las contraseñas.

Muchas veces es mucha más perjudicial brindar un ambiente de falsa sensación de seguridad que no brindar ninguna, debido a que muchos usuarios se confían que su información está segura y no le prestan mayor importancia a aspectos de seguridad complementarios.

18. “Para resolver un problema interesante, comience por encontrar un problema que le resulte interesante.”

Prácticamente Raymond vuelve a plantear la regla N°1, pero reformulándola, y quiere decir que la mayoría de proyectos de software surgen de una necesidad propia del desarrollador, y se difunden cuando afecta también a un amplio número de usuarios, esta es la base para la etapa siguiente que es la evolución del programa, cuando hay una comunidad amplia y activa de usuarios y colaboradores.

19. Si el coordinador de desarrollo tiene un medio al menos tan bueno como lo es Internet, y sabe dirigir sin coerción, muchas cabezas serán, inevitablemente, mejor que una.

El sistema GNU con Linux fue el primero en demostrar que es posible utilizar el mundo entero como su base de talento, este sistema se adaptó a las nuevas reglas que el internet dispuso, al permitir que millones de personas puedan colaborar con este proyecto, parte importante del éxito de este proyecto fue que las personas que participaban de él no se veían presionadas de ninguna forma, sino lo hacían por su ego, por dar realce a su nombre y demostrar su capacidad, esto sin duda es mucho más motivador que dedicar su tiempo a base de un salario. Además otro aspecto importante para poder lograr el éxito es el renunciar a intereses mezquinos y mercantilistas, sino más bien colaborar para el bienestar común de todos, aceptar la

colaboración de varias personas y su experiencia es parte fundamental del éxito de estos proyectos de software.

En este ensayo se menciona que este estilo de diseño es muy útil para cuando se parte de algún otro proyecto, pues es complicado el iniciar desde cero un proyecto de software, pero si se cuenta con una base robusta, los colaboradores realizan un trabajo excepcional, algo que difícilmente puede ser superado por el estilo de la catedral, es decir a la hora de programar, puede más el ego, su satisfacción y el reconocimiento para los programadores que el dinero.

Además cuenta que el éxito del software es un buen diseño y la capacidad de ingeniería que tenga el desarrollador del proyecto, en este ensayo Raymond afirma que Linus Torvalds pudo crear el núcleo Linux debido a un gran diseño y la colaboración de miles de personas, y que su merito era hacer que sus colaborados participen activamente, y prácticamente el recibió el merito, pero el trabajo fue de sus colaboradores.

5.6 Cuadro Comparativo de las Metodologías analizadas

	Características	Ventajas	Desventajas	Cuando Usar
EXTREME PROGRAMMING	Desarrollo incremental y en iteraciones	Se adapta muy bien a los requisitos cambiantes	Dificultad para determinar el costo y tiempo del proyecto	Cuando los requisitos cambian constantemente (clientes indecisos)
	Programación por parejas	Disminuye la tasa de errores		Cuando se tiene un equipo de desarrollo motivado y responsable.
	El usuario es parte del equipo de desarrollo	Permite superar con mayor facilidad circunstancias imprevistas en el desarrollo de software	Se aplica solo para proyectos pequeños	Cuando se cuenta con Clientes que entienden y participan activamente en el proyecto
	Simplicidad del código	Garantiza que el software funcione como el usuario lo requiere.		
	Entregas rápidas y Frecuentes	Permite una mejor y más eficiente comunicación entre todos los integrantes del proyecto		
	Pruebas unitarias continuas	Calidad de software en el menor tiempo		
RATIONAL UNIFIED PROCESS	Desarrollo iterativo en etapas	Utiliza las seis mejores prácticas de desarrollo para lograr la eficiencia en la producción de software	no menciona nada con respecto a la gestión y acuerdo de suministros	Cuando existe una comunicación entre los equipos de desarrollo del proyecto
	Programación por equipos	Permite definir de manera clara quién debe hacer las cosas, qué debe hacerse, cuándo y cómo	La gestión y evaluación de la calidad no se la realiza de forma muy detallada en la fase de transición	Cuando existen proyectos de desarrollo complejos

	Características	Ventajas	Desventajas	Cuando Usar
	Se describe en dos dimensiones, estática y dinámica; la dinámica representa el tiempo y muestra aspectos del ciclo de vida; la estática muestra los diferentes recursos utilizados	Su enfoque le permite utilizar modelos en lugar de gran cantidad de documentación	La medición y análisis de los procesos en cada fase no están contemplados de forma detallada	Cuando se requiere una correcta configuración y control de cambios en los documentos generados en el proyecto
	Empieza el UML como lenguaje de modelado	Permite la incorporación de enfoques y artefactos complementarios como patrones de diseño y patrones de implementación	Se necesita mayor esfuerzo en la construcción de los modelos	
	Proporciona la documentación del ciclo de vida en el mismo proceso	Puede utilizarse en una variedad de tipos de sistemas, diferentes áreas de aplicación, tipos de organizaciones y dimensiones de proyectos.		
METODO WATCH	Cada etapa del ciclo de vida se organiza en forma de un reloj.	Indica las actividades a realizar (Proceso), las actividades de los miembros del equipo (Actores) y los productos que cada actividad produce (Producto)	Método de desarrollo poco aplicable debido al nivel de detalle que exige para su implementación	Cuando se requiere la asistencia completa de un método para desarrollar software que proporcione un marco metodológico detallado en el que se indique de manera transparente todas las actividades a realizar en la ejecución del proyecto
	Desarrollo incremental y en iteraciones	Se especifican los procesos de gestión con los procesos técnicos y de soporte de forma detallada		
	Empieza el UML como lenguaje de modelado	Integra aspectos de desarrollo del modelo espiral y desarrollo incremental		
	Integra procesos de gestión con los procesos técnicos y de soporte usando las mejores prácticas de la ingeniería de software y gestión de proyectos	La estructura de procesos del modelo fue creado a través del estándar IEEE 1074		

Tabla 5.6.1. Cuadro Comparativo de las Metodologías analizadas

CAPITULO VI

METODOLOGÍA DE DESARROLLO APLICADA AL SOFTWARE LIBRE

Introducción

En la actualidad los productos de software tienen un papel muy importante en la sociedad, por ello es indispensable contar con mecanismos claros y precisos en sus diferentes fases de producción que aseguren la calidad de los mismos, por ello este capítulo presenta como propuesta una metodología de desarrollo aplicada al Software Libre que servirá como guía para la construcción de aplicaciones y que puede ser empleada y adaptada según los requerimientos de quién lo utilice. La documentación generada por la misma provee un punto de referencia para los documentos utilizados en proyectos de desarrollo de software, ayudando así a los desarrolladores a trabajar más rápido y a tomar en cuenta todos los aspectos más importantes del proceso de desarrollo.

Esta metodología se fundamenta en algunas de las metodologías analizadas en el capítulo anterior, entre ellas *Extreme Programming*, *Rational Unified Process* y el *Método Watch*, adoptando el UML (Unified Modeling Language) como lenguaje de modelado. También se combinan características del modelo de desarrollo de Eric S. Raymond quién escribió varios ensayos a favor del Software Libre, entre ellos el más conocido “La Catedral y el Bazar”, en que se expresa que el desarrollo de software del estilo Catedral está dirigido de manera centralizada y el proceso de desarrollo está restringido a un grupo de programadores, mientras que en el estilo del Bazar el desarrollo de software no es dirigido de manera centralizada y se la realiza con la participación de una comunidad que libera el código de la versión desarrollada.

Cabe recalcar que esta metodología propuesta no pretende ser un estándar rígido exento a cambios, sino una herramienta de apoyo para los desarrolladores de aplicaciones proporcionando herramientas básicas para que estos cumplan con un proceso de desarrollo adecuado y así logren el éxito esperado en sus proyectos.

La metodología propuesta en si está planteada para proyectos grandes, pero puede adaptarse a proyectos medianos y pequeños, para lo cual se deben suprimir algunos roles y actividades que no sean indispensables, por lo que sugerimos leer y analizar

esta metodología con el fin de que se pueda comprender de mejor manera cuales roles y actividades son necesarios para un proyecto de software.

6.1 Estructura de la Metodología Propuesta

Para definir la estructura de esta metodología adaptaremos algunas de las características de la metodología RUP, primero, las seis mejores prácticas de desarrollo de software que servirán de línea base para el proyecto, las cuales se encuentran descritas en el apartado 5.3.4 del capítulo anterior, y segundo, la manera de distinguir un proceso de desarrollo según sus dimensiones estática y dinámica.

La dimensión estática representa las diferentes actividades, el personal encargado de las mismas, los roles, artefactos, disciplinas y los flujos de trabajo del proyecto; mientras que la dinámica muestra las distintas fases del ciclo de vida que se presentan en el transcurso del proyecto, estas fases son: fase de análisis del dominio de la aplicación, especificación de requerimientos, diseño, construcción, pruebas y liberación, todas estas heredan el concepto del método Watch que consiste en organizarse en forma de un reloj en sentido horario facilitando el cumplimiento de todas las actividades y planes definidos en el proyecto garantizando así la calidad de la aplicación.

De la metodología Extreme Programming se adoptan conceptos como la programación en parejas en las que el código se discute, se analiza y se revisa por ambos desarrolladores, así el código generado es de mejor calidad, con menos errores y es producto de la genialidad y fusión de conocimientos de dos personas.

6.1.1 Dimensión Estática de la Metodología Propuesta

La dimensión estática de la metodología propuesta está basada en la dimensión estática de la metodología RUP, estos aspectos dividen al proceso en términos de componentes de proceso, disciplinas, flujos de trabajo, actividades, artefactos y roles, es decir representan a la organización a lo largo del contenido agrupando las actividades de manera lógica de acuerdo a su naturaleza. En un proceso de desarrollo de software se definen las siguientes preguntas:

- **¿Quién hace?**, para responder esta pregunta están los **Roles**, los cuales definen el comportamiento y la responsabilidad de un individuo o equipo dentro del proyecto.
- **¿Cómo lo hace?**, la respuesta lo encontramos en las **actividades**, las cuales son unidades de trabajo elaboradas por una persona que desempeña un rol.
- **¿Qué hace?**, para responder la pregunta están los **artefactos o productos**, los cuales son los resultados del proyecto, es decir elementos que se crean y se utilizan hasta el final del proyecto.
- **¿Cuándo lo hace?**, esta pregunta se responde con los **flujos de trabajo**, los cuales cuentan la secuencia de actividades realizadas por diferentes roles y los artefactos utilizados.

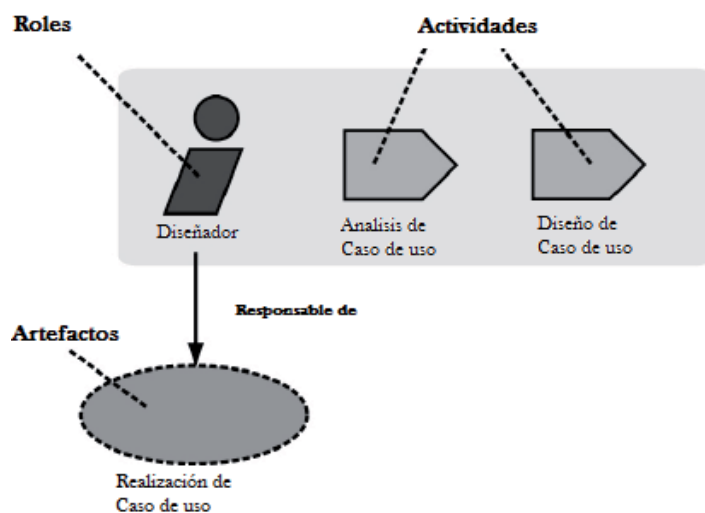


Figura 6.1 Ejemplo de relación entre roles, actividades y artefactos²²

6.1.1.1 Roles

Especifica la función que cumple un individuo o un grupo que trabaja como equipo dentro del proyecto de software, es decir definen el comportamiento y las responsabilidades de los mismos. Una persona puede desempeñar diversos roles dentro del proyecto, así como un mismo rol puede ser representado por varias personas.

²² Traducido de: Rational Unified Process , "Best Practices for Software Development Teams"
http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf,
 Fecha de Consulta: 29 de Noviembre del 2009

Según se asignen los roles, el individuo o grupo debe cumplir con tareas las cuales generaran posteriormente artefactos, también pueden existir artefactos que para su elaboración requieran de un rol. Las responsabilidades de un rol son tanto el llevar a cabo un conjunto de actividades como el ser el dueño de un conjunto de artefactos y realizar tareas comunes. Siempre es importante establecer los roles de acuerdo a las características del proyecto a desarrollar, los roles deben estar agrupados por categorías según las tareas a realizar.

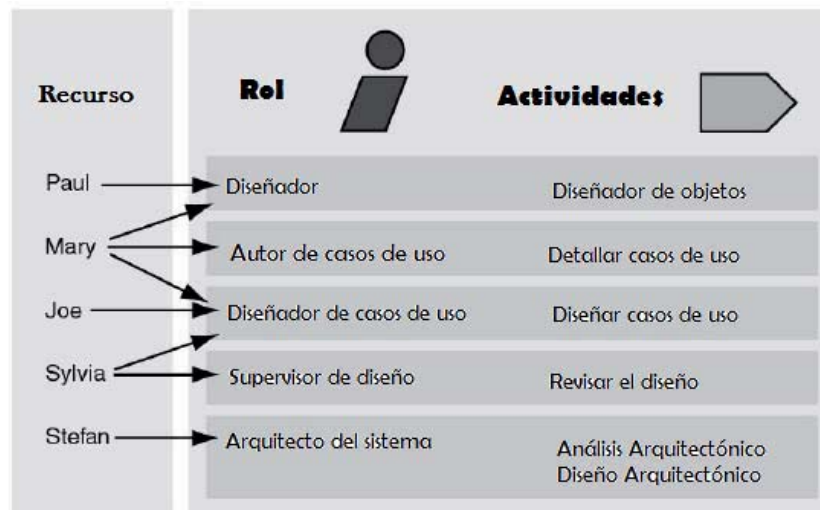


Figura 6.2 Ejemplo de la asignación de roles a personas.¹

Los Roles propuestos para esta metodología son los siguientes:

- Analistas
 - Analista de calidad
 - Analista de producto (Especificador de requerimientos)
- Desarrolladores
 - Arquitecto de software
 - Desarrollador
- Gestores
 - Mentor
 - Líder del proyecto
- Apoyo
 - Involucrados

- Pruebas
 - Probador

Los roles anteriormente listados serán detallados a continuación.

6.1.1.1.1 Roles Analistas

Los analistas cumplen un rol vital en el proceso de desarrollo, son responsables de investigar, planear, coordinar y recomendar opciones de software para cumplir los requerimientos del usuario. Un analista de sistemas para ser exitoso debe adquirir cuatro habilidades:

- **Analítica:** permiten entender a la organización y sus funciones, las cuales le ayudan a identificar oportunidades, analizar y resolver problemas.
- **Técnica:** ayudan al analista de sistemas a entender el potencial y las limitaciones de las tecnologías de la información. El analista de sistemas debe ser capaz de trabajar con varios lenguajes de programación, sistemas operativos, y plataformas hardware de computadoras.
- **Gerencial:** ayudan al analista de sistemas a administrar proyectos, recursos, riesgos, y cambio.
- **Interpersonal:** ayudan al analista de sistemas a trabajar con los usuarios finales así como con analistas, programadores, y otros profesionales de los sistemas, es decir, son un enlace entre el equipo de desarrollo y los usuarios.

Analista de Calidad: Se encarga de revisar los documentos (artefactos) que contienen información del avance y seguimiento del proyecto, además de verificar que los objetivos del marco de desarrollo se cumplan.

Analista de producto: Se encarga de la dirección del proceso de captura de requerimientos, definir los actores y casos de uso para estructurar el modelo de casos de uso, de esta forma establece la forma en que funcionará el sistema y cuáles son las restricciones del mismo.

6.1.1.1.2 Roles Desarrolladores

Su función consiste en trasladar las especificaciones del software en una aplicación, para ello es necesario primero diseñar una arquitectura la cual guíe la construcción de software describiendo en general el cómo se construirá una aplicación de software mediante el uso de diagramas.

Arquitecto de Software: Es responsable de definir la arquitectura que guiará el desarrollo y la refinación continua de la misma en cada iteración, además de definir los lineamientos generales para el diseño e implementación, para probar aspectos riesgosos desde el punto de vista técnico, debe construir prototipos. Este rol se puede descomponer en los subroles siguientes:

- Diseñador
- Diseñador de base de datos
- Diseñador de interfaz de usuario

Desarrollador: Este rol es responsable del código fuente del programa, tiene las tareas de desarrollar, depurar, mantener y documentar el código del programa, además de elaborar y ejecutar las pruebas unitarias sobre el código. Este rol se puede descomponer en los subroles siguientes:

- Implementador
- Integrador

6.1.1.1.3 Roles Gestores

Este grupo tiene la responsabilidad del planeamiento y la ejecución acertados del proyecto. Las personas en estos roles deben poseer una combinación de habilidades tales como dirigir y asignar recursos y resolver conflictos interpersonales. Una de sus tareas más importantes es el reconocimiento de los riesgos que pueden afectar el éxito del proyecto y la medición constante de dicho riesgo a lo largo del ciclo de vida del proyecto. Cada decisión tomada por los gestores de proyecto debe involucrar un beneficio directo hacia el proyecto.

Líder del proyecto: Este rol se encarga de establecer las condiciones de trabajo, dirigir y asignar recursos, coordina las interacciones con los clientes y usuarios finales, planifica las iteraciones, planifica y asigna el trabajo, define la organización del proyecto, establece las prácticas que aseguran la integridad y calidad de los artefactos del proyecto, entre otras responsabilidades. Este rol se puede descomponer en los siguientes subroles:

- Ingeniero de procesos
- Jefe de configuración
- Jefe de implantación
- Jefe de pruebas
- Revisor de gestión del proyecto

Mentor: Es una persona muy ligada con el proceso de desarrollo de software, conoce las prácticas utilizadas y él porque de su uso, apoya a los equipos de trabajo mediante la revisión de los artefactos generados y hace recomendaciones de cómo mejorarlos. También aclara dudas que puedan surgir en el desarrollo del proyecto y contribuye a que la calidad del software se mantenga. Este rol se puede descomponer en los siguientes subroles:

- Revisor técnico
- Revisor

6.1.1.1.4 Roles de Apoyo

Son personas interesadas en que sus necesidades sean satisfechas con la elaboración del software, estos roles son muy importantes para definir el alcance y los requerimientos del proyecto, entre ellos se encuentran:

- Cliente
- Cualquier rol
- Desarrollador de cursos
- Directores de usuarios

- Diseñador gráfico
- Documentador técnico
- Especialista en herramientas
- Experto del Negocio
- Usuarios

6.1.1.1.5 Pruebas

Las pruebas son parte muy importante del software, pues mediante ellas se puede comprobar el correcto funcionamiento del software, cumpliéndose así las necesidades del usuario.

Probador: En este rol están las personas que tienen que ver con la fase de pruebas del proyecto, es decir los que diseñan y ejecutan las pruebas de cada funcionalidad implementada y de su correcta integración con los demás componentes del software. Este rol se puede descomponer en los siguientes subroles:

- Analista de pruebas.
- Diseñador de pruebas.
- Especialista en Pruebas.

6.1.1.2 Equipo de Trabajo

Existen muchas consideraciones a tomar en cuenta para definir un equipo de trabajo y los roles que se asignaran a los integrantes del mismo, entre las principales están la distribución geográfica de los integrantes, su disponibilidad de tiempo, la cantidad de integrantes del equipo, el tamaño del proyecto, entre otros. Por ello según la cantidad de personas en el equipo de trabajo se puede asignar varios roles a una persona, así como un rol puede ser asignado a varias personas.

El talento humano es lo más valioso dentro de esta metodología, y si se aprovecha al máximo las capacidades y cualidades de la persona, seguramente se podrá obtener resultados de altísima calidad, por ello es muy importante que se gestione bien los recursos humanos del proyecto, siendo muy importante una estructura de trabajo, en

la que la coordinación y colaboración son esenciales. Planteamos un equipo para el desarrollo del proyecto, el cual será dividido en 3 Equipo de gestión, de desarrollo y de soporte.

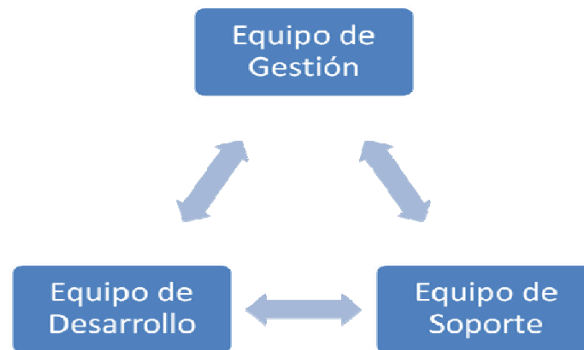


Figura 6.3 Comunicación entre los Equipos de Trabajo

Equipo de gestión: Su objetivo es establecer y mantener los lineamientos del proyecto además de controlar la calidad del mismo durante el ciclo de vida del software.

Este equipo básicamente debe contar con los siguientes roles:

- Líder del proyecto: Su función es encargarse de establecer las condiciones de trabajo, sobretodo dirigir el trabajo.
- Mentor: También aclara dudas que puedan surgir en el desarrollo del proyecto y contribuye a que la calidad del software se mantenga.
- Analista de calidad: El cual verifica la calidad de los artefactos generados, para obtener resultados óptimos.

Este equipo es esencial para poder alcanzar el éxito en el desarrollo de software, una buena gestión significa que el proyecto de software tenga alta calidad, y desarrollo continuo.

El rol líder del proyecto es el rol principal dentro de este equipo, una o varias personas comprometidas con el proyecto pueden ejercer este rol, motivadas para trabajar y con una gran visión para guiar al proyecto en la dirección correcta.

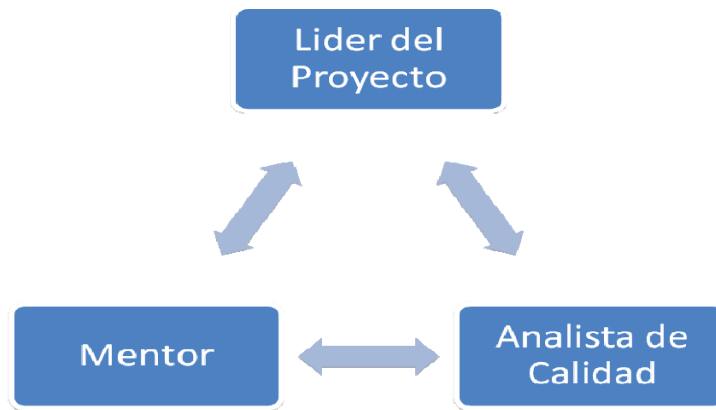


Figura 6.4. Equipo de gestión del proyecto

Equipo de desarrollo: Puede existir uno o varios equipos, estos realizan el análisis, diseño e implementación del programa, teniendo que cooperar coordinadamente y con una comunicación continua para poder maximizar la utilidad de su trabajo, pueden ser personas geográficamente cercanas o distantes.

- Arquitecto de software: El arquitecto de software es un rol vital para el éxito del desarrollo del software, un buen diseño y de alta calidad, nos permite tener un software adaptable y escalable.
- Desarrollador: Su función es desarrollar las especificaciones generadas por los diseñadores o arquitectos del software.
- Analista del producto: El objetivo de este rol es definir las funciones del software y las restricciones del mismo, partiendo de los requerimientos de los usuarios.

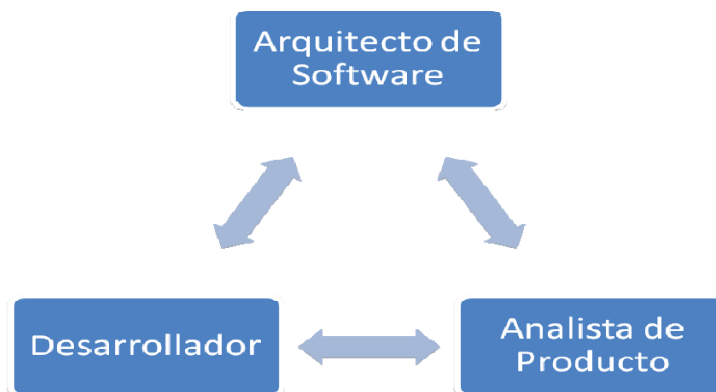


Figura 6.5. Equipo de desarrollo del proyecto

Equipo de soporte: Su función es ayudar a que el software sea de buena calidad y que sea implementado de manera correcta para que satisfaga las necesidades de los usuarios. Está formado por:

- Probador: personal de pruebas del sistema.
- Involucrados: expertos del negocio y usuarios que ayudan a comprender los requerimientos del software.

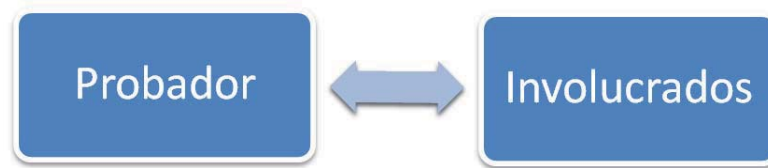


Figura 6.6. Equipo de soporte del proyecto

Estos equipos de trabajo están basados en una buena comunicación, cooperación y coordinación, y se pretende que pueda ser aplicado a proyectos con restricciones geográficas, así como posibles cambios de entrada y salida de personas.

6.1.1.3 Actividades

Una actividad es una unidad de trabajo que puede ser realizada por una persona que desempeñe un rol. Las actividades tienen un objetivo concreto, normalmente expresado en términos de crear o actualizar algún producto o artefacto tales como un modelo o una clase. Todas las actividades son asignadas a un rol específico, la duración puede ir desde pocas horas hasta días y afecta a uno o un pequeño número de artefactos.

Una actividad debería poder ser utilizada como un elemento de planificación y progreso, si esta es muy pequeña podría ser abandonada o descuidada, y si es muy larga el progreso de la misma puede ser expresada como partes de una actividad. Las actividades dentro de un proyecto de software varían según el alcance del mismo, deben plantearse solo las necesarias para el correcto desarrollo del software y en lo posible deben estar agrupadas por categorías según como se muestra a continuación:

- Modelado del negocio
 - Evaluar el estado del negocio
 - Describir el negocio actual
 - Identificar y Diseñar los procesos de negocio
 - Identificar roles y responsabilidades
 - Explorar la automatización de los procesos
- Requerimientos
 - Analizar el problema
 - Entender los requerimientos de los usuarios del sistema
 - Definir el sistema
 - Gestionar el alcance del sistema
 - Gestionar cambios requeridos
- Análisis y Diseño
 - Definir una arquitectura candidata
 - Analizar el comportamiento del sistema
 - Refinar la arquitectura del sistema
 - Diseñar servicios
 - Diseñar componentes
 - Diseñar bases de datos
- Implementación
 - Definir modelo de implementación
 - Planificar la integración
 - Implementar componentes
 - Integrar subsistemas
 - Integrar el sistema
- Pruebas
 - Definir la misión de las pruebas
 - Validar escalabilidad de componentes
 - Probar y evaluar el sistema
 - Mejorar los recursos de prueba
 - Verificar enfoque de las pruebas
- Gestión de Configuración y Cambios
 - Gestionar los Cambios de Requerimientos

- Planear la Configuración del Proyecto y el Control de Cambios
- Registrar y Almacenar los Cambios
- Gestión del proyecto
 - Concebir un Nuevo Proyecto
 - Evaluar el Alcance y los Riesgos del Proyecto
 - Planear el Proyecto
 - Gestionar Iteración
 - Reevaluar el Alcance y los Riesgos del Proyecto
 - Monitorear y Controlar el Proyecto
 - Finalización de cada Fase
 - Planificar la próxima Iteración

Estas actividades dependiendo de las características del proyecto pueden ser reducidas o aumentadas. Para cada una de ellas, al final del capítulo se incluye como sugerencia una plantilla modelo para la elaboración de los diferentes artefactos a lo largo del proyecto.

6.1.1.4 Artefactos

Un artefacto es parte de la información que es producida, modificada o usada por un proceso durante el desarrollo del software, los artefactos son piezas que hacen tangible al proyecto, van desde el código fuente del software hasta la documentación aportada por el cliente y la entregada por el equipo de desarrollo al culminar cada hito dentro del proyecto.

Los artefactos sirven de entrada para que los roles puedan realizar una actividad y también son la salida de dichas actividades realizadas. Muchos de estos artefactos pueden ser documentos, los cuales deben permanecer actualizados y consistentes durante todo el proyecto. Es importante elaborar los artefactos necesarios, el exceso de ellos puede hacer más lento y burocrático el proceso de desarrollo de software. Los artefactos que creemos convenientes realizar son los siguientes:

- Documento de Arquitectura del Software (DAS)
- Especificación de Requerimientos del Software (ERS)
- Glosario del Sistema
- Modelo de Diseño

- Plan de Implantación
- Plan de Pruebas
- Planificación del Proyecto
- Repositorio de Versiones
- Términos de Referencia para el Equipo de Desarrolladores del Sistema
- Términos de Referencia del Sistema
- Visión del Sistema
- El Sistema mismo

Además un artefacto puede estar compuesto por varios artefactos, a continuación lo ilustramos con los siguientes ejemplos:

El Sistema

- Lista de Materiales
- Artefactos de Instalación

Especificación de Requerimientos del Software (ERS)

- Modelo de Caso de Uso
- Especificaciones Suplementarias

Modelo de Análisis del Negocio

- Entidad del Negocio
- Trabajador del Negocio
- Reglas del Negocio

6.1.1.5 Flujos de Trabajo

Son una secuencia de actividades realizadas por los diferentes roles, así como la relación que existe entre los mismos. Un flujo de trabajo es una relación de actividades que producen resultados observables. Los flujos de trabajo más importantes son:

6.1.1.5.1 Modelo de negocios

Con este flujo de trabajo pretendemos llegar a un mejor entendimiento de la organización donde se va a implantar el software (denominada organización objetivo). Se busca establecer una nueva perspectiva de la organización objetivo a través de definir procesos, roles y responsabilidades de la organización por medio de un modelo de Casos de Uso del Negocio y un Modelo de Objetos del Negocio. También estos modelos con complementados con el desarrollo de otras especificaciones tales como un Glosario. Los objetivos a cumplir con este flujo de trabajo son:

- Entender la estructura y la dinámica de la organización objetivo.
- Entender el problema actual en la organización objetivo e identificar potenciales mejoras.
- Asegurar que clientes, usuarios finales y desarrolladores tengan un entendimiento común de la organización objetivo.
- Derivar los requisitos del sistema necesarios para apoyar a la organización objetivo.

6.1.1.5.2 Requerimientos

Es uno de los flujos de trabajo más importantes, pues mediante el mismo se establece las características del sistema que se va a construir, por ello es importante que los usuarios entiendan y acepten los requerimientos del sistema que serán especificados. Los requerimientos se dividen en dos.

- Los requisitos funcionales representan la funcionalidad del sistema. Se modelan mediante diagramas de Casos de Uso.
- Los requisitos no funcionales representan aquellos atributos que debe presentar el sistema, pero que no son una funcionalidad específica. Por ejemplo requisitos de facilidad de uso, fiabilidad, eficiencia, portabilidad, escalabilidad entre otros.

Para capturar los requisitos es necesario entrevistar a todos los interesados en el proyecto, no sólo a los usuarios finales, y tomar en cuenta todas sus peticiones. Siendo un requisito la facilidad de uso, se diseña la interfaz gráfica de usuario

considerando los heurísticos de usabilidad. Se lo hace a través de la construcción de prototipos de la interfaz gráfica de usuario. Los objetivos a cumplir con este flujo de trabajo son:

- Llegar a un acuerdo con los usuarios sobre las funcionalidades que tendrá el software.
- Facilitar el mejor entendimiento de los requisitos del sistema.
- Definir el ámbito del sistema.
- Poder establecer una base para la planeación de los contenidos técnicos de las iteraciones.
- Poder establecer una base para estimar costos y tiempo de desarrollo del sistema.
- Definir una interfaz de usuarios para el sistema, enfocada a las necesidades y metas del usuario.

6.1.1.5.3 Análisis y Diseño

Mediante este flujo de trabajo se pretende convertir los requerimientos del sistema a una especificación de cómo implementarlo. El análisis consiste en lo que hará el sistema, por lo tanto sólo se interesa por los requisitos funcionales. El diseño es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales, es decir cómo cumple el sistema sus objetivos.

Antes de la fase de construcción hay que definir una arquitectura candidata, la que durante esta fase se irá refinando hasta llegar a su forma definitiva, además si el sistema usa una base de datos, habrá que diseñarla también obteniendo un modelo de datos. Los objetivos a cumplir con este flujo de trabajo son:

- Convertir los requisitos al diseño del sistema.
- Desarrollar una arquitectura para el sistema.
- Adaptar el diseño para poder brindar características funcionales, tales como eficiencia, entre otras al sistema.

6.1.1.5.4 Implementación

En este flujo de trabajo se obtiene una versión ejecutable del sistema en la que se van reflejando los requerimientos del usuario según el análisis y diseño elaborado, además para las funcionalidades implementadas los desarrolladores deben realizar las pruebas de unidades.

La estructura de todos los elementos implementados forman el modelo de implementación, la integración debe ser incremental, es decir, sólo se añade un elemento a la vez. De este modo es más fácil localizar fallos y los componentes se prueban más a fondo. En fases tempranas del proceso se pueden implementar prototipos para reducir los riesgos sobre el proyecto. Los objetivos a cumplir con este flujo de trabajo son:

- Planificar qué subsistemas deben ser implementados y en qué orden deben ser integrados, obteniendo así un plan de Integración.
- Decidir en qué orden se implementan los elementos del subsistema.
- Notificar si encuentra errores de diseño.
- Probar los subsistemas individualmente, mediante las pruebas de unidades.
- Integrar el sistema.

6.1.1.5.5 Pruebas

Este flujo de trabajo es el encargado de evaluar la calidad del software que se está desarrollando y si cumple las necesidades y expectativas del usuario, por ello es importante que esté presente en todo el ciclo de vida del software y no solo al final. Este flujo de trabajo consiste en:

- Planificar que es lo que hay que probar.
- Diseñar cómo se va a probar.
- Implementar lo necesario para realizar las pruebas.

- Ejecutarlas en los niveles necesarios.
- Refinar el software, mediante la interpretación de los resultados obtenidos.

Los objetivos a cumplir con este flujo de trabajo son:

- Establecer criterios de aceptación
- Encontrar y documentar defectos del software.
- Verificar que el software cumpla con los diseños.
- Verificar que los requerimientos del usuario tengan una apropiada implementación.
- Realizar la validación de los supuestos realizados en el diseño y especificación de requisitos por medio de demostraciones concretas.
- Probar el rendimiento y calidad del software.

6.1.1.5.6 Producción

Este flujo de trabajo se da en la etapa de liberación antes de que entre a operar el nuevo sistema, su principal objetivo es asegurar su adaptación y aceptación sin complicaciones, sobre todo por parte de los usuarios.

Su ejecución inicia en fases anteriores para tener menos resistencia por parte de los usuarios y solventar problemas no previstos, se lo realiza mediante actividades de planificación y la elaboración del manual de usuario y tutoriales. Este flujo de trabajo incluye las siguientes actividades:

- Probar el producto en su entorno operativo real.
- Empaquetar el software para su distribución.
- Distribuir el software.
- Instalar el software.
- Proveer asistencia y ayuda a los usuarios.

- Formar a los usuarios.
- Migrar la información existente.

6.1.1.5.7 Gestión del proyecto

Este flujo de trabajo busca organizar y administrar objetivos, riesgos y restricciones para conseguir desarrollar un software de acuerdo a las necesidades del usuario, estos deben realizarse en dos niveles, uno para cada iteración y otro para las fases del proyecto. Los objetivos a cumplir con este flujo de trabajo son:

- Tener un marco de trabajo para la gestión del proyecto de software.
- Tener guías prácticas para la planeación, ejecución y monitoreo del proyecto.
- Tener un marco de trabajo para gestionar riesgos.

6.1.1.5.8 Gestión y Configuración de Cambios

Este flujo de trabajo pretende realizar tareas de tal manera que los artefactos y los documentos generados se mantengan íntegros, y que también se pueda ir registrando como ha sido la evolución del sistema en las diferentes iteraciones. Los objetivos a cumplir con este flujo de trabajo son:

- Tener un plan de gestión de configuración de cambios
- Poseer un repositorio de versiones.

6.1.1.5.9 Gestión del Entorno

Este flujo de trabajo da soporte al proyecto mediante herramientas, procesos y métodos adecuados, tanto en la preparación del entorno del proyecto, así como en la preparación del ambiente de cada iteración. Los objetivos a cumplir con este flujo de trabajo son:

- Tener herramientas adecuadas para el desarrollo del proyecto.
- Tener lineamientos para el desarrollo del proyecto.
- Tener plantillas para la elaboración del proyecto.

6.1.2 Dimensión Dinámica de la Metodología Propuesta

La dimensión dinámica de la metodología permite apreciar los aspectos dinámicos del proceso de desarrollo expresados en términos de fases y ciclos de vida de un proyecto, cada ciclo termina con la generación de un producto y consta de seis fases, el ciclo puede terminar en una o más iteraciones dependiendo de la complejidad del proyecto. Para pasar de una fase a otra siguiente, es necesario cumplir con todas las metas clave e hitos de control que determinan el avance de dicha fase y permiten planear su continuidad, de esto se encarga el equipo gestor del proyecto quién realiza una evaluación para determinar si los objetivos se cumplieron.

Generar una versión de un producto implica pasar por todas las fases de desarrollo indicadas en la figura 6.7, comenzando por la fase de Análisis del Dominio de la Aplicación para luego terminar en la fase de Liberación, en el caso de que la versión no supere las pruebas de funcionalidad no se puede dar por concluida la iteración.

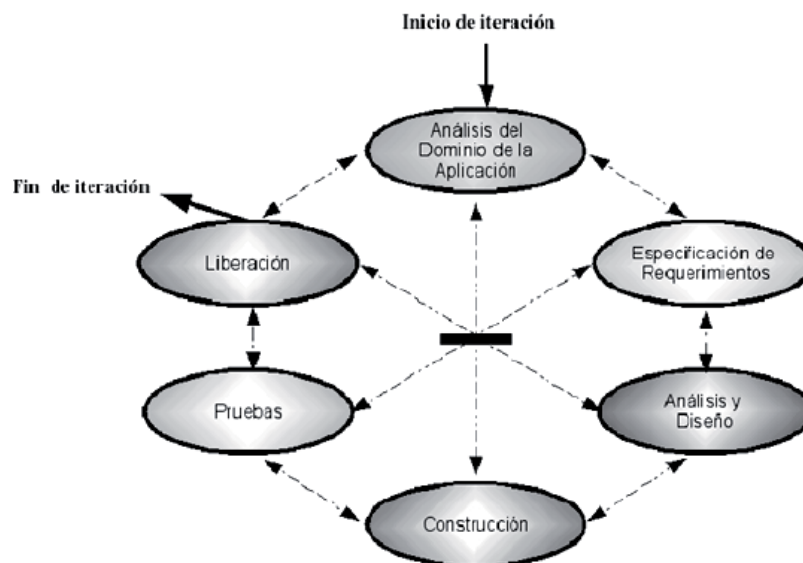


Figura 6.7 Ciclo de Vida de la Metodología Propuesta

6.1.2.1 Fase de Análisis del Dominio de la Aplicación

Es la primera fase del ciclo de vida y en ella se analiza el dominio o contexto en el cual operará la aplicación a desarrollarse, su alcance organizacional y los problemas actuales que requieren de una solución tecnológica, es importante que el equipo de trabajo involucrado en el proyecto conozca, entienda y analice el funcionamiento de todos los procesos del dominio de la aplicación, para ello puede basarse en la

revisión de documentos relacionados a los procesos que se requieren automatizar o incluso realizar entrevistas con usuarios que conozcan los procesos de negocio más a fondo; esto asegurará la calidad del producto y el éxito esperado en el desarrollo.

De esta manera se definirá el modelo de negocio y el alcance del proyecto estableciendo sus límites y escenarios básicos que definen la funcionalidad. Como resultado de esta fase se obtendrá un documento de estudio del negocio cuya plantilla de diseño se encuentra en el apartado 6.3.1.

6.1.2.2 Fase de Especificación de Requerimientos

En esta segunda fase del desarrollo se especifican a detalle las restricciones operativas del producto a desarrollar, estas restricciones reflejan las necesidades actuales del modelo del negocio que el sistema pretende resolver. Los requerimientos pueden ser funcionales o no funcionales, los funcionales describen lo que el sistema debe y no debe hacer, indicando sus entradas, salidas y funciones específicas que proporciona el sistema; y los no funcionales incluyen restricciones de tiempo, restricciones en los procesos de desarrollo y demás propiedades que surgen en torno al proyecto como la fiabilidad, costos, tiempo de respuesta, escalabilidad, rendimiento y disponibilidad del sistema.

Es importante tomar en cuenta todos los requerimientos del proyecto, ya que un incumplimiento de alguno de estos acarrearía a un posible fracaso y la inutilización del sistema. Como resultado de esta fase se obtendrá un documento de requerimientos cuya plantilla de diseño se encuentra en el apartado 6.3.2, el cual irá evolucionando con el desarrollo de cada iteración.

6.1.2.3 Fase de Diseño

En esta fase se traducen todos los requerimientos de la fase anterior a una especificación de diseño en el que se construyen modelos del sistema con diferentes niveles de abstracción en los cuales se vea reflejada la estructura del software que se va a implementar en la siguiente etapa, todos estos representan las vistas arquitectónicas de la aplicación, los mismos que se irán enriqueciendo en cada iteración según se vayan añadiendo funcionalidades al sistema; este prototipo de la arquitectura contiene en detalle los casos de uso, diagramas de clases, de secuencia, de estados y Modelo E-R, utilizando el UML como lenguaje de modelado. Esta etapa

finaliza con la obtención de una línea base de la arquitectura del sistema debidamente documentada. Como resultado de esta fase se obtendrá un documento de análisis y diseño cuya plantilla de diseño se encuentra en el apartado 6.3.3.

6.1.2.4 Fase de Construcción

Esta fase comprende el desarrollo operacional del producto de forma incremental a través de sucesivas iteraciones en el que se integran e implementan todas las características, componentes, clases, objetos y ejecutables, obteniendo una versión aceptable del producto comúnmente denominada versión beta; en cada iteración se refina la interfaz del usuario, base de datos y demás funcionalidades de la aplicación.

Como resultado de esta fase se obtendrá un documento de implementación del proyecto cuya plantilla de diseño se encuentra en el apartado 6.3.4, en el cual se identifican las herramientas a utilizar, lenguajes de programación, el sistema gestor de bases de datos, los módulos o subsistemas y la forma de integrarlos de manera que en conjunto cumplan los estamentos establecidos en la fase de especificación de requerimientos.

Un aspecto importante a tener en cuenta es que esta fase debe desarrollarse en base a las seis mejores prácticas descritas en la metodología RUP de la sección 5.3.4 quienes con su uso permiten reducir los riesgos del proyecto y así tener un sistema ejecutable en el menor tiempo descritos en una o más versiones ejecutables.

6.1.2.5 Fase de Pruebas

El objetivo de esta fase es eliminar las falencias existentes en la aplicación de manera que se elaboran y aplican pruebas unitarias y de integración, funcionales y no funcionales en las que se garantiza el correcto funcionamiento de la aplicación y la satisfacción del cliente. La detección temprana de errores o incompatibilidad del código permite mejorar la calidad del sistema, para ello se debe contar con datos de prueba, resultados esperados y actores participantes (ver la plantilla de diseño del documento de pruebas en el apartado 6.3.5).

Las pruebas unitarias son diseñadas y aplicadas por los desarrolladores conforme se escribe el código, mientras que las pruebas de integración funcionales y no funcionales son diseñadas y aplicadas por personal especial designado al proyecto;

todos los errores detectados deben ser reportados en un documento a los encargados del levantamiento de requisitos del proyecto a que procedan a realizar las correcciones del caso.

6.1.2.6 Fase de Liberación

Es la última fase del ciclo de vida y comprende la liberación tanto de las versiones de prueba como las versiones estables de la aplicación, las primeras se liberan con la finalidad de dar a conocer el producto a la comunidad para que así puedan validarlo y en muchos de los casos mejorarlo, así la comunidad participa en las pruebas y detección de errores a través de foros en la web u otros medios.

Una vez reportados y corregidos los errores presentes en estas primeras versiones, se liberan finalmente las versiones estables de la aplicación, se entrega el producto funcional a los usuarios finales y se los entrena en el manejo del sistema; es importante entregar al usuario la documentación del sistema en el que se especifiquen tareas como configuración, instalación y uso del producto, con ello se garantiza que el usuario aprenda a operar y mantener el sistema y que el producto final cumpla con los requerimientos establecidos.

Ahora bien, el usuario de Software Libre, es libre de poder acceder, usar y modificar el código fuente del software, sin embargo es muy importante tener en cuenta algunos aspectos legales que estas prácticas conllevan, por ello en esta fase se debe incluir el tipo de licencia de Software Libre que regirá en el producto, para conocer la más adecuada, referirse al capítulo tres de “Licencias del Software Libre”.

6.2 Como utilizar ésta Metodología en un Proyecto Pequeño

A continuación explicaremos una propuesta para desarrollar software orientada a proyectos pequeños y medianos, la misma está basada en los puntos anteriores de este capítulo, la cual será utilizada para la elaboración de un caso práctico de modo de poder evaluar su eficacia.

6.2.1 Equipos de Trabajo y Roles

Proponemos la utilización de equipos de trabajos, los cuales deben tener una buena y continua comunicación tanto en el mismo equipo, como entre diferentes equipos. Los equipos de trabajo propuestos son:

6.2.1.1 Equipo de Gestión

Está compuesto por:

- **Líder del Proyecto:** este rol es responsable de establecer las condiciones de trabajo, dirigir y asignar recursos, planifica las iteraciones, planifica y asigna el trabajo, define la organización del proyecto, establece las prácticas que aseguran la integridad y calidad de los artefactos del proyecto.
- **Analista de Calidad:** Este rol es responsable de revisar los artefactos y documentación generada en el desarrollo del proyecto. Su objetivo es verificar la calidad de los mismos, antes de que pasen a formar parte del proyecto.

(Véase Literal 6.1.1.1.3 Roles Gestores)

6.2.1.2 Equipo de desarrollo

Está compuesto por:

- **Arquitecto de software:** Es responsable de definir la arquitectura que guiará el desarrollo, la refinación continua de la misma en cada iteración. además de definir los lineamientos generales para el diseño e implementación.
- **Desarrollador:** Este rol es responsable del código fuente del programa, tiene las tareas de desarrollar, depurar, mantener y documentar el código del programa, además de elaborar y ejecutar las pruebas unitarias sobre el código.

(Véase Literal 6.1.1.1.2 Roles Desarrolladores)

6.2.1.3 Equipo de Soporte

Está compuesto por:

- **Probadores:** Su función es validar el software generado, a través de la ejecución de diferentes planes de prueba y el control de su respectiva documentación.

- **Involucrados:** Su función es ayudar a que el software sea de buena calidad y que sea implementado de manera correcta para que satisfaga las necesidades de los usuarios.

(Véase Literal 6.1.1.1.4 Roles de Apoyo)

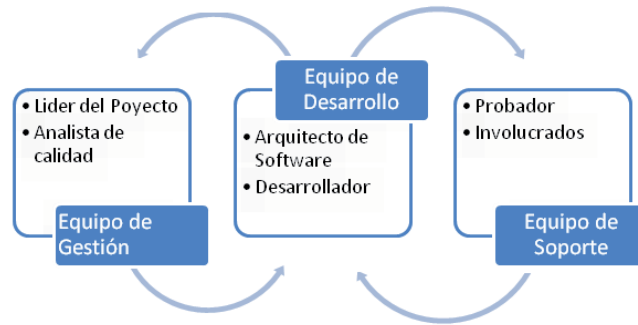


Figura 6.8 Equipos de Trabajo y Roles presentes en un proyecto pequeño

6.2.2 Flujos de Trabajo y Actividades

Luego de conformar los equipos de trabajos y asignar los roles a las personas que participan del proyecto de software, es necesario definir las tareas asignadas a cada equipo de trabajo, las cuales están agrupadas en categorías llamadas flujos de trabajo. A continuación enumeraremos las actividades que componen los principales flujos de trabajo:

6.2.2.1 Modelo de Negocios

Con este flujo de trabajo pretendemos llegar a un mejor entendimiento de la organización donde se va a implantar el software. Se busca establecer una nueva perspectiva de la organización objetivo a través de definir procesos, roles y responsabilidades de la organización por medio de un modelo de Casos de Uso del negocio. Las actividades que componen este flujo de trabajo son:

- Evaluar el estado del negocio: Analizar la estructura y la dinámica de la entidad en la que se va a implementar el software.
- Describir el negocio actual: Definir el entorno en el que labora la organización y cuál es su campo de acción.

- Identificar procesos de negocio: Muestra los procesos del negocio y quienes intervienen en los mismos.
- Diseñar como se realizan los procesos del negocio: Realizar abstracciones de cómo son los procesos del negocio para poderlos comprender de mejor manera y prepararlos para ser automatizados planteando posibles alternativas de automatización.
- Identificar roles y responsabilidades: Identificar los roles que actúan dentro de la organización y de que actividades son responsables, por Ejemplo Rol vendedor, su responsabilidad es emitir comprobantes de venta, etc.

6.2.2.2 Requerimientos

Mediante este flujo de trabajo se establece las características del sistema que se va a construir. Para capturar los requisitos es necesario entrevistar a todos los interesados en el proyecto, no sólo a los usuarios finales y tomar en cuenta todas sus peticiones. Siendo un requisito a cumplir la facilidad de uso. Las actividades que componen este flujo de trabajo son:

- Analizar el problema: Formula los problemas de los usuarios, y plantea posibles alternativas para solucionarlos.
- Entender los requerimientos de los usuarios del sistema: Analizar las necesidades del usuario y sus expectativas de cómo serán satisfechas.
- Definir el sistema. Se debe definir los requerimientos funcionales y no funcionales del sistema.
- Gestionar el alcance del sistema: Ponerse de acuerdo con los usuarios sobre las tareas que realizara el sistema.
- Gestionar cambios requeridos: Administrar los cambios de requerimientos que se presenten mientras se desarrolla el proyecto.

6.2.2.3 Análisis y Diseño

El objetivo principal de esta fase es la de convertir los requerimientos del sistema a una especificación de cómo implementarlos. El análisis consiste en lo que hará el sistema, por lo tanto sólo se interesa por los requisitos funcionales. El diseño es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales, es decir cómo cumple el sistema sus objetivos. Las actividades que componen este flujo de trabajo son:

- Definir una arquitectura candidata: Definir si el sistema será Clientes/Servidor, a 3 Capas, Monolítico, etc. Y por qué.
- Analizar el comportamiento del sistema: Cual será el comportamiento esperado del sistema en su entorno y su reacción a eventos externos.
- Refinar la arquitectura del sistema: Si es necesario cambiar detalles en la arquitectura del sistema, según avance el proyecto.
- Diseñar componentes: Como se construirán los componentes del sistema.
- Diseñar bases de datos: Crear un soporte adecuado para que la información del sistema satisfaga las necesidades del usuario de manera consistente.

Los modelos que se deben generar son los que el equipo de análisis considere necesarios para poder construir correctamente el sistema, nosotros sugerimos los siguientes:

- Modelos de Casos de uso
- Diagramas de clases (En caso de programación orientada a objetos)
- Diagramas de estados
- Diagramas de Actividades
- Diagramas de Secuencia
- Diagrama de Componentes
- Diagrama de Despliegue
- Diagrama de Base de datos

6.2.2.4 Implementación

En este flujo de trabajo se obtiene una versión ejecutable del sistema, además para las funcionalidades implementadas los desarrolladores deben realizar las pruebas de unidades. Las actividades que componen este flujo de trabajo son:

- Definir modelo de implementación: Este modelo debe guiar a los programadores para desarrollar el software, con indicaciones tales como que el uso de estándares para escribir código, estandarización de interfaces, entre otras.
- Planificar la integración: En el desarrollo de componentes se debe hacer pensando en su comunicación con otros, por ejemplo es utilizando estándares, que faciliten su posterior integración.
- Implementar componentes: Construir los módulos y componentes del sistema.
- Integrar el sistema: Juntar todos los módulos y partes en un solo sistema.

6.2.2.5 Pruebas

Este flujo de trabajo es el encargado de evaluar la calidad del software que se está desarrollando y si cumple las necesidades y expectativas del usuario, por ello es importante que esté presente en todo el ciclo de vida del software, tales como las pruebas de unidades y no solo al final. Las actividades que componen este flujo de trabajo son:

- Definir la misión de las pruebas: Cuales son los objetivos de las pruebas.
- Validar escalabilidad de componentes: La facilidad de los componentes para seguir creciendo en funcionalidades, según sea necesario.
- Probar y evaluar el sistema: Demostrar que el sistema haga las tareas para las que fue construido de manera correcta.

6.2.2.6 Gestión de Configuración y Cambios

Este flujo de trabajo pretende realizar tareas de tal manera que los artefactos y los documentos generados se mantengan íntegros, y que también se pueda ir registrando como ha sido la evolución del sistema en las diferentes iteraciones. Las actividades que componen este flujo de trabajo son:

- Gestionar los Cambios de Requerimientos
- Registrar y Almacenar los Cambios

6.2.2.7 Gestión del Proyecto

Este flujo de trabajo busca organizar y administrar objetivos, riesgos y restricciones para conseguir desarrollar un software de acuerdo a las necesidades del usuario, estos deben realizarse en dos niveles, uno para cada iteración y otro para las fases del proyecto.

- Planear el Proyecto: Como se realizara el proyecto, calcular tiempo y recursos necesarios.
- Gestionar Iteración: Planear y definir el alcance de cada iteración.
- Monitorear y Controlar el Proyecto: Mediante informes de actividades, reuniones, y agendas diarias.

6.2.3 Artefactos

Se refieren a los resultados de cada actividad de los diferentes flujos de trabajo, pueden ser documentación, código fuente, diagramas etc. Los principales artefactos con los que debe contar el sistema son:

- **Documento de Estudio del negocio:** Es el resultado del flujo de trabajo de modelado del negocio.
- **Documento de Requerimientos:** Es el resultado del flujo de trabajo de requerimientos, en el mismo se detallan los requerimientos funcionales y no funcionales del sistema.

- **Documento de Análisis y Diseño:** Es el resultado del flujo de trabajo de Análisis y diseño, contiene información sobre los diferentes modelos del sistema, tales como casos de uso, diagrama de bases de datos entre otros.
- **Artefactos de Implementación:** Es el resultado del flujo de trabajo de implementación, puede ser el código fuente del software, además de documentación sobre cómo fue implementado e integrado los diferentes componentes de software.
- **Documento de pruebas:** Es el resultado del flujo de trabajo de pruebas, puede ser el plan de pruebas del sistema, es importante especificar que se probó, Como se probó, y cuál fue el resultado.
- **Documento de Control de cambios:** Aquí se registraran los diferentes cambios que sufra el software en su ciclo de vida, es importante contar con un repositorio de versiones a fin de tener un mejor control sobre las mismas.
- **Artefactos de gestión del proyecto:** Son diferentes documentos sobre los cuales se debe contener información de la planeación del proyecto, tales como cronograma de actividades, asignación de tareas a los deferentes roles, entre otros.

6.3 Plantillas para la Elaboración de los diferentes Artefactos del Sistema

6.3.1 Plantilla para Elaborar un Documento de Estudio del Negocio

EMPRESA “XYZ” DOCUMENTO DE ESTUDIO DEL NEGOCIO
<p>Fecha de elaboración: [fecha]</p> <p>Responsable: [Rol responsable de la tarea y persona responsable del artefacto]</p> <p>Colaboradores: [Nombres de roles y personas que colaboran con la tarea]</p>
<p>1. Evaluación del estado actual del negocio [Describa el estado actual del negocio y cuáles son las necesidades que se pueden superar al realizar el software]</p> <p>2. Detalle del negocio actual [Cuales son las actividades que realiza el negocio, como las realiza]</p> <p>3. Identificar y Diseñar los procesos de negocio [Cuales son los procesos que se ejecutan dentro del negocio, proponer abstracciones de cómo es el proceso]</p> <p>4. Identificar roles y responsabilidades [Identificar a las personas responsables de los diferentes procesos, que tareas realizan y asignarles roles.]</p> <p>5. Explorar la automatización de los Procesos [Se describen los procesos de negocio que se van a automatizar en el sistema, definiendo los módulos que integrarán las bases de datos.]</p>

6.3.2 Plantilla para Elaborar un Documento de Requerimientos

EMPRESA “XYZ” DOCUMENTO DE REQUERIMIENTOS
<p>Fecha de elaboración: [fecha]</p> <p>Responsable: [Rol responsable de la tarea y persona responsable del artefacto]</p> <p>Colaboradores: [Nombres de roles y personas que colaboran con la tarea]</p> <p>Versión: [X.Y Dos dígitos para indicar la versión del documento]</p> <p><i>[En caso de cambios de requerimientos, solo realizar las tareas que sufran cambios respecto a la versión anterior, por ejemplo si el requerimiento no cambia el problema, no es necesario volverlo a analizar].</i></p>
<p>1. Analizar el problema [Formular los problemas de los usuarios, y plantear posibles alternativas para solucionarlos.]</p> <p>2. Entender los requerimientos de los usuarios del sistema [Analizar las necesidades del usuario y sus expectativas de cómo serán satisfechas.]</p> <p>3. Definir el sistema [Se debe definir los requerimientos funcionales y no funcionales del sistema]</p> <p>3.1 Requerimientos funcionales del sistema [Detallar funcionalidades específicas del sistema, como por ejemplo: requerimientos de cálculos, manipulación de datos, etc.]</p> <p>3.2 Requerimientos no funcionales del sistema [Detallar requerimientos sobre cómo será la operación del sistema, como por ejemplo: Escalabilidad, portabilidad, etc.]</p> <p>4. Alcance del sistema [Detallar cuáles requerimientos del usuario serán implementados y cuáles no]</p>

6.3.3 Plantilla para Elaborar un Documento de Análisis y Diseño

EMPRESA “XYZ” DOCUMENTO DE ANÁLISIS Y DISEÑO
<p>Fecha de elaboración: [fecha]</p> <p>Responsable: [Rol responsable de la tarea y persona responsable del artefacto]</p> <p>Colaboradores: [Nombres de roles y personas que colaboran con la tarea]</p> <p>Versión: [X.Y Dos dígitos para indicar la versión del documento]</p> <p><i>[En caso de cambios de requerimientos, solo realizar las tareas que sufran cambios respecto a la versión anterior, por ejemplo si el requerimiento no cambia la arquitectura candidata del sistema, no es necesario volverla a definir]</i></p>
<p>1. Definir una arquitectura candidata</p> <p>[Describir diferentes aspectos del software, que lleven a elegir una arquitectura que podría ser implementada en el proyecto de software. Por ejemplo Monolítica, Cliente servidor, Programación por capas, Orientada a servicios, etc. Definir de manera abstracta los componentes que llevan a cabo alguna tarea de computación.]</p> <p>2. Analizar el comportamiento del sistema</p> <p>[Describir la manera de comportamiento del sistema de acuerdo a su entorno y eventos que controle, esto mediante casos de uso, actores, flujos del sistema, condiciones y salidas]</p> <p>3. Refinar la arquitectura del sistema</p> <p>[En caso de ser necesario cambios en la arquitectura del sistema, se debe refinarla, describiendo los cambios que se realizaran, y guardando las versiones anteriores.]</p> <p>4. Diseño del sistema y sus componentes</p> <p>[Detallar mediante el uso de diagramas UML como se realizaran los procesos del</p>

sistema mediante los siguientes diagramas:

- Diagrama de Clases
- Diagramas de Estados
- Diagramas de Actividades
- Diagramas de Secuencia
- Diagramas de Componentes
- Diagramas de Despliegue

]

5. Diseñar bases de datos

[Incluir diagrama de base de datos con nomenclatura estándar ya sea Entidad-Relación, Case Method u otro estándar.

Es necesario que estos den soporte adecuado para la información que manipulara el sistema, el diseño debe estar orientado a mantener la consistencia de los datos.]

6.3.4 Plantilla para Elaborar un Documento de Implementación del Proyecto

EMPRESA “XYZ”

DOCUMENTO DE IMPLEMENTACIÓN DEL PROYECTO

Fecha de elaboración: [fecha]

Responsable: [Rol responsable de la tarea y persona responsable del artefacto]

Colaboradores: [Nombres de roles y personas que colaboran con la tarea]

Versión: [X.Y Dos dígitos para indicar la versión del documento]

[Realizar las tareas que competan, según la versión del software, por ejemplo la integración de componentes será cuando estén terminados los mismos, mientras que

el modelo de implementación se define antes de la construcción de los componentes.]

1. Definir modelo de implementación

[Se debe especificar:

- Lenguaje de Programación.
- Sistema Gestor de Base de Datos.
- Uso de normas de estandarización, en caso de estandarización personalizada, especificar las normas a cumplir, por ejemplo como escribir un comentario, como nombrar las variables, Diseño de interfaces graficas etc.
- Herramientas a utilizar, por ejemplo IDE's, Administradores de Base de datos, etc.]

2. Planificar la integración

[Describir como se integraran los módulos o componentes del sistema]

3. Implementar componentes

4. Integrar componentes

6.3.5 Plantilla para Elaborar un Documento de Pruebas del Proyecto

EMPRESA "XYZ"

DOCUMENTOS DE PRUEBAS DEL SISTEMA

Fecha de elaboración: [fecha]

Responsable: [Rol responsable de la tarea y persona responsable del artefacto]

Colaboradores: [Nombres de roles y personas que colaboran con la tarea]

Versión: [X.Y Dos dígitos para indicar la versión del documento de Pruebas]

[Realizar las pruebas que competan según la versión del software.]

1. Definir la misión de las pruebas

[Se describe la finalidad con la cual es creado el plan de pruebas, por ejemplo: Comprobar el correcto desempeño del modulo de compras del sistema.....]

2. Validar escalabilidad de componentes

[Describir si el software sometido a prueba tiene la capacidad de crecer o adaptarse a nuevos requerimientos, conforme cambien las necesidades del sistema. Explicar el porqué de la escalabilidad o la no escalabilidad del sistema.]

3. Probar y evaluar el sistema

[Se deberá detallar las pruebas a las que es sometido el software]

Prueba: [Nombre de la Prueba y una breve descripción de la misma]
Resultado Obtenido: [Detallar la salida del sistema al someterle a la prueba]
Resultado Esperado: [Indicar la salida esperada que debe dar sistema]
Estado de la prueba: [Indicar si al prueba fue o no superada]
Posibles Soluciones: [Indicar las posibles soluciones para corregir el Error]

6.3.6 Plantilla para Elaborar un Documento de Control de Cambios

EMPRESA “XYZ”
DOCUMENTO DE CONTROL DE CAMBIOS
Fecha de elaboración: [fecha]
Responsable: [Rol responsable de la tarea y persona responsable del artefacto]
Colaboradores: [Nombres de roles y personas que colaboran con la tarea]
Versión: [X.Y Dos dígitos para indicar la versión del documento]
<i>[Realizar las tareas que competan, ya sea por cambio de requerimientos de usuario, o por cambio de versión de software.]</i>

1. Cambios de requerimientos de Usuario

[Registrar cambios en requerimientos tanto funcionales, como no funcionales, la fecha en la que surge el nuevo requerimiento, el usuario que pide el cambio de requerimiento, y si es factible cambiarlo.]

2. Control de versiones

[Se debe contar con un repositorio de versiones. Sugerimos utilizar la nomenclatura x.y para las diferentes versiones. Se deberá cambiar el primer dígito de la nomenclatura cuando el sistema sufra una modificación grande.

Se deberá cambiar el segundo dígito de la nomenclatura cuando el sistema sufra una modificación leve. Se debe iniciar desde 0.y, con lo que se especifica que la versión aun no esta lista o no cumple con los requerimientos mínimos.

Se deberá registrar el cambio de versión, indicando las funcionalidades agregadas, corregidas o eliminadas de la versión de la cual se partió.]

6.3.7 Plantilla para Elaborar un Documento de Gestión del Proyecto

EMPRESA “XYZ” DOCUMENTOS DE GESTIÓN DEL PROYECTO
Fecha de elaboración: [fecha]
Responsable: [Rol responsable de la tarea y persona responsable del artefacto]
Colaboradores: [Nombres de roles y personas que colaboran con la tarea]
1. Concebir un Nuevo Proyecto [Detallar el por qué del proyecto y cuáles son sus objetivos]

2. Evaluar el Alcance y los Riesgos del Proyecto

[Detallar lo que se hará y no se hará en el proyecto e identificar los potenciales riesgos de su elaboración]

3. Planear el Proyecto

[Planear como se realizara el proyecto, estimando tiempos, recursos, etc. Se puede utilizar Diagramas de Grant, de asignación de recursos, carga de recursos, etc.]

4. Gestionar Iteración

[Planear tiempos de ejecución de cada iteración y los recursos necesarios para llevarlas a cabo]

5. Reevaluar el Alcance y los Riesgos del Proyecto

[Cuando surjan cambios dentro del proyecto, describir como estos pueden cambiar el alcance y que riesgos conllevaría implementarlos]

6. Monitorear y Controlar el Proyecto

[Establecer parámetros para poder medir como se desarrolla el proyecto, planear reuniones y revisiones periódicas.]

7. Finalización de cada Fase

[Detallar los resultados obtenidos al fin de cada iteración, las dificultades que se presentaron, y hacer una evaluación del proyecto]

8. Planificar la Próxima Iteración

[Detallar las actividades que se ejecutaran en la siguiente iteración y los recursos necesarios para la misma.]

CAPITULO VII

ELABORACION DEL CASO PRÁCTICO

Introducción

El proyecto a desarrollar a continuación tiene como objeto implementar un software en el que se gestione de manera práctica las compras y ventas, dos procesos esenciales dentro de pequeños y medianos negocios de la actualidad, y que además brinde asistencia en el manejo de impuestos llevando un registro mensual de ingresos y egresos conforme se vayan dando las transacciones diarias.

En muchas empresas pequeñas estos procesos todavía se los lleva de forma manual, impidiéndoles crecer y competir en el mercado empresarial, por ello pretendemos brindar a la comunidad una herramienta tecnológica que sirva de soporte y agilice sus procesos básicos. Dentro de las funcionalidades del software destacamos la facilidad de listar, modificar, eliminar y actualizar información de clientes, proveedores y productos, así mismo registrar los diferentes movimientos en los procesos de compras, ventas y cálculo de valores correspondientes a impuestos.

El proyecto se realizará aplicando la metodología definida en el capítulo seis usando herramientas de desarrollo de Software Libre y se lo validará al aplicarlo en la empresa de venta de Repuestos “Ferrorepuestos JW”.

7.1 Documento de Estudio del Negocio

FERROREPUESTOS “JW” DOCUMENTO DE ESTUDIO DEL NEGOCIO
Fecha de elaboración: 15 de Diciembre del 2009 Responsables: Andrés Argudo; William Astudillo Colaboradores:

1. Evaluación del Estado actual del Negocio

Ferrorepuestos “JW” lleva en funcionamiento seis años, durante este tiempo todos los procesos tanto para la compra y venta de mercaderías se han realizado de manera rudimentaria, utilizando como único medio el papel para el almacenaje de dichos movimientos como documentos históricos, ahora bien, el negocio ha crecido y la lista de clientes que adquieren sus productos se ha multiplicado y la forma análoga de llevar sus transacciones ya no brinda el mismo soporte que antes, por ello es necesario realizar un cambio al negocio e integrar soluciones tecnológicas que permitan un manejo y administración más eficiente del mismo.

Este cambio implica una reingeniería de procesos en el que se debe conocer a fondo el manejo actual de los procesos para automatizarlos de forma adecuada, de manera que no altere el normal funcionamiento del negocio y que el usuario se familiarice rápidamente con la aplicación.

2. Descripción del Negocio Actual

El negocio actual ofrece a sus clientes una gama completa en repuestos genuinos para carros en todas las marcas existentes en el mercado, de esta manera el cliente adquiere un repuesto, se le da una factura y se almacena una copia de la misma en una carpeta de documentos históricos para las respectivas cuentas mensuales que consisten en una suma total de todas ellas para el cálculo de ingresos y de los impuestos respectivos.

Para la adquisición de mercaderías el proceso es similar, se obtiene una factura del proveedor que pasa a almacenarse en una carpeta de compras y al final del mes se suman todas y se calcula el total de egresos del negocio.

3. Identificación y Diseño de los procesos del Negocio

Para los procesos de compras y ventas se cuenta con un documento de inventario en el que consta un listado de todos los productos existentes, así este se va modificando según el proceso que se realice. De forma análoga, para realizar una venta, el vendedor primero verifica que se tenga el producto en stock, consulta el precio y

realiza la venta.

Para las compras el administrador del local revisa el documento de inventario y consulta el número de productos existentes en el local, dependiendo de este número realiza el pedido al proveedor correspondiente.

Con respecto al manejo de proveedores, el local cuenta únicamente con un tarjetero en el que constan un informativo de direcciones, números de teléfono y nombres de los proveedores.

Los impuestos son calculados de forma manual mediante un cálculo entre todo el resumen de las compras y ventas del mes, así se obtiene un valor que va incluido en el formulario 104 para contribuyentes que realizan ventas con tarifa 12% al SRI.

4. Identificación de Roles y Responsabilidades

Los roles existentes actualmente en el negocio son dos, el vendedor quien tiene contacto directo con los clientes y es el que despacha los productos y además notifica al administrador cuando un producto está cerca a terminarse.

El administrador quien se encarga de la parte gerencial del negocio realizando actividades como pedidos, pago de impuestos y la administración general del local.

5. Exploración de la automatización de los procesos

Para dar soporte adecuado al local se implementará una base datos de clientes, proveedores y productos los cuales se integrarán con los procesos de compras y ventas del negocio, de manera que en conjunto formen un sistema adecuado que satisfaga las necesidades actuales del negocio automatizando los procesos que antes se hacían en papel, agilizando los tiempos de entrega y teniendo un mejor control de los productos.

El software incluirá los módulos de compras y ventas, en cada uno de ellos el cliente podrá realizar ingresos, actualizaciones, modificaciones y eliminaciones de registros, interactuando así con la base de datos según el proceso que esté realizando.

7.2 Documento de Requerimientos

FERROREPUESTOS “JW” DOCUMENTO DE REQUERIMIENTOS
Fecha de elaboración: 18 de Diciembre del 2009 Responsables: Andrés Argudo; William Astudillo Colaboradores: Versión: 1.0
<p>1. Análisis del problema</p> <p>Los procesos se realizan de manera manual retardando los tiempos de atención al cliente en la venta de productos, además los cálculos de valores de ingresos y egresos de cada mes se realizan de forma inadecuada donde muchas de las veces se han producido errores humanos en los mismos. Estos cálculos que determinan el valor en impuestos a cancelar al SRI, se vuelven una tarea tediosa que dura horas e incluso días debido a las numerosas transacciones que se encuentran en papeles almacenados en carpetas. El sistema actual es lento, y posibilita la inconsistencia de información debido a que los inventarios no son actualizados constantemente.</p> <p>2. Requerimientos de los usuarios del Sistema</p> <p>El usuario requiere un mejor soporte para las transacciones diarias debido a que el negocio ha crecido y el realizar las tareas solo en papel dificulta su realización, el usuario quiere además mejorar la atención a sus clientes con un sistema rápido y moderno que facilite sus tareas eliminando la tediosa labor de almacenar y consultar papeles que muchas de las veces no tienen un adecuado tratamiento.</p> <p>Se requiere también el soporte de una herramienta en el manejo de impuestos, la cual simplifique los procesos de cálculo de los mismos, ya que actualmente el administrador no la puede realizar de manera eficiente.</p>

3. Definición del Sistema

Requerimientos no Funcionales del Sistema

- El acceso al sistema será controlado mediante usuarios registrados, permisos y roles, solo el usuario que se desempeñe como administrador podrá acceder a funciones administrativas como realización de pedidos en el módulo de compras, pago de impuestos y la administración general del local, los demás usuarios solo podrán acceder al modulo de ventas. Se definirán políticas para el uso de claves.
- La interfaz de usuario deberá ser tan familiar como sea posible a los usuarios que han usado otras aplicaciones de software, para ello la distribución de menús, botones y cajas de texto deben ser las más idóneas y su uso debe ser el más simple. Las interfaces serán construidas basándose en estándares.
- El sistema contará con un manual de usuario en el que se incluya una guía completa de resolución de problemas.
- El tiempo de actualización de la pantalla será alrededor de dos segundos.
- El sistema procesará alrededor de 50 transacciones por segundo en casos críticos como búsquedas o listados.
- Se dispondrá de un tiempo promedio de respuesta alrededor de 3 a 4 segundos aproximadamente, debido a que se estima que el sistema no tendrá una cantidad excesivamente grande de datos, por a su orientación a empresas pequeñas y medianas.
- El tiempo de reinicio después de fallos será alrededor de los 2 minutos.
- El almacenamiento de datos requerirá que el servidor siempre tenga a disposición espacio suficiente de disco duro durante su tiempo de servicio.
- La cantidad de memoria RAM de procesamiento recomendada mínima es de 512 Mb para poder tener un correcto desempeño del sistema.
- La disponibilidad de funcionamiento del sistema será del 95%.
- El sistema será desarrollado en el lenguaje de programación Java y utilizará una Base de Datos MySQL. En el documento de Implementación, apartado

7.4, se explica el porqué de esta selección.

- El entrenamiento a los usuarios del sistema durará una semana calendario, en el que se impartirán técnicas de uso y procedimientos en casos de falla del sistema, con esto se busca la pronta familiarización de los usuarios con el sistema.

Requerimientos Funcionales del Sistema

El sistema debe satisfacer necesidades muy importantes para el mejor funcionamiento del negocio para poder agilizar los procesos del mismo y brindar una mejor atención a los clientes, además de permitir una administración mucho más eficiente. Los requisitos funcionales que deberá cumplir el sistema son los siguientes:

- Debe registrar, modificar y consultar los datos de clientes.
- Debe registrar, modificar y consultar los datos de proveedores.
- Debe registrar, modificar y consultar los datos de productos.
- Debe registrar y consultar las transacciones generadas por cada venta. Es decir que productos se vendieron y su valor.
- Debe permitir anular cierto registro de venta, pero solo con el consentimiento del administrador.
- Debe poder presentar los valores totalizados de las transacciones de ventas, en ciertos rangos de fechas.
- Debe registrar, y consultar las compras realizadas a los diferentes proveedores, registrando el valor de la compra, y los productos adquiridos.
- Debe poder presentar los valores totalizados de las transacciones de compras, en ciertos rangos de fechas.
- Poner obtener un registro de ingresos y gastos del negocio.
- Debe automatizar el cálculo del valor a cancelar por concepto de impuesto al valor agregado IVA, basándose en los datos totalizados del mes correspondiente.

4. Alcance del Proyecto

El sistema a desarrollar comprende una automatización de los procesos que actualmente se están realizando en la empresa pero de forma manual, estos procesos hacen referencia a las compras y a las ventas, que son las principales actividades que realiza la empresa. El software está comprendido por una base de datos en la que se podrán almacenar datos de los diferentes clientes, proveedores y productos; así el usuario podrá gestionar los mismos.

Los usuarios del sistema tendrán asignado un rol según su función, y su respectiva clave de acceso, así el administrador tendrá acceso a todo el sistema pudiendo realizar pedidos y gestionar la parte administrativa del negocio, el usuario vendedor solo podrá utilizar el sistema para realizar las ventas correspondientes, así tendrá acceso a la información de productos y clientes pudiendo solo ingresar, modificar o actualizar datos de las transacciones que lleve a cabo.

El administrador del local tendrá acceso a toda la información del sistema pudiendo modificar, eliminar (lógica) y listar los registros que desee, de igual forma es el único que puede realizar los pedidos a los proveedores.

7.3 Documento de Análisis y Diseño

FERROREPUESTOS “JW”

DOCUMENTO DE ANÁLISIS Y DISEÑO

Fecha de elaboración: 21 de Diciembre del 2009

Responsable: Andrés Argudo; William Astudillo

Colaboradores:

Versión: 1.0

1. Definición de la arquitectura candidata

La arquitectura candidata a utilizar está constituida por capas, descritas de la siguiente manera:

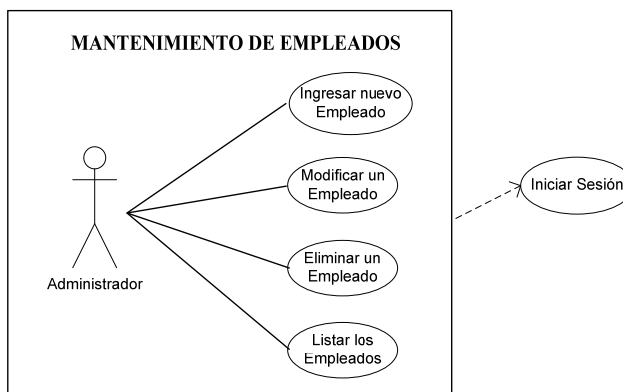
- Capa de persistencia de datos: Esta capa realiza una interfaz de comunicación con el sistema gestor de base de datos, mediante la manipulación de datos a través de un API (Interfaz de programación de aplicaciones).
- Capa de lógica de negocio: Se encarga de gestionar las solicitudes con la capa de datos y presentar los resultados de las mismas, interactuando así con el gestor de base de datos para almacenar o recuperar datos de él.
- Capa de aplicación: Esta capa será vista por el usuario, y le permitirá realizar las diferentes operaciones sobre los datos (Ingresos, modificaciones, listados, etc.).

2. Analizar el comportamiento del sistema

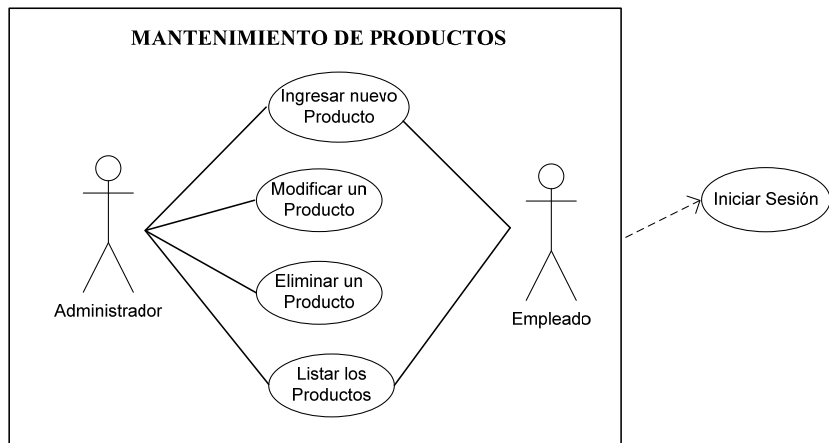
El comportamiento del sistema será descrito mediante los diagramas de casos de uso y sus flujos de trabajos ya sean normales o alternativos en caso de que se presente algún problema.

Se describen también las condiciones iniciales para que un caso de uso pueda entrar en funcionamiento, quienes intervienen en el caso de uso y las salidas que se obtienen del mismo.

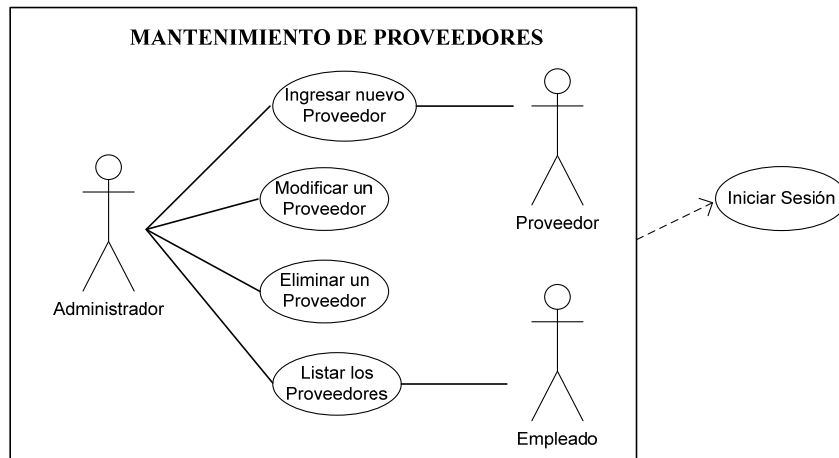
Casos de Uso



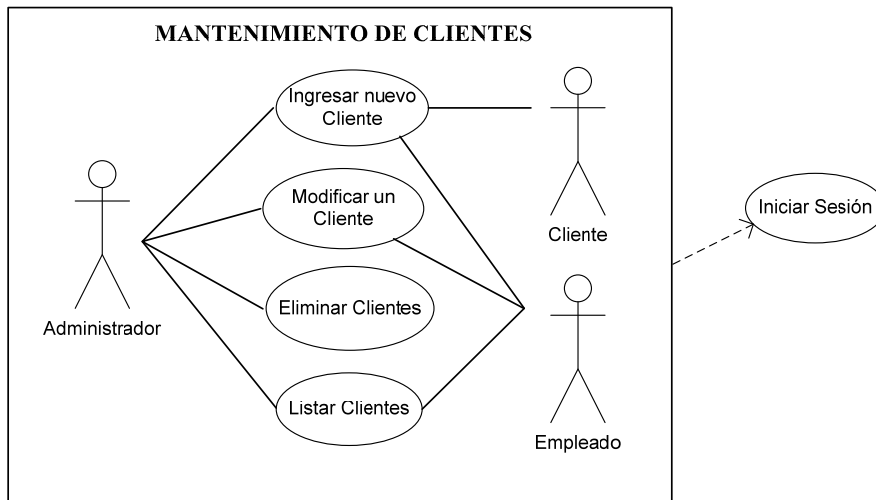
Nombre:	<i>Mantenimiento de Empleados</i>
Descripción: Permite ingresar, modificar, eliminar y listar los Empleados del local.	
Actores: Administrador.	
Precondiciones: El Administrador debe haber iniciado sesión para poder realizar el mantenimiento de Empleados.	
Flujo Normal: <ol style="list-style-type: none"> 1. El Administrador se conecta al sistema. 2. Accede al menú de Empleados y selecciona la opción ingresar, modificar, eliminar o listar Empleados. 3. El sistema ejecuta la acción. 4. El sistema presenta un mensaje al usuario informándole que la operación se realizó exitosamente. 	
Flujo Alternativo: <ol style="list-style-type: none"> 1. Si hay algún problema con los datos ingresados se informa al usuario y se le permite realizar modificaciones, o cancelar la acción. 2. No se ejecuta la acción por cuestiones de consistencia de la información. 	
Poscondiciones: Los datos del nuevo Empleado se almacenan en la Base de Datos.	
Datos de Salida: <ul style="list-style-type: none"> • Mensaje que informe al usuario si la operación se realizó o no con éxito. 	



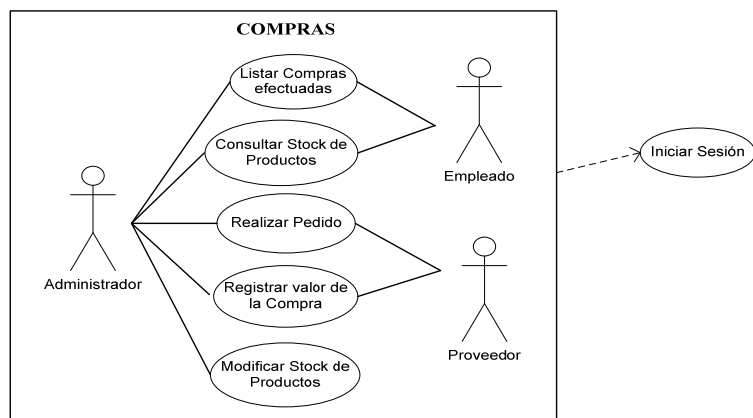
Nombre:	<i>Mantenimiento de Productos</i>
Descripción:	Permite ingresar, modificar, eliminar y listar los Productos del local.
Actores:	Administrador, Empleado.
Precondiciones:	El Administrador o el Empleado deben haber iniciado sesión para poder realizar las operaciones con los Productos.
Flujo Normal:	<ol style="list-style-type: none"> 1. El Administrador o el Empleado se conectan al sistema. 2. Acceden al menú de Productos y seleccionan la opción ingresar, modificar, eliminar o listar Productos, (El rol Empleado solo tendrá permisos para ingresar o listar Productos). 3. El sistema ejecuta la acción. 4. El sistema presenta un mensaje al usuario informándole que la operación se realizó exitosamente.
Flujo Alternativo:	<ol style="list-style-type: none"> 1. Si hay algún problema con los datos ingresados se informa al usuario y se le permite realizar modificaciones. 2. No se ejecuta la acción por cuestiones de consistencia de la información.
Poscondiciones:	Los datos del nuevo Producto se almacenan en la Base de Datos.
Datos de Salida:	<ul style="list-style-type: none"> • Mensaje que informe al usuario si la operación se realizó o no con éxito.



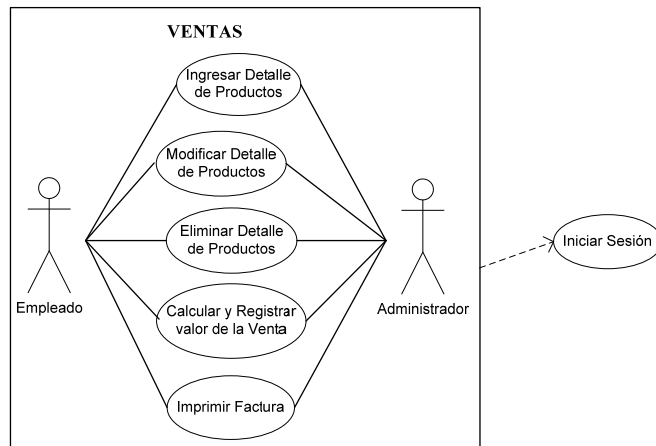
Nombre:	<i>Mantenimiento de Proveedores</i>
Descripción:	Permite ingresar, modificar, eliminar y listar los Proveedores.
Actores:	Administrador, Proveedor, Empleado.
Precondiciones:	El Administrador o el Empleado deben haber iniciado sesión para poder realizar las operaciones con los Proveedores.
Flujo Normal:	<ol style="list-style-type: none"> 1. El Administrador o el Empleado se conectan al sistema. 2. Acceden al menú de Proveedores y seleccionan la opción ingresar, modificar, eliminar o listar Proveedores, (El rol Empleado solo tendrá permisos para listar los Proveedores). Cuando se ingresa un nuevo Proveedor al sistema, este actor interactúa directamente con el actor Administrador. 3. El sistema ejecuta la acción. 4. El sistema presenta un mensaje al usuario informándole que la operación se realizó exitosamente.
Flujo Alternativo:	<ol style="list-style-type: none"> 1. Si hay algún problema con los datos ingresados se informa al usuario y se le permite realizar modificaciones. 2. No se ejecuta la acción por cuestiones de consistencia de la información.
Poscondiciones:	Los datos del nuevo Proveedor se almacenan en la Base de Datos.
Datos de Salida:	<ul style="list-style-type: none"> • Mensaje que informe al usuario si la operación se realizó o no con éxito.



Nombre:	<i>Mantenimiento de Clientes</i>
Descripción:	Permite ingresar, modificar, eliminar y listar los Clientes del local.
Actores:	Administrador, Cliente, Empleado.
Precondiciones:	El Administrador o el Empleado deben haber iniciado sesión para poder realizar las operaciones con los Clientes.
Flujo Normal:	<ol style="list-style-type: none"> 1. El Administrador o el Empleado se conectan al sistema. 2. Acceden al menú de Clientes y seleccionan la opción ingresar, modificar, eliminar o listar Clientes, (el rol Empleado no tiene permisos para eliminar Clientes). 3. El sistema ejecuta la acción. 4. El sistema presenta un mensaje al usuario informándole que la operación se realizó exitosamente.
Flujo Alternativo:	<ol style="list-style-type: none"> 1. Si hay algún problema con los datos ingresados se informa al usuario y se le permite realizar modificaciones. 2. No se ejecuta la acción por cuestiones de consistencia de la información.
Poscondiciones:	Los datos del nuevo Cliente se almacenan en la Base de Datos.
Datos de Salida:	<ul style="list-style-type: none"> • Mensaje que informe al usuario si la operación se realizó o no con éxito.



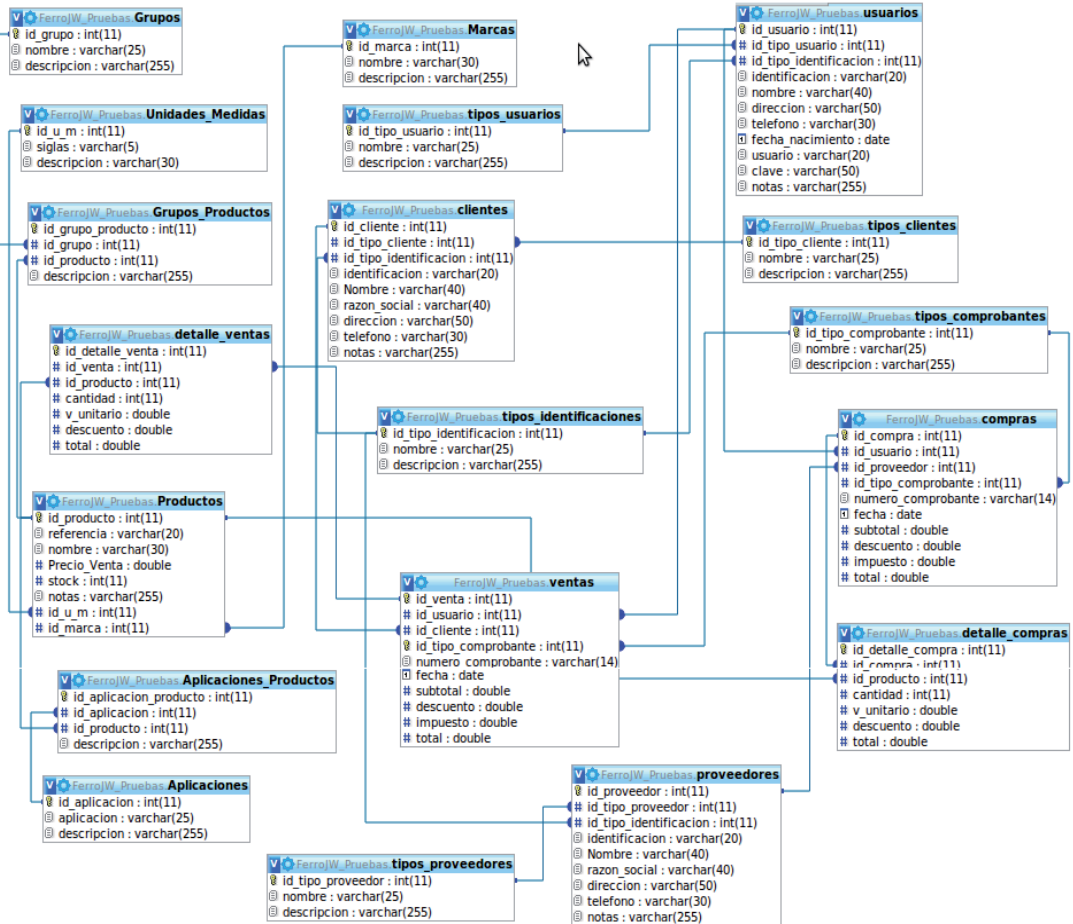
Nombre:	<i>Realizar una Compra a un Proveedor</i>
Descripción:	Permite efectuar un Pedido de uno o varios Productos a un Proveedor.
Actores:	Administrador, Proveedor, Empleado.
Precondiciones:	El Administrador o el Empleado deben haber iniciado sesión para poder ingresar al módulo de Compras.
Flujo Normal:	<ol style="list-style-type: none"> 1. El Administrador se conecta al sistema. 2. Abre la página de búsqueda y digita el artículo a consultar en stock. 3. Selecciona un artículo de la lista con la cantidad para el pedido. 4. El sistema ejecuta la acción y presenta una descripción de los artículos listos para el pedido. 5. El Administrador indica al sistema que concluyó el ingreso de Productos. 6. El Empleado entrega el Pedido al Proveedor. 7. El proveedor entrega la mercancía al Administrador. 8. El Administrador efectúa el pago y recibe la factura de la Compra. 9. El sistema registra la Compra. 10. El sistema presenta un mensaje al usuario informándole que la operación se realizó exitosamente.
Flujo Alternativo:	<ol style="list-style-type: none"> 1. Si hay algún problema con los datos ingresados se informa al usuario y se le permite realizar modificaciones o de igual manera se puede cancelar la Compra. 2. No se ejecuta la acción por cuestiones de consistencia de la información.
Poscondiciones:	Los datos del nuevo Pedido se almacenan en la Base de Datos.
Datos de Salida:	<ul style="list-style-type: none"> • Mensaje que informe al usuario si la operación se realizó o no con éxito.



Nombre:	<i>Realizar una Venta a un Cliente</i>
Descripción:	Permite efectuar una Venta de uno o varios Productos a un Cliente.
Actores:	Administrador, Empleado.
Precondiciones:	El Administrador o Empleado debe haber iniciado sesión para poder ingresar al módulo de Ventas.
Flujo Normal:	<ol style="list-style-type: none"> 1. El Administrador o Empleado se conectan al sistema. 2. Abre la página de búsqueda y digita el artículo a vender. 3. Selecciona un artículo de la lista con la cantidad para la Venta. 4. El sistema ejecuta la acción y presenta una descripción de los artículos, precio y suma parcial. 5. El usuario indica al sistema que concluyó el ingreso de Productos. 6. El sistema calcula y presenta el total con impuestos de la Venta. 7. El usuario gestiona el pago de la Venta. 8. El sistema registra la Venta y genera una Factura. 9. El sistema presenta un mensaje al usuario informándole que la operación se realizó exitosamente.
Flujo Alternativo:	<ol style="list-style-type: none"> 1. Si hay algún problema con los datos ingresados se informa al usuario y se le permite realizar modificaciones o sino de igual forma se puede cancelar la Venta. 2. No se ejecuta la acción por cuestiones de consistencia de la información.
Poscondiciones:	Los datos de la Venta se almacenan en la Base de Datos.
Datos de Salida:	<ul style="list-style-type: none"> • Mensaje que informe al usuario si la operación se realizó o no con éxito.

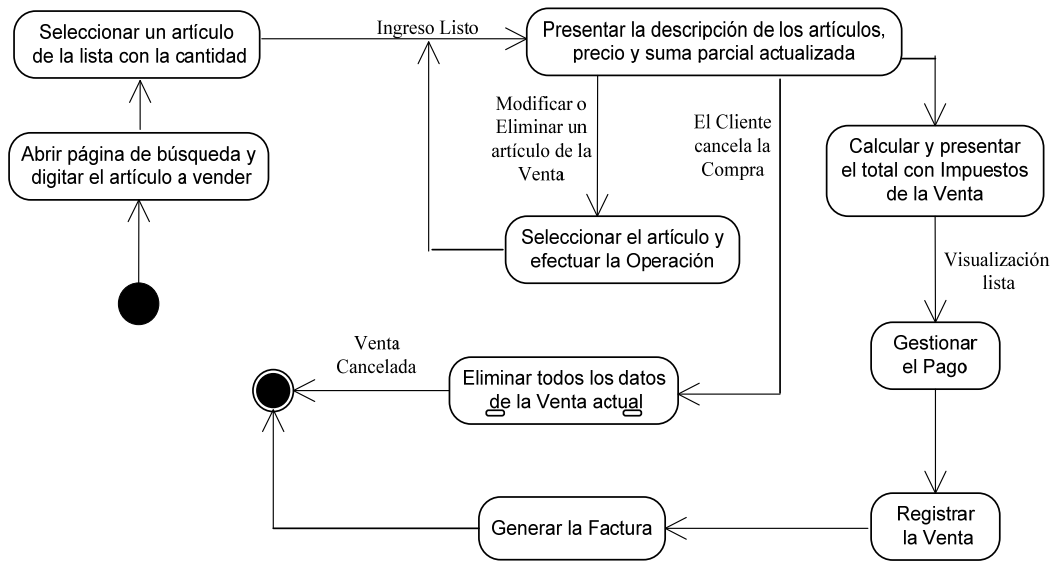
3. Diseño del sistema y sus componentes

3.1 Diagrama de la Base de Datos

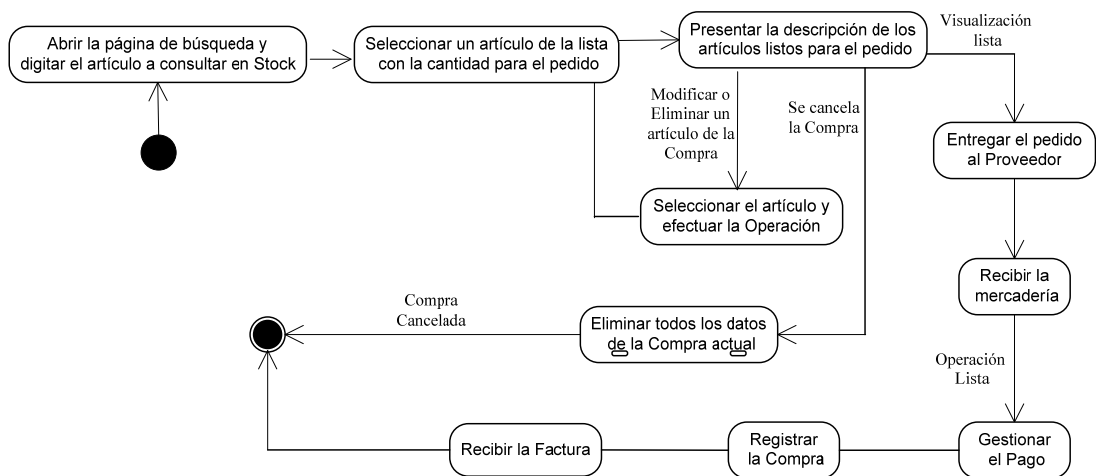


3.2 Diagrama de Estados

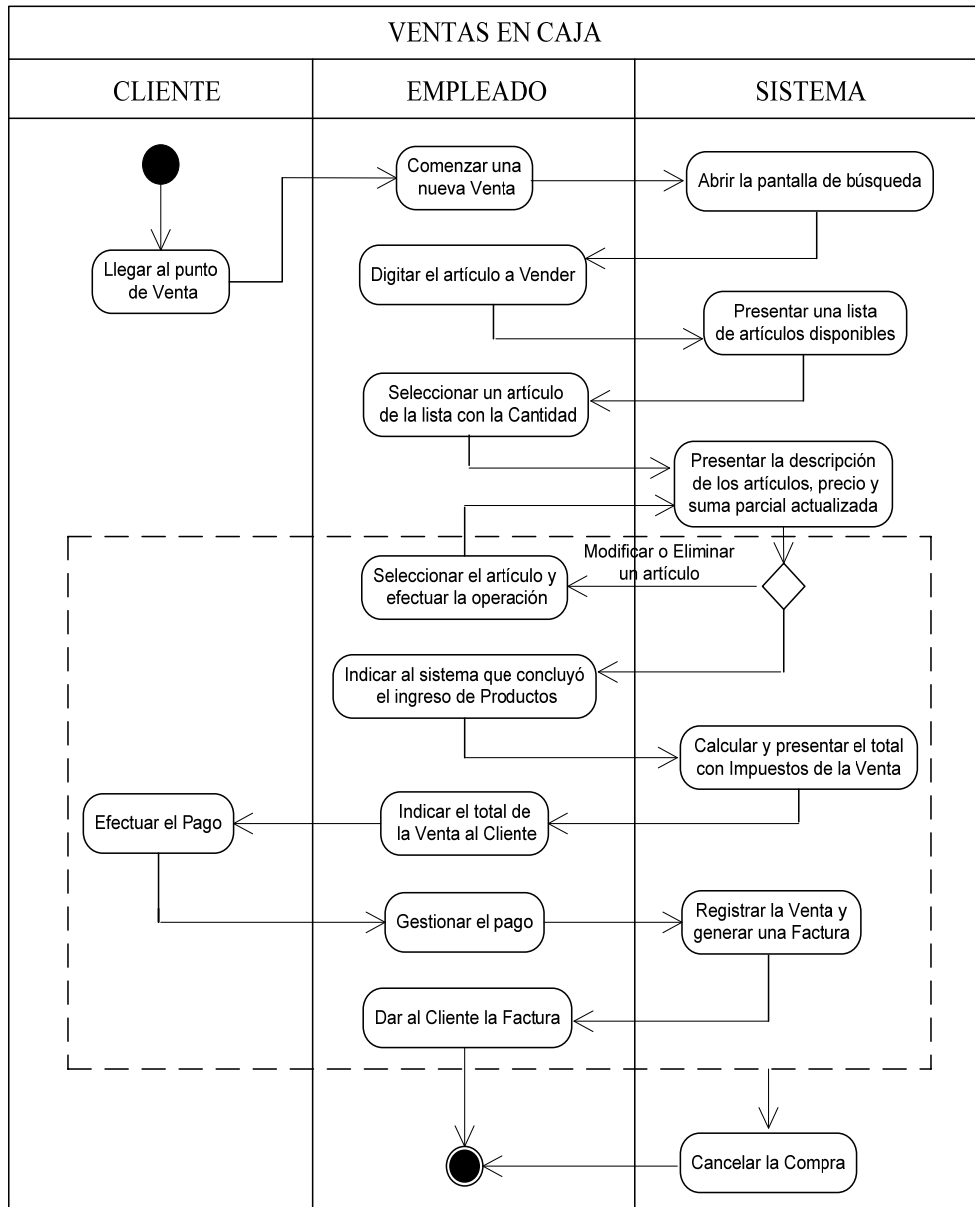
Ventas en Caja

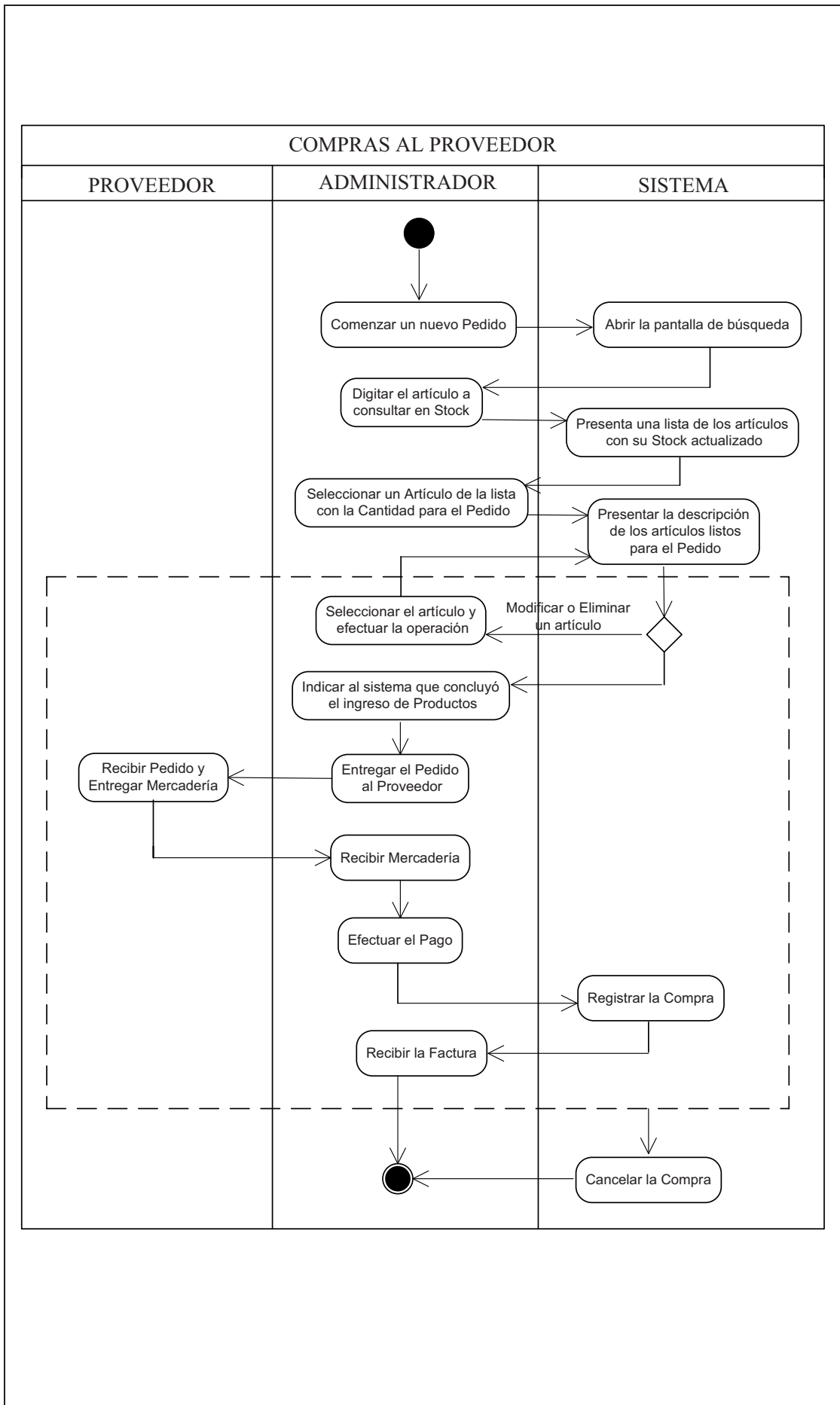


Compras al Proveedor



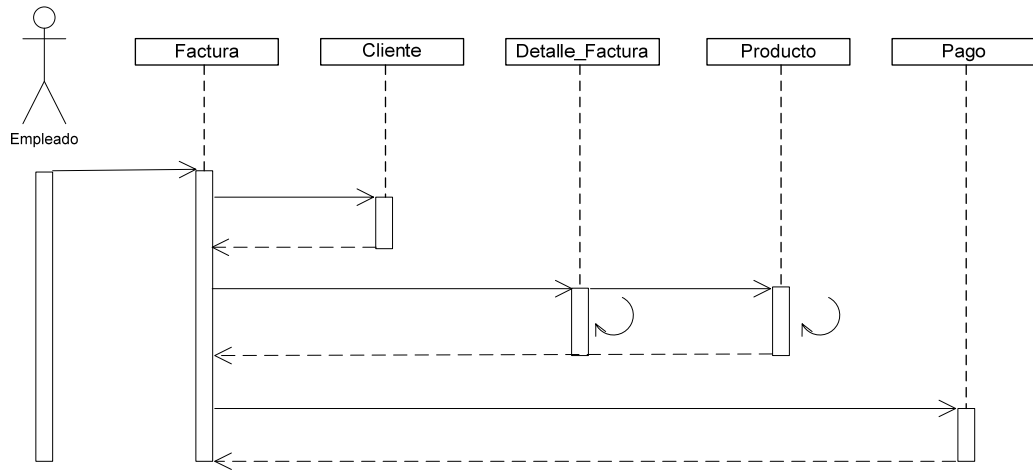
3.3 Diagrama de Actividades



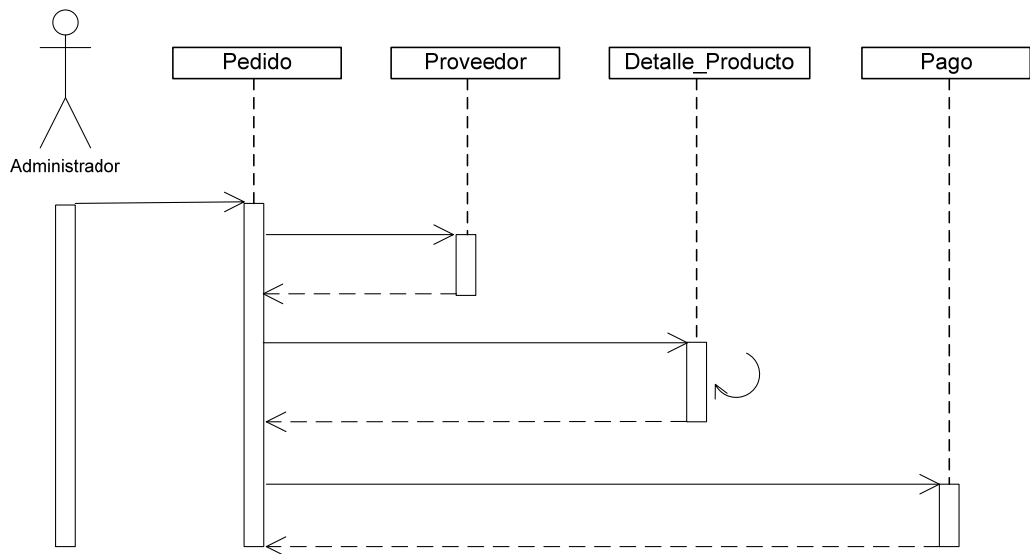


3.4 Diagramas de Secuencia

Ventas en Caja



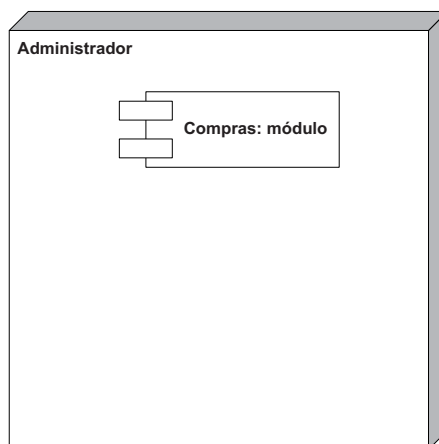
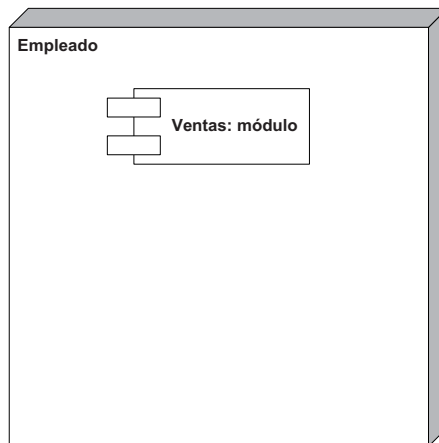
Compras al Proveedor



3.5 Diagrama de Componentes



3.6 Diagramas de Despliegue



7.4 Documento de Implementación del Proyecto

FERROREPUESTOS “JW” DOCUMENTO DE IMPLEMENTACIÓN DEL PROYECTO
Fecha de elaboración: 05 de Enero del 2010 Responsables: Andrés Argudo, William Astudillo Colaboradores: Versión: 1.0
1. Definición del Modelo de Implementación ESPECIFICACIÓN DEL LENGUAJE DE PROGRAMACIÓN Y SISTEMA GESTOR DE BASE DE DATOS Lenguaje de Programación: El lenguaje de programación resulta de vital importancia al momento de construir aplicaciones, pues según las bondades y herramientas que nos brinde el mismo, podremos implantar soluciones eficaces, y eficientes. El Lenguaje de programación a utilizar en nuestro proyecto será JAVA ya que presenta las siguientes virtudes: <ul style="list-style-type: none">• Orientación a Objetos: Lo que nos permite crear abstracciones que facilitan la tarea de programación pudiendo asemejarla con objetos de la vida real.• Gran Portabilidad: Nos permite ejecutar nuestro programa en diferentes Sistemas Operativos, sin tener que reescribirlo o tener que realizar grandes modificaciones. Debido a su máquina virtual.• Brinda una plataforma completa de desarrollo: Consta con muchos componentes que pueden ser reutilizados y extendidos, facilitando mucho la tarea de programación.• Lenguaje de programación libre: Nos permite crear aplicaciones con licencias de Software Libre.

- Evolución constante y acelerada: Java ha evolucionando muchísimo en los últimos años, y ha sido elegido como lenguaje de programación para grandes proyectos de software de varias empresas, por lo que se han generado diversos Frameworks y herramientas muy poderosas.
- JAVA es el lenguaje que más conocemos y manejamos, lo hemos aprendido a lo largo de nuestra carrera universitaria.

Base de Datos

Sera la encargada de dar soporte a todos los datos de nuestro proyecto, por lo que resulta muy importante poder contar con un Sistema Gestor de Base de Datos que sea rápido, eficiente y de fácil utilización, por ello hemos elegido el Sistema Gestor de Base de Datos MySql debido a las siguientes virtudes.

- Escalabilidad y Flexibilidad: Se puede partir de bases de datos pequeñas y pueden convertirse en bases de datos gigantescas (con capacidad de almacenamiento de hasta TeraBytes), y el Sistema Gestor de base de datos las sigue administrando muy eficientemente.
- Alto Rendimiento: Cuenta con una serie de mecanismos de mejora de rendimiento, como índices full text que le permiten brindar un rendimiento muy bueno en diversas aplicaciones.
- Alta Disponibilidad: Tiene como característica Solidez y Disponibilidad, brindando la posibilidad de desarrollar sistemas de replicación de manera rápida y sencilla.
- Soporte Transaccional: Tiene una motor transaccional, que brinda soporte completo de ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad).
- Evolución constante y acelerada: MySql ha evolucionando muchísimo en los últimos años, y ha sido elegido como el gestor de bases de datos para grandes proyectos de software, así como también de páginas web de varias empresas, debido a su carácter libre, lo que significa gran rendimiento a costos mínimos.

ESTÁNDARES PARA EL DESARROLLO DE CÓDIGO DE PROGRAMAS

Envío de Parámetros

Para el envío de parámetros se utilizan palabras en minúscula separadas por mayúscula, todas ellas dentro de un paréntesis.

Variables globales y locales

La declaración de variables locales comienza con palabras en minúscula separadas por mayúscula.

Nombres de Clases

Los nombres de las clases comienzan con letra mayúscula, con las palabras que la forman en minúscula y separadas por mayúsculas.

Nombres de Métodos

Los nombres de los métodos miembros de una clase comienzan con minúscula y separando las palabras con mayúscula.

Indentación

No usar tabs ya que algunos editores muestran dichos caracteres con distinto ancho según la configuración del usuario, en lugar de ello utilizar espacios.

Uso de Llaves

Luego de cualquier sentencia if, else, for, etc, la llave de apertura se coloca al mismo nivel de indentación que la palabra anterior, en la línea inmediatamente debajo de ella, la llave de cierre se coloca a ese mismo nivel de indentación. El código dentro de las llaves sube un nivel de indentación con respecto a las llaves.

Uso de Paréntesis

No poner paréntesis pegados a las sentencias ya que puede generar confusión en el código, se los debe poner pegados a las funciones o variables, además no se los debe poner en valores de retorno cuando no sea necesario.

Condiciones complicadas dentro de un if, while, for, etc.

En muchos de los casos las condiciones dentro de un if, while o for, se torna compleja y con muchos niveles de paréntesis. En estos casos es importante la legibilidad y poder ver claramente la correspondencia entre los pares de paréntesis. Una solución es la de extender la sentencia en varias líneas, indentando cada nivel de paréntesis curvos como sea necesario.

HERRAMIENTAS A UTILIZAR

IDE

La Construcción de nuestro proyecto requiere de un Entorno de Desarrollo integrado para poder trabajar de manera rápida y eficiente, un IDE nos permite escribir las clases, construir interfaces de usuarios mediante la utilización de APIs, además de permitirnos la fácil integración de diferentes componentes de software. El IDE utilizado para nuestro proyecto es NetBeans ya que presenta las siguientes características:

- Administración de interfaces de usuario: Facilidades para construir ventanas, menús, barra de herramientas, etc.
- Asistente de configuración de Frameworks: Cuenta con una serie de asistentes para poder configurar varios Frameworks de desarrollo paso a paso y de manera muy fácil.
- Add-ons Packs: Nos permite agregar paquetes adicionales, según nuestras necesidades.
- Evolución constante y acelerada: NetBeans ha evolucionando muchísimo, adaptándose a los nuevos Frameworks de desarrollo de manera exitosa y siendo una opción libre muy buena y fácil de utilizar para desarrollar proyectos de software.

phpMyAdmin

Es una herramienta libre que gestiona de manera práctica y sencilla la Base de Datos MySQL permitiendo crear o borrar bases de datos, tablas, relaciones, etc. Además permite ejecutar cualquier sentencia SQL, administrar claves, privilegios y se encuentra disponible bajo licencia GPL.

7.5 Documento de Pruebas

FERROREPUESTOS “JW”

DOCUMENTO DE PRUEBAS DEL SISTEMA

Fecha de elaboración: 26 de Enero del 2010

Responsables: Andrés Argudo, William Astudillo

Colaboradores:

Versión: 1.0

1. Definición de la misión de las Pruebas

Comprobar el correcto desempeño de los módulos de Compras y Ventas del sistema a entregar a FERROREPUESTOS “JW” mediante la aplicación de pruebas a través de situaciones controladas que incluyan contextos normales como anormales, los cuales permitan encontrar errores con el fin de mitigarlos.

2. Escalabilidad de los Componentes

El sistema es escalable ya que permite la modificación de su estructura de datos, permitiendo realizar cambios en la base de datos los cuales se ven reflejados de manera eficiente en sus componentes, esto gracias a la utilización de un Framework Hibernate que facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación.

3. Evaluación del Sistema

Ejecutar el sistema una vez comprobado el correcto funcionamiento de la Base de Datos y los módulos respectivos

Descripción: Se comprobará el correcto funcionamiento del sistema, base de datos y módulos que lo integran una vez iniciada la aplicación.

Resultado Obtenido: El sistema inició normalmente sin complicaciones y listo para usarse.

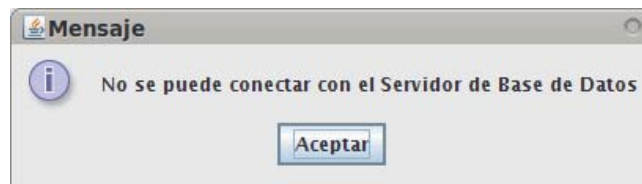
Resultado Esperado: El sistema debe ejecutarse sin problemas y en el menor tiempo posible.

Estado de la prueba: Superada

Ejecutar el sistema teniendo deshabilitada la Base de Datos

Descripción: La prueba consiste en ver el comportamiento del sistema cuando se carga, teniendo deshabilitada la base de datos.

Resultado Obtenido: El sistema inició de forma anormal indicando el siguiente mensaje:



Resultado Esperado: El sistema no debe ejecutarse correctamente, debe mostrar un mensaje de notificación de error de conexión con la base de datos.

Estado de la prueba: Superada

Validar el Ingreso al Sistema solo por Usuarios Registrados

Descripción: Se comprobará el acceso al sistema solo de usuarios registrados cuyas claves consten en la base de datos. La prueba consiste en ingresar un parámetro erróneo en la ventana de login sea este el id o la clave, de la cual se obtenga un mensaje de error impidiendo el acceso al usuario.

Resultado Obtenido: El sistema produjo un error de acceso a través de un mensaje en el que se indicaba que la información digitada no corresponde a la que se encuentra en la base de datos.

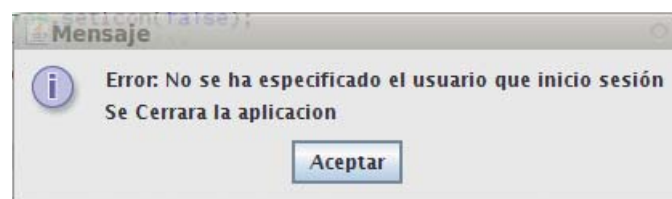
Resultado Esperado: El sistema debe impedir el acceso si los datos que provee el usuario no corresponden a los que están almacenados en la base de datos.

Estado de la prueba: Superada

Invocar directamente a un módulo o clase sin antes autenticarse.

Descripción: Se comprobará el comportamiento del sistema al tratar de acceder directamente a los módulos y clases de la aplicación sin antes autenticarse.

Resultado Obtenido: El sistema produjo un error de acceso a través de un mensaje en el que se indicaba lo siguiente:



Resultado Esperado: El sistema debe impedir el acceso a los módulos y clases de la aplicación si el usuario no se autentica primero.

Estado de la prueba: Superada

Validar que una Ventana se abra por una sola vez en la Aplicación

Descripción: La prueba consiste en validar que una ventana se instancie una sola vez en la aplicación, para esto se intentará abrir varias veces la misma ventana para ver cómo reacciona el sistema.

Resultado Obtenido: El sistema instanció una sola vez la ventana al intentar abrirla varias veces.

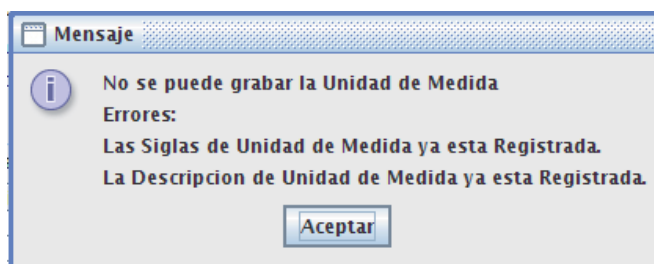
Resultado Esperado: El sistema debe instanciar una sola vez una ventana en la aplicación, así el usuario intente abrirla varias veces.

Estado de la prueba: Superada

Validar que un campo en la Base de datos sea ingresado una sola vez, solo para ciertas tablas

Descripción: La prueba consiste en validar que el sistema acepte el ingreso de un solo campo en la base de datos según su aplicación, por ejemplo, validar que una misma unidad de medida no sea ingresada dos veces.

Resultado Obtenido: El sistema no permite el ingreso de dos campos con el mismo nombre, el error capturado es el siguiente:



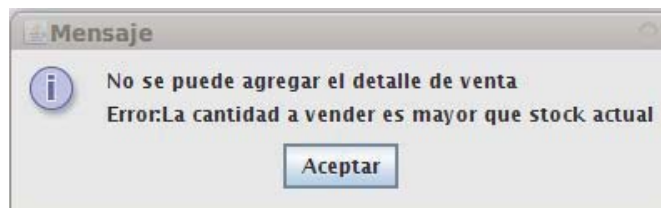
Resultado Esperado: El sistema no debe permitir ingresar en la base de datos dos campos con el mismo nombre, por ejemplo, una misma unidad de medida no debe estar dos veces en el Sistema.

Estado de la prueba: Superada

Validar que la cantidad a vender no sea mayor que la cantidad en Stock

Descripción: La prueba consiste en validar que el sistema realice una venta solo si la cantidad a vender no supere la cantidad en stock de productos, para ello se manipularon los datos de stock para que el sistema no pudiera continuar con la venta.

Resultado Obtenido: El sistema no pudo continuar con la venta ya que el valor del stock no es mayor que la cantidad demandada, se produjo el siguiente error:



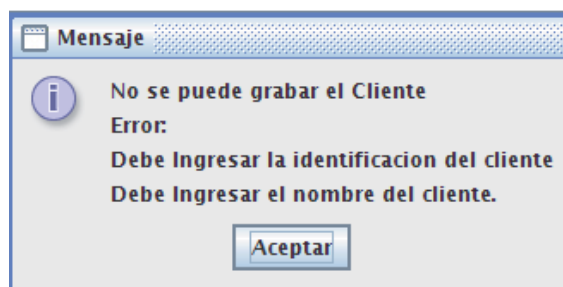
Resultado Esperado: El sistema no debe permitir continuar con una venta si el valor en stock es inferior a la cantidad solicitada.

Estado de la prueba: Superada

Validar que los registros de un Cliente sean guardados siempre y cuando sean ingresados

Descripción: La prueba consiste en validar que guarden los datos del cliente siempre y cuando hayan sido ingresados.

Resultado Obtenido: El sistema no pudo continuar con el registro del cliente ya que no se ingresó la información del caso, surgió el siguiente error:



Resultado Esperado: El sistema no debe permitir guardar los datos que no

han sido ingresados.

Estado de la prueba: Superada

Validar que después de una Venta se actualice el Stock

Descripción: La prueba consiste en validar que el sistema actualice el Stock de un producto luego de una Venta, para ello se realizará una Venta con una cantidad considerable y al final se comprobará el Stock en la base de datos.

Resultado Obtenido: El sistema actualizó el Stock del producto luego de la Venta.

Resultado Esperado: El sistema debe actualizar el campo Stock de la tabla de productos luego de cualquier Venta.

Estado de la prueba: Superada

Validar que los cálculos de valores en la Venta sean los correctos

Descripción: La prueba consiste en validar que el sistema realice correctamente todos los cálculos correspondientes con la venta, se verificarán valores totales, parciales y con IVA.

Resultado Obtenido: El sistema no respondió correctamente con los cálculos procesados. El valor total calculado no corresponde a valores reales tal como se muestra en la siguiente imagen:

Codigo	16	Producto	FRAM PH3387A FILTRO DE ACEITE CAJA				
Cantidad	1	Precio	3.33	Descuento	0.1	Total	2.910

Aceptar Cancelar

Resultado Esperado: El sistema debe responder correctamente con los valores de la venta.

Estado de la prueba: No Superada

Observaciones: Tuvimos que verificar el manejo y operación con los decimales.

Validar que después de una Compra se actualice el Stock

Descripción: La prueba consiste en validar que el sistema actualice el Stock de un producto después que se realiza una Compra, para ello se simulará un proceso de Compra con una cantidad considerable y al final se comprobará el Stock en la base de datos.

Resultado Obtenido: El sistema actualizó el Stock del producto luego de la Compra.

Resultado Esperado: El sistema debe actualizar el campo Stock de la tabla de productos luego de cualquier Compra.

Estado de la prueba: Superada

Validar que en la cantidad a vender solo se ingresen números

Descripción: La prueba consiste en validar que cuando se vaya a vender un producto, en la caja de texto de cantidad, solo se ingresen números, es decir, nada de letras, caracteres especiales, ni teclas de control como Alt, Shift, Ctrl.

Resultado Obtenido: Cuando se ingresaron números en la caja de texto, el sistema respondió correctamente, al ingresar letras y caracteres especiales el sistema no las mostraba, pero cuando se presionaban las teclas de control se borraron los números ingresados inicialmente.

Resultado Esperado: La caja de texto de cantidad solo debe permitir el ingreso de números, nada de letras, caracteres especiales ni teclas de control.

Estado de la prueba: No Superada

Observaciones: Tuvimos que implementar un nuevo método que valide que

cada vez que salte de la caja de texto, verifique que lo que se ha ingresado sea un número.

7.6 Manual de Usuario

INICIO DE SESIÓN

Al iniciar la aplicación el usuario encontrará la siguiente ventana:



Aquí se debe ingresar el nombre de usuario y clave de un usuario registrado en el sistema. El nombre de usuario y clave son campos obligatorios.

Botón Iniciar:



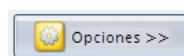
Al Presionar el Botón Iniciar el sistema valida que el nombre del usuario y la clave sean correctas, de ser así se muestra la ventana principal de la aplicación, caso contrario se indicará que no se ha podido establecer la conexión por un error de la información ingresada.

Botón Cancelar:



Al presionar el botón Cancelar el usuario finaliza la aplicación.

Botón Opciones:

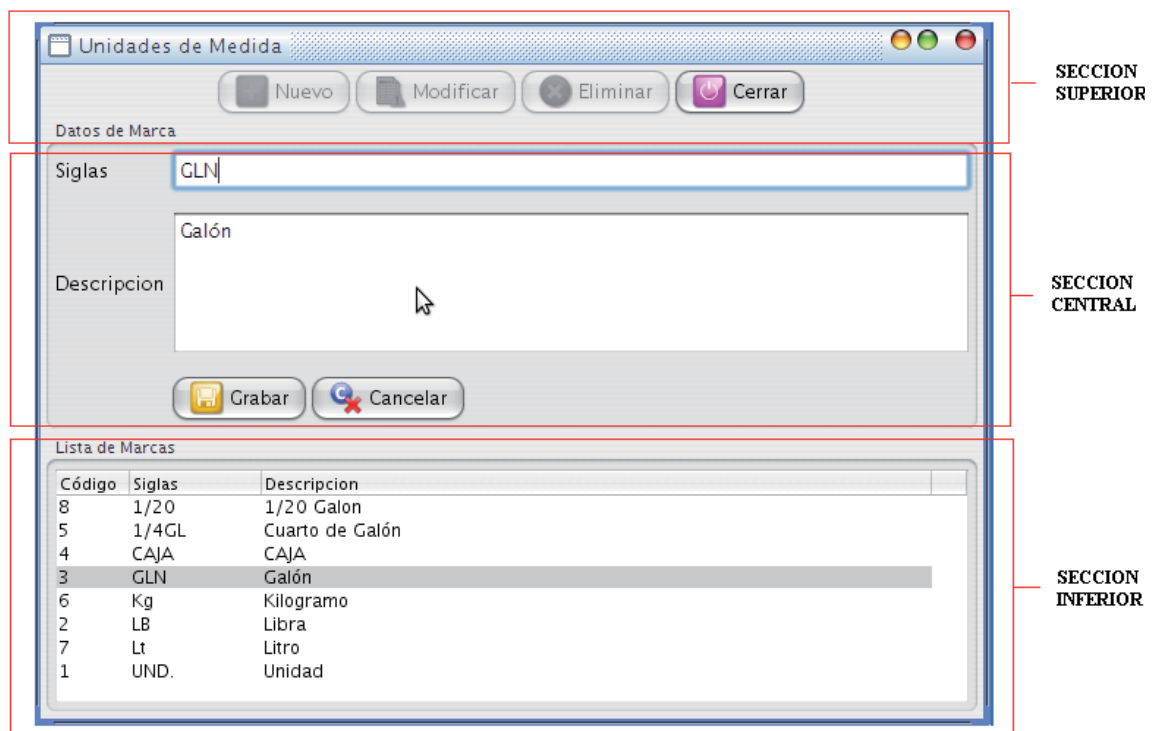


Al presionar este botón el usuario accede a los atributos de dirección y puerto del servidor de Base de Datos. La dirección del servidor puede ser la dirección IP del servidor o su nombre. A continuación la pantalla de Opciones:



ESTRUCTURA DE LAS VENTANAS DEL SISTEMA

Las ventanas del sistema tienen la siguiente estructura:



SECCIÓN SUPERIOR

Aquí se encuentran las diferentes operaciones que se pueden realizar con la aplicación, los botones se describen a continuación:

Botón Nuevo:

Crea un nuevo registro, este puede ser por ejemplo una nueva unidad de medida, una aplicación, etc.

Botón Modificar:

Esta opción se habilita al seleccionar un registro desde la tabla ubicada en la parte inferior de la ventana, al elegir esta opción se permitirá modificar la información del registro seleccionado.

Botón Eliminar:

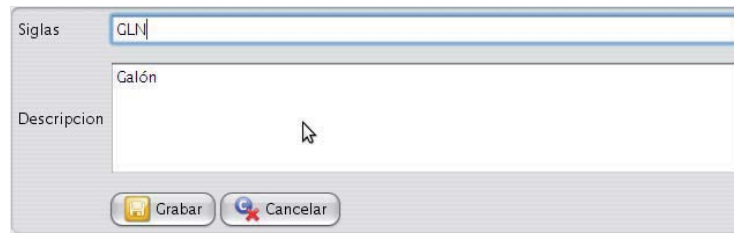
Esta opción se habilita al seleccionar un registro desde la tabla ubicada en la parte inferior de la ventana, al elegir esta opción se mostrará una ventana de confirmación sobre la eliminación del registro seleccionado. La eliminación es lógica, es decir, los datos no se borran físicamente.

Botón Cerrar:

Este botón cierra la ventana, en caso de estar efectuándose alguna operación como un nuevo ingreso o modificación, estos son cancelados.

SECCIÓN CENTRAL

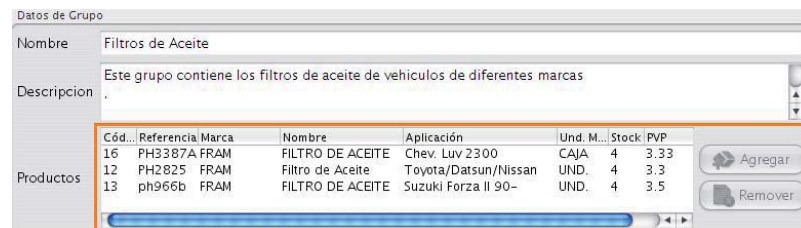
Aquí se muestra e ingresa información según la ventana abierta, en este ejemplo mostramos la información de unidades de medida y de grupos de productos.



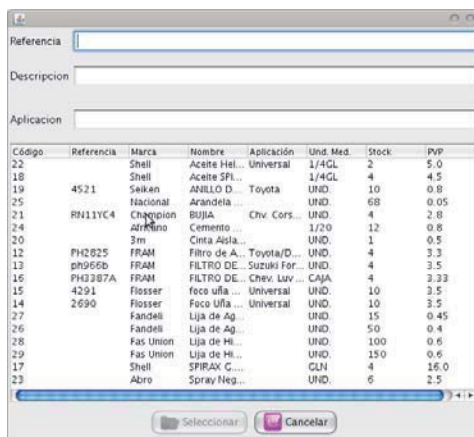
Además aquí se muestran los botones Grabar y Cancelar para las operaciones Nuevo y Modificar. Los diferentes controles de edición estarán bloqueados a menos que se esté ingresando un nuevo registro o modificando alguno existente.

Entidades con información detallada

Para las entidades que contengan información en forma de detalles, tales como las aplicaciones de un producto, o los productos que pertenecen a un grupo por ejemplo. El usuario encontrará tablas que muestran dicha información y en la parte derecha de la misma los siguientes botones:



Agregar: Muestra Un diálogo que ayuda al usuario a buscar una aplicación, producto, cliente, proveedor, etc. según sea el caso. En el mismo se busca la información por sus atributos más importantes.



La información que coincide con los criterios de búsqueda es mostrada en la parte inferior de la pantalla, para seleccionar cual es el registro buscado se selecciona de la tabla, cuando se selecciona algún registro se

habilita el botón Seleccionar, al presionar dicho botón la información es pasada a la ventana desde la cual se abrió el Dialogo de selección. En caso de presionar cancelar se cierra el dialogo y se anula la selección realizada.

Según como se vaya especificando el criterio de búsqueda, automáticamente se van presentando los resultados en la pantalla.

Eliminar: Esta opción se habilita cuando se presiona sobre alguna fila de la tabla, al presionarla se borra la información de la selección actual, desapareciendo la fila seleccionada.

SECCIÓN INFERIOR

Aquí se encuentra la información detallada de todos los registros, los mismos están ordenados por el atributo más importante, por ejemplo para la entidad marcas, clientes y proveedores se encuentra ordenado por su nombre, así como para unidades de medida por sus siglas, etc.

Desde aquí se puede seleccionar cualquier registro para modificarlo o eliminarlo.

Código	Siglas	Descripcion
8	1/20	1/20 Galon
5	1/4GL	Cuarto de Galón
4	CAJA	CAJA
3	GLN	Galón
6	Kg	Kilogramo
2	LB	Libra
7	Lt	Litro
1	UND.	Unidad

ESTRUCTURA DE LA VENTANA DE UN PROCESO DE VENTAS

Para efectuar el proceso de una venta el usuario contará con la siguiente ventana:

Esta está compuesta por tres secciones, en la parte superior se ingresará la información sobre el comprobante de venta, tal como la fecha, número de comprobante, y se seleccionará el tipo de comprobante.

Elegir el cliente

Cód...	Tipo Ide...	Num. Docu...	Nombre	Razón Social	Tipo	D
4	RUC	01017362...	Angelita Quituisac...	FerroRepuestos JW	Client...	A
5	Cedula	01050352...	Bernarda Vintimill...	Libeli Dent	Client...	A
3	Cedula	99999999...	Consumidor Final		Client...	
6	Cedula	01041425...	Juan Pelaez		Client...	D
2	Cedula	01041426...	Juan Rodriguez		Client...	A

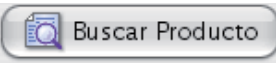
Para elegir el cliente al cual se le genera la venta, se presiona el botón **Buscar Clientes**, el cual abre un cuadro de diálogo para buscar al cliente, ya sea por su identificación (RUC, Cédula,

etc.), Nombre o Razón Social. Luego se selecciona el cliente desde los resultados obtenidos. Inicialmente se muestran todos los clientes en la tabla.

Agregar Productos a la Venta

Para agregar productos a la venta, el usuario debe presionar el botón Buscar

Código	Referencia	Marca	Nombre	Aplicación	Und. Med.	Stock	PVP
22		Shell	Acete Hel...	Universal	1/4GL	2	5.0
18		Shell	Acete SPL...	1/4GL	4	4.5	
19	4521	Selken	ANILLO D...	Toyota	UND.	10	0.8
25		Nacional	Arandela...		UND.	68	0.05
21	RN11YC4	Champion	BUJIA	Chv. Cors...	UND.	4	2.8
24		Africano	Cemento ...		1/20	12	0.8
20		3m	Cinta Aisla...		UND.	1	0.5
12	PH2825	FRAM	Filtro de A...	Toyota/D...	UND.	4	3.3
13	ph9660	FRAM	FILTRO DE...	Suzuki For...	UND.	4	3.5
16	PH3387A	FRAM	FILTRO DE...	Chev. Luv...	CAJA	4	3.33
15	4291	Flosser	foco uña...	Universal	UND.	10	3.5
14	2690	Flosser	Foco Uña...	Universal	UND.	10	3.5
27		Fandeli	Lija de Ag...		UND.	15	0.45
26		Fandeli	Lija de Ag...		UND.	50	0.4
28		Fas Union	Lija de Hi...		UND.	100	0.6
29		Fas Union	Lija de Hi...		UND.	150	0.6
17		Shell	SPIRAV G...		GLN	4	16.0
23		Abro	Spray Neg...		UND.	6	2.5

Producto , el cual abrirá un diálogo para seleccionar el producto a vender, se puede buscar el producto por su referencia, por su nombre o por su aplicación, también se puede hacer una búsqueda combinada por ejemplo se puede buscar un producto por su nombre y su aplicación, ejemplo en Descripción puede escribir “filtro”, y en aplicación “forza”, con lo que se buscará

todos los productos cuyo nombre contenga filtro y sea para las aplicaciones que contengan la palabra “forza”.

Inicialmente se muestran todos los productos, cabe destacar que las búsquedas ignoran mayúsculas y minúsculas. Para seleccionar el producto a vender se debe seleccionar de la tabla y presionar el botón Seleccionar.

Luego de seleccionar el producto, se carga la información de Código y Descripción del Producto, las mismas no se pueden modificar, estos valores son llenados con la información del producto seleccionado del cuadro de diálogo; mientras que cantidad, precio y descuento son editables.

Cantidad: Especifica el número de productos a vender.

Precio: Indica el precio de venta del producto, sin contar los impuestos que serán calculados al final de la venta. En la caja de texto de precio se carga el valor venta del producto, pero el mismo puede ser cambiado.

Descuento: Indica el valor de descuento individual para el producto, este puede ser escrito en porcentaje, por ejemplo si se pretende dar el 5% de descuento a un cliente, en esta caja de texto se escribe 5%, y cuando el cursor sale de la caja, este porcentaje es calculado automáticamente.

Total: Este valor es calculado automáticamente según los datos de cantidad, precio y descuento, y se muestra cada vez que se cambie alguno de los valores anteriormente indicados.

Codigo	<input type="text" value="25"/>	Producto	<input type="text" value="Nacional Arandela Plana 1/4 UND."/>				
Cantidad	<input type="text" value="1"/>	Precio	<input type="text" value="0.050"/>	Descuento	<input type="text" value="5%"/>	Total	<input type="text" value="0.05"/>
<input type="button" value="Aceptar"/> <input type="button" value="Cancelar"/>							

Botón Aceptar: Transporta los valores que se encontraban en las cajas de textos a la tabla de detalle de venta, que se encuentra a continuación de los mismos en la parte inferior.

Botón Cancelar: Borra la información de las cajas de texto.

Eliminar o Modificar un detalle de venta

Para modificar o eliminar un detalle de venta se debe seleccionar de la tabla de detalle, al seleccionarlo se activan los botones Modificar y Eliminar que se encuentran en la parte derecha de la tabla, al presionar el botón Modificar la información es subida nuevamente a las cajas de texto para realizar las modificaciones necesarias. Mientras que al presionar el botón Eliminar es eliminado el detalle de venta seleccionado.

Cantidad	Producto	Valor U.	Descuento	Total	
1	Arandela Plana 1/4...	0.05	0.00	0.05	<input type="button" value="Modificar"/> <input type="button" value="Eliminar"/>

Restricciones




- El Usuario no puede agregar dos veces el mismo producto.
- La cantidad a vender no puede ser mayor al stock actual del producto, o ser menor que 1.
- El descuento no puede ser mayor que el 30% del valor del producto.

Totalización de la Venta

Este proceso se realiza automáticamente cada vez que se agregan o eliminan detalles de la venta, el único parámetro que puede ser cambiado es el porcentaje de impuesto IVA, que de manera predeterminada es 12%.

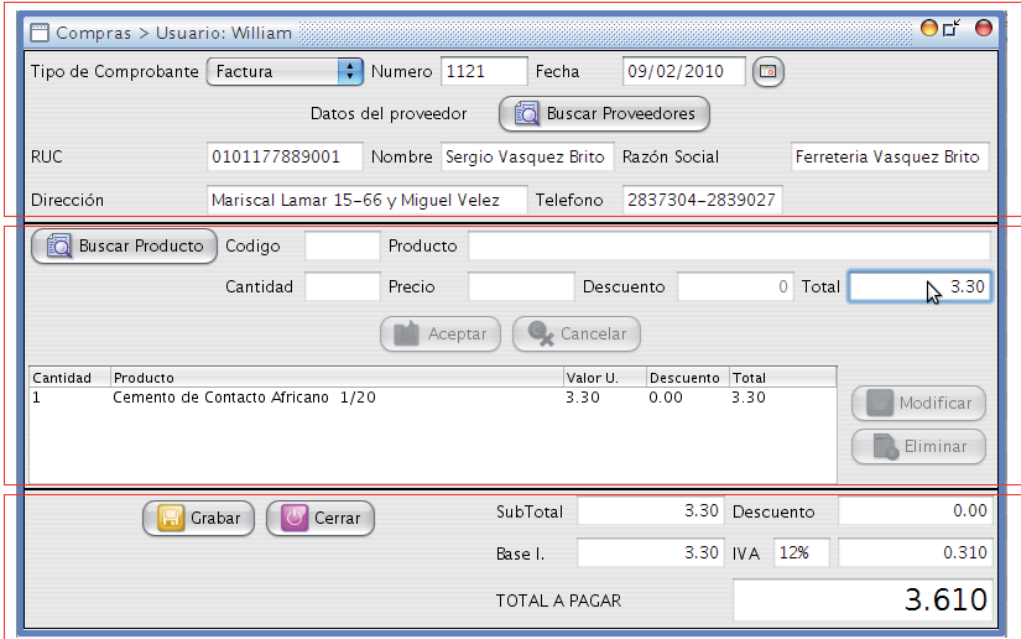
Luego que se han agregado todos los detalles de la venta se debe presionar el botón Grabar, el cual graba la venta con toda la información que está en pantalla. Una vez guardada la venta, esta no puede ser modificada y en los productos de la venta el stock es disminuido, según la cantidad vendida.

Al grabar la compra se habilita el botón Imprimir el cual permite imprimir el comprobante de venta correspondiente.

 Imprimir	 Grabar	 Cerrar	SubTotal	0.05	Descuento	0.00
			Base I.	0.05	IVA 12%	0.01
			TOTAL A PAGAR	0.05		

ESTRUCTURA DE LA VENTANA DE UN PROCESO DE COMPRAS

Para registrar una compra el usuario contará con la siguiente ventana:



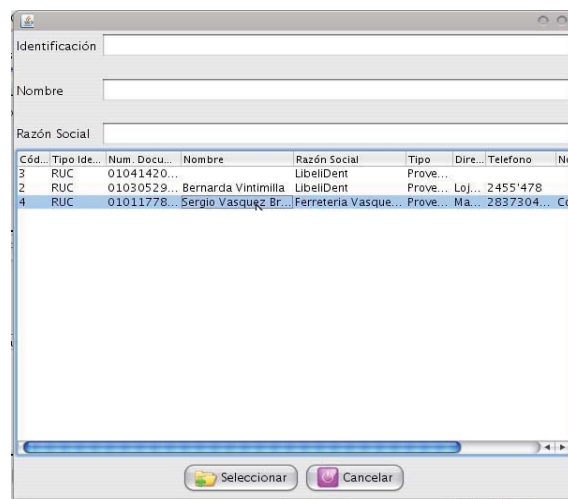
The screenshot shows a software window titled "Compras > Usuario: William". It is divided into three sections:


- SECCION SUPERIOR:** Contains fields for "Tipo de Comprobante" (Factura), "Numero" (1121), "Fecha" (09/02/2010), and "Datos del proveedor" (RUC: 0101177889001, Nombre: Sergio Vasquez Brito, Razón Social: Ferreteria Vasquez Brito, Dirección: Mariscal Lamar 15-66 y Miguel Velez, Telefono: 2837304-2839027). A "Buscar Proveedores" button is also present.
- SECCION CENTRAL:** Contains a "Buscar Producto" section with fields for "Codigo", "Producto", "Cantidad", "Precio", "Descuento", and "Total" (3.30). Below this is a table with one row: "Cemento de Contacto Africano 1/20" with a unit price of 3.30 and a total of 3.30. Buttons for "Aceptar", "Cancelar", "Modificar", and "Eliminar" are also visible.
- SECCION INFERIOR:** Contains a summary section with "Grabar" and "Cerrar" buttons, and a calculation table: SubTotal (3.30), Descuento (0.00), Base I. (3.30), IVA 12% (0.310), and TOTAL A PAGAR (3.610).

Esta está compuesta por tres secciones, en la parte superior se ingresará la información sobre el comprobante de venta, tal como la fecha, número de comprobante y el tipo de comprobante.

Elegir el Proveedor

Para elegir al proveedor al cual se le realizará la Compra, se presiona el botón Buscar

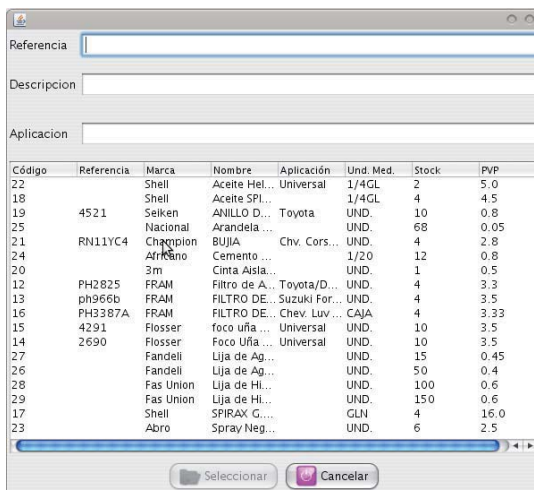


Proveedores 

el cual abre un cuadro de diálogo para buscar al proveedor, ya sea por su identificación (RUC, Cédula, etc.), Nombre o Razón Social. Luego se selecciona el proveedor desde los resultados obtenidos. Inicialmente se muestran todos los proveedores en la tabla.

Agregar Productos a la Compra

Para agregar productos a la compra, el usuario debe presionar el botón Buscar



Producto 

el cual abrirá un diálogo para seleccionar el producto a vender, se puede buscar el producto por su referencia, por su nombre, o por su aplicación, también se puede hacer una búsqueda combinada por ejemplo se puede buscar un producto por su nombre y su aplicación. Inicialmente se muestran todos los productos, cabe destacar que las búsquedas ignoran mayúsculas y

minúsculas. Para seleccionar el producto a comprar se debe seleccionar de la tabla y presionar el botón Seleccionar.

Luego de seleccionar el producto, se carga la información de Código y Descripción del Producto, las mismas no se pueden modificar, estos valores son llenados con la

información del producto seleccionado del cuadro de diálogo, mientras que cantidad, precio y descuento son editables.

Cantidad: Especifica el número de productos a comprar.

Precio: Indica el precio de compra del producto, sin contar los impuestos que serán calculados al final de la compra.

Descuento: Indica el valor de descuento individual para el producto, este puede ser escrito en porcentaje, por ejemplo si el proveedor nos da un descuento del 5%, en la caja de texto de descuento escribimos “5%” y el programa calculará el valor de descuento correspondiente.

Total: Este valor es calculado automáticamente según los datos de cantidad, precio y descuento, y se muestra cada vez que se cambie alguno de los valores anteriormente indicados.

Codigo	<input type="text" value="25"/>	Producto	Nacional Arandela Plana 1/4 UND.				
Cantidad	<input type="text" value="1"/>	Precio	<input type="text" value="0.050"/>	Descuento	<input type="text" value="5%"/>	Total	<input type="text" value="0.05"/>

Botón Aceptar: Transporta los valores que se encontraban en las cajas de texto a la tabla de detalle de compra, que se encuentra a continuación de los mismos en la parte inferior.

Botón Cancelar: Borra la información de las cajas de texto.

Eliminar o Modificar un Detalle de Compra

Para modificar o eliminar un detalle de compra se debe seleccionar de la tabla de detalle, al seleccionarlo se activan los botones Modificar y Eliminar que se encuentran en la parte derecha de la tabla, al presionar modificar la información es subida nuevamente a las cajas de texto, para realizar las modificaciones necesarias. Mientras que al presionar el botón eliminar es eliminado el detalle de compra seleccionado.

Cantidad	Producto	Valor U.	Descuento	Total
1	Arandela Plana 1/4...	0.05	0.00	0.05

Restricciones

- El Usuario no puede agregar dos veces el mismo producto.
- La cantidad a comprar no puede ser menor que 1.

Totalización de la compra

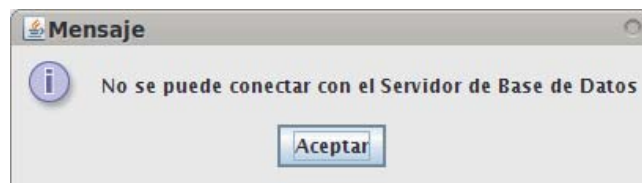
Este proceso se realiza automáticamente cada vez que se agregan o eliminan detalles de la compra, el único parámetro que puede ser cambiado es el porcentaje de impuesto IVA, que de manera predeterminada es 12%.

Luego que se han agregado todos los detalles de la compra se debe presionar la tecla Grabar, la cual graba la compra con toda la información que está en pantalla. Una vez Guardada la compra, esta no puede ser modificada. Al grabar la compra se habilita el botón Imprimir el cual permite imprimir el comprobante de compra correspondiente.

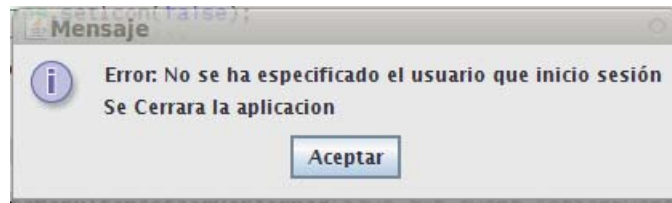
<input type="button" value="Imprimir"/> <input type="button" value="Grabar"/> <input type="button" value="Cerrar"/>	SubTotal <input type="text" value="0.05"/> Descuento <input type="text" value="0.00"/>
	Base I. <input type="text" value="0.05"/> IVA <input type="text" value="12%"/> <input type="text" value="0.01"/>
	TOTAL A PAGAR <input type="text" value="0.05"/>

MENSAJES DE ADVERTENCIA

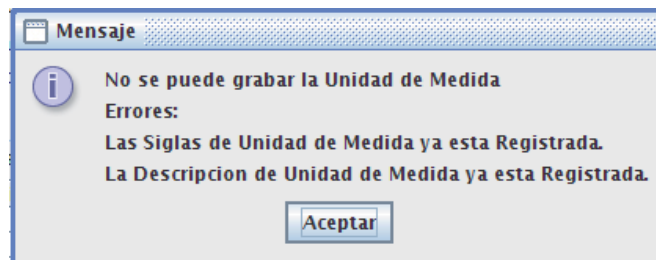
Ejecución del sistema teniendo deshabilitada la Base de Datos:



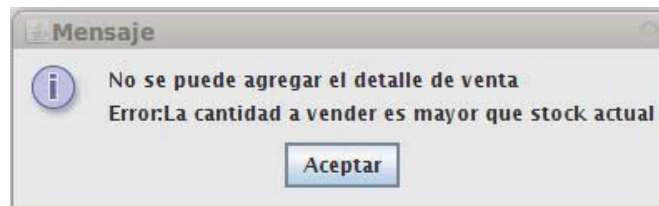
Invocación directa a un módulo o clase sin antes autenticarse:



Cuando un mismo campo es ingresado dos veces, por ejemplo para unidades de medida:



Cuando la cantidad a vender es mayor que la cantidad en Stock:



CONCLUSIONES

Concluido el proyecto de estudio y en base a las pruebas finales, así como los diferentes módulos implementados como parte del desarrollo de la aplicación y del sistema completo, podemos terminar diciendo que se ha realizado con éxito el estudio en el que se determinó las principales metodologías de desarrollo de Software Libre, se analizó las ventajas y desventajas de cada una de ellas y además se determinó los casos en las que pueden ser usadas, este análisis nos permitió elaborar una metodología propia constituida como un abstracto de todas las estudiadas en las que se enfocan ideas de cómo desarrollar un software de alta calidad desde proyectos pequeños hasta grandes aplicaciones.

El objetivo con esta metodología propuesta no es formar un estándar para el desarrollo, sino más bien ser una guía técnica que marque el camino a los desarrolladores para lograr un producto final de calidad en su estructura y funcionamiento y de provecho para quién lo utilice.

Durante el estudio tuvimos la gran oportunidad de conversar con la persona más influyente del Software Libre, Richard Stallman, quién a través de sus comentarios y vivencias nos dio su punto de vista acerca de las metodologías de desarrollo de Software Libre, su comentario no fue muy alentador ya que concluyó diciendo que un software al ser libre no tiene porque ser realizado a través de una metodología, desafortunadamente no compartimos su idea y por ello decidimos demostrar que si realizamos una aplicación con una metodología en la cual basarnos, de seguro lograremos un producto de calidad y en el menor tiempo.

Con respecto al desarrollo de la aplicación, se implementó un software en el que se gestionó de manera práctica las compras, ventas y la asistencia en el manejo de impuestos usando herramientas de desarrollo de Software Libre y se lo validó al aplicarlo en la empresa de venta de Repuestos “Ferrorepuestos JW”; dicho proyecto se lo realizó siguiendo los lineamientos de la metodología propuesta, la cual nos permitió la generación de la siguiente documentación: un documento de estudio del negocio, documento de requerimientos, documento de análisis y diseño, documento

de implementación del proyecto y un documento de pruebas, los cuales están plasmados en el Capítulo 7.

Como toda solución de software que se encuentra en el mercado, siempre sigue estando sujeta a mejoras, y esta no es la excepción; dejamos liberado el código fuente para que reciba mejoras de toda la comunidad, siempre y cuando se respeten los términos legales que involucra adquirir un producto de software regido por una licencia de Software Libre.

Lo antes mencionado junto al sistema desarrollado cubren los objetivos planteados, contando finalmente con una guía técnica de apoyo la cual está presta a ayudar a quién quiera beneficiarse de ella. Es importante no abandonar el desarrollo y crecimiento del sistema, por lo que recomendamos aplicar la metodología para proyectos grandes, lo que garantizará un óptimo desarrollo y un producto final de alta calidad.

RECOMENDACIONES

Al término del proyecto hemos logrado obtener una guía técnica fundamentada por los conceptos y prácticas más influyentes de varias de las metodologías existentes hoy en día, por ello como estudiantes que fuimos una vez, nos atrevemos a recomendarla para el estudio en la materia de Ingeniería de Software de nuestra carrera.

Si bien es cierto, una metodología no puede quedarse como un estándar rígido, sino que siempre debe estar adaptándose al mundo cambiante en el que vivimos, invitamos a seguirla y conocer todos sus componentes producto de la investigación que hemos realizado a lo largo del proyecto.

De igual manera sugerimos a nuestra universidad, incentivar a todos los estudiantes de la carrera a descubrir el mundo del Software Libre, darles a conocer todas las ventajas que conlleva desarrollar o mejorar un producto sin las ataduras que antes teníamos con el software no libre. Podrían implementar un portal donde los estudiantes tengan su propio espacio y puedan subir sus aplicaciones con el fin de que sean mejoradas y así fomentar la integración de nuevos miembros a la gran comunidad de usuarios de Software Libre.

Invitamos a todos los estudiantes a la investigación y continuo aprendizaje de nuevas tecnologías en Software Libre, solo así formaremos una sociedad tecnológica libre de poder acceder, usar y modificar el código fuente de un software, siempre y cuando tengamos en cuenta las implicaciones legales que estas prácticas conllevan.

ANEXOS

Anexo 1

Decreto presidencial 1014

El 10 de abril del 2008 se firmó el decreto presidencial 1014 en el que se establece como política pública para las entidades de la Administración Pública Central la utilización de Software Libre en sus sistemas y equipamientos informáticos.

RAFAEL CORREA DELGADO

PRESIDENTE CONSTITUCIONAL DE LA REPÚBLICA

CONSIDERANDO:

Que en el apartado g) del numeral 6 de la Carta Iberoamericana de Gobierno Electrónico, aprobada por el IX Conferencia Iberoamericana de Ministros de Administración Pública y Reforma del Estado, realizada en Chile el 1 de Junio de 2007, se recomienda el uso de estándares abiertos y software libre, como herramientas informáticas;

Que es el interés del Gobierno alcanzar soberanía y autonomía tecnológica, así como un significativo ahorro de recursos públicos y que el Software Libre es en muchas instancias un instrumento para alcanzar estos objetivos;

Que el 18 de Julio del 2007 se creó e incorporó a la estructura orgánica de la Presidencia de la República la Subsecretaría de Informática, dependiente de la Secretaría General de la Administración, mediante Acuerdo Nº119 publicado en el Registro Oficial No. 139 de 1 de Agosto del 2007;

Que el numeral 1 del artículo 6 del Acuerdo Nº 119, faculta a la Subsecretaría de Informática a elaborar y ejecutar planes, programas, proyectos, estrategias, políticas, proyectos de leyes y reglamentos para el uso de Software Libre en las dependencias del gobierno central; y,

En ejercicio de la atribución que le confiere el numeral 9 del artículo 171 de la Constitución Política de la República;

DECRETA:

Artículo 1.- Establecer como política pública para las Entidades de la Administración Pública Central la utilización de Software Libre en sus sistemas y equipamientos informáticos.

Artículo 2.- Se entiende por Software Libre, a los programas de computación que se pueden utilizar y distribuir sin restricción alguna, que permitan su acceso a los códigos fuentes y que sus aplicaciones puedan ser mejoradas.

Estos programas de computación tienen las siguientes libertades:

- a) Utilización del programa con cualquier propósito de uso común
- b) Distribución de copias sin restricción alguna.
- c) Estudio y modificación del programa (Requisito: código fuente disponible)
- d) Publicación del programa mejorado (Requisito: código fuente disponible).

Artículo 3.- Las entidades de la Administración Pública Central previa a la instalación del software libre en sus equipos, deberán verificar la existencia de capacidad técnica que brinde el soporte necesario para el uso de este tipo de software.

Artículo 4.- Se faculta la utilización de software propietario (no libre) únicamente cuando no exista una solución de Software Libre que supla las necesidades requeridas, o cuando esté en riesgo la seguridad nacional, o cuando el proyecto informático se encuentre en un punto de no retorno.

Para efectos de este decreto se comprende como seguridad nacional, las garantías para la supervivencia de la colectividad y la defensa del patrimonio nacional.

RAFAEL CORREA DELGADO

PRESIDENTE CONSTITUCIONAL DE LA REPÚBLICA

Para efectos de este decreto se entiende por un punto de no retorno, cuando el sistema o proyecto informático se encuentre en cualquiera de estas condiciones:

- a) Sistema en producción funcionando satisfactoriamente y que un análisis de costo beneficio muestre que no es razonable ni conveniente una migración a Software Libre.
- b) Proyecto en estado de desarrollo y que un análisis de costo - beneficio muestre que no es conveniente modificar el proyecto y utilizar Software Libre.

Periódicamente se evaluarán los sistemas informáticos que utilizan software propietario con la finalidad de migrarlos a Software Libre.

Artículo 5.- Tanto para software libre como software propietario, siempre y cuando se satisfagan los requerimientos, se debe preferir las soluciones en este orden:

- a) Nacionales que permitan autonomía y soberanía tecnológica.
- b) Regionales con componente nacional.
- c) Regionales con proveedores nacionales.
- d) Internacionales con componente nacional.
- e) Internacionales con proveedores nacionales.
- f) Internacionales.

Artículo 6.- La Subsecretaría de Informática como órgano regulador y ejecutor de las políticas y proyectos informáticos en las entidades del Gobierno Central deberá realizar el control y seguimiento de este Decreto.

Para todas las evaluaciones constantes en este decreto la Subsecretaría de Informática establecerá los parámetros y metodología obligatorias.

Artículo 7.- Encárguese de la ejecución de este decreto los señores Ministros Coordinadores y el señor Secretario General de la Administración Pública y Comunicación.

Dado en el Palacio Nacional en la ciudad de San Francisco de Quito, Distrito Metropolitano, el día de hoy **10 de abril de 2008**

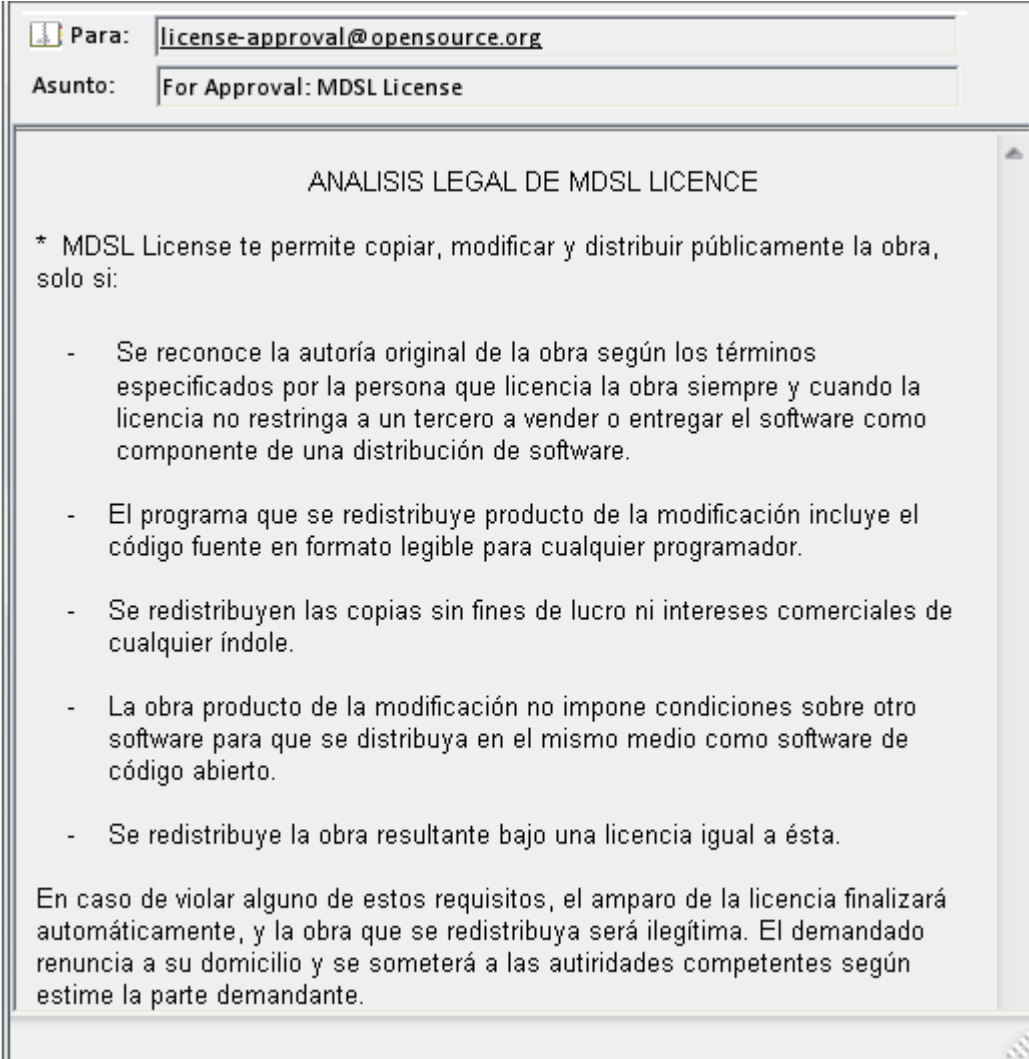


Rafael Correa Delgado
PRESIDENTE CONSTITUCIONAL DE LA REPÚBLICA

Anexo 2

Enviar un email a opensource.org

A continuación se muestra un ejemplo de cómo enviar un email a opensource.org para aprobar una licencia nueva.



Para: license-approval@opensource.org

Asunto: For Approval: MDSL License

ANALISIS LEGAL DE MDSL LICENCE

* MDSL License te permite copiar, modificar y distribuir públicamente la obra, solo si:

- Se reconoce la autoría original de la obra según los términos especificados por la persona que licencia la obra siempre y cuando la licencia no restrinja a un tercero a vender o entregar el software como componente de una distribución de software.
- El programa que se redistribuye producto de la modificación incluye el código fuente en formato legible para cualquier programador.
- Se redistribuyen las copias sin fines de lucro ni intereses comerciales de cualquier índole.
- La obra producto de la modificación no impone condiciones sobre otro software para que se distribuya en el mismo medio como software de código abierto.
- Se redistribuye la obra resultante bajo una licencia igual a ésta.

En caso de violar alguno de estos requisitos, el amparo de la licencia finalizará automáticamente, y la obra que se redistribuya será ilegítima. El demandado renuncia a su domicilio y se someterá a las autoridades competentes según estime la parte demandante.

Anexo 3

Fotografía junto a Richard Stallman, padre del Software Libre



BIBLIOGRAFIA

- Decreto Presidencial 1014, Rafael Correa Delgado, 10-abril-2008,
http://ia360931.us.archive.org/0/items/decreto/Decreto_1014_software_libre_Ecuador.pdf,
Fecha de Consulta: 25 de Junio del 2009.
- Manual de Copyleft, “Porqué se produce software libre”,
http://www.manualcopyleft.net/wiki/index.php/Gu%C3%ADa_del_software_libre#.C2.BFPor_qu.C3.A9_se_produce_software_libre.3F,
Fecha de Consulta: 03 de Julio del 2009.
- Diseño de Sistemas, “Racional Unified Process”,
http://www.slideshare.net/punk.kekito/diseo-de-sistemas-presentation?src=related_normal&rel=1341543,
Fecha de Consulta: 06 de Noviembre del 2009.
- Wikipedia.org, “Metodología de desarrollo de software”,
http://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software,
Fecha de Consulta: 15 de Agosto del 2009.
- Rational Unified Process , "Best Practices for Software Development Teams",
http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf,
Fecha de Consulta: 29 de Noviembre del 2009.
- Ingeniería del Software II, “Introducción a Extreme Programming”,
Gerardo Fernández Escribano,
<http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf>,
Fecha de consulta: 15 de noviembre de 2009.
- INEN Instituto Ecuatoriano de Normalización, “NORMA TECNICA ECUATORIANA NTE ISO/IEC 26300:2009”,
http://www.inen.gov.ec/normas/norma.php?COD_NORMA=2844,
Fecha de Consulta 15 de Septiembre del 2009.

- Open Source Initiative, “OSI Certification Mark and Program”,
http://opensource.linux-mirror.org/docs/certification_mark.php,
Fecha de Consulta: 25 de Septiembre del 2009.
- Charity Navigator, “Calificación de la FSF”,
<http://charitynavigator.org/index.cfm?bay=search.summary&orgid=8557>,
Fecha de Consulta 18 de Septiembre del 2009.
- NONIUS, Jorge: Introducción a la propiedad intelectual,
<http://www.laespiral.org/articulos/propintlect/propintlect.pdf>, página 43,
Fecha de Consulta 09 de Septiembre del 2009.
- Free Software Foundation, “Copyleft”,
<http://www.gnu.org/copyleft/copyleft.es.html>,
Fecha de Consulta 07 de Septiembre del 2009.
- Wikipedia.org, “Validez Jurídica”,
http://es.wikipedia.org/wiki/Validez_jur%C3%ADdica.html,
Fecha de Consulta 06 de Septiembre del 2009.
- FSF Free Software Foundation, “What is Copyleft”,
<http://www.fsf.org/licensing/essays/copyleft.html/view?searchterm=copyleft>,
Fecha de Consulta 10 de Septiembre del 2009.
- GNU Operating System, “El problema de la licencia BSD”,
<http://www.gnu.org/philosophy/bsd.es.html>,
Fecha de Consulta 08 de Septiembre del 2009.
- Ley de propiedad intelectual Ecuador,
<http://www.gocomputer.com.ec/faqdemo/pdf.php?cat=4&id=4&lang=es>,
Fecha de Consulta 16 de Julio del 2009.
- Microsoft Licencias, “Explicación de las Licencias de Microsoft”,
<http://www.microsoft.com/spain/licencias/novedades/explicacion.msp>,
Fecha de consulta 26 de Junio del 2009.
- OMPI Servicios, “Preguntas Frecuentes”,
http://www.wipo.int/patentscope/es/patents_faq.html#patent,
Fecha de Consulta 01 de Julio del 2009.