DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

TESIS PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS E INFORMÁTICA

TEMA: DESARROLLO DE UNA APLICACIÓN DISTRIBUIDA SOBRE ARQUITECTURA MULTI CAPAS, CASO PRÁCTICO MÓDULO EVALUACIÓN RR.HH BASADO EN COMPETENCIAS – ZEUZ SISTEMAS

AUTOR: NARVÁEZ VARGAS GUILLERMO GERMÁN

DIRECTOR: ING. GERMÁN ÑACATO CAIZA

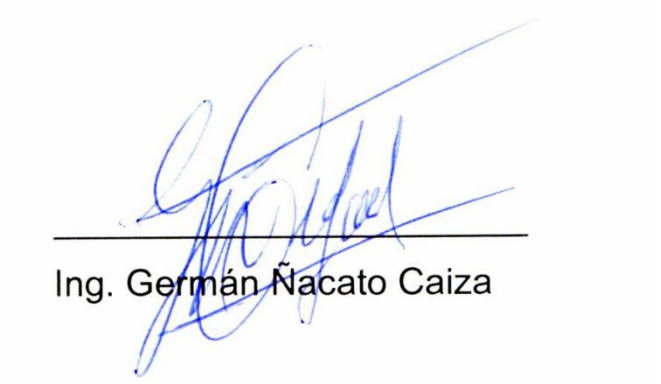
SANGOLQUÍ, MAYO 2015

UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE
CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

CERTIFICACIÓN

Certifico que el presente trabajo titulado "DESARROLLO DE UNA APLICACIÓN DISTRIBUIDA SOBRE ARQUITECTURA MULTI CAPAS, CASO PRÁCTICO MÓDULO EVALUACIÓN RR.HH BASADO EN COMPETENCIAS – ZEUZ SISTEMAS" fue realizado en su totalidad por el Sr. GUILLERMO GERMÁN NARVÁEZ VARGAS, el cual ha sido periódicamente revisado, guiado y cumple las normas estatutarias establecidas por la Universidad de las Fuerzas Armadas – ESPE, como requerimiento parcial a la obtención del título de INGENIERO EN SISTEMAS E INFORMÁTICA.

Sangolquí, Mayo 2015



Ing. Germán Ñacato Caiza

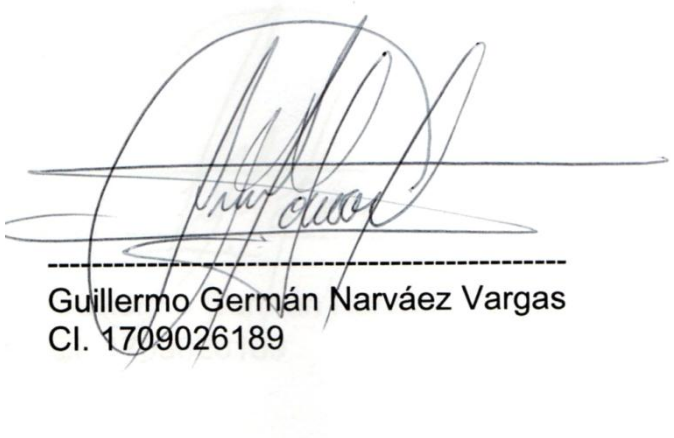
UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE
CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

DECLARACIÓN DE RESPONSABILIDAD

Yo, GUILLERMO GERMÁN NARVÄEZ VARGAS, declaro que el presente trabajo de tesis de grado titulado “DESARROLLO DE UNA APLICACIÓN DISTRIBUIDA SOBRE ARQUITECTURA MULTI CAPAS, CASO PRÁCTICO MÓDULO EVALUACIÓN RR.HH BASADO EN COMPETENCIAS – ZEUZ SISTEMAS”, ha sido desarrollado en base a una investigación exhaustiva, respetando derechos intelectuales de terceros, conforme citas que constan en la bibliografía.

Consecuentemente este trabajo es de mi autoría, y me responsabilizo del contenido, veracidad y alcance del proyecto de grado en mención.

Sangolquí, Mayo 2015



Guillermo Germán Narváez Vargas
CI. 1709026189

UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE
CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

AUTORIZACIÓN DE PUBLICACIÓN

Guillermo Germán Narváez Vargas

AUTORIZA

A la Universidad de las Fuerzas Armadas – ESPE la publicación en la Biblioteca Virtual de la Institución del trabajo “DESARROLLO DE UNA APLICACIÓN DISTRIBUIDA SOBRE ARQUITECTURA MULTI CAPAS, CASO PRÁCTICO MÓDULO EVALUACIÓN RR.HH BASADO EN COMPETENCIAS – ZEUZ SISTEMAS”, cuyo contenido, ideas y criterio son de mi responsabilidad y autoría.

Consecuentemente este trabajo es de mi autoría, en virtud de esta declaración, me responsabilizó del contenido, veracidad y alcance del proyecto de grado en mención.

Sangolquí, Mayo de 2015

Guillermo Germán Narváez Vargas
CI. 1709026189

DEDICATORIA

Este trabajo está dedicado con mucho amor a mi madre Gladicita Vargas Parra, que con su apoyo, dedicación y fortaleza siempre ha estado a mi lado en forma incondicional, y a mi hija Kamila Narváez que es el motor de mi vida para ser su ejemplo de superación. Les dedico este trabajo con mucho amor para su alegría y complacencia.

Guillermo Germán Narváez Vargas

AGRADECIMIENTO

A Dios en primer lugar porque es mi guía y soporte en todas mis actividades diarias, que conoce mis anhelos, me bendice y siempre protege mi vida.

A mis padres Guillermo y Gladys por ser ejemplo de constancia y perseverancia, y por confortarme en los momentos de angustia.

A mi hija Kamila, que con su corta edad me apoya y siempre está para darme esos momentos de felicidad.

A toda mi familia que de una u otra forma siempre estuvieron pendientes de mi crecimiento personal y profesional.

Al mis directores del proyecto y más colaboradores por ayudarme a conseguir un trabajo de excelente calidad.

Guillermo Germán Narváez Vargas

ÍNDICE DE CONTENIDO

LISTADO DE TABLAS

LISTADO DE FIGURAS

RESUMEN

La presente tesis tiene como propósito revisar, analizar, comparar las tres metodologías de desarrollo de software (RUP, XP, MFS), con lo cual se define una guía metodológica basada en fases, disciplinas, roles, artefactos, patrones, los cuales se utilizaran para el desarrollo de software, que podrán ser utilizados en proyectos de distinto tamaño y complejidad. Para llevar a cabo este trabajo se recabó información detallada de: Procesos de Desarrollo, Metodologías Ágiles, Sistemas Operativos, Bases Distribuidas, Paradigma Objetos, Herramientas de Análisis y Diseño y Recursos Humanos basado en Competencias, lo que permitió tener una visión global de toda la problemática y el conocimiento necesario para la definición de los aspecto relevantes y la elaboración del Sistema de Evaluación Basado en Competencias. Este marco teórico definido se utilizó en el desarrollo de la aplicación, la cual es un software a la medida que cumple los objetivos y requerimientos del cliente.. El producto de software resultante es un sistema web distribuido multi capa, en lenguaje de programación Visual Basic.Net, almacenando la información en SQLServer 2008 como motor de base de datos y Crystal Reporte como visor de reportes. El sistema desarrollado Evaluación de Recursos Humanos Basados en Competencias, permite identificar el clima laboral relacionado directamente con el puesto y estima el rendimiento global, esta evaluación se basa en competencias claramente identificadas utilizando el método 360 grados.

PALABRAS CLAVES:

- METODOLOGÍA
- APLICACIÓN
- DISTRIBUIDA
- EVALUACION
- COMPETENCIAS

ABSTRACT

This thesis aims to review, analyze , compare the three software development methodologies (RUP , XP , MFS) , whereby a methodological guide built in phases, disciplines , roles, artifacts , patterns , defined which will be used for software development , which can be used in projects of different size and complexity. To carry out this work was gathered detailed information : Process Development, Agile , Operating Systems , Distributed Databases , Object Paradigm , Design and Analysis Tools Human Resources and Skills based , allowing an overall view of the entire issues and the need for the definition of the relevant aspect and the development of evaluation system based on knowledge skills. This defined framework was used in the development of the application, which is a custom software that meets the objectives and customer requirements .. The resulting software is a distributed multi- layer web system , programming language Visual Basic .net , storing information in SQLServer 2008 as database engine and as Crystal Report viewer reports. The assessment system developed Competency Based Human Resources , identifies the work environment directly related to the position and estimated the overall performance , this assessment is based on clearly identified competences using the method 360 degrees.

KEYWORDS:

- METHODOLOGY
- APLICATION
- DISTRIBUTED
- EVALUATION
- COMPETITIONS

CAPÍTULO 1: INTRODUCCIÓN

1.1. TEMA

Desarrollo de una Aplicación Distribuida sobre Arquitectura Multi-Capas, caso práctico Módulo Evaluación de RR.HH basado en Competencias – "Zeuz Sistemas"

1.2. INTRODUCCIÓN

Según la definición del IEEE, citada por Lewis 1994, "**Software** es la suma total de los programas de computadora, procedimientos, reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo". Según el mismo autor, "**un producto de software** es un producto diseñado para el usuario". En este contexto, la Ingeniería de Software es un enfoque sistemático del desarrollo, operación, mantenimiento y retiro del software, que en palabras más llanas, se considera que la "**Ingeniería de Software**" es la rama de la ingeniería que aplica los principios de la ciencia de la computación y las matemáticas para lograr soluciones costo-efectivas a los problemas de desarrollo de software, es decir permite elaborar consistentemente productos correctos, utilizables y costo-efectivos.

El proceso de Ingeniería de Software se define como "un conjunto de etapas parcialmente ordenadas con la intención de lograr un objetivo, en este caso, la obtención de un producto de software de calidad". El proceso de desarrollo de software es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo. Concretamente define "Quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo"

El proceso de desarrollo de software requiere por un lado un conjunto de conceptos, una metodología y un lenguaje propio. A este proceso también se lo llama "**ciclo de vida del software**" que comprende cuatro grandes fases: concepción, elaboración, construcción y transición. La

concepción define el alcance del proyecto y desarrolla un caso de negocio. La elaboración define un plan del proyecto, especifica las características y fundamenta la arquitectura. La construcción crea el producto y la transición transfiere el producto a los usuarios.

1.3. POSICIONAMIENTO DEL TEMA

La tendencia actual en el campo empresario, es la formación de los consorcios, esto es, varias empresas con una administración de alto nivel común. Para lograr alcanzar las metas del consorcio, se selecciona una de ellas como eje central de la organización quien es la encargada de controlar los procesos administrativo-financieros del consorcio, sin que esto afecte a la independencia y autonomía de las empresas individuales, dando como resultado la centralización de la información y una evidente disminución de costos administrativos, pero a la vez, todos los procesos de transferencia de información son manuales, teniendo como consecuencia que la información no este actualizada y tengan que pasar largos lapsos de tiempo para que la misma sea procesada.

Con la **Distribución de la Información** y el enfoque **Orientado a Objetos**, se pretende optimizar la ventajas y disminuir las desventajas de los Sistemas Centralizados y Distribuidos, es decir, mantener una independencia y autonomía de las empresas, centralizar ciertos procesos administrativo-financieros, utilizar los recursos de hardware existentes y crear Sistemas Integrados de Información en Línea, de tal forma que permitan la toma oportuna de decisiones.

1.4. JUSTIFICACIÓN E IMPORTANCIA

En una sociedad capitalista la competencia y la rentabilidad son factores decisivos en el éxito o fracaso de las organizaciones, razón por la cual es de mucha importancia que el tratamiento de la información y la disminución de costos se conviertan en herramientas que permitan estar a la vanguardia.

Un sistema de Bases de Datos Relacionales Distribuidas (RDDB) está formado por un conjunto de servidores, cada uno de los cuales mantiene una Base de Datos Local, en el cual se procesa **transacciones locales** y además participa en **transacciones globales**; estas últimas utilizan diferentes servidores. Los datos se almacenan en varios computadores, estos computadores se comunican entre si a través de diversos medios de comunicación, tales como cables de alta velocidad, líneas telefónicas, enlaces de radio frecuencia, etc.

Actualmente se encuentra en una etapa de madurez el enfoque **Orientado a Objetos** (OO) como paradigma de desarrollo de sistemas de información. El **Object Management Group** (OMG) es un consorcio a nivel internacional que integra a los principales representantes de la industria de la tecnología de información OO. El OMG tiene como objetivo central la promoción, fortalecimiento e impulso de la industria OO. EL OMG propone y adopta por consenso especificaciones entorno a la tecnología OO. Una de las especificaciones más importantes es la adopción en 1998 del Lenguaje de Modelado Unificado (UML) como un estándar, que junto con el Proceso Unificado están consolidando la tecnología OO.

Por lo tanto, desde el punto de vista técnico es muy importante desarrollar el tema, por cuanto todos saben **¿Qué hacer?** Y muy pocos **¿Cómo Hacer?**. La distribución de la Información en una arquitectura Multi Capas, debe ser completamente entendida a tal punto de diseñar sistemas que permitan:

Maximizar:

- Compartición de datos y recursos
- Disminución de Costos
- Agilitar el proceso de Consultas
- Transparencia y Autonomía
- Mayor Control

Minimizar:

- Costo de desarrollo de Software
- Probabilidad de errores

- Tiempo extra de procesamiento
- Tiempos de respuesta

1.5. OBJETIVOS

1.5.1. Objetivo General

- Desarrollar una Aplicación Distribuida Aplicar los aspectos relevantes en las metodologías para desarrollar Aplicaciones Distribuidas, sobre Arquitectura Multi Capas aplicado al Módulo de Evaluación de Recursos Humanos basado en Competencias de "Zeuz Sistemas"

1.5.2. Objetivos Específicos

- Recopilar, Revisar y Documentar información de Sistemas Distribuidos, Bases de Datos Distribuidas y Metodologías de Desarrollo de Software.
- Definir una guía metodológica mediante la comparación de las características y componentes de 3 Metodologías (XP, MSF, RUP).
- Diseñar y Desarrollar el Módulo de Evaluación de RR.HH, utilizando la guía metodológica definida para Zeuz Sistemas.
- Implementar y Puesta en Marcha del Módulo desarrollado en Zeuz Sistemas.

1.6. ALCANCE

Realizar el Análisis de Aplicaciones Distribuidas, Bases de Datos Distribuidas y Metodologías en Arquitectura Multi Capas y Orientada a Objetos usando el protocolo TCP/IP. Con estas bases teóricas, se definirá los aspectos relevantes de las metodólogas que permita desarrollar sistemas distribuidos utilizando de hardware existente (PC's).

Con los aspectos propuestos se desarrollará el "Módulo Evaluación de Recursos Humanos basado en Competencias para "Zeuz Sistemas"

El trabajo en mención se llevará a efecto en 6 fases.

- Relevamiento de la Información
- Definición de guía Metodológica
- Relevamiento de Procesos
- Diseño del Sistema y Red
- Desarrollo y Programación
- Implementación y Puesta en Marcha
- Conclusiones y Recomendaciones

1.7. DESCRIPCIÓN DE LAS FASES

1.7.1. Relevamiento de Información

En esta fase se recopilará toda la información necesaria referente al software, hardware, comunicación y análisis.

- Sistemas Distribuidos
- Protocolo TCP/IP
- Bases de Datos Relacionales Distribuidas
- Paradigma de lo Orientado a Objetos
- Herramientas Análisis y Diseño O.O.
- Especificación de Requerimientos según IEEE
- Evaluación Recursos Humanos Basado en Competencias

1.7.2. Análisis y Definición de Guía Metodología

En función de la información recopilada acerca de Sistemas Distribuidos y la Tecnología Orientada a Objetos, se realizará el análisis y definición de la guía metodología de desarrollo e implementación de sistemas, especialmente aplicado a los requerimientos de “Zeuz Sistemas”

- Fundamentos de lo Orientado a Objetos
- Análisis Orientado a Objetos (OOA)
- Diseño Orientado a Objetos (OOD)
- Programación Orientada a Objetos (OOP)
- Lenguaje de Modelado Unificado (UML)
- Análisis y Requerimiento de Hardware
- Sistemas Operativos

- Sistemas de Administración de Bases de Datos
- Software de Comunicación

1.7.3. Relevamiento Procesos

En esta fase se realizará toda la recopilación de la información y los procesos que intervienen en la Evaluación de los Recursos Humanos basado en Competencias. Los procesos que intervienen son:

- Competencias
- Estructura Organizacional
- Cuestionarios de Evaluación
- Valoración de Cuestionarios
- Procesos Evaluación 360 grados
- Índices y Parámetros de Evaluación
- Índices y Parámetros de Desempeño

1.7.4. Diseño de Sistema y Red

En base a la guía metodológica seleccionada y el relevamiento de procesos, se procederá al diseño del Sistema de Evaluación de Recursos Humanos basado en Competencias y de la red, incluyendo el hardware existente.

1.7.5. Desarrollo y Programación

En esta fase se desarrollará y codificará el sistema propuesto para lo cual se utilizará herramientas de análisis y programación orientada a Objetos.

1.7.6. Implementación y Puesta en Marcha

En esta fase se implementará y pondrá en marcha los sistemas desarrollados, considerando el plan de implementación definido con el usuario, se incluye la conversión de datos históricos y la capacitación a usuarios finales, para esta fase se tendrá la colaboración del personal de “Zeuz Sistemas”

1.7.7. Elaboración de Conclusiones y Recomendaciones

En esta fase se elaborará un análisis de costo-beneficio de la utilización de la guía metodología definida.

CAPÍTULO 2: RELEVAMIENTO DE INFORMACIÓN

2.1. INTRODUCCIÓN

Según la IEEE Computer Society, Ingeniería de Software se define como: “La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del Software”.

Esta definición permite la profesionalización de la disciplina de la informática dentro del conjunto de las ingenierías, mediante un constante esfuerzo de la comunidad informática en el cambio y su mejoramiento. El objetivo está en la definición de un conjunto de patrones y estándares a ser aplicados en el desarrollo de software. Según Frederick Brooks [Brooks, 1987] las características esenciales de una metodología son las siguientes:

- **Complejidad**, las entidades de software son complejas de acuerdo a su tamaño.
- **Conformidad**, la complejidad lleva a conformar múltiples interfaces heterogéneas que involucran a distintas personas
- **Maleabilidad**, las entidades de software están sujetas a cambios constantemente; dado que el software es materia puramente intelectual y que su cambio es percibido como algo sencillo el mismo suele sufrir este efecto
- **Invisibilidad,** lo que genera dificultades en la comunicación para el modelado de los sistemas

Una METODOLOGÍA, es un conjunto de métodos empleados para el desarrollo de sistemas automatizados. Una metodología completa es algo más que una notación, un proceso y herramientas. Además de una "notación, de un proceso, y de herramientas," estas "metodologías completas" proporcionan:

- Guías para estimar costos,
- Manejo del proyecto las tareas y entregas,
- Medidas y métricas,
- Formas definidas y dirección las entregas de la construcción.

- Políticas y procedimientos para garantizar la calidad del software,
- Descripciones de los roles y programas de entrenamiento detallados,
- Ejemplos totalmente trabajados,
- Ejercicios de entrenamiento,
- Técnicas para adaptar el método, y
- Técnicas definidas

2.2. EVOLUCIÓN DE LAS METODOLOGÍAS DE DESARROLLO

Uno de los grandes pasos dados en la industria del software fue aquel en que se plasmó el denominado modelo en **cascada**. Dicho modelo sirvió como base para la formulación del análisis estructurado, el cual fue uno de los precursores en este camino hacia la aplicación de prácticas estandarizadas dentro de la ingeniería de software. Este modelo se basaba en el desarrollo de etapas las cuales tenían un input y un output predefinido. El proceso era desarrollado en forma de cascada ya que se requería la finalización de la etapa anterior para empezar la siguiente. Esto degeneraba en un "congelamiento" temprano de los requerimientos, los cuales al presentar cambios requerían gran esfuerzo en retrabajo.

De esta forma y en forma bastante temprana en algunos casos, fueron surgiendo diversos procesos denominados **iterativos** que proponían lidiar con la impredictibilidad del software (subsanando muchas de las falencias del modelo en cascada) mitigando los riesgos en forma temprana. Los procesos iterativos de los cuales se desprenderían diversas instancias, como son el modelo iterativo e incremental, el modelo en espiral, el modelo basado en prototipo, el modelo SLCD, el MBASE, el RUP, etc.

TABLA 1
Evolución Metodologías de Desarrollo

Modelo	Versión de origen	Características
Modelo en cascada	Secuencial: Bennington 1956 – Iterativo: Royce 1970 – Estándar DoD 2167-A	Secuencia de requerimiento, diseño del sistema, diseño de programa, codificación, pruebas, operación y mantenimiento

CONTINÚA →

Modelo en cascada c/ fases superpuestas	Cf. McConnell 1996:143-144	Cascada con eventuales desarrollos en paralelo (Modelo Sashimi)
Modelo iterado con prototipado	Brooks 1975	Iterativo – Desarrollo incremental
Desarrollo rápido (RAD)	J. Martin 1991 – Kerr/Hunter 1994 – McConnell 1996	Modelo lineal secuencial con ciclos de desarrollo breves
Modelo V	Ministerio de Defensa de Alemania 1992	Coordinación de cascada con iteraciones
Modelo en espiral	Barry Boehm 1988	Iterativo – Desarrollo incremental. Cada fase no es lineal, pero el conjunto de fases sí lo es.
Modelo en espiral win-win	Barry Boehm 1998	Iterativo – Desarrollo incremental – Aplica teoría-W a cada etapa
Modelo de desarrollo concurrente	Davis y Sitaram 1994	Modelo cíclico con análisis de estado
Modelo de entrega incremental *(Staged delivery)*	McConnell 1996: 148	Fases tempranas en cascada – Fases ulteriores descompuestas en etapas

Fuente: Autor

2.3. METODOLOGÍAS DE DESARROLLO

A principios de la década del ’90, surgió un enfoque de lograr obtener software en tiempo, costo y con la requerida calidad, este consistía en un entorno de desarrollo altamente productivo, en el que participaban grupos pequeños de programadores utilizando herramientas que generaban código en forma automática tomando como entradas sintaxis de alto nivel. Cabe mencionar que tanto las metodologías tradicionales como las no tradicionales no inventaron la noción de los procesos iterativos e incrementales, los cuales eran usados desde décadas pasadas inclusive en momentos en que el modelo en cascada era el estándar. Entre las metodologías de desarrollo más destacadas hasta el momento se puede nombrar: XP, MSF, RUP

2.3.1. METODOLOGÍA EXTREME PROGRAMING (XP)

XP es una metodología centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en la realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define

como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

A mediados de la década de 1980, Kent Beck y Ward Cunningham trabajaban en un grupo de investigación de Tektronix; allí idearon las tarjetas CRC y sentaron las bases de lo que después serían los patrones de diseño y XP, como se muestra en la figura 1.

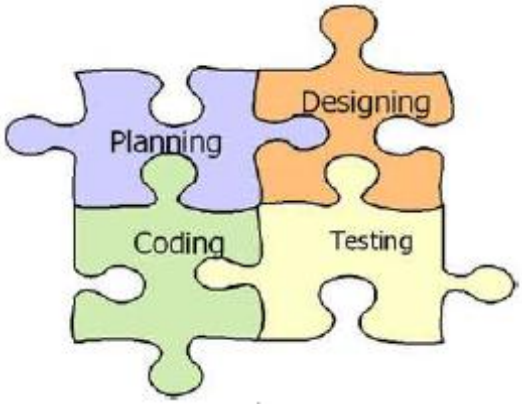


FIGURA 1 Esquema Extreme Programing XP

2.3.1.1. Valores XP

El proceso de desarrollo se fundamenta en una serie de valores y principios que lo guían, los valores representan aquellos aspectos que los autores de XP han considerado como fundamentales para garantizar el éxito de un proyecto de desarrollo de software. Los cuatro valores de XP son:

- Comunicación,
- Simplicidad,
- Realimentación y
- Coraje

Los partidarios de la programación extrema dicen que son los necesarios para conseguir diseños y códigos simples, métodos eficientes de desarrollo software y clientes contentos. Los valores deben ser intrínsecos al equipo de desarrollo. De los cuatro valores, quizás el que llame más la atención es el de coraje. Detrás de este valor se encuentra el lema "si funciona, mejóralo", que choca con la práctica habitual de no tocar algo que funciona, por si acaso. Aunque también es cierto que se tiene las pruebas

unitarias, de modo que no se pide a los desarrolladores una heroicidad, sino sólo coraje.

2.3.1.2. Principios XP

Los principios fundamentales se apoyan en los valores y también son cuatro. Se busca:

- Realimentación veloz,
- Modificaciones incrementales,
- Trabajo de calidad y
- Asunción de simplicidad.

Los principios suponen un puente entre los valores (algo intrínseco al equipo de desarrollo) y las prácticas, que se verán a continuación, y que están más ligadas a las técnicas que se han de seguir.

2.3.1.3. Proceso XP

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

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso (1)

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

- ***Derechos del Cliente***
    - Decidir que se implementa
    - Saber el estado real y el progreso del proyecto
    - Añadir, cambiar o quitar requerimientos en cualquier momento
    - Obtener lo máximo de cada semana de trabajo
    - Obtener un sistema funcionando cada 3 o 4 meses

- ***Derechos del Desarrollador***
    - Decidir cómo se implementan los procesos
    - Crear el sistema con la mejor calidad posible
    - Pedir al cliente en cualquier momento aclaraciones
    - Estimar el esfuerzo para implementar el sistema
    - Cambiar los requerimientos en base a nuevos descubrimientos

Lo fundamental en este tipo de metodología es:

- La comunicación, entre los usuarios y los desarrolladores
- La simplicidad, al desarrollar y codificar los módulos del sistema
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales

2.3.1.4. Ciclo de Vida XP

El ciclo de vida ideal de XP consiste de seis fases:

1. Exploración,
2. Planificación de la Entrega (Release),
3. Iteraciones,
4. Producción,
5. Mantenimiento
6. Muerte del Proyecto.

2.3.1.5. Practicas XP

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

- ***El juego de la planificación***.- Hay una comunicación frecuente el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.

- ***Entregas pequeñas.-*** Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más 3 meses.

- ***Metáfora.-*** El sistema es definido mediante una metáfora o un conjunto de metáforas Compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia Compartida que describe como debería funcionar el sistema (conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).

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

- ***Pruebas.-*** La producción de código está dirigida por las pruebas unitarias. …éstas son establecidas por el cliente antes de escribirse el

código y son ejecutadas constantemente ante cada modificación del sistema.

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

- ***Programación en parejas.-*** Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores)

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

- ***Integración continua.-*** Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día .

- ***40 horas por semana.***- Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente este ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.

- ***Cliente in-situ.***- El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Este es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportar mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada, la comunicación oral es más efectiva que la escrita.

- ***Estándares de programación.-*** XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

- ***Espacio abierto.-*** Es preferible una sala grande con pequeños cubículos o, mejor todavía, sin divisiones. Los pares de programadores deben estar en el centro. En la periferia se ubican las máquinas privadas. En un encuentro de espacio abierto la agenda no se establece verticalmente.

- ***Reglas justas.***- El equipo tiene sus propias reglas a seguir, pero se pueden cambiar en cualquier momento. En XP se piensa que no existe un proceso que sirva para todos los proyectos; lo que se hace habitualmente es adaptar un conjunto de prácticas simples a las características de cada proyecto.

- El mayor beneficio de las prácticas se consigue con su aplicación conjunta y equilibrada puesto que se apoyan unas en otras, esto significa que las dos prácticas se refuerzan entre si. La mayoría de las prácticas propuestas por XP no son novedosas sino que en alguna forma ya habían sido propuestas en ingeniería del software e incluso demostrado su valor en la práctica. El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo.

2.3.1.6. Artefactos XP

Entre los artefactos que se utilizan en XP podemos mencionar:

- Tarjetas CRC
- Tarjetas Históricas
- Listas de Tareas

- ***Las tarjetas CRC.-*** se trata de simples tarjetas de papel para fichado, de 4x6 pulgadas; CRC denota “Clase-Responsabilidad-Colaboración”, y es una técnica que reemplaza a los diagramas en la representación de modelos. En las tarjetas se escriben las Responsabilidades, una descripción de alto nivel del propósito de una clase y las dependencias primarias.

- ***Tarjetas Históricas (story cards).-*** son tarjetas comunes de papel en que se escriben breves requerimientos de un rasgo, jamás casos de uso; pueden adoptar el esquema CRC. Las tarjetas tienen una granularidad de diez o veinte días. Las tarjetas se usan para estimar prioridades, alcance y tiempo de realización; en caso de discrepancia, gana la estimación más optimista.

- ***Listas de tareas.-*** en papel o en una pizarra (jamás en computadora) y gráficos visibles pegados en la pared.

2.3.1.7. Roles XP

Los roles de acuerdo con la propuesta original de Beck son:

- ***Cliente.-*** Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuales se implementan en cada iteración centrándose en aportar mayor valor al negocio.

- ***Programador.-*** El programador escribe las pruebas unitarias y produce el código del sistema, trabajan en pares.

- ***Verificador (Tester).-*** Ayuda al cliente a escribir las pruebas funcionales. Ejecuta
- las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

- ***Consultor Técnico.-*** Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.

- ***Seguidor de rastros (Tracker).-*** Proporciona realimentación al equipo. Verifica el
- grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.

- ***Entrenado (Coach).-*** Es responsable del proceso global. Debe proveer guìas al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente. Lo importante es que el coach se vea como un facilitador, antes que como quien dá las órdenes.

- ***Gestor (Big boss).-*** Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

2.3.1.8. Ventajas / Beneficios XP

- Maximización la comunicación entre las personas,
- Mayor transferencia de conocimiento entre el desarrollador y el cliente
- Disposición física contigua del lugar de trabajo.
- Alto nivel de disciplina entre los programadores.
- Mínimo nivel de documentación
- Gran velocidad en el desarrollo.
- Mayor responsabilidades a programadores para estimular su trabajo.
- Mayor entendimiento del diseño y de todo el código producido
- Mantener una metáfora mediante la cual se nombra las clases y métodos de forma consistente.

- Necesidad de mantener un horario fijo, sin horas extras ya que esto conlleva al desgaste del equipo y a la posible deserción de sus miembros.
- Empieza en pequeño y añade funcionalidad con retroalimentación continua
- El manejo del cambio se convierte en parte sustantiva del proceso
- El costo del cambio no depende de la fase o etapa
- No introduce funcionalidades antes que sean necesarias
- El cliente o el usuario se convierte en miembro del equipo

2.3.1.9. Desventajas / Obstáculos XP

- La falta de documentación genera incapacidad de persistir la arquitectura y demás cuestiones de análisis, diseño e implementación.
- Continúo análisis de estimaciones de cada iteración, acorde a la capacidad de desarrollo del equipo
- Cuidar el alto nivel de disciplina de las personas que participan en el proyecto.
- Pretender que el cliente se quede en el sitio y con total disponibilidad.
- Resistencia de muchos programadores a trabajar en pares.
- Ausencia de énfasis en la arquitectura durante las primeras iteraciones (no hay arquitectos en XP) y por ende la falta de métodos de diseño arquitectónico.

2.3.2. METODOLOGÍA MICROSOFT SOLUTION FRAMEWORK (MSF)

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos desarrollada por Microsoft. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. Se compone de principios, modelos y disciplinas, como se muestra en la figura 2.

FIGURA 2 Esquema Microsoft Solution FrameWork MSF

2.3.2.1. Características MSF

- **Adaptable:** es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- **Escalable:** puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.
- **Flexible:** es utilizada en el ambiente de desarrollo de cualquier cliente.
- **Tecnología Agnóstica:** porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

2.3.2.2. Principios MSF

1.- Promover comunicaciones abiertas
2.- Trabajar con una visión compartida
3.- Fortalecer los miembros del equipo
4.- Establecer responsabilidades claras y compartidas
5.- Focalizarse en agregar valor al negocio
6.- Permanecer ágil y esperar los cambios
7.- Invertir en calidad
8.- Aprender de todas las experiencias

2.3.2.3. Disciplinas MSF

- **Gestión de Proyectos.-** Esta disciplina describe el rol de la gestión del proyecto dentro del modelo de equipo de MSF, y como permite mayor escalabilidad desde proyectos pequeños a proyectos largos y complejos. Se basa en:
    - Planificar sobre entregas cortas
    - Incorporar nuevas características sucesivamente
    - Identificar cambios ajustando el cronograma

- **Control de Riesgos.-** Esta disciplina permite ayudar al equipo a identificar las prioridades, tomar decisiones estratégicas correctas y controlar las emergencias que puedan surgir, este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.

- **Control de Cambios.-** Esta disciplina orienta a que el equipo sea proactivo en lugar de reactivo. Los cambios deben considerarse riesgos inherentes y además deben registrarse y hacerse evidentes.

2.3.2.4. Modelos MSF

- **Modelo Equipo de Trabajo.-** Modelo que alienta la agilidad para hacer frente a nuevos cambios involucrando a todo el equipo en las decisiones fundamentales, asegurando así que se exploran y revisan los elementos de juicio desde todas las perspectivas críticas. Este modelo no es rígido, puede ser escalado dependiendo del tamaño del proyecto y equipo de personas disponibles.
    - *Gerencia de Proyecto.-* Controla y administra
    - *Gerencia de Producto.-* Enfoque al cliente
    - *Experiencia del Usuario.-* Apoyo al usuario
    - *Desarrollo.-* Ejecución de las especificaciones
    - *Pruebas.-* Aprobación y verificación
    - *Puesta en Operación.-* Planeación y puesta en producción

- **Modelo de Proceso.-** Este modelo a través de su estrategia iterativa en la construcción de productos del proyecto, suministra una imagen más clara del estado en cada etapa sucesiva. El equipo puede identificar con mayor facilidad el impacto de cualquier cambo y administrarlo efectivamente, minimizando los efectos colaterales negativos mientras optimiza los beneficios. Este modelo ha sido diseñado para mejorar el control del proyecto, minimizando el riesgo y aumentar la calidad acortando el tiempo de entrega.

2.3.2.5. Fases e Hitos MSF

- **Visión.-** Obtener una visión del proyecto compartida, comunicada, entendida y alineada con los objetivos del negocio. Además identificar los beneficios, requerimientos funcionales, sus alcances y restricciones; y los riesgos inherentes al proceso

- **Planeación.-** Obtener un cronograma de trabajo que cumpla con lo especificado en la fase de visión dentro del presupuesto, tiempo y recursos acordados. Este cronograma debe identificas los puntos de control específicos que permitan generar entregas funcionales y cortas n el tiempo.

- **Desarrollo.-** Obtener interactivamente de la mano de las fases de plantación y estabilización, versiones del producto entregables y medibles que permitan al cliente probar características nuevas sucesivamente. Esto incluye ajustes en el cronograma.

- **Estabilización.-** Obtener una versión final del producto, esta debe ser probada, ajustada y aprobada en su totalidad

- **Instalación.-** Instalar y entregar al cliente el producto finalizado totalmente, como garantía se han superado con éxito las etapas anteriores.

- **Soporte.-** Brindar soporte y garantía al producto durante el tiempo estipulado en el contrato, registrando los reportes de soporte y mantenimiento recibidos, así como los ajustes y versiones ajustadas obtenidas. En esta fase se identifican las características y requerimientos no tomados en cuenta en la fase visión, generando así un nuevo proyecto.

2.3.2.6. Entregables MSF

- **Visión.-**
  - Documento Visión.- antecedentes, visión, criterios de diseño
  - Detalle de Visión.- beneficios, metas, objetivos y restricciones
  - Perfiles de usuario
  - Casos de uso
  - Requerimientos funcionales y no funcionales
  - Plan de Instalación
  - Arquitectura lógica (diagrama de componentes UML)
  - Arquitectura física (diagrama d despliegue UML)
  - Documento Requerimientos Funcionales
  - Descripción Detallada de Requerimientos
  - Características que componen cada caso de uso
  - Perfiles asociados
  - Recursos del equipo de proyecto
  - Riesgos y observaciones
  - Scripts de pruebas
  - Documento Matriz e Riesgos
  - Identificar posibles riesgos acerca de requerimientos
  - Acciones a tomar en cada escenario
  - Acta de Aprobación de Visión
- **Planeación**
  - Cronograma de actividades
  - Acta de Aprobación de Cronograma
- **Desarrollo**

- Fuentes y ejecutables según lo acordado
- Manuales técnicos, de usuario e instalación
- Acta de Finalización de desarrollo

- **Estabilización**
  - Registro de Pruebas
  - Acta de aprobación de versión definitiva
- **Instalación**
  - Conjunto de archivos propios, programas ejecutables, directorio, bases de datos, scripts, instaladores, manuales, licencias, etc.
  - Acta de entrega y finalización de proyecto
- **Soporte**
  - Registro de soporte y atención al usuario
  - Fechas de actualizaciones y ajustes realizados

2.3.2.7. Ventajas y Beneficios MSF

- MSF reconoce la naturaleza caórdica, una combinación de caos y orden de los proyectos de tecnología, toma como punto de partida el supuesto de que debe esperarse cambio continuo y de que es imposible aislar un proyecto de solución de estos cambios.
- El Modelo de Equipo y el Modelo de Proceso está diseñado para anticiparse al cambio y controlarlo
- Alienta la agilidad para enfrentar los cambios involucrando a todo el equipo en decisiones fundamentales.
- El modelo de Proceso, suministra una imagen más clara del estado de cada iteración
- El equipo de proyecto identifica con mayor agilidad el impacto de los cambios, minimiza los efectos y optimiza los beneficios.
- Marco de desarrollo fuertemente documentado, sus disciplinas de ningún modo admiten la validez de modelos no iterativos o no incrementales.
- Prioriza la gestión de riesgos, tratar áreas más riesgosas primero

2.3.2.8. Desventajas y Obstáculos MSF

- Los requerimientos de un proyecto pueden ser difíciles de articular de antemano y a menudo sufren modificaciones significativas a medida que los participantes van viendo sus posibilidades con mayor claridad.

- Alto costo de herramientas de software

- Alto nivel de conocimiento en el equipo de proyecto

- El proyecto puede llegar a ser muy costoso

2.3.3. METODOLOGÍA RATIONAL UNIFIED PROCESS (RUP)

Es una metodología de Ingeniería de Software planteado por Kruchten (1996) cuyo objetivo es producir software de alta calidad, es decir, que cumpla con los requerimientos de los usuarios dentro de una planificación y presupuesto establecido, cubre el ciclo de vida de desarrollo de software. Cuenta las mejores prácticas en el modelo de desarrollo de software en particular las siguientes:

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

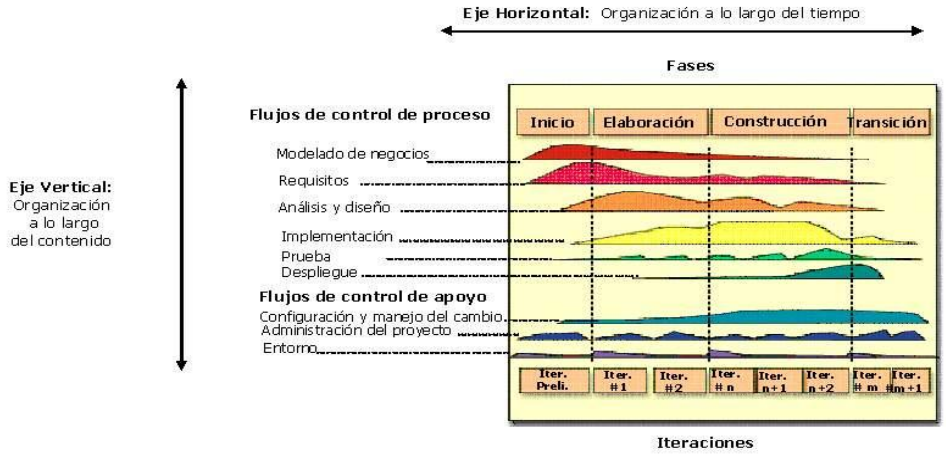


FIGURA 3 Esquema Rational Unified Process RUP

2.3.3.1. Características RUP

Proceso dirigido por Casos de Uso.- Los Casos de Uso son una técnica que permiten capturar los requerimientos e información del usuario, basado en funciones y su importancia. Además los CU permiten,

- Definir fragmento de la funcionalidad del sistema
- Representar los requisitos funcionales del sistema
- Guiar el diseño, implementación y pruebas
- Integrar elementos y establecer la trazabilidad entre artefactos
- Iniciar el proceso de desarrollo

Proceso centrado en la Arquitectura.- La arquitectura de un sistema es la organización o estructura de sus partes más relevantes, lo que permite tener una visión común del proyecto tanto para los desarrolladores y como para los usuarios, dando una perspectiva clara del sistema completo, Además permite:

- Facilitar el control del desarrollo
- Involucrar aspectos estáticos y dinámicos significativos del sistema
- Tomar decisiones sobre la construcción del sistema,
- Ayudar a determinar el orden y seguimiento
- Tomar en consideración elementos de: calidad, rendimiento y reutilización
- Flexibilidad durante todo el proceso de desarrollo.
- La arquitectura se ve influenciada por la plataforma software, sistema operativo, gestor de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados.
- Muchas de estas restricciones constituyen requisitos no funcionales del sistema.

Proceso iterativo e incremental.- Este proceso es un conjunto de iteraciones, que dividen el trabajo en partes más pequeñas o mini proyectos, lograr el equilibrio entre Casos de Uso y arquitectura durante todo el proceso de desarrollo. Cada mini proyecto se puede ver como una iteración, un

recorrido más o menos completo a lo largo de todos los flujos de trabajo fundamentales, del cual se obtiene un incremento que produce un crecimiento en el producto. Además una iteración permite:

- Abarcar una parte de la funcionalidad total de l proyecto
- Pasar por todos los flujos de trabajo relevantes y refinando la arquitectura.
- Cada iteración se analiza cuando termina.
- Determinar si han aparecido nuevos requisitos o han cambiado los existentes,
- Afectar a las iteraciones siguientes.
- Examinar los riesgos que un quedan en el proyecto
- Retroalimentar información para reajustar objetivos de las siguientes iteraciones

2.3.3.2. Prácticas RUP

- **Gestión de requisitos.-** Guía para encontrar, organizar, documentar, y seguir los cambios de los requisitos funcionales y restricciones. Utiliza una notación de Caso de Uso y escenarios para representar los requisitos.
- **Desarrollo de software iterativo.-** Desarrollo del producto mediante iteraciones con hitos bien definidos, en las cuales se repiten las actividades pero con distinto énfasis, según la fase del proyecto.
- **Desarrollo basado en componentes.-** La creación de sistemas intensivos en software requiere dividir el sistema en componentes con interfaces bien definidas, que posteriormente serán ensamblados para generar el sistema. Esta característica en un proceso de desarrollo permite que el sistema se vaya creando a medida que se obtienen o se desarrollan sus componentes.
- **Modelado visual (usando UML).-** UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema software, es un estándar de la OMG. Utilizar herramientas de modelado visual facilita la gestión de dichos modelos, permitiendo

ocultar o exponer detalles cuando sea necesario. El modelado visual también ayuda a mantener la consistencia entre los artefactos del sistema: requisitos, diseños e implementaciones. En resumen, el modelado visual ayuda a mejorar la capacidad del equipo para gestionar la complejidad del software.

- **Verificación continua de la calidad.-** Es importante que la calidad de todos los artefactos se evalúe en varios puntos durante el proceso de desarrollo, especialmente al final de cada iteración. En esta verificación las pruebas juegan un papel fundamental y se integran a lo largo de todo el proceso. Para todos los artefactos no ejecutables las revisiones e inspecciones también deben ser continuas.
- **Gestión de los cambios.-** El cambio es un factor de riesgo crítico en los proyectos de software. Los artefactos software cambian no sólo debido a acciones de mantenimiento posteriores a la entrega del producto, sino que durante el proceso de desarrollo, especialmente importantes por su posible impacto son los cambios en los requisitos. Por otra parte, otro gran desafío que debe abordarse es la construcción de software con la participación de múltiples desarrolladores, posiblemente distribuidos geográficamente, trabajando a la vez en una versión, y quizás en distintas plataformas. La ausencia de disciplina rápidamente conduciría al caos. La Gestión de Cambios y de Configuración es la disciplina de RUP encargada de este aspecto.

2.3.3.3. Proceso RUP

- ***Eje horizontal***: Representa el tiempo y es considerado el eje de los aspectos dinámicos del proceso. Indica las características del ciclo de vida del proceso expresado en términos de fases, iteraciones e hitos. RUP consta de cuatro fases: Inicio, Elaboración, Construcción y Transición. Como se mencionó anteriormente cada fase se subdivide a la vez en iteraciones, como se muestra en la figura 3.

- ***Eje vertical:*** Representa los aspectos estáticos del proceso. Describe el proceso en términos de componentes de proceso, disciplinas, flujos de trabajo, actividades, artefactos y roles, como se muestra en la figura 3.

2.3.3.4. Fases e Iteraciones RUP

EL ciclo de vida de cuatro fases: Inicio, Elaboración, Construcción y Transición. Cada fase se subdivide a la vez en iteraciones, el número de iteraciones en cada fase es variable y cada ciclo concluye con una generación del producto para los clientes.

Inicio.- Esta fase define el modelo del negocio y el alcance del proyecto. Se identifican todos los actores y Casos de Uso, y se diseñan los Casos de Uso más esenciales (aproximadamente el 20% del modelo completo). Se desarrolla, un plan de negocio para determinar que recursos deben ser asignados al proyecto. Los objetivos son:

- Establecer el ámbito del proyecto y sus límites.
- Encontrar los Casos de Uso críticos del sistema, los escenarios básicos que definen la funcionalidad.
- Mostrar al menos una arquitectura para los escenarios principales.
- Estimar el coste en recursos y tiempo de todo el proyecto.
- Estimar los riesgos, las fuentes de incertidumbre.
- Al terminar la fase de inicio se deben comprobar los criterios de evaluación para continuar:
- Todos los interesados en el proyecto coinciden en la definición del ámbito del sistema y las estimaciones de agenda.
- Entendimiento de los requisitos, como evidencia de la fidelidad de los Casos de Uso principales.
- Las estimaciones de tiempo, coste y riesgo son creíbles.
- Comprensión total de cualquier prototipo de la arquitectura desarrollado.
- Los gastos hasta el momento se asemejan a los planeados.

Elaboración.- El propósito de la fase es analizar el dominio del problema, establecer los cimientos de la arquitectura, desarrollar el plan del proyecto y eliminar los mayores riesgos. En esta fase se construye un prototipo de la arquitectura, que debe evolucionar en iteraciones sucesivas hasta convertirse en el sistema final. Este prototipo debe contener los Casos de Uso críticos identificados en la fase de inicio. También debe demostrarse que se han evitado los riesgos más graves. Los objetivos de esta fase son:

- Definir, validar y cimentar la arquitectura.
- Completar la visión.
- Crear un plan fiable para la fase de construcción. Este plan puede evolucionar en sucesivas iteraciones. Debe incluir los costes si procede.
- Demostrar que la arquitectura propuesta soportará la visión con un coste razonable y en un tiempo razonable.
- Los criterios de evaluación de esta fase son los siguientes:
- La visión del producto es estable.
- La arquitectura es estable.
- Se ha demostrado mediante la ejecución del prototipo que los principales elementos de riesgo han sido abordados y resueltos.
- El plan para la fase de construcción es detallado y preciso. Las estimaciones son creíbles.
- Todos los interesados coinciden en que la visión actual será alcanzada si se siguen los planes actuales en el contexto de la arquitectura actual.
- Los gastos hasta ahora son aceptables, comparados con los previstos.

Construcción.- La finalidad principal de esta fase es alcanzar la capacidad operacional del producto de forma incremental a través de las sucesivas iteraciones. Durante esta fase todos los componentes, características y requisitos deben ser implementados, integrados y probados en su totalidad, obteniendo una versión aceptable del producto. Los objetivos son:

- Minimizar los costes de desarrollo mediante la optimización de recursos y evitando el tener que rehacer un trabajo o incluso desecharlo.
- Conseguir una calidad adecuada tan rápido como sea práctico.
- Conseguir versiones funcionales (alfa, beta, y otras versiones de prueba) tan rápido como sea práctico.
- Los criterios de evaluación de esta fase son los siguientes:
- El producto es estable y maduro como para ser entregado a la comunidad de usuario para ser probado.
- Todos los usuarios expertos están listos para la transición en la comunidad de usuarios.
- Son aceptables los gastos actuales versus los gastos planeados.

Transición.- La finalidad de la fase de transición es poner el producto en manos de los usuarios finales, para lo que se requiere desarrollar nuevas versiones actualizadas del producto, completar la documentación, entrenar al usuario en el manejo del producto, y en general tareas relacionadas con el ajuste, configuración, instalación y facilidad de uso del producto. En se citan algunas de las cosas que puede incluir esta fase:

- Prueba de la versión Beta para validar el nuevo sistema frente a las expectativas de los usuarios
- Funcionamiento paralelo con los sistemas legados que están siendo sustituidos por nuestro proyecto.
- Conversión de las bases de datos operacionales.
- Entrenamiento de los usuarios y técnicos de mantenimiento.
- Traspaso del producto a los equipos de marketing, distribución y venta.

Los principales objetivos de esta fase son:

- Conseguir que el usuario se valga por si mismo.
- Un producto final que cumpla los requisitos esperados, que funcione y satisfaga suficientemente al usuario.

Los criterios de evaluación de esta fase son los siguientes:

- El usuario se encuentra satisfecho.
- Son aceptables los gastos actuales versus los gastos planificados.

2.3.3.5. Entregables RUP

Inicio.-

- Un documento de visión: Una visión general de los requerimientos del proyecto, características clave y restricciones principales.
- Modelo inicial de Casos de Uso (10-20% completado).
- Un glosario inicial: Terminología clave del dominio.
- El caso de negocio.
- Lista de riesgos y plan de contingencia.
- Plan del proyecto, mostrando fases e iteraciones.
- Modelo de negocio, si es necesario
- Prototipos exploratorios para probar conceptos o la arquitectura candidata.

Elaboración.-

- Un modelo de Casos de Uso completa al menos hasta el 80%: todos los casos y actores identificados, la mayoría de los casos desarrollados.
- Requisitos adicionales que capturan los requisitos no funcionales y cualquier requisito no asociado con un Caso de Uso específico.
- Descripción de la arquitectura software.
- Un prototipo ejecutable de la arquitectura.
- Lista de riesgos y caso de negocio revisados.
- Plan de desarrollo para el proyecto.
- Un caso de desarrollo actualizado que especifica el proceso a seguir.
- Un manual de usuario preliminar (opcional).

Construcción.-

- Modelos Completos (Casos de Uso, Análisis, Diseño, Despliegue e

Implementación)

- Arquitectura íntegra (mantenida y mínimamente actualizada)
- Riesgos Presentados Mitigados
- Plan del Proyecto para la fase de Transición.
- Manual Inicial de Usuario (con suficiente detalle)
- Prototipo Operacional – beta
- Caso del Negocio Actualizado

Transición.-

- Prototipo Operacional
- Documentos Legales
- Caso del Negocio Completo
- Línea de Base del Producto completa y corregida que incluye todos los modelos del sistema
- Descripción de la Arquitectura completa y corregida
- Las iteraciones están dirigidas a conseguir una nueva versión.

2.3.3.6. Roles RUP

Analistas.-

- Analista de procesos de negocio.
- Diseñador del negocio.
- Analista de sistema.
- Especificador de requisitos.

Desarrolladores.-

- Arquitecto de software.
- Diseñador
- Diseñador de interfaz de usuario
- Diseñador de cápsulas.
- Diseñador de base de datos.
- Implementador.
- Integrador.

Gestores.-

- Jefe de proyecto
- Jefe de control de cambios.
- Jefe de configuración.
- Jefe de pruebas
- Jefe de despliegue
- Ingeniero de procesos
- Revisor de gestión del proyecto
- Gestor de pruebas.

Apoyo.-

- Documentador técnico
- Administrador de sistema
- Especialista en herramientas
- Desarrollador de cursos
- Artista gráfico

Especialista en pruebas.-

- Especialista en Pruebas (testeador)
- Analista de pruebas
- Diseñador de pruebas

Otros roles.-

- Stakeholders
- Revisor
- Coordinación de revisiones
- Revisor técnico
- Cualquier rol

2.3.3.7. Actividades RUP

Una actividad en concreto es una unidad de trabajo que una persona que desempeñe un rol puede ser solicitado a que realice. Las actividades

tienen un objetivo concreto, normalmente expresado en términos de crear o actualizar algún producto.

2.3.3.8. Artefactos RUP

- Un documento, como el documento de la arquitectura del software.
- Un modelo, como el modelo de Casos de Uso o el modelo de diseño.
- Un elemento del modelo, un elemento que pertenece a un modelo como una clase, un Caso de Uso o un subsistema.

2.3.3.9. Flujos de Trabajo RUP

Con la enumeración de roles, actividades y artefactos no se define un proceso, se necesita contar con una secuencia de actividades realizadas por los diferentes roles, así como la relación entre ellos. Un flujo de trabajo es una relación de actividades que producen unos resultados observables. A continuación se dará una explicación de cada flujo de trabajo.

Modelado del Negocio.- Con este flujo de trabajo se pretende llegar a un mejor entendimiento de la organización donde se va a implantar el producto.

- Entender la estructura y la dinámica de la organización para la cual el sistema va ser desarrollado (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.

Requisitos.- Este es uno de los flujos de trabajo más importantes, porque en él se establece qué hacer exactamente en sistema que se construirá. En esta línea los requisitos son el contrato que se debe cumplir, de modo que los usuarios finales tienen que comprender y aceptar los requisitos especificados.

- Establecer y mantener un acuerdo entre clientes y otros stakeholders sobre lo que el sistema podría hacer.
- Proveer a los desarrolladores un mejor entendimiento de los requisitos del sistema.
- Definir el ámbito del sistema.
- Proveer una base para la planeación de los contenidos técnicos de las iteraciones.
- Proveer 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.

Análisis y Diseño.- El objetivo de este flujo de trabajo es traducir los requisitos a una especificación que describe cómo implementar el sistema.

- Transformar los requisitos al diseño del futuro sistema.
- Desarrollar una arquitectura para el sistema.
- Adaptar el diseño para que sea consistente con el entorno de implementación, diseñando para el rendimiento.

Implementación.- En este flujo de trabajo se implementan las clases y objetos en ficheros fuente, binarios, ejecutables y demás. Además se deben hacer las pruebas de unidad: cada implementador es responsable de probar las unidades que produzca. El resultado final de este flujo de trabajo es un sistema ejecutable. En cada iteración se debe lo siguiente:

- Planificar qué subsistemas deben ser implementados y en que orden deben ser integrados, formando el Plan de Integración.
- Cada implementador decide en que orden implementa los elementos del subsistema.
- Si encuentra errores de diseño, los notifica.
- Se prueban los subsistemas individualmente.
- Se integra el sistema siguiendo el plan.

Pruebas.- Este flujo de trabajo es el encargado de evaluar la calidad del producto que a desarrollar, pero no para aceptar o rechazar el producto al final del proceso de desarrollo, sino que debe ir integrado en todo el ciclo de vida.

- Encontrar y documentar defectos en la calidad del software.
- Generalmente asesora sobre la calidad del software percibida.
- Provee la validación de los supuestos realizados en el diseño y especificación de requisitos por medio de demostraciones concretas.
- Verificar las funciones del producto de software según lo diseñado.
- Verificar que los requisitos tengan su apropiada implementación.

Despliegue.- El objetivo de este flujo de trabajo es producir con éxito distribuciones del producto y distribuirlo a los usuarios. Las actividades implicadas incluyen:

- Probar el producto en su entorno de ejecución final.
- Empaquetar el software para su distribución.
- Distribuir el software.
- Instalar el software.
- Proveer asistencia y ayuda a los usuarios.
- Formar a los usuarios y al cuerpo de ventas.
- Migrar el software existente o convertir bases de datos.

Gestión del proyecto.- La Gestión del proyecto es el arte de lograr un balance al gestionar objetivos, riesgos y restricciones para desarrollar un producto que sea acorde a los requisitos de los clientes y los usuarios.

- Proveer un marco de trabajo para la gestión de proyectos de software intensivos.
- Proveer guías prácticas realizar planeación, contratar personal, ejecutar y monitorear el proyecto.
- Proveer un marco de trabajo para gestionar riesgos.

Configuración y control de cambios.- La finalidad de este flujo de trabajo es mantener la integridad de todos los artefactos que se crean en el proceso, así como de mantener información del proceso evolutivo que han seguido.

Entorno.- La finalidad de este flujo de trabajo es dar soporte al proyecto con las adecuadas herramientas, procesos y métodos. Brinda una especificación de las herramientas que se van a necesitar en cada momento, así como definir la instancia concreta del proceso que se va a seguir. En concreto las responsabilidades de este flujo de trabajo incluyen:

- Selección y adquisición de herramientas
- Establecer y configurar las herramientas para que se ajusten a la organización.
- Configuración del proceso.
- Mejora del proceso.
- Servicios técnicos.

2.3.4. MATRICES DE COMPARACIÓN

2.3.4.1. Comparación Valores, Principios y Características

La tabla 2 permite comparar los valores, principios y características de las metodologías.

TABLA 2

Comparación Valores, Principios y Características

VALORES, PRINCIPIOS Y CARACTERÍSTICAS		
RUP	**XP**	**MSF**
Calidad en Software Planificación y Presupuesto Desarrollo Iteractivo	Comunicación Simplicidad Realimentación Coraje	Adaptable Escalable Flexible Tecnología Agnóstica
Dirigido a Casos de Uso Centralizado a la Arquitectura Proceso Iteractivo e Incremental	Realimentación Veloz Modificaciones Incrementales Trabajo de calidad Asunción de Simplicidad	Promover Comunicaciones Abiertas Trabajar con Visión Compartida Fortalecer miembros del Equipo Responsabilidades Claras y Compartidas Agregar valor al negocio Agilidad y esperar cambios Inversión en Calidad Aprender de experiencias

2.3.4.2. Comparación Disciplinas

La tabla 3 permite comparar las disciplinas de las metodologías.

TABLA 3

Comparación Disciplinas

DISCIPLINAS		
RUP	**XP**	**MSF**
Gestión de requisitos Desarrollo de software iteractivo Desarrollo basado en Componentes Modelado visual (uml) Verificación Continua Gestión de Cambios	Juego de la Planificación Entregas pequeñas Metáfora Diseño Simple Pruebas Refactorización Programación en Parejas Propiedad Colectiva del Código Integración Continua 40 horas por semana Cliente in-situ Estándares de Programación Espacio Abierto Reglas Justas	Gestión de Proyectos Planificar entregas cortas Incorporar nuevas características Identificar Cambios sucesivos Ajustar Cronograma Control de Riesgos Control de Cambios

2.3.4.3. Comparación Entregables y Artefactos

La tabla 4 permite comparar los documentos entregables y artefactos de las metodologías.

TABLA 4

Comparación Entregables y Artefactos

ENTREGABLES O ARTEFACTOS		
RUP	**XP**	**MSF**
Inicio Requerimientos y Características iniciales Casos de uso (20%) Caso del Negocio Lista de riesgos y contingencias Modelo de negocio Plan de Proyecto	Tarjetas CRC Tarjetas Históricas Listas de Tareas Código fuente Manuales	Visión Perfiles de usuario y Recursos Casos de uso Requerimientos funcionales y no funcionales Plan de Instalación Diagramas arquitectura lógica y física Riesgos, Observaciones y Acciones Scripts de Pruebas Acta Aprobación
Elaboración Modelo casos de Uso (80%) Requisitos adicionales y no funcionales Descripción Arquitectura		Plantación Cronograma de Actividades Acta de Aprobación

CONTINÚA →

software Prototipo ejecutable Lista de riesgos y casos de uso revisados Plan de desarrollo		
Construcción Casos de uso completo Diagramas de análisis y diseño Arquitectura integra Riesgos mitigados Plan de instalación Manual inicial de usuario Prototipo Operacional - beta Caso de negocio Actualizado		Desarrollo Fuentes y ejecutables Manuales de usuario y técnicos Acta finalización desarrollo
		Estabilización Registro de Pruebas Acta aprobación **CONTINUA →**
Transición Prototipo Operacional Caso de negocio completo Documentos entrega-recepción Arquitectura completa y corregida		Instalación y Soporte Directorios, Programas, Bases de Datos Manuales, Instaladores y Licencias Acta de entrega Registro de Soporte y Atención al usuario Fechas de Actualizaciones y Ajustes

2.3.4.4. Comparación Roles

La tabla 5 permite la comparación de los roles de las metodologías.

TABLA 5

Comparación Roles

ROLES		
RUP	**XP**	**MSF**
Analistas Desarrolladores Gestores Apoyo Verificadores	Cliente Programador Verificador Consultor Seguidor de Rastros Entrenador Gestor	Gerente de Proyecto Gerente de Producto (cliente) Usuario experto Desarrollador Verificador Implementador

2.4. ESTANDARIZACIÓN DE METODOLOGÍAS DE DESARROLLO

Para la estandarización de las metodologías de desarrollo se utiliza “El Manifiesto para el Desarrollo de Software”, este es de suma importancia

ya que representa una iniciativa conjunta entre los principales responsables y promotores, para lograr unificar principios compartidos por las diversas metodologías de manera de crear un framework de trabajo que contribuya al mejoramiento del desarrollo eficiente.

Uno de los principales objetivos del encuentro en que se generó el Manifiesto fue el de extraer un factor común de los principios esenciales que servirían de guía para cualquier metodología. Esto concluyó en la declaración de lo que se denomina el prólogo del Manifiesto [Manifesto, 2001]:

"*Estamos descubriendo mejores maneras de desarrollar software mediante su construcción y ayudando a que otras personas lo construyan. A través de este trabajo se llegado a valorar:*

- Individuos e interacciones sobre procesos y personas.

- Software funcionando sobre documentación comprensiva

- Colaboración del cliente sobre negociación de contrato

- Responder al cambio sobre seguir un plan

Esto es, mientras que existe valor en los ítems de la derecha, se valora más los ítems de la izquierda."

Analizando los factores mencionados en detalle se observa que estos forman parte esencial de las metodologías descritas.

- El énfasis sobre las personas está identificado en la primera oración respecto a los valores de las metodologías.
- El énfasis en el código por sobre los documentos es una propuesta eficaz para lograr el rápido feedback requerido por la agilidad.
- La participación del cliente es fundamental para el éxito de los proyectos ya que ellos definen la funcionalidad a construir y guían el desarrollo de las pruebas funcionales.
- La importancia de responder al cambio indica que el desarrollo de software no es más que un proceso de aprendizaje en que cada cambio permite conocer más en detalle el dominio de la aplicación.

- Para definir una nueva metodología, la misma deberá estar completamente circunscripta a los valores detallados en el Manifiesto. Es por esto que se hará continuas referencias al mismo para comentar la conformidad del proceso a los principios y valores.

2.5. DESCRIPCIÓN DEL PROBLEMA

2.5.1. ELECCIÓN DE UN PROCESO

La elección de un proceso es obligatorio, pues este permitirá que el conocimiento y el esfuerzo de los involucrados en el proyecto sea aplicado en forma uniforme y óptima, además permitirá disminuir el riesgo y obtener el resultado esperado.

Existen 2 enfoques bien definidos en el desarrollo de software, Burocracia y Adhocracia, que permiten identificar ***El Grado de Formalidad*** de las metodologías.

Enfoque BUROCRACIA.- representado por las organizaciones con procesos rígidos y definidos hasta el más mínimo nivel de detalles, poseen manuales de funciones y procedimientos para cada una de las actividades relacionadas con el desarrollo de software, estas actividades realizadas por cada persona están descritas en detalle y en cualquier momento la documentación podrá ser consultada si tienen dudas respecto a algún punto del proceso. Además, estas organizaciones poseen una estructura tradicional que utilizan el modelo Comando-Control en todos los niveles jerárquicos, utilizan procesos rígidos para generar hitos y entregas bien definidos, este proceso es tan pautado que al final termina sin satisfacer las necesidades de los usuarios.

Enfoque de ADHOCRACIA.- representa el desarrollo caótico sin ningún proceso ni visibilidad sobre el estado y el rumbo de los proyectos, esta oscuridad completa y aleatoriedad total que deviene del caos permite únicamente beneficios o pérdidas a corto plazo, es decir, si una empresa está interesada en que cada proyecto sea como una aventura, en que se

desconozca el resultado y existan una gran cantidad de riesgos, no necesitará proceso alguno. Simplemente dejará en manos de los profesionales de sistemas el desarrollo de proyectos mediante las técnicas que utilice cada individuo en su trabajo, la aleatoriedad de cada proyecto será una característica inherente, se realizarán planificaciones a muy corto plazo, ya que no existe visibilidad para las personas externas al equipo de desarrollo. Indudablemente, cualquier organización que se considere como tal jamás elegiría este enfoque como primera instancia ya que uno de los objetivos de la misma sería *maximizar ganancias a largo plazo* y para esto debería mantener procesos que aseguren el éxito en todos los sectores en que está compuesta, sin embargo, en la realidad la mayoría de las empresas termina eligiendo esta opción sin ser plenamente consciente de la elección y de los resultados que esta acarrea. La figura 4 permite visualizar la ubicación de las metodologías según el grado de formalidad, desde burocracia hasta adhocracia.

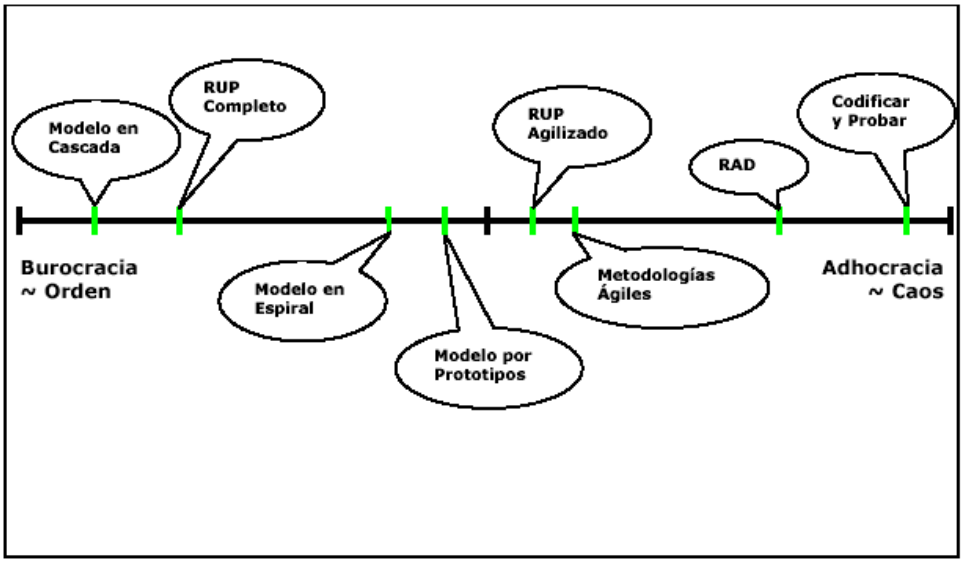


FIGURA 4 Ubicación de las Metodologías por Formalidad

El Modelo en Cascada es el que más contribuye a un régimen burocrático, en el que los proyectos son planificados en base a tareas a realizar por los recursos, fomenta el orden de la organización mediante la clara división en etapas con sus inputs/outputs correspondientes, es un

modelo orientado a documentos, en donde el progreso se mide en base a un conjunto de fases bien definidas y al grado de avance de la documentación correspondiente. Las personas son partes reemplazables en este esquema, y como tales alcanza con definir los roles y las actividades que estas realizan para completar el marco de desarrollo.

El RUP en su forma completa, construye todos los artefactos especificados y realiza todas las actividades planteadas, estos procesos definen todas las actividades, roles, entregables, y demás aspectos relacionados en un proyecto de software, para generar resultados predecibles a lo largo del tiempo.

El RUP Agilizado no es otra cosa que una versión del RUP customizada para grupos pequeños y con feedback frecuente de los interesados.

Las Metodologías Ágiles reconocen las características inherentes de complejidad del software, y el carácter empírico que debe tener un proceso de desarrollo, aquí las personas son de primera magnitud y se prefiere sacrificar el proceso en pos de darles libertad a las personas. Sin embargo, esto no significa que se termina en un proceso de Codificar y Probar, liderado por programadores expertos que se pasan el día generando código. Es por esto que el punto en que se ubican las mismas está bastante alejado del Caos en que la aleatoriedad es la norma. Otra cuestión que diferencia estos procesos de los anteriores es el cambio existente entre un proceso empírico y un proceso determinista o definido.

Los Procesos Empíricos utilizan las buenas prácticas probadas con la experiencia, y los resultados de outputs que surgen generalmente de tomar ciertos inputs o de realizar ciertas actividades. Estos modelos surgen de la observación y la experiencia obtenida a lo largo de los años, sin basarse en leyes de la naturaleza o principios teóricos fundamentados. Es por esto que se los asocia a un enfoque pragmático para desarrollar software.

El RAD en el que se tenía un lenguaje de alto nivel lo suficientemente poderoso como para permitir generar código de forma rápida, un pequeño equipo trabajando en conjunto con buenos canales de comunicación, y un

plan de entregas de funcionalidad en forma iterativa. El mismo no poseía pautas claras en relación a la calidad del producto, la satisfacción de los requerimientos de los usuarios, la clara definición de la arquitectura del sistema. Por estas causas terminó siendo asociado con la generación rápida de aplicaciones con bajos niveles de calidad, y que resultaban en pesadillas de mantenimiento. Sin embargo, se debe reconocer el aporte que hicieron a la ingeniería de software ya que gracias a estas resurgieron las herramientas CASE y comenzó el uso masivo de lenguajes de alto nivel, sirviendo como base para el posterior advenimiento de las metodologías ágiles.

El modelo Codificar y Probar es caótico por excelencia, se basa en el siguiente flujo de actividades: un programador trabajando en su computadora, genera código fuente; una vez que termina, lo compila, corrige los errores y lo vuelve a compilar; cuando ya no tiene más errores, genera el ejecutable y lo corre; ahí prueba el correcto funcionamiento del mismo, ejecutando instrucción por instrucción el código al instante hasta corregir aquellos errores que detecta; cuando este proceso iterativo finaliza, continúa con el siguiente componente/módulo/programa. El mismo es totalmente impredecible y la visibilidad para las personas involucradas en el mismo es inexistente.

El Grado de Predictibilidad de una metodología depende de ciertos aspectos inherentes en el desarrollo, a medida que aumenta la cantidad de personas asignadas a un proyecto se debe contar con metodologías más pesadas tendientes a suplir la falta de canales anchos de comunicación entre los integrantes del equipo de desarrollo.

El Grado de Criticidad de la Aplicación, a medida que aumenta el perjuicio para el negocio ante una posible falla en el sistema, se deberá utilizar una metodología más pesada, con procesos de revisión bien pautados, de forma de asegurarse la calidad del producto. No es lo mismo construir una aplicación empaquetada para bajar archivos de Internet automáticamente, que el software utilizado en un equipo de resonancia magnética.

2.5.2. ORIENTACIÓN PROCESOS VS PERSONAS

Un par de décadas atrás, en el campo de la Ingeniería de Software solo existían los denominados procesos tradicionales. Comenzando por el modelo en cascada con su división en etapas y roles bien definidos, continuando por el modelo en espiral, donde se mitigaban los riesgos en forma temprana, todos estos procesos proponían un enfoque secundario en relación a las personas. Consideraban que teniendo un proceso predecible, con etapas y tareas bien definidas, el éxito estaba garantizado independientemente de las personas que cubrieran los roles. Estos modelos de procesos están orientados al proceso en sí, y suponen que al seguir los principios propuestos en cualquier organización y con cualquier equipo de desarrollo, se obtendrán los resultados predichos en la teoría. El resultado de aplicar un proceso de calidad, será la calidad en el producto.

Actualmente se encuentra con esta visión orientada al proceso en muchas organizaciones con altos niveles de burocracia y en organizaciones basadas en el modelo Codificar y Probar. Es decir, en ningún momento existen consideraciones respecto a cómo la productividad se ve afectada por la gente que trabaja y por las condiciones laborales existentes. Para evaluar la importancia de este elemento se toma uno de los frameworks de estimación más importantes en la actualidad COCOMO II desarrollado por Barry Boehm entre otros [Boehm, 2000].

El método en cuestión propone siete factores de personal que ajustan la estimación de un proyecto, están relacionados con:

- Experiencia en aplicaciones
- Factores de comunicación
- Experiencia en el lenguaje y las herramientas
- Continuidad del personal
- Experiencia en la plataforma
- Capacidad de los programadores
- Capacidad de los analistas

Dependiendo de factores humanos el proyecto puede variar en relación al equipo de desarrollo adecuado y sin deficiencias.

2.5.3. ASPECTOS HUMANOS

El hecho que una metodología base sus principios en las personas no significa que se han resuelto todos los problemas del desarrollo y que teniendo un buen equipo el éxito está garantizado. Todo lo contrario, el énfasis propuesto requiere tomar ciertas consideraciones en relación a la naturaleza de las personas que no son necesarias en las metodologías tradicionales. La base formulada por [Cockburn, 1999] que las personas son organismos vivientes impredecibles, inconsistentes pero con buena voluntad y un sentido de hacer las cosas bien, establece ciertas suposiciones que han sido esenciales en las prácticas tradicionales de management basadas en la ideología de *Command and Control*.

Uno de los elementos fundamentales asociados a las personas y las interacciones es el *Equipo de Desarrollo*. Dado el incremento continuo en la complejidad del software, no alcanza con esfuerzos individuales para tener éxito. Se deben contar con equipos productivos que pueden construir software en forma iterativa, en los que exista un alto grado de comunicación y donde cada integrante se sienta parte de una comunidad de personas con un interés en común: construir software de calidad. Estos equipos de desarrollo deben cubrir dos aspectos que según [Royce, 1998] definen la excelencia de un equipo: *balance* y *cobertura*.

2.5.4. ORIENTACIÓN PRODUCTOS VS TAREAS

El término timeboxes o iteración indican la clara tendencia a focalizarse en los productos a ser entregados, restando importancia a las tareas necesarias para obtener los correspondientes entregables. Por ejemplo en la pruebas o testing, las metodologías tradicionales utilizan dos formas: *Caja Negra y Caja Blanca*, el primero se refiere a la noción de tomar ciertas entradas y verificar las correspondientes salidas, y el segundo refiere a tomar ciertas entradas, analizar el proceso interno y verificar las correspondientes salidas.

Las metodologías tradicionales al considerar a las personas como

partes reemplazables de un desarrollo, definen en detalle entradas, salidas y tareas necesarias para llevar a cabo la correspondiente transformación, proponen que basta con tener una clara definición de las tareas en un proyecto para garantizar el éxito del mismo. AL considerar a las personas como componentes predecibles y reemplazables, el desarrollo de *caja blanca* se convierte en la forma más adecuada de llevar un proyecto al éxito en organizaciones orientadas al proceso.

El principio de *Máxima Comunicación* es la necesidad de tener un feedback continuo por parte del cliente, el énfasis de cualquier iteración esta puesto en los artefactos que esta genera, las actividades necesarias para la transformación de entradas en salidas queda a discreción del equipo de desarrollo. Para lograr que el desarrollo de *caja negra* tenga éxito se debe tener un equipo integrado, balanceado y altamente motivado.

La motivación es el factor que logra sacar lo mejor de un equipo de desarrollo; que permite saber que las personas involucradas utilizarán las técnicas que les sean más efectivas para llegar al final de la iteración con los artefactos más críticos construidos de acuerdo a las estimaciones. La motivación es por lejos el mayor contribuyente a la productividad [Boehm, 1981].

2.5.5. ITERACTIVIDAD

Con el avance de la tecnología, los tiempos de procesamiento han disminuido, mientras que los sistemas han crecido en complejidad, esto genera mayores necesidades de distribución, seguridad, mantenimiento, flexibilidad, crecimiento, etc.

En la actualidad los sistemas deben contar con atributos como: multiprocesamiento, sistemas distribuidos, middleware, web services, modelo en tres capas, tolerancia 24x7, multiplataforma, Virtual Machine, que son requerimientos no funcionales típicos de sistemas de software/hardware. Los “nuevos” clientes de los actuales sistemas esperan tener su sistema en el menor tiempo posible, con la máxima calidad y

conforme a los requerimientos no funcionales necesarios para la operación diaria del negocio.

El desarrollo Iterativo es factor crítico en una guía metodológica, ya que en cada iteración, tarea o actividad, la comunicación entre el usuario y los desarrolladores es continua, esto permite:

- Determinar el alcance de lo que se va a construir por iteración
- Funcionalidad del producto y resolver las necesidades del cliente
- Feedback del desarrollo al cliente
- Menor nivel de capacitación
- Tareas analizables y predecibles
- Mínima variabilidad
- Minimización de riesgo y corrección de errores
- Continua priorización de la funcionalidad
- Grupos de trabajo por iteración
- Estandarización de entregables

2.6. SISTEMAS DISTRIBUIDOS

Un Sistema Distribuido es un conjunto de componentes localizados en computadores, conectados en red, que comunican y coordinan sus acciones únicamente mediante el paso de mensajes. Los computadores que están conectados mediante una red pueden estar separados espacialmente por cualquier distancia, pueden estar en continentes distintos, en el mismo edificio o en la misma habitación. Sus características son: Concurrencia de los Componentes, Carencia de un reloj global y Fallos independientes de los componentes. Como ejemplo se tiene: Internet, Intranet, Computación móvil y ubicua.

2.6.1. Concurrencia

En una red de computadores, la ejecución de programas concurrentes es la norma. El usuario puede realizar su trabajo en su computador, mientras otro realiza su trabajo en la suya, compartiendo recursos como páginas Web o ficheros, cuando es necesario. La capacidad para manejar recursos

compartidos se puede incrementar añadiendo más recursos (computadores) a la red.

2.6.2. Inexistencia de reloj global

Cuando los programas necesitan cooperar coordinan sus acciones mediante el intercambio de mensajes, la coordinación estrecha depende a menudo de una idea compartida del instante en el que ocurren las acciones de los programas. Pero resulta que hay límites a la precisión con lo que los computadores en una red pueden sincronizar sus relojes, no hay una única noción global del tiempo correcto. Esto es una consecuencia directa del hecho que la única comunicación se realiza enviando mensajes a través de la red, lo que puede presentar problemas de temporización, para lo cual si hay soluciones.

2.6.3. Fallos independientes

Todos los sistemas informáticos pueden fallar y los diseñadores de Sistemas tienen la responsabilidad de planificar las consecuencias de posibles fallos. Los sistemas distribuidos pueden fallar de nuevas forma y los fallos en la red producen el aislamiento de los computadores conectados a él, pero eso no significa que detengan su ejecución. De hecho, los programas que se ejecutan en ellos pueden no ser capaces de detectar cuando la red ha fallado o está excesivamente lenta. De forma similar, la parada de un computador o la terminación inesperada de un programa en alguna parte del sistema *(crash)* no se da a conocer inmediatamente a lo demás componentes con los que se comunica. Cada componente del sistema puede fallar independientemente, permitiendo que los demás continúen su ejecución.

Los sistemas distribuidos están por todas partes. Internet permite que los usuarios de todo el mundo accedan a sus servicios donde quiera que estén situados. Cada organización administra una intranet, que provee servicios locales y servicios de Internet a los usuarios locales y

habitualmente proporciona servicios a otros usuarios de Internet. Es posible construir pequeños sistemas distribuidos con computadores portátiles y otros dispositivos computacionales pequeños conectados a una red inalámbrica.

La compartición de recursos es el principal factor que motiva la construcción de sistemas Distribuidos. Recursos como impresoras, archivos, páginas Web o registros de bases de datos se administran mediante servidores del tipo apropiado. Por ejemplo los servidores Web administran páginas y otros recursos Web. Los recursos son accedidos por clientes, por ejemplo, los clientes de los servidores Web se llaman normalmente visualizadores o navegadores Web. La construcción de los sistemas distribuidos presenta muchos desafíos:

2.6.4. Heterogeneidad

Debe construirse desde una variedad de diferentes redes, sistemas operativos, hardware de computador y lenguajes de programación. Los protocolos de comunicación de Internet enmascaran las diferencias entre redes y el middleware puede tratar con las diferencias restantes.

2.6.5. Extensibilidad

Los sistemas distribuidos deberían ser extensibles, el primer paso es la publicación de las interfaces de sus componentes, pero la integración de componentes escritos por diferentes programadores es un auténtico reto.

2.6.6. Seguridad

Se puede emplear encriptación para proporcionar una protección adecuada a los recursos compartidos y mantener secreta la información sensible cuando se transmite un mensaje a través de la red. Los ataques de denegación de servicio son aún un problema.

2.6.7. Escalabilidad

Un sistema distribuido es escalable si el costo de añadir un usuario es una cantidad constante en términos de recursos que se deberán añadir. Los algoritmos empleados para acceder a los datos compartidos deberían evitar cuellos de botella y los datos deberían estar estructurados jerárquicamente para dar los mejores tiempos de acceso. Los datos frecuentemente accedidos pudieran estar replicados.

2.6.8. Tratamiento de fallos:

Cualquier proceso, computador o red puede fallar independientemente de los otros. En consecuencia cada componente necesita estar al tanto de las formas posibles en que pueden fallar los componentes de los que depende y estar diseñado para tratar apropiadamente con cada uno de estos fallos.

2.6.9. Concurrencia

La presencia de múltiples usuarios en un sistema distribuido es una fuente de peticiones concurrentes a sus recursos. Cada recurso debe estar diseñado para ser seguro en un entorno concurrente.

2.6.10. Transparencia

El objetivo es que ciertos aspectos de la distribución sean invisibles al programador de aplicaciones, de modo que sólo necesite ocuparse del diseño de su aplicación particular. Por ejemplo, no debe ocuparse de su ubicación o los detalles sobre cómo se accede a sus operaciones por otros componentes, o si será replicado o migrado. Incluso los fallos de las redes y los procesos pueden presentarse a los programadores de aplicaciones en forma de excepciones, aunque deban de ser tratados.

2.7. PROTOCOLO TCP/IP

2.7.1. Protocolo de Red o Comunicación

Es el conjunto de reglas que especifican el intercambio de datos u órdenes durante la comunicación entre las entidades que forman parte de una red. Un protocolo es una convención, o estándar, o acuerdo entre partes que regulan la conexión, la comunicación y la transferencia de datos entre dos sistemas. En su forma más simple, un protocolo se puede definir como las reglas que gobiernan la semántica (significado de lo que se comunica), la sintaxis (forma en que se expresa) y la sincronización (quién y cuándo transmite) de la comunicación.

Los protocolos pueden estar implementados bien en hardware (tarjetas de red), software (drivers), o una combinación de ambos.

2.7.2. Propiedades Típicas

Al hablar de protocolos no se puede generalizar, debido a la gran amplitud de campos que cubren, tanto en propósito como en especificaciones, no obstante, la mayoría de los protocolos especifican una o más de las siguientes propiedades:

- Detección de la conexión física sobre la que se realiza la conexión (cableada o sin cables)
- Pasos necesarios para comenzar a comunicarse (Handshaking)
- Negociación de las características de la conexión.
- Cómo se inicia y cómo termina un mensaje.
- Formato de los mensajes.
- Qué hacer con los mensajes erróneos o corruptos (corrección de errores)
- Cómo detectar la pérdida inesperada de la conexión, y qué hacer en ese caso.
- Terminación de la sesión de conexión.
- Estrategias para asegurar la seguridad (autenticación, encriptación)

El Protocolo TCP/IP es un conjunto de protocolos de red que implementa la pila de protocolos en la que se basa Internet y que permiten la transmisión de datos entre redes de computadoras. En referencia a los dos protocolos más importantes que la componen: Protocolo de Control de Transmisión (TCP) y Protocolo de Internet (IP), que fueron los dos primeros en definirse, y que son los más utilizados de la familia.

Existen tantos protocolos en este conjunto que llegan a ser más de 100 diferentes, entre ellos se encuentra:

- HTTP (HyperText Transfer Protocol), el más popular, que es el que se utiliza para acceder a las páginas Web
- ARP (Address Resolution Protocol) para la resolución de direcciones
- FTP (File Transfer Protocol) para transferencia de archivos
- SMTP (Simple Mail Transfer Protocol) y el POP (Post Office Protocol) para correo electrónico,
- TELNET para acceder a equipos remotos, entre otros.
- El TCP/IP es la base de Internet, y sirve para enlazar computadores que utilizan diferentes sistemas operativos, incluyendo PC, dispositivos y servidores sobre redes de área local (LAN) y área extensa (WAN).

La familia de protocolos de internet puede describirse por analogía con el modelo OSI, que describe los niveles o capas de la pila de protocolos, aunque en la práctica no corresponde exactamente con el modelo en Internet. En una pila de protocolos, cada nivel soluciona una serie de problemas relacionados con la transmisión de datos, y proporciona un servicio bien definido a los niveles más altos. Los niveles superiores son los más cercanos al usuario y tratan con datos más abstractos, dejando a los niveles más bajos la labor de traducir los datos de forma que sean físicamente manipulables.

2.7.3. Niveles de la Pila TCP/IP

El modelo OSI no está lo suficientemente dotado en los niveles inferiores como para detallar la auténtica estratificación en niveles: necesitaría tener una capa extra (el nivel de Interred) entre los niveles de transporte y red. Protocolos específicos de un tipo concreto de red, que se sitúan por encima del marco de hardware básico, pertenecen al nivel de red, pero sin serlo. Ejemplos de estos protocolos son el ARP (Protocolo de resolución de direcciones) y el STP (Spanning Tree Protocol). De todas formas, estos son protocolos locales, y trabajan por debajo de las capas de Intered. Cierto es que situar ambos grupos (sin mencionar los protocolos que forman parte del nivel de Interred pero se sitúan por encima de los protocolos de Interred, como ICMP) todos en la misma capa puede producir confusión, pero el modelo OSI no llega a ese nivel de complejidad para ser más útil como modelo de referencia.

El siguiente diagrama intenta mostrar la pila TCP/IP y otros protocolos relacionados con el modelo OSI original:

Normalmente, los tres niveles superiores del modelo OSI (Aplicación, Presentación y Sesión) son considerados simplemente como el nivel de aplicación en el conjunto TCP/IP. Como TCP/IP no tiene un nivel de sesión unificado sobre el que los niveles superiores se sostengan, estas funciones son típicamente desempeñadas (o ignoradas) por las aplicaciones de usuario. La diferencia más notable entre los modelos de TCP/IP y OSI es el nivel de Aplicación, en TCP/IP se integran algunos niveles del modelo OSI en su nivel de Aplicación. Una interpretación simplificada de la pila se muestra debajo:

2.7.3.1. Nivel Físico

Describe las características físicas de la comunicación, las convenciones sobre la naturaleza del medio usado para la comunicación (como las comunicaciones por cable, fibra óptica o radio), y todo lo relativo a

los detalles como los conectores, código de canales y modulación, potencias de señal, longitudes de onda, sincronización y temporización y distancias máximas. La familia de protocolos de Internet no cubre el nivel físico de ninguna red

2.7.3.2. Nivel de Enlace de datos

Describe como son transportados los paquetes sobre el nivel físico, incluido los delimitadores (patrones de bits concretos que marcan el comienzo y el fin de cada trama). Ethetnet, por ejemplo, incluye campos en la cabecera de la trama que especifican que máquina o máquinas de la red son las destinatarias de la trama. Ejemplos de protocolos de nivel de red de datos son: Ethernet, Gíreles Ethernet, SLIP, Token Ring y ATM. PPP es un poco más complejo y originalmente fue diseñado como un protocolo separado que funcionaba sobre otro nivel de enlace, HDLC/SDLC.Este nivel es a veces subdividido en Control de enlace lógico (Logical Link Control) y Control de acceso al medio (Media Access Control).

2.7.3.3. Nivel de Interred

El nivel de red soluciona el problema de conseguir transportar paquetes a través de una red sencilla. Ejemplos de protocolos son X.25 y Host/IMP Protocol de ARPANET. El concepto de Interred, incluye nuevas funcionalidades, basadas en el intercambio de datos entre una red origen y una red destino, incluye un enrutamiento de paquetes a través de una red de redes.

En la familia de protocolos de Internet, IP realiza las tareas básicas para conseguir transportar datos desde un origen a un destino. IP puede pasar los datos a una serie de protocolos superiores; cada uno de esos protocolos es identificado con un único "Número de protocolo IP". ICMP y IGMP son los protocolos 1 y 2, respectivamente.

Algunos de los protocolos por encima de IP como ICMP (usado para transmitir información de diagnóstico sobre transmisiones IP) e IGMP (usado

para dirigir tráfico multicast) van en niveles superiores a IP pero realizan funciones del nivel de red e ilustran una incompatibilidad entre los modelos de Internet y OSI. Todos los protocolos de enrutamiento, como BGP, OSPF, y RIP son realmente también parte del nivel de red, aunque ellos parecen pertenecer a niveles más altos en la pila.

2.7.3.4. Nivel de Transporte

Determinan a que aplicación va destinados los datos, cubre las necesidades de fiabilidad a quien va dirigido y la seguridad de que los datos llegan en el orden correcto.

Los protocolos de enrutamiento dinámico que técnicamente encajan en el conjunto de protocolos TCP/IP (ya que funcionan sobre IP) son generalmente considerados parte del nivel de red; un ejemplo es OSPF (protocolo IP número 89). TCP (protocolo IP número 6) es un mecanismo de transporte fiable y orientado a conexión, que proporciona un flujo fiable de bytes, que asegura que los datos llegan completos, sin daños y en orden. TCP realiza continuamente medidas sobre el estado de la red para evitar sobrecargarla con demasiado tráfico. Además, TCP trata de enviar todos los datos correctamente en la secuencia especificada. Esta es una de las principales diferencias con UDP, y puede convertirse en una desventaja en flujos en tiempo real (muy sensibles a la variación del retardo) o aplicaciones de enrutamiento con porcentajes altos de pérdida en el nivel de interred.

Más reciente es SCTP, también un mecanismo fiable y orientado a conexión. Está relacionado con la orientación a byte, y proporciona múltiples sub-flujos multiplexados sobre la misma conexión. También proporciona soporte de multihoming, donde una conexión puede ser representada por múltiples direcciones IP (representando múltiples interfaces físicas), así si una falla la conexión no se interrumpe. Fue desarrollado inicialmente para aplicaciones telefónicas (para transportar SS7 sobre IP), pero también fue usado para otras aplicaciones.

UDP (protocolo IP número 17) es un protocolo de datagramas sin conexión. Es un protocolo no fiable (best effort al igual que IP) - no porque sea particularmente malo, sino porque no verifica que los paquetes lleguen a su destino, y no da garantías de que lleguen en orden. Si una aplicación requiere estas características, debe llevarlas a cabo por sí misma o usar TCP.

UDP es usado normalmente para aplicaciones de streaming (audio, video, etc) donde la llegada a tiempo de los paquetes es más importante que la fiabilidad, o para aplicaciones simples de tipo petición/respuesta como el servicio DNS, donde la sobrecarga de las cabeceras que aportan la fiabilidad es desproporcionada para el tamaño de los paquetes. DCCP está actualmente bajo desarrollo por el IETF. Proporciona semántica de control para flujos TCP, mientras de cara al usuario se da un servicio de datagramas UDP..

TCP y UDP son usados para dar servicio a una serie de aplicaciones de alto nivel. Las aplicaciones con una dirección de red dada son distinguibles entre sí por su numero de puerto TCP o UDP. Por convención, los puertos bien conocidos (well-known ports) son asociados con aplicaciones específicas.

RTP es un protocolo de datagramas que ha sido diseñado para datos en tiempo real como el streaming de audio y video que se monta sobre UDP.

2.7.3.5. Nivel de Aplicación

Describe la comunicación a través de la red con otros programas. Los procesos que acontecen en este nivel son aplicaciones específicas que pasan los datos al nivel de aplicación en el formato que internamente use el programa y es codificado de acuerdo con un protocolo estándar.

Algunos programas específicos se considera que se ejecutan en este nivel. Proporcionan servicios que directamente trabajan con las aplicaciones

de usuario. Estos programas y sus correspondientes protocolos incluyen a HTTP (HyperText Transfer Protocol), FTP (Transferencia de archivos), SMTP (correo electrónico), SSH (login remoto seguro), DNS (Resolución de nombres de dominio) y a muchos otros.

Una vez que los datos de la aplicación han sido codificados en un protocolo estándar del nivel de aplicación son pasados hacia abajo al siguiente nivel de la pila de protocolos TCP/IP.

2.8. BASES DE DATOS DISTRIBUIDAS

2.8.1. CARACTERÍSTICAS

Un Sistema de Bases de Datos Distribuidas es almacenar la información en un conjunto de computadores que están interconectados entre sí, poseen las siguientes características:

- Mantienen un sistema de Base de Datos Local
- Pueden procesar transacciones locales o globales entre varios computadores,
- No comparten la memoria principal ni el reloj.
- Pueden variar en cuanto su tamaño y función.
- Reciben diferentes nombres, tales como localidades, nodos o computadores.
- Se interconectan por diferentes medios físicos o inalámbricos
- Forman redes de larga distancia (WAN) o locales (LAN)

Las localidades pueden conectarse físicamente de diversas formas, las principales son:

- Red totalmente conectada
- Red prácticamente conectada
- Red con estructura de árbol
- Red de estrella
- Red de anillo

Las diferencias principales entre estas configuraciones son:

- ***Costo de instalación***, conectar físicamente las localidades.
- ***Costo de comunicación***, tiempo y dinero en enviar un mensaje.
- ***Fiabilidad***, frecuencia con que falla una línea de comunicación.
- ***Disponibilidad*****,** posibilidad de acceder a pesar de fallos.

2.8.2. CONSIDERACIONES PARA DISTRIBUIR

Existen varias razones para construir Sistemas Distribuidos de Bases de Datos que incluyen: compartir la información, fiabilidad y disponibilidad y agilizar el procesamiento de las consultas. Pero también tiene sus desventajas, como desarrollos de software más costosos, mayor posibilidad de errores y costos extras de procesamiento.

2.8.2.1. Ventajas de la distribución de datos

- Capacidad de compartir y acceder a la información de una forma fiable y eficaz.
- Distribución de la responsabilidad y control hasta cierto punto en cada localidad
- Cada Administrador local tiene su propia Autonomía
- Si se produce un fallo en una localidad, las otras pueden seguir operando y no implica una desactivación del sistema
- Los datos se pueden encontrar en diferentes localidades, siempre están disponibles
- División de una consulta extensa en varias subconsultas que se ejecuten en paralelo en distintas localidades, para agilitar el proceso.
- El sistema puede pasar la consulta a las localidades más ligeras de carga.

2.8.2.2. Desventajas de la distribución de los datos

- Mayor complejidad para garantizar la coordinación entre localidades
- Costo del desarrollo de software
- Costo de comunicaciones y mantenimiento

- Mayor posibilidad de errores, puesto que las localidades operan en paralelo
- Es más difícil estructurar un sistema de bases de datos distribuidos
- Es más difícil garantizar que los algoritmos sean correctos.
- Mayor tiempo extra de procesamiento según número de localidades
- Disminución del tiempo de respuesta, caudado por intercambio de mensajes y los cálculos adicionales.

2.8.3. TRANSPARENCIA Y AUTONOMÍA

La transparencia de la red es el grado hasta el cual los usuarios del sistema deben ignorar los detalles del diseño distribuido y como esta almacenada la infamación. La autonomía local es el grado hasta el cual el diseñador o administrador de una localidad puede ser independientes del resto del sistema distribuido. La transparencia y autonomía se aplica a los siguientes puntos:

- Nombre de los datos
- Repetición de los datos.
- Fragmentación de los datos.
- Localización de los fragmentos y copias.

2.8.3.1. Asignación de Nombres y Autonomía Local

Todo elemento de información de una base de datos debe tener un nombre único y las distintas localidades deben asegurar el no utilizar el mismo nombre para dos datos diferentes. Es posible que un asignador de nombres centralizado solucione el problema pero puede convertirse en un cuello de botella y dependiente. Otra solución es utilizar un identificador por localidad, pero no genera transparencia en la red, además cada copia y fragmento de un elemento debe tener también un nombre único.

2.8.3.2. Transparencia de la repetición y la fragmentación

El sistema debe ser quien determine a qué copia debe acceder cuando se le solicite su lectura, y debe modificar todas las copias cuando se

produzca una petición de escritura, para lo cual se utiliza una tabla-catálogo para determinar cuáles son todas las copias de ese dato.

2.8.3.3. Transparencia de localización

El sistema debe ser transparente en cuanto la repetición y fragmentación, es decir ocultará al usuario el esquema de la base de datos distribuida, para lo cual se utiliza seudónimos o alias, y el administrador de la base de datos puede cambiar un dato de una localidad a otra sin afectar a los usuarios.

2.8.4. DISEÑO Y ALTERNATIVAS

El diseño de un sistema de base de datos distribuido implica la toma de decisiones sobre la ubicación de los programas que accederán a la base de datos y sobre los propios a lo largo de las diferentes localidades. La distribución se basa en 3 niveles:

- Nivel de Compartición,
- Nivel de Acceso a Datos
- Nivel de Conocimiento de Acceso

2.8.4.1. Nivel de Compartición

- ***Inexistencia*****,** cada aplicación y sus datos se ejecutan en un ordenador con ausencia total de comunicación con otros programas u otros datos
- ***Compartir sólo los datos y no los programas*****,** en tal caso existe una réplica de las aplicaciones en cada máquina y los datos viajan por la red;
- ***Compartir datos y programas*****,** dado un programa ubicado en un determinado sitio, éste puede solicitar un servicio a otro programa localizado en un segundo lugar, el cual podrá acceder a los datos situados en un tercer emplazamiento.

2.8.4.2. Nivel de Acceso a los Datos

- ***Estático*****,** el acceso a los datos no cambia a lo largo del tiempo
- ***Dinámico*****,** al acceso sufre variación a lo largo del tiempo

Se establece la relación entre el diseño de bases de datos distribuidas y el procesamiento de consultas.

2.8.4.3. Nivel de Conocimiento de Acceso

- ***Carecer de información,*** sobre cómo los usuarios acceden a la base de datos, difícil de diseñar
- ***Conocer con detenimiento*****,** la forma de acceso de los usuarios o, en el caso de su imposibilidad, conformarnos con una información parcial de ésta.

2.8.4.4. Alternativas

Existe 2 procesos fundamentales sobre las cuales se basa el diseño: fragmentación y asignación, como se muestra en la figura 5.

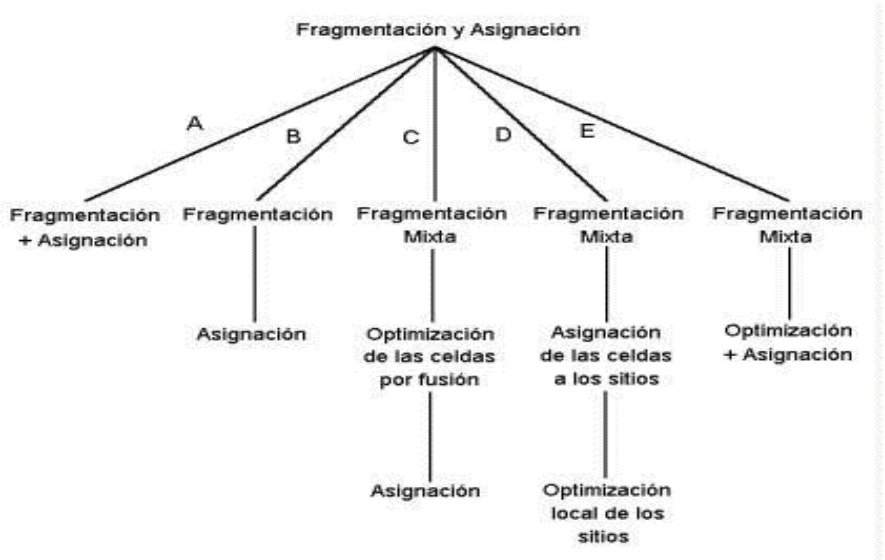


FIGURA 5 Diagrama de Fragmentación y Asignación

2.8.5. TIPOS DE FRAGMENTACIÓN

Una relación se corresponde esencialmente con una tabla y la cuestión consiste en dividirla en fragmentos menores, para lo cual existe:

- ***Fragmentación Horizontal,*** trabaja sobre las tuplas, divide la relación en subrelaciones que contienen un subconjunto de las tuplas que alberga la primera
- ***Fragmentación Vertical*****,** se basa en los atributos de la relación para efectuar la división.
- ***Fragmentación Mixta VH,*** primero fragmentación vertical y luego partición horizontal sobre los fragmentos verticales.
- ***Fragmentación Mixta HV****,* primero fragmentación horizontal y luego partición vertical sobre los fragmentos horizontales.
- ***Fragmentación Mixta Celdas,*** fragmentación horizontal y vertical en forma simultánea y no secuencial, genera celdas que alcanzan el grado máximo de fragmentación.

2.8.5.1. Grado de Fragmentación

El grado de fragmentación que una base de datos alcanza, es un factor que influye notablemente en el acceso y ejecución de consultas, el grado puede variar desde la ausencia de división que considera a las relaciones como unidades de fragmentación, hasta el grado en el que cada tupla o atributo forme un fragmento. Lo óptimo es buscar el punto intermedio basado en las características de las aplicaciones.

2.8.5.2. Reglas de Corrección de la Fragmentación

Estas reglas permiten asegurar la ausencia de cambios semánticos en la base de datos durante el proceso.

- ***Complación,*** esta regla asegura que los datos de la relación global se proyecten sobre los fragmentos sin pérdida alguna, tener en cuenta que en horizontal el elemento de datos es una tupla y en vertical es un atributo.
- ***Reconstrucción,*** esta regla asegura la reconstrucción de la relación a partir de sus fragmentos y la preservación de las restricciones definidas sobre los datos en forma de dependencias.

- ***Disyunción,*** esta regla asegura que los fragmentos horizontales sean disjuntos. Si una relación se descompone verticalmente, sus atributos primarios clave normalmente se repiten en todos sus fragmentos.

2.8.5.3. Alternativas de Asignación

Luego de fragmentar la base de datos, se debe asignar los fragmentos a los distintos sitios de la red, existe 3 alternativas: replica total, replica parcial y partición, y los factores sobre los cuales se analiza son: procesamiento de consultas, gestión de directorio, control de concurrencia, seguridad y realidad. En la tabla 6 se muestra el grado de asignación.

TABLA 6

Grado de Asignación de Fragmentos

	Réplica total	**Répli.Parcial**	**Partición**
Procesamiento de consultas	Fácil	dificultad	Similar
Gestión del directorio	fácil o inexistente	Dificultad	Similar
Control de concurrencia	Moderado	Difícil	Fácil
Seguridad	muy alta	Alta	Baja
Realidad	posible aplicación	Realista	Posible

2.8.6. FRAGMENTACIÓN HORIZONTAL

La fragmentación horizontal se realiza sobre las tuplas de la relación y cada fragmento es un subconjunto de las tuplas de la relación. Existen dos variaciones: fragmentación horizontal primaria, esta se desarrolla empleando los predicados definidos en esa relación; y fragmentación horizontal derivada, que consiste en dividir una relación partiendo de los predicados definidos sobre alguna otra.

Para fragmentar adecuadamente y optimizar los algoritmos, es necesario reunir información sobre la base de datos (estructura, PK, FK) e información sobre las aplicaciones (frecuencia de accesos, consultas y actualizaciones, índices)

2.8.7. FRAGMENTACIÓN VERTICAL

La fragmentación vertical consiste en producir una serie de fragmentos que contienen un subconjunto de los atributos de una relación, incluida la clave primaria. De tal forma que minimice el tiempo de ejecución de las aplicaciones que utilicen estos fragmentos. La partición vertical resulta más complicada que la horizontal, debido al aumento del número total de alternativas que tenemos disponibles. Existen dos enfoques heurísticos para la fragmentación vertical de relaciones:

- ***Agrupación*****,** comienza asignando cada atributo a un fragmento, y en cada paso, junta algunos de los fragmentos hasta que satisface un determinado criterio.
- ***Escisión,*** a partir de la relación se deciden que fragmentos resultan mejores, basándose en las características de acceso de las aplicaciones a los atributos.

Para fragmentar adecuadamente y optimizar los algoritmos, es necesario reunir información sobre la afinidad de los atributos y la estrecha relación entre ellos, además la frecuencia de acceso simultaneo.

2.8.8. FRAGMENTACIÓN MIXTA O HIBRIDA

Este tipo de fragmentación es una mezcla de las dos anteriores, la cuales se aplican en forma secuencial o simultánea. Cuando al proceso de fragmentación vertical le sigue una horizontal, es decir, se fragmentan horizontalmente los fragmentos verticales resultantes, se habla de la fragmentación mixta HV, caso contrario, es una fragmentación VH. Una característica común a ambas es la generación de árboles que representan la estructura de fragmentación, se debe tener en cuenta el número de niveles del árbol ya que este influirá en el rendimiento.

La fragmentación mixta basada en celdas, es la generación de fragmentos horizontales y verticales de forma simultanea sobre la relación, los algoritmos como primer paso realizan un fragmentación máxima (un

atributo y una tupla), para luego optimizar la celda mediante la fusión o desfragmentación, acorde con las aplicaciones y localidades en la red.

Para fragmentar adecuadamente y optimizar los algoritmos, es necesario reunir información sobre la base de datos, las aplicaciones que funcionan sobre ella, la red de comunicaciones, las características de proceso, y el límite de almacenamiento de cada sitio de la red.

2.9. PARADIGMA DE LO ORIENTADO A OBJETOS

2.9.1. INTRODUCCIÓN AL ENFOQUE O.O

El término OOP (Programación Orientado a Objetos) indica más una forma de diseño y una metodología de desarrollo que un lenguaje de programación, ya que en realidad se pueden aplicar los principios del paradigma de programación orientada a objetos (encapsulación, herencia y polimorfismo) en cualquier lenguaje de programación. Para conseguirlo, la clave consiste en aplicar los principios de la POO en el momento del diseño del software.

A menudo se confunde la Orientación a Objetos con sistemas de ventanas, iconos y similares, Interfaces Gráficas de Usuario. Esto es debido a que se usan técnicas orientadas a objetos para construir estos entornos. También es razonable pensar que todo lo programado por lenguajes orientados a objetos es O.O.P. Esto no es cierto, ya que incluso en un lenguaje orientado a objetos puro, es posible que un mal diseño lleve a una implementación que en realidad no siga los principios de Orientación a Objetos mencionados.

El Objetivo es, dado un problema a automatizar (“resolver” mediante una aplicación), hacer una aplicación de la mejor manera posible en todos los aspectos del desarrollo de software. Los Aspectos importantes que determinan el desarrollo de software son:

- ***Portabilidad.-*** El software desarrollado debe funcionar independientemente del hardware y del entorno de funcionamiento o Sistema Operativo sobre el cual se ejecute.

- ***Productividad.-*** Uno de los principales objetivos es simplificar el proceso de desarrollo de Software, para lo cual es fundamental **reutilizar** componentes.
- ***Mantenimiento.-*** Constituye una de las etapas más costosas en el desarrollo de software que, por tanto, habrá de intentarse optimizar. Es fundamental estructurar bien el programa para prever futuros cambios o modificaciones.
- ***Calidad.-*** A la hora de producir software, no hay que olvidar que se deben de desarrollar productos de calidad, dada a partir de una serie de aspectos como robustez, fiabilidad, eficiencia, flexibilidad.

2.9.2. CARACTERÍSTICAS

Las metodologías orientadas a objetos pueden facilitar la producción de sistemas cada vez más complejos, permiten modelar problemas no estructurados, incrementan la productividad gracias a la reutilización de objetos y facilitan el mantenimiento.

La orientación a objetos puede describirse cómo el *conjunto de disciplinas que desarrollan y modelizan software que facilitan la construcción de sistemas complejos a partir de componentes*.

Su característica más importante, frente a la programación estructurada, es que no trata de hacer más sencillo el problema a resolver (dividiéndolo), sino que trata de simular el *problema* a resolver. Así, en todo problema existen una serie de *entidades* (físicas o no) que interaccionan para resolverlo, de manera que simulando estas entidades, y su comportamiento, es posible resolver el problema en cuestión. Cada una de estas entidades es un objeto en el sistema desarrollado mediante OOP, de ahí el nombre de programación orientada a objetos.

El atractivo de la orientación a objetos es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible.

2.9.3. VENTAJAS SOBRE OTROS LENGUAJES

La OOP proporciona las siguientes ventajas sobre otros lenguajes de programación:

- **Uniformidad.-** pues la representación de los objetos lleva implícita tanto el análisis como el diseño y codificación. Así, todo el programa está compuesto de objetos que interaccionan entre sí.
- **Comprensión.-** Los programas están escritos mediante definiciones de clases, que representan las entidades que son necesarias para resolver el problema que se esté modelando. Los programas son más fáciles de comprender porque las clases modelan los tipos de entidades involucradas en el problema a resolver, simulando ese problema.
- **Flexibilidad.-** Al tener relacionados los procedimientos que manipulan los datos con los datos a tratar, cualquier cambio que se realice sobre ellos quedará reflejado automáticamente en cualquier lugar donde éstos aparezcan.
- **Reusabilidad.-** La noción de **objeto** permite que programas que traten las mismas estructuras de datos reutilicen las definiciones de clases empleadas en otros programas e incluso los procedimientos que los manipulan. De forma ideal, el desarrollo de un programa "nuevo" debería poder llegar a ser una simple combinación de objetos ya definidos en otros programas que se desarrollaron anteriormente.

La OOP no sustituye a ninguna metodología ni lenguaje de programación anterior, es un enfoque distinto. Todos los programas que se realizan según los principios de la OOP se pueden realizar igualmente mediante programación estructurada, aunque de forma más costosa, puesto que los lenguajes Orientados a Objetos soportan de forma natural la OOP. A la vez, las limitaciones observadas en el paradigma de programación estructurada son eliminadas (al menos, parcialmente), ya que un lenguaje Orientado a Objetos ya fuerza e invita a emplear los principios de OOP.

2.9.4. CONCEPTOS BÁSICOS DE O.O

2.9.4.1. Clases

Una clase permite describir objetos similares mediante la definición de sus estructuras de datos y métodos comunes. Las clases son plantillas para objetos, permitiendo la agrupación de objetos que comparten las mismas propiedades y comportamiento. Una **superclase** es una clase de más alto nivel que agrupa otras clases con propiedades y funciones comunes, por tanto, los objetos (también llamados **instancias** de una clase determinada) de esas otras clases son también objetos (**instancias**) de la superclase. Las clases que son agrupadas por una superclase son sus **subclases**.

2.9.4.2. Métodos y Mensajes

Los métodos, son *funciones miembro* de una clase, estas funciones tratan con los atributos del objeto como si fuesen variables locales a la función.

Los métodos, son el mecanismo de acceso y manipulación de los datos del objeto, constituyen la interfaz del objeto con el resto de objetos. Cada método está constituido por un conjunto de instrucciones escritas en un lenguaje de programación estructurado determinado, asociadas a un objeto determinado y cuya ejecución sólo puede desencadenarse a través de un mensaje recibido por este o por sus descendientes.

Los mensajes permiten la comunicación entre los objetos. Un mensaje le dice a un objeto lo que tiene que hacer, identificando un método y los operandos necesarios para su ejecución. Los objetos reciben, interpretan y responden a mensajes procedentes de otros objetos.

2.9.4.3. Atributos

Los **atributos de una clase** no son más que *datos miembro* de una clase y sirven para identificar claramente y de manera única a cada uno de los objetos. *El número de propiedades debe ser el mínimo para realizar todas las operaciones que requiere la aplicación*.

2.9.4.4. Objetos

Un objeto representa un elemento, unidad o entidad individual e identificable, ya sea real o abstracta, con un papel bien definido en el dominio del problema. Un OBJETO es la integración de tres aspectos:

- ***Relaciones****,* permiten que el objeto se integre en la organización de la que forma parte junto con otros objetos. Las relaciones pueden ser de tipo jerárquico (herencia) o semántico.
- ***Propiedades (atributos)****,* distinguen un objeto de los restantes (tamaño, posición, color). Cada propiedad tendrá un determinado valor. Las propiedades de un objeto pueden ser heredadas por sus descendientes.
- ***Métodos (operaciones)****,* conjunto de cosas que puede hacer un objeto, es un procedimiento o función que altera el estado de un objeto o hace que el objeto envíe un mensaje, es decir, que devuelva valores. Los métodos pueden ser heredados por sus descendientes.

2.9.4.5. Identidad

La identidad expresa en cada objeto, aunque dos objetos sean exactamente iguales en sus atributos, son distintos entre sí, cada objeto tiene un **controlador** por el cual se identifica. Éste puede ser una *variable,* una *estructura de datos,* una *cadena de caracteres,* etc. El controlador será distinto para cada uno de los objetos, aunque las referencias a éstos sean uniformes e independientes del contenido, permitiendo crear agrupaciones de objetos con el mismo tratamiento (clases).

2.9.4.6. Encapsulación y Ocultación de datos

Cada objeto mantiene sus propios datos internos y los usuarios de un objeto pueden acceder y manipular los datos internos solamente a través de sus métodos, de tal forma que la *interfaz* de un objeto es su conjunto de métodos públicos. Así, detalles internos de la implementación de un objeto como algoritmos o representaciones internas de datos, no son visibles. Esto

refuerza el ocultamiento de la información y permite un alto nivel de **abstracción**.

Esto no quiere decir que sea imposible conocer información de un objeto, sino que las peticiones de información a un objeto deben cursarse a través de **mensajes** al objeto con el fin de realizar determinada operación. La respuesta a esa petición será la información solicitada siempre que el objeto compruebe que el solicitante tiene acceso a ella.

2.9.4.7. Herencia

La herencia designa a los objetos la facultad de compartir propiedades y operaciones entre ellos. Como la herencia humana, existen unos determinados objetos **hijo** que heredan las propiedades y atributos de otros objetos **padres**. Existen dos tipos fundamentales de herencia: **simple** y **múltiple.**

La clase que se sitúa en el primer nivel superior se denomina clase **raíz**, mientras las clases de los niveles inferiores se denominan clases **hoja** o **terminales** y el resto se denominan clases intermedias.

Los objetos se instancian de clases terminales o muy cercanas a ellas, por tanto los objetos contienen miembros (atributos y métodos) propios y (definidos en su clase) como heredados.

2.9.4.8. Abstracción

La abstracción es una técnica de programación que permite definir nuevos tipos de datos por el usuario. La esencia de la abstracción es similar al uso de un tipo de dato de un lenguaje de programación, cuyo uso se realiza sin saber como está representado o implementado.

Las abstracciones deben ser completas, es decir, los objetos deben abstraer y encapsular tanto los datos como sus procesos (de hecho, la abstracción y la encapsulación están íntimamente ligadas). Una abstracción se centra en la vista externa de un objeto, de modo que sirva para separar el comportamiento esencial de un objeto de su implementación.

2.9.4.9. Polimorfismo

El *polimorfismo* consiste en la posibilidad de aplicar una misma operación sobre distintos objetos, y que esta operación varíe según el objeto al que se le aplique. Esto sucede en tiempo de ejecución, **NO** en tiempo de compilación, por lo que el polimorfismo está íntimamente ligado al enlace dinámico. La operación suele ser un método. Los objetos a los que se les puede aplicar este método están limitados por herencia: ese método se define en una clase y es aplicable, con los refinamientos subsiguientes, a todas sus subclases.

2.9.4.10. Ligadura Dinámica

Al proceso de seleccionar el método apropiado para enviar un mensaje según la clase del objeto se llama **ligadura** o **enlace.** La *ligadura estática* significa que se fijan los tipos de todas las variables y expresiones en tiempo de compilación; la *ligadura (o enlace) dinámica* (también llamada *ligadura tardía*) se produce cuando no es posible conocer la clase del objeto hasta el tiempo de ejecución. La *ligadura dinámica* es la capacidad de retrasar hasta el instante de la recepción del mensaje la decisión sobre la clase del objeto que lo recibe y el método concreto que debe de ejecutarse.

2.9.5. VENTAJAS DE PROGRAMAR O.O

- Un lenguaje de programación que soporta el paradigma OO proporciona al desarrollador de software una forma natural de modelar el mundo real, utilizando para ello clases de objetos.
- Los objetos bien diseñados en los sistemas OO constituyen la base para otros sistemas que se ensamblan, en gran parte, a partir de módulos reutilizables, lo que redunda en una mayor productividad. Esta es, quizás, la ventaja más conocida de la tecnología de objetos.
- La reutilización de clases existentes, que han sido probadas en proyectos anteriores,

- Conduce a la elaboración de sistemas de mayor calidad, que satisfacen mejor los requisitos de negocios y contienen menos errores.
- El trabajo de programación es más fácil en base al uso de bibliotecas de clases predefinidas.
- La POO, y la herencia en particular, hacen que sea posible utilizar y definir de forma clara módulos funcionalmente incompletos y, luego, permiten su extensión sin trastornar la operación de otros módulos o de sus clientes. Esto hace que los sistemas sean más flexibles, más fácilmente extensibles y de mantenimiento menos costoso.
- La convención de paso de mensajes para la comunicación entre objetos lleva a que las descripciones de la interfaz entre módulos y sistemas externos se haga más fácil. También facilita la descripción y la construcción de Interfaces Gráficas de Usuario.
- El ocultamiento de información contribuye a construir sistemas seguros.

Los posibles inconvenientes son:

- Un programa orientado a objetos se ejecuta más despacio sobre todo debido a la creación de objetos, la ligadura dinámica y al polimorfismo.
- El desarrollador de software tiene que conocer las bibliotecas de clases para sacar rendimiento al lenguaje.

2.9.6. MATRIZ COMPARATIVA DE LENGUAJES O.O

TABLA 7

Comparación de Lenguajes O.O

Características	SIMULA	SMALLTALK	OBJETIVE-C	C++
Tipo de Ligadura		Tardia	temprana o tardía	temprana o tardía
Polimorfismo	Si	Si	Si	Si
Ocultamiento información	Si	Si	Si	Si
Concurrencia	Si	escasa	escasa	escasa
Herencia	Si	Si	Si	Si
Herencia Múltiple	No	No	No	Si
Recolección de basura	Si	Si	Si	No
Persistencia	No	No	No	No
Genericidad	No	No	No	plantillas
Biblioteca de Objetos	Simulación	graficas	pocas	Bastantes

CONTIÚA →

Características	ADA	O.O.Pascal	Turbo Pascal	Java
Tipo de Ligadura	Temprana	tardía	temprana o tardía	temprana o tardía
Polimorfismo	Si	Si	Si	Si
Ocultamiento información	Si	Si	Si	Si
Concurrencia	Difícil	No	No	Si
Herencia	No	clases	Si	Si
Herencia Múltiple	No	No	No	No
Recolección de basura	No	Si	No	Si
Persistencia	No	No	No	Si
Genericidad	Si	No	No	Si
Biblioteca de Objetos	Pocas	pocas	Pocas	Si

2.10. HERRAMIENTAS DE ÁNALISIS Y DISEÑO O.O

2.10.1. MODELADO UML (Unified Modeling Lenguaje)

El modelado, o *modelo de objetos*, describe los conceptos principales de la orientación a objetos: las estructuras estáticas y sus relaciones. Las principales estructuras estáticas son los *objetos* y *clases*, los cuales están compuestos de *atributos* y *operaciones*, mientras que las principales relaciones entre objetos y entre clases corresponden a las *ligas* y *asociaciones*, respectivamente.

UML es una técnica basada en una serie de modelos estándares, que sirven como guía según el tipo de aplicación que se va a construir. Estos modelos UML sirven para describir distintas vistas de la arquitectura necesarias para la comunicación dentro del equipo. Entre los modelos necesario que se debe considerar son:

2.10.1.1. Diagrama de Despliegue

Este diagrama muestra cómo y donde el sistema será desplegado, en el mismo se muestran nodos físicos, procesadores, y los componentes que corren en cada uno. Como se observa este diagrama puede tener distintas implementaciones de acuerdo al grado de detalle que se quiera llegar. Sin embargo, es muy recomendable ya que muestra como la aplicación será llevada al ambiente de producción mediante la separación de los componentes en nodos.

2.10.1.2. Diagrama de Componentes

Este diagrama muestra las piezas de software o entidades que

conforman al sistema; en general, un componentes es de más alto nivel que una clase y suele ser implementado en tiempo de ejecución por un número de clases.

2.10.1.3. Diagrama de Secuencia

Este diagrama muestra una representación de las interacciones entre clases y objetos para llevar a cabo una operatoria a lo largo del tiempo. Se utiliza para mostrar flujos de trabajo, pasaje de mensajes y la cooperación necesaria para brindar un determinado resultado.

2.10.1.4. Diagrama de Estado

Este diagrama sirve para mostrar como un elemento (clase) cambia de estado a lo largo del tiempo y las transiciones que le son permitidas en cada caso con las correspondientes condiciones.

2.10.1.5. Diagrama de Clases

Este diagrama permite capturar la estructura lógica del sistema, las clases y/o entidades que componen al modelo. Es considerado un modelo estático ya que muestra las clases existentes y los atributos, métodos de cada una.

2.10.1.6. Diagrama de Datos

Este diagrama sirve para mostrar como el modelo será persistido en un RDBMS. El mismo está dentro del perfil de UML para modelado de datos y maneja estereotipos como Tabla, Primary Key, Foreign Key, etc.

Se debe tener en cuenta que debe existir un balance entre el esfuerzo en realizar estos diagramas y la posibilidad de comunicar dichas ideas en forma directa, de acuerdo al patrón de *Máxima Comunicación* conviene que estos diagramas estén en algún radiador de información ya sea mediante impresiones en hojas grandes colgadas, en una intranet de

administración de conocimiento, o simplemente dibujados en pizarrones en los casos que se desee más informalidad. Una vez que el propósito comunicacional es servido no tiene sentido seguir detallando y/o manteniendo estos diagramas.

2.11. RECURSOS HUMANOS BASADO EN COMPETENCIAS

2.11.1. INTRODUCCIÓN

Actualmente la competencia de los mercados, las condiciones y factores tradicionales, como la mano de obra, el acceso a recursos financieros y la materia prima, ofrecen ventajas competitivas menores que en el pasado. Las tendencias actuales exigen plantear nuevos conceptos para las organizaciones, es necesario visualizar a la empresa como un conjunto de recursos, capacidades y aptitudes centrales heterogéneos que pueden utilizarse para crear una ventaja con relación a otras empresas del mercado, lo que supone que cada empresa tiene recursos y capacidades que no poseen otras empresas, al menos no en la misma combinación.

Los recursos son la fuente de capacidades, que llevan al desarrollo de aptitudes centrales, con la cuales las empresas pueden desarrollar mejor sus actividades que sus competidores.

2.11.2. RECURSOS, CAPACIDADES Y APTITUDES

Los recursos, las capacidades y las aptitudes centrales son la base de la ventaja competitiva.

2.11.2.1. Recursos.

Son los insumos en el proceso de producción de la empresa, pueden ser tangibles o intangibles. Los recursos tangibles son los activos que se pueden ver y contar, la capacidad de pedir dinero prestado, las condiciones de su planta, etc. Generalmente el valor de los recursos tangibles se establece a través de los estados financieros. Los recursos intangibles van desde el derecho de propiedad intelectual, las patentes, las marcas registradas o dependen de ciertas personas con conocimientos prácticos,

etc. El valor estratégico de los recursos está determinado por el grado en que pueden contribuir al desarrollo de capacidades y aptitudes centrales y, finalmente, al logro de una ventaja competitiva. Debido a que son menos visibles, y más difíciles de comprender o imitar, generalmente se utilizan los recursos intangibles como base de las capacidades y aptitudes centrales.

2.11.2.2. Capacidades

Representan la habilidad de una empresa para aprovechar los recursos que se han integrado en forma intencional para lograr una condición deseada. Las capacidades están dadas por las habilidades y conocimientos de sus empleados, por ello no se debe subestimar el valor del Capital humano en el desarrollo y aplicación de las capacidades y obviamente en la creación de las aptitudes centrales.

2.11.2.3. Aptitudes Centrales

Consiste en acumular y aprender a aprovechar los diversos recursos y capacidades de la organización, es la habilidad para tomar acciones y decisiones que constituyen la esencia de una organización única para ofrecer valor a sus clientes durante un largo periodo. No todos los recursos y capacidades son estratégicos ni sirven como fuente de ventaja competitiva, por esto es necesario conocer todos los recursos de la organización para crear nuevas aptitudes centrales.

Existen cuatro criterios específicos para determinar cuáles recursos y capacidades son aptitudes centrales. Si una capacidad cumple con estos cuatro criterios se considera una ventaja competitiva sostenible.

- ***Valiosa.-*** Ayuda a neutralizar peligros o aprovechar las oportunidades
- ***Costosa de Imitar.-*** otras empresas no pueden desarrollar con facilidad
- ***Rara.-*** son poseídas por algunos competidores actuales o potenciales
- ***Insustituible.-*** no tienen equivalente estratégicos.

2.11.2.4. Cadena de Valor

La cadena de valor se utiliza para identificar y evaluar los recursos y capacidades de la empresa. Estudiando sus actividades primarias y de apoyo, las compañías entienden mejor su estructura de costos y las actividades mediante las cuales pueden crear y captar valor.

2.11.3. DEFINICIONES CONCEPTUALES

2.11.3.1. Competencias

Las competencias pueden consistir en motivos, rasgos de carácter, conceptos de uno mismo, actitudes o valores, contenido de conocimiento, o capacidades cognoscitivas o de conducta: cualquier característica individual que se pueda medir de un modo fiable, y que se pueda demostrar que diferencia de una manera significativa entre los trabajadores que mantienen un desempeño excelente de los adecuados o entre los trabajadores eficaces e ineficaces.

Algunos autores les designan con las siglas CHAI (conocimientos, habilidades, actitudes e intereses) que puestas en acción diferencian a unas personas de otras.

- **Competencias:** todas aquellas habilidades, cualidades, conocimientos, actitudes que permitan al trabajador tener un desempeño superior (sobre la media) en cualquier puesto de trabajo, que pueda ser medidas y controladas y que de esta forma diferencia a un trabajador distinguido, de un trabajador meramente hacedor de su trabajo.
- **Habilidades/destrezas:** es la capacidad adquirida de ejecutar labores, tareas o acciones en forma destacada producto de la práctica y del conocimiento.
- **Cualidades:** rasgos del carácter de los individuos que le predisponen a realizar determinado tipo de tareas, acciones o labores en forma excelente.
- **Conocimiento:** es la información que se adquiere en forma teórica o empírica y que es procesada en el ámbito mental de acuerdo a las

experiencias anteriores del sujeto poseedor de este conocimiento y que son la base cognitiva que le permiten desarrollar labores, acciones o tareas.

- **Actitudes:** Inclinación de las personas a realizar determinado tipo de labores, tareas o acciones, que se generan por las motivaciones y conocimientos del individuo

2.11.3.2. Gestión por Competencias

- Detectar las competencias que requiere un puesto de trabajo para que quien lo desarrolle mantenga un rendimiento elevado o superior a la media.
- Determinar a la persona que cumpla con estas competencias
- Favorecer el desarrollo de competencias tendientes a mejorar aún más el desempeño superior (sobre la media) en el puesto de trabajo
- Permitir que el recurso humano de la organización se transforme en una aptitud central y de cuyo desarrollo se obtendrá una ventaja competitiva para la empresa.

2.11.4. DIMENSIONES DE GESTIÓN

Al referirse a competencia laboral es conveniente distinguir entre una de cuatro dimensiones que pueden diferenciarse y significar aplicaciones prácticas del concepto de competencia:

2.11.4.1. Identificación de Competencias

Es el proceso que se sigue para establecer, a partir de una actividad de trabajo, las competencias que se ponen en juego con el fin de desempeñar tal actividad, en forma excelente. La cobertura de la identificación puede ir desde el puesto de trabajo hasta un concepto más amplio de área ocupacional o ámbito de trabajo.

2.11.4.2. Normalización de Competencias

Una vez identificadas las competencias, su descripción puede ser de mucha utilidad para aclarar las transacciones entre empleadores, trabajadores, y entidades educativas. Usualmente, cuando se organizan sistemas normalizados, se desarrolla un procedimiento de estandarización ligado a una futura institucional, de forma tal que la competencia identificada y descrita con un procedimiento común, se convierta en una norma, un referente válido para las instituciones educativas, los trabajadores y los empleadores. Este procedimiento creado y formalizado institucionalmente, normaliza las competencias y las convierte en un estándar al nivel en que se haya acordado (empresa, sector, país)

2.11.4.3. Formación Basada en Competencias

Una vez dispuesta la descripción de la competencia y su normalización; la elaboración de estuviera de formación para el trabajo será mucho más eficiente si considera la orientación hacia la norma. Esto significa que la formación orientada a generar competencia con referentes claros en normas existentes tendrá mucha más eficiencia e impacto que aquella desvinculada de las necesidades del sector empresarial.

2.11.4.4. Certificación de Competencias

La emisión de un certificado implica la realización previa de un proceso de evaluación de competencias. El certificado, es un sistema normalizado, no es un diploma que acredita estudios realizados; es una constancia de una competencia demostrada; se basa obviamente en el estándar definido.

2.11.5. FASES DE IMPLEMENTACIÓN

2.11.5.1. Detección de Problemas y Necesidades

La detección de necesidades y problemas es posible en 2 instancias:

- **Medición de Clima Organizacional.-** Es un diagnóstico de clima organizacional, con esta medición se sabe la percepción de los trabajadores de: su jefatura, de la estructura organizacional, del medio

ambiente que lo rodea y de los conflictos internos. Existen muchos instrumentos para la medición del clima organizacional que evalúan las variables antes nombradas, sin embargo hemos desarrollado un nuevo instrumento que lo encontrará en el

- **Detección y Análisis de Problemas.-** Es revisar las funciones realizadas por todos los cargos y los conflictos, problemas y necesidades que se suscitan en ellos, entorpeciendo de forma directa o indirecta el desarrollo o accionar del trabajador, a la vez de identificar el origen, y las consecuencias de estos "eventos". Si la empresa es pequeña, entonces bastaría con aplicar un cuestionario que permita al trabajador informar sobre los problemas que ocurren al realizar el trabajo, en este cuestionario se debe pedir también al trabajador que identifique las causas del problema y sus consecuencias directas (sobre su labor) y las consecuencias indirectas (sobre la labor de otros) y cotejarlos con los que su jefatura directa observa. Si la empresa es mediana o grande (sobre 30 trabajadores) entonces será necesario realizar esta búsqueda de los problemas por equipos de trabajo o áreas de trabajo, según sea la estructura de la organización, pero el análisis debe realizarlo la jefatura directa junto a los jefes de otras áreas.

2.11.5.2. Análisis de Tareas

Una vez detectadas las necesidades especificadas en problemas, se debe dar paso al análisis de cada una de las tareas que se realizan en la actualidad, tal análisis consiste en especificar las tareas que están entregando problemas, realizar un seguimiento de las causas del mal desarrollo de las tareas y proporcionar una solución, es decir determinar el modo de modificarlas entregando propuestas para su resolución, o también puede darse la posibilidad que tras haber analizado cada tarea exista una o algunas que deban ser eliminadas. El análisis de las tareas que se desarrollan en las organizaciones junto con el análisis de problemas, determinará por qué los objetivos no se consiguen de forma exitosa; por

medio de éstos dos estudios obtendrán respuestas a la ineficiencia la que se refleja en dos aspectos:

- **Ineficiencia por Agentes Externos.-** problemas del medio ambiente en el cual desarrollan sus tareas, carencias de material, carencia de suministros, equipos, etc.
- **Ineficiencia Personal.-** es en este punto donde recae la responsabilidad en el trabajador de no estar desarrollando bien su labor, por ello se puede analizar el porqué de tal desempeño, de acuerdo a esto se pueden arrojar resultados como qué competencias pueden ser necesarias introducir en los trabajadores para elevar su desempeño, o en caso de que el trabajador presente las conductas apropiadas, desarrollar nuevas competencias.

2.11.5.3. Definición de Unidades y Perfiles

Aquí se define de unidades de competencia y perfiles profesionales para todos los cargos de la empresa. Las unidades de competencias son funciones integradas por una serie de elementos de competencia y criterios de desempeño asociados, los cuales forman una actividad que puede ser aprendida, evaluada y certificada. Las competencias deben reflejar la conducta que se necesita para el futuro éxito de la empresa u organización, en la tabla 8 se presenta un proceso muy utilizado por los consultores, para ayudar a las empresas a traducir los retos estratégicos en formas de conducta requerida y competencias de las personas.

2.11.5.4. Proceso Definición de Competencias

TABLA 8

Proceso Definición de Competencias

Sec	Tarea	Detalle
1	Panel de Expertos	Lo conforma un grupo de directores especialistas en recursos humanos y con sólidos conocimientos en el tema de las competencias, empleados de alto nivel y especialistas en diversos puestos de trabajo con una clara visión de futuro. El proceso consiste en: Análisis FODA de la organización. Acordar la misión de los puestos de trabajo. **CONTINÚA →**

2	Identificación de Competencia	La identificación de las competencias y conductas requeridas para esos puestos de trabajo, se da a través del inventario de competencias que en términos generales es una lista de comprobación con conductas y competencias o también se puede realizar a través de un sistema experto que permita a las personas que están en los puestos de trabajo, dar respuesta a las preguntas planteadas por el sistema, estas preguntas están registradas en una base de datos comprendiendo competencias identificadas anteriormente. **CONTINÚA →**
3	Muestra Representativa	De acuerdo a los resultados obtenidos por el panel de expertos se selecciona a un número de empleados que presentan las competencias y conductas identificadas.
4	Entrevistas de Incidentes	Realizar entrevistas de incidente críticos, debido a que las descripciones de conducta no es una expresión tangible que proporciona una base sólida al momento de gestionar el recurso humano es necesario tomar ejemplos de conductas de la vida real, esto se lleva a cabo mediante una serie de entrevistas de incidentes críticos a la muestra representativa seleccionada por el panel de expertos. Estas entrevistas proporcionan abundantes datos e información para la identificación de las competencias, y unas descripciones muy concretas de las conductas críticas de trabajo en situaciones específicas. Por medio de esto se puede hacer una estimación acerca de cuándo, cómo, dónde, adquirieron sus competencias clave.
5	Análisis de Datos	Todas las transcripciones obtenidas en las entrevistas anteriores se analizan por contenidos, a fin de obtener una clara comprensión y descripción de las competencias que serán utilizadas como base para las aplicaciones de recursos humanos.
6	Validación	El modelo se puede validar mediante una segunda serie de BEI con un nuevo grupo de personas, y comprobar si las competencias identificadas se relacionan efectivamente con la actuación superior, tal como la identificó el panel de expertos. **CONTINÚA →**
7	Planificación de Aplicaciones	Los modelos de competencias forman un buen núcleo en torno al cual se puede crear un conjunto de políticas y técnicas de recursos humanos, lógicamente interrelacionadas, este tema será revisado con más detalles más adelante en este estudio.
8	Informe Final	Corresponde al informe escrito donde se detalla las competencias requeridas para que la organización alcance sus metas y propósitos establecidos en la visión - misión.

2.11.6. APLICACIÓN DE COMPETENCIAS

2.11.6.1. Descripción y Análisis de Puestos

Es una explicación escrita de los deberes, las condiciones de trabajo y otros aspectos relevantes de un puesto específico. El análisis de puesto consiste en la obtención, evaluación, y organización de información sobre los puestos de una organización.

La información sobre análisis de puestos es importante porque comunica a los especialistas en personal, que deberes y responsabilidades se asocian a cada puesto; esta información se utiliza posteriormente cuando se lleva a cabo actividades como el diseño de puestos, el reclutamiento y la selección de personal. Los puestos constituyen el nexo entre las organizaciones y sus competencias.

2.11.6.2. Plan de Carrera

Es el proceso mediante el cual los empleados individuales identifican y ponen en marcha las acciones para alcanzar sus metas de carrera. Comprender que las aspiraciones e intereses de cada individuo cambiarán y diferirán de acuerdo con ciertos patrones y las etapas de su carrera, puede ayudar a los individuos y directores a comprender los tipos de oportunidades y asistencia más efectivos para apoyar la planificación de carrera del individuo.

2.11.6.3. Planeación y Distribución

Es una técnica para determinar en forma sistemática la provisión y demanda de empleados que tendrá una organización. Al determinar el número y tipo de empleados que serán necesarios, el departamento de personal puede planear sus labores de reclutamiento, selección, capacitación y otras más. Esto permite al departamento de personal suministrar a la organización al personal adecuado en el momento adecuado.

2.11.6.4. Evaluación de Desempeño

Constituye el proceso por el cual se estima el rendimiento global del empleado. La mayor parte de los empleados procura obtener retroalimentación sobre la manera en que cumple sus actividades y las personas que tiene a su cargo la dirección de las labores de otros empleados deben evaluar el desempeño individual para decidir las acciones que deben tomar. Un sistema de evaluación de desempeño debe identificar los elementos relacionados con el desempeño, medirlos y proporcionar retroalimentación a los empleados y al departamento de personal. Si las normas para la evaluación del desempeño no se basan en elementos relacionados con el puesto, pueden traducirse en resultados imprecisos o subjetivos. El objetivo de la evaluación de desempeño está directamente relacionado con el puesto, entendido como: que el sistema califica únicamente elementos de importancia vital para obtener éxito en el puesto, si la evaluación no se relaciona con el puesto, carece de validez.

2.11.6.5. Capacitación

Después de que los empleados han sido seleccionados y orientados, es posible que aún deban adquirir las habilidades, el conocimiento y las actitudes necesarias para desempeñarse adecuadamente. Si la organización desea promover a esos empleados a puestos con mayores responsabilidades en el futuro, las actividades de desarrollo resultan imprescindibles para muchos efectos prácticos.

La capacitación sobre la base del modelo de gestión de las competencias conlleva tres áreas para su control eficaz, una parte teoría, una parte de laboratorio donde se entregan herramientas teóricas para desarrollar las competencias necesarias y una evaluación en terreno en el puesto de trabajo, donde se medirá la efectividad de lo aprendido en forma teórica y en forma empírica a escala en el laboratorio. Lo anterior sería una reevaluación de los conocimientos y técnicas aprendidos

CAPÍTULO 3: DEFINICIÓN DE GUÍA METODOLÓGICA

3.1. INTRODUCCIÓN

En el capítulo anterior se ha descrito en qué consiste un modelo de proceso de desarrollo de software, cuál es su propósito, definición, las ventajas y el impacto de su utilización dentro de una organización. Además, se ha realizado una detallada investigación y análisis de las metodologías más relevantes en la actualidad, presentando un resumen de cada una y analizando sus características.

En este capítulo se verá en detalle la guía metodología propuesta, que da una solución al problema del desarrollo de software distribuido de alta calidad, la misma que es concebida con la idea de ser aplicada en pequeños, medianos y grandes proyectos de desarrollo, sin necesidad de invertir en costosas herramientas o de capacitar en forma extensiva a las personas involucradas.

También se anexa a este capítulo los templates que serán utilizados por el equipo de desarrollo para estandarizar los entregables que genera cada fase, esto permitirá que la transición hacia la metodología sea lo menos resistente posible, logrando disminuir la curva de aprendizaje inherente a un cambio de proceso.

3.2. CARACTERÍSTICAS DE LA GUÍA METODOLÓGICA

La guía metodológica propuesta se basa en 10 elementos [Cockburn, 1998], que son:

- **Roles,** las descripciones a los trabajos que se solicitan en las búsquedas laborales cuando se necesitan tomar gente
- **Destrezas,** las destrezas que presupone poseen los recursos que llevan a cabo el proyecto
- **Entregables,** lo que se quiere que cada persona o equipo entregue a otra persona o equipo: casos de uso, especificaciones de diseño, documentación de framework, diagramas de secuencia.
- **Actividades,** las actividades e hitos de los diferentes equipos, como

se integran para producir los entregables finales.

- **Valores**, los valores del equipo y de la guía metodológica.
- **Equipos**, la forma en que se agrupan a las personas.
- **Asignación de Tareas,** son tareas asignadas a los múltiples roles.
- **Técnicas,** las técnicas que las personas utilizan en su trabajo
- **Herramientas**, las herramientas que usan a diario, ya sea dentro de una técnica o para producir un entregable de acuerdo al estándar.
- **Estándares**, lo que está o no permitido en el diseño y los entregables, esto incluye notación, convenciones del proyecto y cuestiones de calidad.

3.3. ROLES Y DESTREZAS

Los roles y destrezas definen el conjunto cohesivo de actividades a realizadas y artefactos mantenidos que son llevados a cabo por personas o por grupos de personas. No sólo se refieren a personas internas al desarrollo del software, sino que también involucran a los usuarios u otras personas que se vean afectadas por el proyecto, cualquiera de los denominados *stakeholders*.

Los roles a mencionar pueden ser o no exclusivos, es decir asociados a una única persona (patrocinante) o grupo de personas (desarrollador) y en algunos casos abarca más de un rol (Arquitecto/Escritor Técnico).

Las asignaciones son llevadas a cabo por el Líder, el cual en base a las aptitudes de los recursos que dispone repartirá las tareas a ser realizadas en cada iteración. Se considera el rol imprescindible conjuntamente con el desarrollador.

3.3.1. Patrocinante

- ***Actividades,*** tiene a su cargo el soporte gerencial del proyecto; es el encargado de proveer, comunicar y mantener actualizada la Visión del proyecto; proveer el presupuesto y la viabilidad económica del desarrollo; es responsable del proyecto del lado del cliente.

- ***Importancia del rol****,* es esencial para el éxito del mismo, ya que un software que no tiene aceptación dentro de la organización que lo financia jamás llegará a ser construido en tiempo, forma y con consentimiento de los usuarios, no siendo utilizado eventualmente si se concreta el proyecto.

3.3.2. Líder de Proyecto

- ***Actividades:*** tiene a su cargo la planificación del proyecto, a lo largo de todo el ciclo de vida, incluida la planificación en detalle de cada iteración; asigna recursos y delega responsabilidades; fomenta la cohesión del grupo y lleva a cabo actividades destinadas a eliminar fricciones; organiza las reuniones a ser realizadas; monitorea el progreso del proyecto y establece estrategias para mitigar los riesgos que se puedan presentar.
- ***Importancia del rol:*** representa la cara visible del equipo de desarrollo, es el nexo existente entre la gerencia y el equipo de desarrollo.

3.3.3. Experto en el Dominio

- ***Actividades:*** tiene a su cargo brindar su conocimiento del negocio contribuyendo al modelado del sistema que llevan a cabo los Analistas durante la disciplina de Requerimientos-Análisis; participará junto con los Testers en la definición del contenido de las pruebas funcionales a ser realizadas; será el responsable de la aprobación de las pruebas de aceptación por cada release entregado.
- ***Importancia del rol:*** permite al Equipo de Desarrollo aprender sobre el negocio para el cual está siendo construida la aplicación; son encargados de resolver cualquier cuestión relacionada con la funcionalidad de la aplicación junto con los Analistas.
- ***Destrezas:*** conocer en detalle el negocio y dar respuesta a cualquier duda en el proyecto, es un miembro de la empresa Cliente.

3.3.4. Coordinador

- ***Actividades:*** tiene a su cargo la supervisión del proceso, y cualquier actividad orientada al mejoramiento del mismo. Durante las primeras etapas de la guía metodológica supervisará la implementación del proceso.
- ***Importancia del rol:*** permite reforzar la adherencia al proceso en aquellos momentos en que se el tiempo apremia y se suele caer en el modelo *Codificar y Probar*.
- ***Destrezas:*** conocer en detalle el negocio para prestar respuesta a cualquier duda que pueda surgir del mismo.

3.3.5. Analista

- ***Actividades:*** tiene a su cargo el relevamiento, mediante el cual se obtienen los requerimientos de la aplicación a ser construidos en cada iteración; realiza la especificación de los requerimientos; prepara el documento de Visión.
- ***Importancia del rol:*** el aprendizaje del dominio de la aplicación y de los requerimientos que deberá tener la misma son claves para el éxito del proyecto y la aceptación del mismo por parte del usuario.
- ***Destrezas:*** tener amplio conocimiento de técnicas de relevamiento, así como aptitudes sociales que le permitan vencer el “Síndrome del Usuario y el Desarrollador” [Leffingwell, 2001].

3.3.6. Arquitecto

- ***Actividades:*** tiene a su cargo la definición de la arquitectura de la aplicación y la arquitectura de distribución que guiará el desarrollo, y de la continua refinación de la misma en cada iteración; deberá construir cualquier prototipo necesario para probar aspectos riesgosos desde el punto de vista técnico en el proyecto; definirá los lineamientos generales del diseño y la implementación.
- ***Importancia del rol:*** la arquitectura es imprescindible en los

proyectos de software distribuido actuales en donde cada vez existe mayor complejidad; el arquitecto puede ser considerado como el Experto en la parte técnica del desarrollo y debe mantener a todo el equipo en conocimiento de los lineamientos fundamentales de la construcción.

- ***Destrezas:*** tener una buena formación técnica, contar con experiencia en las herramientas y técnicas utilizadas; aptitudes comunicacionales son deseadas para que la arquitectura sea comunicada a todos los miembros del equipo; también deberá ser perseverante en conseguir los hitos técnicos planteados mediante entregables para asegurar el progreso de la construcción.

3.3.7. Programador o Desarrollador

- ***Actividades:*** tiene a su cargo la codificación de los componentes a desarrollar en la iteración; debe crear y ejecutar los tests unitarios realizados sobre el código desarrollado; es responsable de las clases que ha desarrollado debiendo documentarlas, actualizarlas ante cambios y mantenerlas bajo el control de configuración de las mismas.
- ***Importancia del rol:*** persona que tiene los materiales y lleva a cabo la implementación de los casos de uso en el lenguaje de programación orientado a objetos elegido.
- ***Destrezas:*** tener amplio conocimiento de las herramientas de desarrollo, del lenguaje de programación, de los aspectos técnicos involucrados.

3.3.8. Verificador o Tester

- ***Actividades:*** tiene a su cargo la generación de pruebas funcionales locales y distribuidas a partir de los requerimientos extraídos por los Analistas. Crea, ejecuta, analiza y mantiene el conjunto de pruebas automatizadas y manuales que son utilizados.

- ***Importancia del rol:*** radica en la necesidad de construir un software distribuido de calidad que cumpla con los requerimientos del usuario; mediante la utilización de un proceso y el armado de un grupo cohesivo de desarrollo, se tienen prácticas para garantizar la calidad en el producto desde el punto de vista técnico
- ***Destrezas:*** tener amplio conocimiento de técnicas de testing, deberá conocer a fondo la aplicación a testear. Asimismo, deberá tener conocimientos de programación para trabajar con las pruebas automatizadas.

3.4. FASES

La guía metodológica propuesta se compone de un conjunto de fases que son realizadas a lo largo del tiempo, las mismas conforman un período de tiempo encuadrado entre hitos significativos para el proyecto. El grado de creatividad requerido en el proceso disminuye con el tiempo, tornándose en un ciclo más predecible.'

3.4.1. Concepción

Consiste en la definición de las características que tendrá la aplicación, el alcance de la misma, la distribución de la información, la identificación de los stakeholders, la customización del proceso a ser usado y llevar a cabo la planificación general del proyecto. Esta fase provee el fundamento en que todo el posterior trabajo es basado. Es crítico en esta fase llevar a cabo los primeros relevamientos que se realizan con el Cliente de forma de adquirir conocimiento del dominio y analizar si los costos del proyecto estarán justificados o bien conviene comprar algún software empaquetado o cancelar la ejecución. Esta fase finaliza con un hito mayor denominado **Objetivos del Ciclo de Vida** en el que se evalúa lo realizado contra las expectativas del Cliente y del Equipo de Desarrollo.

3.4.2. Elaboración

Se refiere a la exploración de los requerimientos más críticos (funcionales y no funcionales) que involucra el proyecto, así como las decisiones técnicas más importantes que quedarán plasmadas en los documentos: Arquitectura de Aplicación y Arquitectura de Distribución. El objetivo principal consiste en asegurar la factibilidad técnica respecto a la realización del proyecto. Es a partir de la próxima fase cuando se comienza con la construcción a gran escala del software, en donde se comprometen la totalidad de los recursos necesarios para que el desarrollo se complete en las iteraciones planificadas. Asimismo, se podrán incorporar recursos en forma limitada tratando de no obstaculizar el normal transcurso del proyecto debido a la capacitación que estos últimos requerirán. Esta fase finaliza con un hito mayor denominado **Arquitectura del Ciclo de Vida** en el que se evalúa lo realizado contra las expectativas del Cliente y del Equipo de Desarrollo.

3.4.3. Construcción

Consiste en terminar de especificar los casos de uso correspondientes a la iteración, se diseñan bajo la arquitectura candidata presentada, y se codifican todos los componentes definidos por los casos de uso, ejecutándose el testing correspondiente y la integración. Se realiza en forma repetitiva por cada iteración.

3.4.4. Transición

Consiste en pasar un cierto conjunto de componentes al entorno productivo, se llevará a cabo las actividades de despliegue necesarias con el release. Esta fase finaliza con un hito mayor denominado **Release del Producto** en el que se evalúa lo realizado contra las expectativas del Cliente y del Equipo de Desarrollo.

Cabe mencionar que dentro de las iteraciones de construcción se realizan actividades relacionadas con Requerimientos, Diseño, Testing, etc.,

se trata de un proceso iterativo e incremental que realiza tareas en paralelo y la aplicación evoluciona hasta cumplir los casos de uso definidos.

3.4.5. Administrador del Conocimiento

- ***Actividades:*** tiene a su cargo la captura, refinamiento, empaquetamiento, y transferencia del conocimiento, ya sea tácito o explícito. Se recomienda que este rol sea llevado a cabo por una persona del equipo de desarrollo con dedicación part-time, esto puede depender del tamaño de la organización y de otros factores.
- ***Importancia del rol:*** es un pilar y su importancia consiste en la capacidad del equipo de desarrollo de aprender de la experiencia que éste y que otros equipos dentro de la organización generan a diario durante el transcurso de los proyectos. Mediante esta disciplina se logra la reutilización de dicho conocimiento.
- ***Destrezas:*** tener aptitudes en comunicación para poder capturar el conocimiento de aquellas personas que lo generan.

3.4.6. Matriz de Roles

TABLA 9

Resumen Roles por Tipo Proyecto

RESUMEN ROLES POR TIPO DE PROYECTO				
Roles	**Descripción**	**Básico**	**Mediano**	**Grande**
Patrocinante	Soporte gerencial del proyecto	Ninguno	Opcional	Obligator
Líder Proyecto	Planificar proyecto, recursos y responsabilidades	Obligatorio	Obligatorio	Obligator
Experto Dominio	Proveer conocimiento del negocio	Ninguno	Obligatorio	Obligator
Coordinador	Supervisar el proyecto y mejoramiento	Ninguno	Opcional	Obligator
Analista	Realizar relevamiento y requerimientos	Opcional	Obligatorio	Obligator
Arquitecto	Definir lineamientos de diseño e implementación	Opcional	Obligatorio	Obligator
Programador	Codificar y desarrollar componentes	Obligatorio	Obligatorio	Obligator
Verificador	Probar funcionalidad y manuales	Ninguno	Opcional	Obligator
Admin.Conocimiento	Capturar y transferir conocimiento	Ninguno	Opcional	Obligator

3.5. DISCIPLINAS DE FASES

La guía metodológica propuesta de desarrollo para tener la visibilidad sobre el progreso y el grado de avance, propone dividir el proceso en disciplinas que conforman agrupaciones de actividades en las cuales se produce un conjunto particular de artefactos. Estas disciplinas se analizan y realizan dentro de las fases con diversos grados de paralelismo entre ellas.

3.5.1. Factibilidad

Se refiere al primer contacto entre el equipo de desarrollo y el cliente. Esta disciplina permite al equipo de desarrollo adquirir todo el conocimiento posible acerca del dominio, las necesidades de los usuarios en las diferentes localidades, los objetivos y expectativas que debe cumplir el software que está siendo construido. Mediante las actividades incluidas en esta disciplina, se estará en la capacidad de responder las siguientes preguntas:

- Es factible distribuir la información?
- Es conveniente construir la aplicación?
- Es conveniente para el cliente el desarrollar un sistema con todas las complejidades inherentes para satisfacer las necesidades del usuario?
- Cuáles son las posibles soluciones que se pueden plantear?

A medida que se ejecuta esta disciplina, se va realizando el Modelado del Dominio tendiente a entender las operaciones que forman parte del dominio del problema, el dominio manejado por el cliente. En forma conjunta, los analistas comenzarán a entender el problema planteado comenzando a bosquejar cuáles son las características que tendrá el sistema y el arquitecto comenzará a realizar modelos y diagramas planteando una arquitectura y distribución preliminar que pueda servir de solución al problema en cuestión.

- ***Exploración del Problema.-*** es esencial el aprendizaje del dominio del cliente, de cómo el sistema puede solucionar los problemas que se presentan, de cómo esta solución puede ser realizada con las herramientas disponibles, de las restricciones impuestas sobre el sistema.
- ***Exploración de la Solución.-*** el entendimiento del negocio permite al equipo de desarrollo conocer explorar soluciones adecuadas a las necesidades de los usuarios; este conocimiento debe ser plasmado en un documento de Modelo del Dominio que debe ser construido conjuntamente con los usuarios y validados por estos.
- ***Exploración de la Distribución.-*** el volumen de la información, la criticidad del tiempo de respuesta y los medios de comunicación entre las distintas localidades permite plantear y definir la necesidad de distribuir la información, el documento Factibilidad de Distribución facilita la toma de decisiones al respecto.

La solución técnica propuesta deberá ajustarse a los estándares que posea el cliente, y el equipo de desarrollo deberá asegurarse de la factibilidad de su implementación, para este punto se puede tener un documento de Descripción de Arquitectura Distribuida que detalle las decisiones técnicas tomadas a lo largo de la *Factibilidad*.

La *Factibilidad* finaliza con la planificación del contenido funcional que será desarrollado en cada una de las iteraciones en que se construirá la aplicación. En dicho plan, se determina cuáles son las iteraciones con versión, las cuales servirán como hitos del progreso del proyecto tanto para el cliente como para el equipo de desarrollo.

Dicha planificación reviste un carácter preliminar ya que los requerimientos pueden presentar cambios que alteren el contenido de las funcionalidades a ser implementadas en cada iteración. Está planificación contendrá los elementos más importantes y será plasmada en un artefacto denominado Plan de Proyecto, el cual será mostrado en detalle en la sección de entregables.

3.5.2. Requerimientos

En esta disciplina se realizan actividades tendientes a capturar las necesidades de los stakeholders, transformar las mismas en características de alto nivel y especificarlas posteriormente en casos de uso. Se debe especificar el *Espectro Funcional* el cual servirá para que los desarrolladores diseñen e implementen los casos de uso, el *Espectro No Funcional* y el *Nivel de Integración de la Información* entre las localidades, que posteriormente utilizará el arquitecto para construir la arquitectura de la aplicación, arquitectura de la distribución y el prototipo.

Se debe considerar el patrón *Orientación al Cliente*, el cual consiste en maximizar la comunicación directa con los usuarios para asegurar el cumplimiento de los requerimientos, en otras palabras el proceso será más efectivo si se tiene al menos un Cliente como parte integral del equipo de desarrollo. Su principal función será la de instruir y transferir su conocimiento al resto del equipo a medida se transcurre el proyecto, muchas veces esto resulta impráctico en cuyo caso se deberá asegurar el involucramiento de los usuarios y una frecuente validación de los artefactos construidos en cada fase.

Para llevar a cabo este relevamiento de información, se recomienda utilizar técnicas que no demanden gran cantidad de recursos o una significativa infraestructura, la técnica más utilizadas es la entrevista, las cuales tendrán un carácter más informal ya que se podrán dar en cualquier momento que el analista deba aclarar cuestiones respecto a los requerimientos, detallar aspectos de un caso de uso, etc. En caso que se necesite hacer participar a un número importante de stakeholders en el relevamiento, se recomiendan sesiones de trabajo dirigidas por el Líder de Proyecto a las cuales asisten los analistas documentando todo las decisiones tomadas.

Posteriormente en la Captura de Requerimientos se realiza la primera aproximación en una forma estructurada que consiste en los casos de uso y diagramas de estructura, estos se han transformado en un estándar en la industria de la informática y han trascendido más allá de la comunidad de

objetos de donde surgieron para ser utilizados ampliamente en proyectos. La principal ventaja es que son escritos en el lenguaje del cliente y no demandan tener conocimientos técnicos para ser entendidos. Son esenciales en los procesos de desarrollo en que se intenta que exista comunicación constante y fluida entre los miembros del equipo y el Cliente.

Para escribir los casos de uso y diagramas se recomienda utilizar estándares predefinidos, esto permitirá que los analistas del equipo de desarrollo sincronicen el nivel de ambigüedad de la narración, los atributos a ser escritos y las convenciones que se utilizarán.

3.5.3. Análisis y Diseño

Esta disciplina consiste en plasmar el problema planteado, que se halla en forma de casos de uso, diagramas y requerimientos contenidos implícitamente por Analistas o Clientes, en el dominio de la solución. Asimismo, también se realiza la definición de la arquitectura de la aplicación y distribución siendo validada mediante algún prototipo.

Durante las primeras iteraciones cabe destacar la necesidad de diseñar la infraestructura sobre la cual será construido el software, para esto el arquitecto tomará del conjunto de casos de uso relevados hasta el momento aquellos que considere más importantes desde un punto de vista técnico y también analizará los requerimientos suplementarios que deberán ser satisfechos por la arquitectura propuesta. Además deberá analizar los medios de comunicación existentes entre las localidades, su disponibilidad y costo. Con esto se construirá los modelos de arquitectura de aplicación y distribución documentados, que especificará los elementos a partir del cual será construido el sistema, las interacciones entre dichos elementos, los patrones que se usaron para su composición, y las restricciones sobre dichos patrones. Finalmente, para probar la factibilidad de la arquitectura el Arquitecto construirá un prototipo ejecutable utilizado para verificar la conformidad a los requerimientos de la aplicación y distribución.

Este proceso es plasmado mediante la generación de Diagramas de Secuencia que incluye todas las localidades, para aquellas interacciones

significativas desde un punto de vista técnico, las cuales podrán ser anexadas al Documento de Arquitectura de Aplicación y Distribución. El nivel de ceremonia estará acorde a la criticidad del sistema que se desarrolla. En casos pequeños y donde se desee tener un mínimo overhead metodológico el diseño podrá ser realizado en papel o en un pizarrón, en casos de mayor ceremonia se utilizarán herramientas de modelado visual que ayuden en este proceso.

Una función esencial es la de asignación de funcionalidad en que el Líder de Proyecto o el Arquitecto se encargarán de nombrar a un responsable por cada unidad de funcionalidad especificada. Puede tratarse de un caso de uso, de un grupo de clases, de un paquete. Lo importante es que exista un responsable para cada artefacto que mantenga actualizada la trazabilidad requerida, pudiendo realizar cualquier tipo de modificación sobre el mismo y manteniendo un adecuado control de cambios.

3.5.4. Implementación y Pruebas

Esta disciplina lleva a cabo la producción del software, el equipo de desarrollo se encarga de implementar la funcionalidad especificada en los casos de uso mediante el lenguaje de programación orientado a objetos elegido.

Para reducir la brecha comunicacional entre las personas que forman el equipo, se recomienda lenguajes de programación que minimicen la brecha entre la realidad y la realización de la misma. Se fomenta el uso de frameworks o tecnologías bajo el paradigma de orientación a objetos, como *Java, Visual NET o Smalltalk*, los cuales permiten el máximo de eficiencia por línea de código para aplicaciones.

En paralelo a la construcción de los componentes se realiza la revisión mediante el testing, siendo está una de las maneras en que se llevan a cabo los mecanismos de SQC (Software Quality Control), la guíoa metodológica define diversas pruebas que podrán ser realizadas en las iteraciones. Las mismas incluyen:

3.5.4.1. Pruebas Unitarias:

- ***Definición:*** son pruebas realizadas a nivel de las clases y métodos construidos para la aplicación, estas pruebas son implementadas por los programadores en el lenguaje de programación utilizando clases de prueba encargadas de verificar el comportamiento correcto de los métodos en una clase. Para ello es conveniente contar con algún framework de testing unitario que le facilite al equipo de desarrollo la creación, mantenimiento y ejecución de una suite de pruebas automatizadas.
- ***Alcance:*** las mismas son utilizadas para controlar la calidad del código construido y son ideales para testing de integración y de regresión.

3.5.4.2. Pruebas Funcionales:

- ***Definición:*** verifican el flujo de interacción de la funcionalidad definida en los casos de uso, esta funcionalidad descrita en forma narrativa en los casos de uso y es plasmada mediante la interacción de objetos enviándose mensajes entre sí en los diagramas de secuencia, debe ser especificada en pruebas automatizadas que serán creadas por el Usuario en conjunción con la ayuda técnica del Tester y serán ejecutadas por el Tester durante cada iteración. Cabe remarcar que debe ser el Usuario el que define el contenido de las mismas ya que de esta forma proveen el feedback que permite al equipo de desarrollo continuar las iteraciones asegurándose de la calidad del producto construido.
- ***Alcance:*** las mismas son utilizadas durante todo el desarrollo y su alcance está definido por el conjunto completo de requerimientos funcionales definidos en los casos de uso.

3.5.4.3. Pruebas de Aceptación:

- ***Definición:*** el objetivo de las Pruebas de Aceptación es la verificación que el software construido en las iteraciones cumple con las funcionalidades solicitadas por el usuario y está listo para ser utilizado en el ambiente del Cliente. Estas pruebas revisten gran relevancia porque implican que el Cliente da conformidad respecto al sistema desarrollado y lo acepta.
- ***Alcance:*** las mismas son realizadas durante las *Iteraciones con Release* en las que se lleva a cabo la transición de la aplicación al ambiente de producción en el Cliente.

3.5.4.4. Pruebas de Distribución:

- ***Definición:*** el objetivo es verificar la comunicación que existe entre las distintas localidades, el volumen de transferencia de información, la frecuencia de errores, el nivel de perdida y recuperación de datos, para lo cual se utiliza el Diagrama de Arquitectura de Distribución. Estas pruebas son indispensables antes de liberar una localidad
- ***Alcance:*** Se realiza en dos fases, la primera a nivel de comunicaciones y la segunda por cada versión de la aplicación.

3.5.5. Puesta en Marcha

La disciplina de *Puesta en Marcha* permite que el sistema construido sea transferido al ambiente de producción para ser utilizado por el usuario en cada localidad. Esto no significa que el software debe estar implementado en su totalidad, cubriendo el 100% de los aspectos funcionales y no funcionales sino que en una iteración determinada se desee liberar una versión con un porcentaje de los casos de uso implementados, pero si debe estar implantado en un 100% el nivel de comunicaciones entre la localidades definido en la Arquitectura de Distribución. Dado el esquema de desarrollo iterativo e incremental, durante

las iteraciones ya mencionadas se va agregando funcionalidad y se hace “crecer” al sistema hasta que cumpla con los requerimientos que se especificaron al principio, sumados a los cambios surgidos a lo largo del desarrollo.

Durante esta disciplina, se deben realizar actividades tendientes a hacer una entrega completa con una funcionalidad previamente determinada la cual será desplegada en producción. Las actividades incluidas dentro de esta disciplina son:

- Instalación y Configuración de Comunicaciones entre localidades
- Generar Manuales del Usuario, de Operación, etc.
- Realizar un empaquetamiento de componentes, junto con scripts de instalación que el Cliente utilice para instalar la aplicación en su entorno.
- Ejecutar el conjunto completo de pruebas funcionales creadas durante el desarrollo, hasta obtener la aceptación del Cliente.
- En caso de ser necesario, definir las actividades de migración al nuevo sistema.
- Capacitar a los Usuarios que utilizarán el nuevo sistema.
- Generar la Nota de Entrega con la totalidad de los artefactos que se le entregan al Cliente (con su correspondiente versionado) al final de la disciplina.

El Líder de Proyecto deberá encargarse de lograr el consenso de todos los involucrados en una fecha de entrega del sistema, para esto coordinará las fechas de forma que el Cliente tenga la oportunidad de verificar el correcto funcionamiento del sistema, mediante las pruebas de aceptación que se han generado hasta el momento. Es conveniente, que se planifique una reunión formal entre las dos partes (Cliente y Equipo de Desarrollo) para que todos estén en conocimiento de los resultados obtenidos, se puedan realizar comentarios respecto a las cuestiones que se

deberían mejorar en el proceso para las siguientes iteraciones. En otras palabras, esta reunión serviría como una revisión Post-Mortem en la que se evaluarían todas las iteraciones de desarrollo que precedieron a la actual, y cuáles fueron las actividades que funcionaron exitosamente y cuáles fueron las dificultades encontradas en el camino.

3.6. DISCIPLINAS DE SOPORTE

Las Disciplinas de Soporte ocurren a lo largo de todo el ciclo de vida del proyecto y dan soporte a las actividades relacionadas con la construcción del software y distribución de la información, estas sirven propósitos organizacionales no directamente relacionados con la producción de sistemas.

3.6.1. Soporte al Proyecto

Esta disciplina engloba todas las actividades relacionadas con la gestión del emprendimiento, esto refiere a las tareas que el Líder de Proyecto lleva a cabo para garantizar un ambiente saludable de trabajo, una medición del grado de avance del proyecto, un continuo monitoreo y mitigación de riesgos. Estas actividades permiten que el proyecto llegue a su éxito, entregando la aplicación en tiempo, forma y dentro de los costos presupuestados.

El Líder es la persona que habilita al equipo de desarrollo para que trabaje cómodamente, sin obstáculos, resolviendo las necesidades que esta tenga, tiene el carácter y las responsabilidades de un facilitador. Deberá reportar y conocer el status del grupo y cómo el desarrollo se va dando en relación a la planificación establecida.

Se recomienda establecer una planificación inicial del proyecto que permita tener objetivos claros respecto a los grandes hitos para obtener feedback del cliente respecto al avance realizado. Se armará un plan de proyecto con las fases y las iteraciones durante las que se irá construyendo el sistema, este plan es armado consultando al equipo de desarrollo; la única tarea del Líder de Proyecto es armarlo en algún formato presentable al

cliente, como por ejemplo un diagrama Gantt o una planilla Excel, dependiendo del nivel de formalidad requerido. El equipo de desarrollo será el que estimará la duración de las tareas a este nivel, entendiendo que la estimación será de una precisión muy limitada, ya que no se cuenta con mucha información de la funcionalidad a desarrollar. En cada iteración el plan será actualizado para reflejar la realidad.

3.6.2. Soporte a la Configuración

Esta disciplina contiene las actividades relacionadas con la administración del repositorio que almacena los ítems de configuración generados en un proyecto. Se recomienda colocar todos los artefactos generados o consumidos en un proyecto bajo una herramienta que permita administrar y controlar las versiones, esto permitirá que en cualquier momento se pueda recuperar una versión determinada de un ítem o ítems de configuración para recrear la cronología del proyecto o volver cambios atrás, o llevar a cabos registros de trazabilidad y/o auditorias entre otras cosas.

El uso de una herramienta de SCM (Software Configuration Management) le da la posibilidad al equipo de desarrollo de tener prácticas eficientes al momento de realizar las actividades en un proyecto. Entre estas prácticas cotidianas presentadas en cualquier desarrollo se puede mencionar:

- Los desarrolladores pueden trabajar en un mismo proyecto, compartiendo todos los ítems de configuración.
- Los desarrolladores pueden compartir el esfuerzo en el desarrollo de cualquier artefacto, sea una clase, un documento, etc.
- Los desarrolladores pueden acceder la versión actual, para verificar si el código generado seguirá funcionando cuando sea integrado.
- Los desarrolladores pueden volver a una versión anterior del sistema para hacer pruebas contra esta.

- Los desarrolladores pueden trabajar en distintas ramas en paralelo sobre los artefactos del sistema, permitiendo tener una rama estable, otra para experimentar, y así sucesivamente.

Las herramientas deberán ser usadas consistentemente y todo el equipo debe tener el conocimiento para trabajar con ellas, esto podrá requerir capacitación a las personas que lo utilizarán de ser necesario. Está práctica recomienda que no existan “dueños” de los ítems de configuración subidos al repositorio sino que cada persona podrá interactuar con cualquier ítem, pudiendo modificarlo y subir una nueva versión, en cierto sentido el equipo de desarrollo es responsable por todo el sistema por lo tanto todos los miembros de este son “dueños” de todos los ítems.

Dentro de esta disciplina se incluye aquellas actividades relacionadas con la Administración de Cambio, es decir, un proceso formal que permita controlar la forma en que los cambios son implementados durante el desarrollo. Los cambios son tomados como una posibilidad de tener un mejor entendimiento de la aplicación a construir por lo cual serán priorizados por el Cliente de acuerdo a su valor del negocio e implementados según corresponda en una iteración determinada. Dependiendo del costo asociado, el Cliente puede decidir descartarlos. Es de suma utilidad disponer de una herramienta que automatice el seguimiento de los cambios a lo largo de su ciclo de vida, notificando al equipo de desarrollo de los cambios de estado de cada uno a lo largo del proyecto.

3.6.3. Soporte al Proceso

Esta disciplina permite que la guía metodológica propuesta sea adaptada a las necesidades de las personas y del proyecto en cuestión. Cuando la organización ya posee una metodología estandarizada en sus proyectos y conocida por todas las personas, la misma deberá ser adaptada de acuerdo al contexto de cada proyecto en que se utiliza, es decir, habrá distintos proyectos con características dispares en cuanto a tamaño del equipo, grado de criticidad del sistema, stakeholders involucrados. Todos

estos factores darán como resultado una necesidad metodológica única, con ciertos roles, ciertos artefactos, ciertas actividades, etc.

En esta disciplina es clave el Coordinador quien empieza con un análisis del proyecto, toma la planificación original, la funcionalidad, el equipo de trabajo, el perfil del cliente, y con todo esto define como se empleará la guía metodológica en el proyecto. El Coordinador debe tener un amplio conocimiento del espectro metodológico existente para poder extraer aquellos elementos de la guía metodológica propuesta.

Cabe remarcar que la tarea de adaptación del proceso es llevada a cabo por el Coordinador en cooperación con el equipo de desarrollo, esto resulta muy relevante ya que será este último el usuario final de la guía metodológica por lo cual el Coordinador consultará con los miembros más especializados para entender los mecanismos de desarrollo involucrados.

Una vez que la guía metodológica haya sido configurada para un proyecto, se harán reuniones esporádicas (una buena práctica consiste en coordinarlas con el fin de cada iteración) en las que se evaluará la adecuación a las características del proyecto. Se analizarán los resultados positivos y negativos, pudiendo el Coordinador hacer cambios sobre la marcha para mejorar la forma de trabajo.

Finalmente, cuando el proyecto esté llegando a su fin, el Coordinador, con la ayuda del Administrador de Conocimiento, almacenará la información acerca de cómo el proceso fue usado, con el propósito de mantener un historial de adaptaciones realizadas por tipo de proyecto.

3.6.4. Soporte a las Personas

Esta disciplina agrupa todas aquellas actividades relacionadas con los aspectos humanos del desarrollo, nuevamente la clave de esta disciplina será el Coordinador, con una importante cooperación del Líder de Proyecto.

Una de las primeras actividades dentro de esta disciplina es la de nivelar a los miembros del equipo de desarrollo, es importante que cada uno tenga todos los conocimientos necesarios para encarar las actividades por las que es responsable. El Coordinador deberá llevar a cabo talleres de

capacitación hasta asegurarse de que las personas han aprendido. Esta capacitación refiere tanto a aspectos técnicos como a los relacionados con las demás disciplinas, incluyendo la descripción del proceso.

Otra de las actividades que será realizada en forma conjunta con el Líder de Proyecto es mantener la salud del equipo de desarrollo, en otras palabras velar porque la motivación del equipo sea alta, garantizar relaciones interpersonales positivas, minimizar las fricciones debidas a distintos tipos de personalidad, entender a cada individuo para poder sacar lo mejor de sí mismo. Existe un solapamiento entre esta disciplina y la de Administración de Proyectos en este punto. Se destaca que el mismo no es tan importante desde un punto de vista conceptual pero la diferencia está marcada en que las actividades de esta disciplina tienen como objetivo el bienestar de las personas en el proyecto mientras que el management se relaciona con la consecución exitosa del proyecto.

La práctica recomendada para esta disciplina tiene que ver con el patrón de Máxima Comunicación, es decir, si tiene canales de comunicación suficientemente ricos las dificultades podrán ser resueltas, en la mayoría de los casos, utilizando el sentido común y la buena predisposición de la gente.

3.6.5. Soporte al Conocimiento

Un proceso de desarrollo de software debe proveer un mecanismo que permita que el conocimiento generado en un proyecto sea mantenido dentro de la organización. Es vital poseer mecanismos para reutilizar el conocimiento que se va generando a medida que la organización va ejecutando distintos tipos de proyectos.

Para ello se define tres niveles de refinamiento hasta llegar al conocimiento; que son: datos, *información* y *conocimiento*. Comenzando con el nivel inferior, los *datos* consisten en valores discretos y objetivos acerca de eventos, pero sin ningún contenido acerca de su importancia o relevancia; a partir de éstos se puede generar información. La *información* son datos organizados de forma de servir de utilidad para las personas que

mediante el contexto la utilizan para encarar tareas o tomar decisiones. Finalmente, el conocimiento requiere entender la información y tiene que ver con la relación entre ítems de información, su clasificación y su meta-dato (información de la información). La *experiencia* es conocimiento aplicado. La misma consiste en una estructura paralela al equipo de desarrollo, la cual tiene personas de dedicación exclusiva dedicadas a capturar, empaquetar, almacenar y distribuir a lo largo de la organización.

Existe un rol especifico que es *Administrador del Conocimiento*, su tarea consiste en documentar todo aquel conocimiento novedoso que haya sigo generado durante el proyecto para que pueda ser reutilizado por otras personas, este no debe generar extrema burocracia dentro de la organización del proyecto.

La organización tiene la necesidad de invertir en la gente y en abocar por la motivación de los equipos de desarrollo. Si una organización no valora a las personas que la conforman, nunca podrá aprender, pues el nivel de rotación es alto y el nivel de aprendizaje es bajo.

3.6.6. Matriz de Fases y Disciplinas

TABLA 10
Matriz Utilización Fases y Disciplinas

		FASES Y % DE UTILIZACION			
		CONCEPCION	**ELABORACION**	**CONSTRUCION**	**TRANSICION**
DISCIPLINAS DE LAS FASES	Factibilidad	70	20	5	5
	Requerimientos	20	70	5	5
	Análisis	5	70	20	5
	Diseño	5	70	20	5
	Implementación	0	5	80	15
	Testing	0	5	80	15
	Despliegue	0	5	15	80
DISCIPLINAS DE SOPORTE	Sop.Proyecto	25	25	25	25
	Sop.Configuración	25	25	25	25
	Sop.Proceso	25	25	25	25
	Sop.Personas	25	25	25	25
	Sop.Conocimiento	25	25	25	25

3.7. ARTEFACTOS

La guía metodológica propuesta define artefactos mínimos necesarios para propósitos de comunicación, diseño, distribución, gestión, sin los cuales un proyecto no llegaría al éxito.

3.7.1. Visión

- ***Definición:*** es realizado durante la fase de Concepción y sirve como contrato entre el Cliente y el Equipo de Desarrollo respecto a lo que se va a construir. En la Visión deberán identificarse los stakeholders del proyecto, la totalidad de características del sistema a ser construido, la distribución de la información y los requerimientos suplementarios que se detectaron en forma temprana. En forma optativa podrá incluirse un resumen de los casos de uso críticos del aplicativo.
- ***Responsable:*** la Visión es construida por un Analista Funcional en conjunto con el Líder de Proyecto, la misma debe ser mantenida para reflejar los cambios al alcance durante el proyecto.

3.7.2. Plan

- ***Definición:*** es un documento mantenido por el Líder de Proyecto con toda la información de gestión, el mismo suele incluir un Gantt con el cronograma y las tareas del proyecto. Dependiendo del nivel de formalidad se generará un WBS y/o algún otro tipo de plan a incluirse dentro de este (plan de testing, de comunicaciones, de distribución, de administración de requerimientos, de administración de cambios, etc.). El Plan de Proyecto es creado en forma temprana durante la Concepción y actualizado durante las fases posteriores.
- ***Responsable:*** el Líder de Proyecto es el responsable de este documento y deberá mantenerlo actualizado hasta la terminación del proyecto.

3.7.3. Riesgos

- ***Definición:*** se utiliza para capturar los riesgos más relevantes que se presentan en el proyecto, y para poder monitorearlos, asignarles prioridad, impacto y probabilidad de ocurrencia. Sirve para planificar las iteraciones y para tener en vista aquellos riesgos que, por su criticidad, deberán ser mitigados lo más tempranamente posible y mucho mas la implantación de las localidades.
- ***Responsable:*** el Líder de Proyecto es el responsable y el principal consumidor de este documento y deberá mantenerlo actualizado hasta la terminación del proyecto.

3.7.4. Casos de Uso

- ***Definición:*** los Casos de Uso se utilizarán para especificar los requerimientos funcionales de la aplicación, servirán para guiar los demás artefactos y para la construcción de los componentes de la aplicación. Se recomienda la generación del Modelo de Casos de uso para tener una visualización gráfica del universo funcional de la aplicación y poder comunicarla tanto internamente como al Cliente para su validación. Los Casos de Uso y el Modelo serán creados en las primeras fases del proyecto, aunque podrán también ser especificados en las iteraciones de construcción.
- ***Responsable:*** los Analistas y programadores son responsables de la creación y el mantenimiento de estos artefactos, los cuales serán consumidos por el resto del equipo de desarrollo.

3.7.5. Especificaciones y Requerimientos

- ***Definición:*** se utilizará para especificar los requerimientos funcionales de carácter más técnico de la aplicación, también en este se incluirán los requerimientos no funcionales, las reglas de negocio, las restricciones que se aplicarán a la solución y la necesidad de

distribuir la información. En este documento se pondrán todos los requerimientos que queden fuera de los casos de uso. La idea es tener algún tipo de constancia documental de todos aquellos requerimientos de carácter técnico que debe contemplar la solución.

- ***Responsable:*** los Analistas son responsables de la creación y el mantenimiento de este artefacto, el cual será consumidos por el resto del equipo de desarrollo.

3.7.6. Arquitectura de la Aplicación

- ***Definición:*** especifica los aspectos técnicos de la solución propuesta por el equipo de desarrollo. Sirve como medio de comunicación entre el Arquitecto y el equipo en relación a cómo deberán ser implementados los casos de uso de la aplicación.
- ***Responsable:*** el Arquitecto es el principal responsable por la realización de este artefacto, así como la creación de un prototipo ejecutable que valide el cumplimiento de los requerimientos funcionales, no funcionales, y de distribución.

3.7.7. Arquitectura de la Distribución

- ***Definición:*** especifica los aspectos técnicos de la distribución y ubicación de la información en las localidades. Sirve como medio de comunicación entre el Arquitecto y el equipo en relación a cómo deberán ser implementados los casos de uso de la aplicación.
- ***Responsable:*** el Arquitecto es el principal responsable por la realización de este artefacto, así como la definición de los medios de comunicación ente las localidades.

3.7.8. Casos de Prueba

- ***Definición:*** los Casos de Prueba contienen la especificación de cómo será validado el sistema. Estos son realizados basándose en los

casos de uso y planteando todos los escenarios posibles que éstos pueden contener. Dado que puede ser bastante burocrático realizar la totalidad de Casos de Prueba existentes en un sistema, se recomienda generar los más importantes y después anotar las variantes en un Caso de Prueba Estándar.

- ***Responsable:*** el Tester es el principal responsable por la realización de este artefacto. El Tester será el encargado del diseño y la ejecución de los Casos de Prueba.

3.7.9. Scripts de Instalación

- ***Definición:*** los Scripts de Despliegue son necesarios para la instalación del producto en un entorno determinado. El mismo automatiza las tareas de empaquetamiento, versionado, compilación, etc. Un ejemplo muy utilizado baja la plataforma J2SE es la herramienta ANT, que permite generar scripts de despliegue con bastantes funcionalidades.
- ***Responsable:*** *algún* Programador será responsable de la generación y mantenimiento de estos scripts.

3.7.10. Bitácora de Incidentes

- ***Definición:*** la Planilla de Incidentes será utilizada para registrar y tener seguimiento de aquellos bugs, mejoras, tareas que el Tester o alguna persona de aseguramiento de la calidad necesite reportar. Son los denominados sistemas de Issue Tracking que pueden ser implementados manualmente o mediante alguna herramienta.
- ***Responsable:*** la Planilla de Incidentes será utilizada por todos los miembros del equipo. será muy utilizada por los Testers quienes cargarán todos los bugs encontrados durante la ejecución de los Casos de Prueba los cuales serán leídos y corregidos por los Programadores.

3.7.11. Repositorio del Versiones

- ***Definición:*** el Repositorio es la herramienta fundamental para cubrir la disciplina de Administración de la Configuración. En el repositorio se almacenan todas las versiones de todos los archivos y directorios del proyecto. Se recomienda que todo artefacto generado durante un proyecto sea puesto bajo control de versiones y no sólo el código fuente de la aplicación, esto permitirá en cualquier momento poder comparar versiones, volver a recuperar versiones anteriores, y generar ramas de desarrollo paralelo.
- ***Responsable:*** algún Programador o alguna persona de Infraestructura de la organización será responsable de la creación, mantenimiento y eliminación (en casos que se desee dar de baja el proyecto) del repositorio.

3.7.12. Acta Entrega/Recepción

- ***Definición:*** la Nota de Entrega sirve para especificar aquello que se le entrega al Cliente en una versión determinada de la aplicación. En la misma se constatará el número de versión, los cambios de la versión anterior, los bugs conocidos y las mejoras efectuadas.
- ***Responsable:*** cualquier miembro del equipo de desarrollo que vaya a entregar algún artefacto al Cliente deberá crear la correspondiente Nota de Entrega.

3.8. PATRONES DE DESARROLLO

Un patrón describe un problema que ocurre una y otra vez en el desarrollo, y después describe la base de la solución a dicho problema de una forma que se puede utilizar dicha solución muchas veces sin realizarla dos veces de la misma manera, además provee un contexto para entender cuándo la solución debe ser aplicada. La guía metodológica propuesta recomienda los siguientes patrones en el desarrollo de software:

1. Comunicación Máxima
2. Comunicación Interna al Equipo
3. Comunicación Externa
4. Participación del Cliente
5. Estimaciones Realistas
6. Enfoque en la Arquitectura Distribuida
7. Integración Continua
8. Recuro Humano
9. Calidad en el Proceso

3.8.1. Comunicación Máxima

Este patrón implica una ***Comunicación Máxima*** entre el equipo de desarrollo y todas las personas involucradas o afectadas en la ejecución del proyecto. La comunicación se refiere a las diversas formas de transmisión de datos/información/conocimiento/experiencia entre individuos. Existen dos tipos de comunicaciones que se toma en cuenta para mejorar el desarrollo:

- ***Comunicación Interna***, entre todos los integrantes del equipo de desarrollo; desde el líder del proyecto hasta el programador
- ***Comunicación Externa,*** entre los integrantes del equipo de desarrollo y el cliente o usuario del sistema o los *stakeholders*.

Este patrón implica la priorización continua de personas sobre el proceso y proveer de mecanismos que fomenten las iteraciones ayudando a mejorar la motivación y la productividad del equipo de desarrollo.

3.8.2. Comunicación Interna

- ***Mínima Dispersión Física*****,** según este patrón el equipo de desarrollo debe estar colocado físicamente en una única oficina que tenga una distribución centralizada al momento de trabajo con lugares privados para cuestiones extra laborales.
- ***Interrupciones Mínimas,*** se refiere a la necesidad de liberar al

equipo de desarrollo de interrupciones referidas a otros ámbitos. Se recomienda que la oficina del equipo de desarrollo este separada mediante paredes del resto de la organización y que las personas no se interrumpan continuamente entre sí.

- ***Radiadores de Información***, es decir tener las paredes de la oficina totalmente libres dispuestas para que se cuelguen en ellas pizarrones, hojas grandes, diagramas, notas post-it, etc. Estos actúan como radiadores de información ya que permiten que cualquier miembro del equipo pueda tener acceso a esa información constantemente sin necesidad de andar indagando a las demás personas. Además lo radiadores sirven para mantener información crítica a la vista de todos la cual puede ser discutida, analizada, modificada y comunicada continuamente.

Una implementación alternativa es tener un sitio web que disponga continuamente de información del proyecto, adicionalmente, puede servir como base de conocimientos del proyecto y de la organización.

De no aplicar estos patrones, se tendrán flujos de información que no refieren al desarrollo y que empeorarán la calidad del ambiente diseñado para maximizar la comunicación.

3.8.3. Comunicación Externa

Tan importante como la comunicación interna resulta la comunicación con los stakeholders del proyecto. Se divide en dos grupos:

- ***Stakeholders claves,*** son aquellos que poseen conocimiento del dominio del sistema a construir y que aceptarán el sistema en cuestión si el mismo está acorde a sus necesidades.
- ***Stakeholders normales***, con los que el equipo interactúa ocasionalmente para algún propósito del proyecto, juegan un rol importante para la salud del proyecto pero que no impacta directamente sobre la funcionalidad de la aplicación construida.

Otro patrón recomendado es ***On-Site Customer***, establece que durante todo el proyecto habrá un cliente real sentado en la misma oficina con el equipo de desarrollo, dispuesto a responder preguntas, resolver disputas, y priorizar las tareas llevadas a cabo. Cabe destacar que en este caso un “cliente real” refiere a una persona que utilizará el sistema una vez puesto en producción y que conozca la aplicación en su extensión para resolver distintas cuestiones que puedan darse.

Ocurre que en la mayoría de los casos los clientes tienen que hacer sus propios trabajos y no disponen de tiempo para estar continuamente formando parte del equipo de desarrollo, dadas las realidades impuestas en la mayor parte de los proyectos, se recomienda designar el rol *Experto en el Dominio,* el cual debe estar dispuesto a contestar cualquier pregunta del equipo de desarrollo ya sea mediante algún canal de comunicación más frío de acuerdo como: teléfono, videoconferencia, mail, etc., en un tiempo razonable que no debería superar algunas horas. En caso de que el problema requiera de una comunicación más personal, el cliente deberá disponer de tiempo para visitar al equipo de desarrollo y discutir el tema personalmente o bien para recibir a algún miembro clave en sus propias oficinas.

Respecto a los stakeholders normales, se recomienda que estos sean identificados en forma temprana y sus necesidades sean relevadas para verificar que el sistema conforme a estos adecuadamente. Si bien no será necesario tener un canal abierto tan flexible la comunicación con estos debe ser fluida y honesta ya que en algún punto el sistema tendrá influencia sobre ellos o viceversa.

3.8.4. Participación del Cliente

Se recomienda la necesidad de contar con una participación incondicional del Cliente durante el proyecto de desarrollo. Dadas las características de las aplicaciones a las que el proceso apunta, los únicos capaces de definir qué debe contener el producto son los usuarios dado que

serán sus necesidades las que se verán afectadas. No hay documento ni modelo ni herramienta que pueda sustituir el feedback provisto en forma directa mediante contacto cara a cara entre el analista y el cliente o usuario del sistema.

No hay forma de alcanzar el éxito en un proyecto si no se cuenta con el compromiso y el soporte del Patrocinante. El apoyo del Líder garantiza que los obstáculos que se presenten (políticos, económicos, sociales) puedan ser removidos mediante su activa participación.

Para lograr esta activa participación del usuario se debe contar con usuarios expertos en el dominio que puedan prestar todo el tiempo necesario para las necesidades del equipo de proyecto. Entre estas se incluye la posibilidad de que los analistas puedan delinear requerimientos con el grado de detalle necesario para que los programadores puedan comenzar con el diseño. También los programadores deben poder consultar a dichos usuarios si se les plantean dudas respecto al dominio del problema o existen omisiones o inconsistencias en los requerimientos.

Los usuarios deberán tener amplio conocimiento del dominio para poder resolver cualquier duda del equipo respecto a las funcionalidades a desarrollar. También deberá priorizar continuamente el valor del negocio de lo que se construye en cada iteración para guiar funcionalmente al equipo.

3.8.5. Estimaciones Reales

La guía metodológica propuesta permite realizar estimaciones cortas descentralizadas en el día a día, para obtener más precisión en la estimación de la entrega de artefactos en cada iteración, sin embargo, es importante realizar estimaciones realistas de la duración de los proyectos sobre todo al principio del mismo.

La técnica utilizada es la estimación por ***Puntos de Caso de Uso,*** la misma mide a los sistemas desde una perspectiva funcional y es independiente de la tecnología. Sin tener consideración del lenguaje, método de desarrollo, o plataforma de hardware utilizada, el número de

puntos de casos de uso de un sistema permanecerá constante. La única variable es la cantidad de esfuerzo necesario para desarrollar un conjunto dado de puntos de caso de uso; por lo tanto el ***Análisis por Puntos de Caso de uso*** puede ser utilizado para determinar si una herramienta, un ambiente, un lenguaje es más productivo comparado con otros dentro de una organización o entre organizaciones. Este es uno de los puntos críticos y uno de los valores más importantes de dicho análisis.

Cada persona estimará el fruto de sus tareas por la técnica que considere necesaria, una consecuencia de esto es que las personas no sienten que se les ha impuesto un cronograma al que deben atenerse pudiendo ser reprendidas en caso de retrasos. El responsable de un producto es la persona más adecuada para medir su ritmo y estimar los tiempos necesarios. Al principio esto podrá resultar difícil para aquellas personas acostumbradas a tener fechas y tareas impuestas desde arriba, pero ahí es donde entra en juego el Coordinador para capacitar a los miembros del equipo y perfeccionarlos en las estimaciones futuras.

La estimación para una iteración dada será realizada durante la iteración previa, de forma de tener la planificación aprobada por el cliente previo a embarcar en las actividades correspondientes al equipo en el desarrollo. Esta estimación será efectuada por todo el equipo junto al cliente y en la misma será validada la funcionalidad a ser próximamente construida.

3.8.6. Arquitectura Distribuida

La guía metodológica propuesta establece una temprana definición de la arquitectura distribuida del sistema. La distribución se explicó detalladamente en capítulos anteriores, y consiste en la suma de aspectos que están relacionados con el diseño de un sistema.

Se debe contar con el Arquitecto, para reflejar y validar las decisiones que se toman en relación al diseño y construcción de la arquitectura, haciendo un paralelismo con el paradigma de objetos, que considera al sistema como un paquete a ser embebido en un ambiente específico. La arquitectura estaría dada por:

- La organización de la misma, definición de la cantidad de localidades, disponibilidad de medios de transmisión y aplicaciones requeridas.
- Volumen de paquetes o clases y las relaciones entre estos.
- Las interfaces que provee al mundo exterior
- Los paquetes y estaciones que requiere para su funcionamiento
- Los servicios que provee, realizados a partir de métodos en clases, que deben dar valor a algún actor que interactúa con el sistema
- Los patrones, estándares, frameworks utilizados para su construcción
- Costo de implementación y tiempo sugerido.

Cabe destacar, que la arquitectura distribuida mantiene un cierto nivel de abstracción dentro de la disciplina de diseño. No contempla los detalles de más bajo nivel, como la implementación interna de las clases y métodos, ni analiza en detalle los protocolos a ser utilizados por los sistemas. Simplemente propone un conjunto de vistas que enfatizan distintos aspectos del software, tienen que ver con:

- Organización lógica
- Funcionalidad
- Concurrencia
- Distribución del software en la plataforma

Gracias a la arquitectura distribuida se logra la ***integridad conceptual*** necesaria para dirigir el proceso y la activa colaboración del Arquitecto.

3.8.7. Integración Continua

La *Integración Continua* consiste en poseer un ambiente automatizado mediante el cual se pueda realizar el check out de la herramienta de Administración de la Configuración que se esté utilizando y

posteriormente tener un entorno que tome el código fuente, genere los instaladores con todos los pasos intermedios que esto involucre, corra los suites de tests automatizados, y emita un reporte con el resultado del proceso. El énfasis está puesto en la automatización del proceso de manera que resulte algo lo más transparente posible para el equipo de desarrollo. Existen en la actualidad herramientas muy potentes y en algunos casos sin costo por ser de código abierto, las cuales permiten que esta práctica pueda ser llevada a la realidad con un mínimo esfuerzo. Un ejemplo de herramienta Open Source de estas características es el **Cruise Control** que funciona en conjunto con **Ant**, **CVS**, y **JUnit** para llevar a cabo los instaladores continuos en una máquina.

Para tener un instalador exitoso es recomendable que el mismo tenga ciertas características que den seguridad respecto a la consistencia e integridad del mismo. En un entorno ideal un instalador solo podrá ser llamado exitoso si se dan todos los siguientes pasos correctamente sin intervención humana:

- Los últimos fuentes han sido bajados del repositorio de versionado
- Cada archivo fue compilado de cero
- Los ejecutables han sido linkeados y empaquetados para su despliegue
- El sistema es ejecutado y las pruebas automatizadas son disparadas.

En casos que no se cuente con una herramienta de instaladores automatizada, el programador hará un check out del módulo correspondiente en la máquina que ha sido designada como repositorio de los archivos maestros. En la misma ejecutará el script que se encargará de compilar y empaquetar las clases para su despliegue. Suponiendo que el resultado del script es exitoso se podrá efectuar el test funcional en caso que se desee liberar la versión. Una vez que el programador deja la máquina de integración tendrá una sensación de logro y una inmediata

recompensa de un trabajo bien realizado.

Sin embargo, recordar que hasta este instante se está verificando la sintaxis y semántica del código, pero no se sabe nada respecto a si cumple las necesidades del usuario. Para esto se tienen las pruebas funcionales automatizadas o manuales que deberá crear o especificar el testeador junto con el cliente.

3.8.8. Recurso Humano

La guía metodológica considera a las personas como partes esenciales del desarrollo y las encuadra en un marco humano necesario para el entendimiento del impacto que las mismas tienen en la construcción de los diversos artefactos. Dicho marco humano se centra en tres áreas de estudio: organización, área de desarrollo e individuo. Dentro de la organización se agrupa aquellas cuestiones externas al área de desarrollo que no son controladas por el mismo.

- El lugar de trabajo.
- Recursos de tecnología disponibles.
- La cultura de la empresa.

Por área de desarrollo se entiende la forma en que se relacionan y organizan los individuos para realizar el desarrollo.

Dentro del análisis del individuo se encuentran los factores que afectan a un individuo al momento de desarrollar software. Las personas necesitan tener un propósito para lo que hacen y esto toma mayor importancia en relación a su trabajo, se debe fomentar que el individuo esté cómodo en su trabajo y que exista una buena relación con el equipo de desarrollo. Para esto es de suma importancia que el Líder de Proyecto monitoree las dinámicas que entran en juego en el grupo así como el comportamiento de los individuos.

La cultura de una empresa se define como el conjunto de ideologías, valores, metas, que perciben los individuos que conforman la misma.

Representa la cultura de las personas que interactúan bajo una misma organización. Una de las primeras cuestiones con las que debe inmiscuirse rápidamente un individuo al comenzar a formar parte de una empresa es su cultura. Gracias a ésta el empleado se irá adaptando, cambiando sus valores, formulando nuevos juicios y terminará poniéndose las mismas metas que el resto de las personas o abandonará el intento por no sentirse cómodo con la misma. La cultura de la empresa tiene tanta relevancia como el trabajo en sí para el que fue contratado el empleado.

3.8.9. Calidad

Una de las razones del fracaso de los proyectos de desarrollo en las organizaciones, es la falta de una metodología clara y transparente. Mientras los hitos se vayan cumpliendo siendo las entregas aceptadas por los clientes el equipo se siente confidente de que su proceso los llevará al éxito y que además lo están siguiendo correctamente. De darse así, todos comentarán la necesidad de mantener un proceso disciplinado mostrando las experiencias pasadas como demostración de los resultados y de los productos entregados.

Podrá llegar el momento en que el proyecto comience a retrasarse en las entregas, a disminuir la calidad o a remover funcionalidad de las mismas. Será en estos momentos en que nacerá un sentimiento de desesperación que comenzará en el management descendiendo en la jerarquía hasta el equipo de desarrollo. Se pedirán entregas imposibles, las cuales llevarán a los programadores a codificar en forma masiva, dejando de lado completamente cualquier viso de proceso, erradicando completamente la calidad del software construido.

Por estas cuestiones es importante que el proceso se tenga en alta estima y que la gente esté convencida de su utilización. En caso contrario, el proceso no cumple su función. Para mantener la salubridad del mismo se incluye en la próxima sección una serie de disciplinas que permiten tener siempre un proceso de calidad que se adecue a las necesidades de la gente que lo utiliza.

3.9. APORTE METODOLÓGICO

A continuación se detalla los aportes que esta tesis hace la universo de las metodologías, en base a un análisis exhaustivo de los procesos existentes y de la identificación de los problemas y soluciones.

3.9.1. Nomenclatura Estandarizada

Una de las primeras facilidades que una persona se encuentra revisar esta tesis, es el uso de nombres estandarizados para todos sus elementos. Muchos de los nombres utilizados han sido tomados del RUP [RUP, 2002] que sirve como base de conocimiento para procesos de desarrollo. Gracias a esta estandarización, se facilita el entendimiento global de los conceptos involucrados lo cual habilita una comunicación más fluida entre las personas involucradas.

3.9.2. Administración del Proyecto

Esta guía metodológica propone distintos patrones enmarcados en una disciplina que pueden ser utilizados para customizar el proceso en distintas etapas. Los patrones sirven para que el proyecto ser implementado exitosamente y para adaptar el mismo a las necesidades particulares del proyecto. Esto logra captar el feedback de la gente que es usuaria del proceso y dinámicamente modificarlo para que cumpla mejor sus objetivos.

3.9.3. Administración del Equipo de Trabajo

Esta guía metodológica propone distintos patrones enmarcados en una disciplina que pueden ser utilizados para manejar mejor el factor humano dentro del desarrollo. Los patrones ayudan a mejorar el ambiente de trabajo de los individuos, mejorar la motivación de los grupos de trabajo y mitigar las fricciones que puedan existir. El resultado será una adopción rápida y eficaz de la guía metodológica, tolerante a la diversidad de personalidades existentes.

3.9.4. Administración del Conocimiento

Esta guía metodológica explícitamente establece una disciplina de Administración del Conocimiento que contribuye al aprendizaje organizacional. La misma establece una serie de prácticas reales que ayudan a que el conocimiento novedoso generado durante un proyecto sea capturado para una posterior recuperación. El objetivo es que el conocimiento se propague por los grupos de trabajo y que se vaya aprendiendo de la experiencia para el mejoramiento de las personas y la agilización del proceso.

3.9.5. Procedimiento de Implementación

Existe una frase en el terreno de la Ingeniería de Software que dice: “Lo difícil no es definir una metodología, sino implementarla en la organización”. Dada esta realidad, se plantea una serie de consideraciones, surgidas a partir de las experiencias de otras áreas de conocimiento, que contribuyen a su implementación en un grupo humano.

3.9.6. Prácticas y Experiencia

Las prácticas propuestas han sido probadas durante suficiente tiempo, las mismas han sido estudiadas en libros mencionados en la bibliografía y se reconoce su valor agregado para mejorar el proceso de construcción de software, además han sido demostradas por experiencia propia en diversos proyectos.

CAPÍTULO 4: DESARROLLO DEL SISTEMA

4.1. CONCEPCION

4.1.1. Visión

Este documento permite recoger, analizar y definir las necesidades de alto nivel y las características del módulo de Evaluación de Recursos Humanos basado en Competencias ya desarrollado por la empresa Zeuz Sistemas, dedicada al desarrollo y mantenimiento de software. Este se centra en la gestión de evaluaciones realizadas al personal de las empresas y su valoración.

Anexo 1 Detalle Artefacto Visión

4.1.2. Plan de Proyecto

Este documento brinda una visión general de la lógica del enfoque del desarrollo propuesto. El proyecto se basa en una serie de artefactos tomados de diferentes metodologías, en las que se toma los elementos necesarios para las fases de Concepción, Elaboración, Construcción y Transición. El enfoque de desarrollo está conformado por procesos de las metodologías RUP, MSF y XP, seleccionando roles indispensables y artefactos necesarios para su posterior construcción.

Anexo 2 Detalle Artefacto Plan de Proyecto

4.1.3. Lista de Riegos

Este documento permite identificar y estimar las características que implican algún tipo de riesgo, su probabilidad y su impacto en el proceso de desarrollo del módulo de Evaluación del Recursos Humanos basado en Competencias ya desarrollado por la empresa Zion Sistemas y analizar sus posibles mitigaciones.

Anexo 3 Detalle Artefacto Riesgo

4.2. ELABORACIÓN

4.2.1. Especificación de Requerimientos

Este documento permite analizar y definir los requisitos del proyecto como producto software. Dichos requerimientos son los obtenidos del cliente y forman la estructura fundamental del proyecto que son los puntos clave a desarrollarse en el proyecto.

Anexo 4 Detalle Artefacto Especificación Requerimientos

4.2.2. Modelo Casos de Uso

Este documento permite definir y analizar los Casos de Uso según actores y las tareas que realizan en la aplicación. Los casos de uso se basan en los requerimientos Funcionales del sistema y están definidos en el documento “Especificación de Requerimientos”.

Anexo 5 Detalle Artefacto Modelo Casos de Uso

4.2.3. Descripción Arquitectura Distribución

La Arquitectura de Distribución hace referencia a las ubicaciones del aplicativo en medios físicos alojados en una red. Este proyecto está desarrollado para cuatro (4) y tres (3) niveles, los cuales son:

- **Nivel de Cliente.-** En este se encuentran las estaciones de trabajo o computadoras desde las cuales se accede a la aplicación web.
- **Nivel de Aplicación.-** En este nivel se encuentra el servidor web donde se aloja la aplicación.
- **Nivel de Datos.-** Nivel en el que se encuentra el servidor con gestor de base de datos y los Datos de la aplicación.
- **Nivel de Nube.-** Nivel Abstracto mediante el cual se conectan ciertos clientes.

Los usuarios clientes pueden conectarse a los datos mediante tres medios: Internet, Intranet y Aplicación de Escritorio. La figura 6 muestra la relación existente entre las capas:

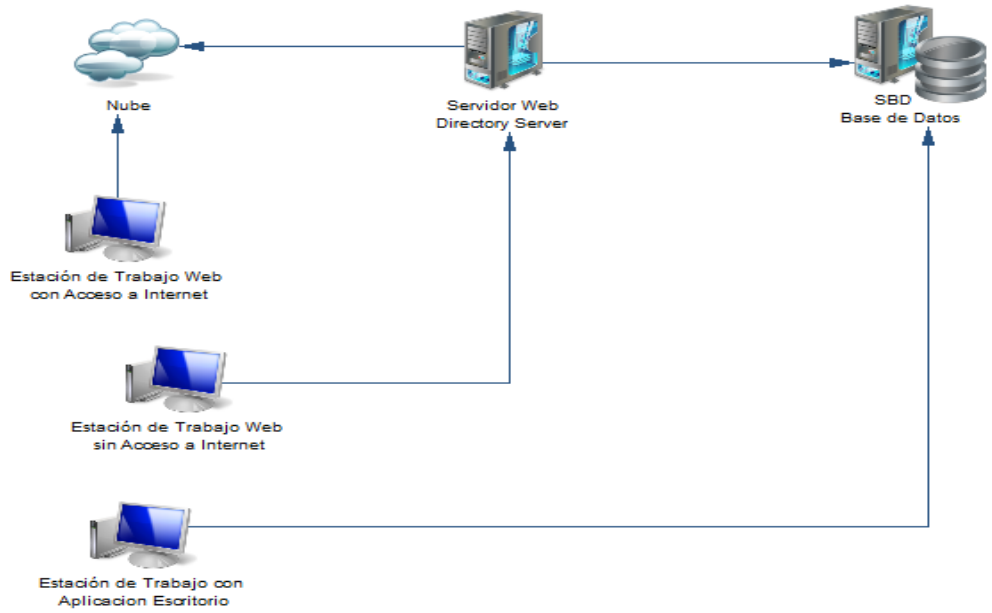


FIGURA 6 Diagrama Arquitectura Distribución

4.2.4. Descripción Arquitectura Aplicación

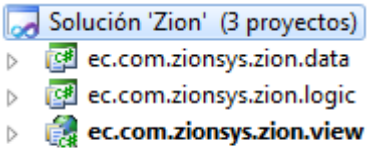
La Arquitectura de Aplicación hace referencia a la distribución lógica de las capas de programación de la aplicación. Este proyecto se encuentra desarrollado bajo una Arquitectura de tres (3), de las cuales algunas poseen subcapas. Las capas son las siguientes:

- **Capa de Datos.-** Capa en las que se encuentran las referencias y métodos para acceder a la Base de Datos. Esta capa también posee funciones de lectura y escritura para la Base de Datos.
- **Capa Lógica.-**
- **Sub capa Negocio.-** Aquí se ubican las clases que contienen los atributos característicos de una entidad.
- **Sub capa Controlador.-** En esta sub capa se encuentran los métodos y funciones representantes de las entidades en la sub capa de negocio.
- **Sub capa Acceso.-** Sub Capa en la que se encuentran las representaciones de entidades correspondientes al usuario y al menú.

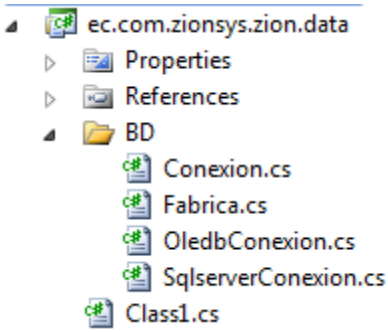
- **Capa de Vista.-** Capa donde se localizan las interfaces gráficas con las cuales interactúan los usuarios. Estas interfaces gráficas son las encargadas de llamar a los controladores en la capa lógica.

A continuación se demuestra mediante imágenes la ubicación de las capas:

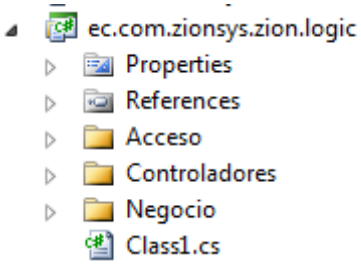
4.2.4.1. Distribución Capas



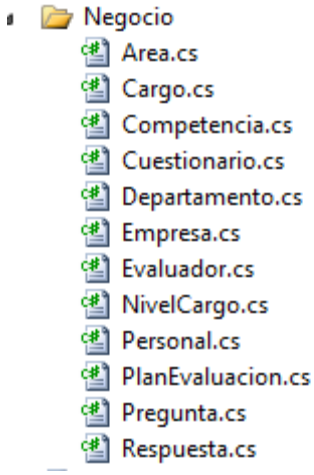
1) Capa de Datos



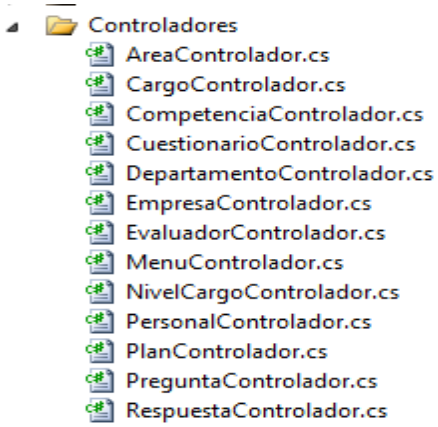
2) Capa Lógica



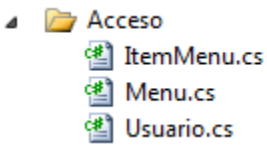
3) Sub Capa Negocio de Capa Lógica



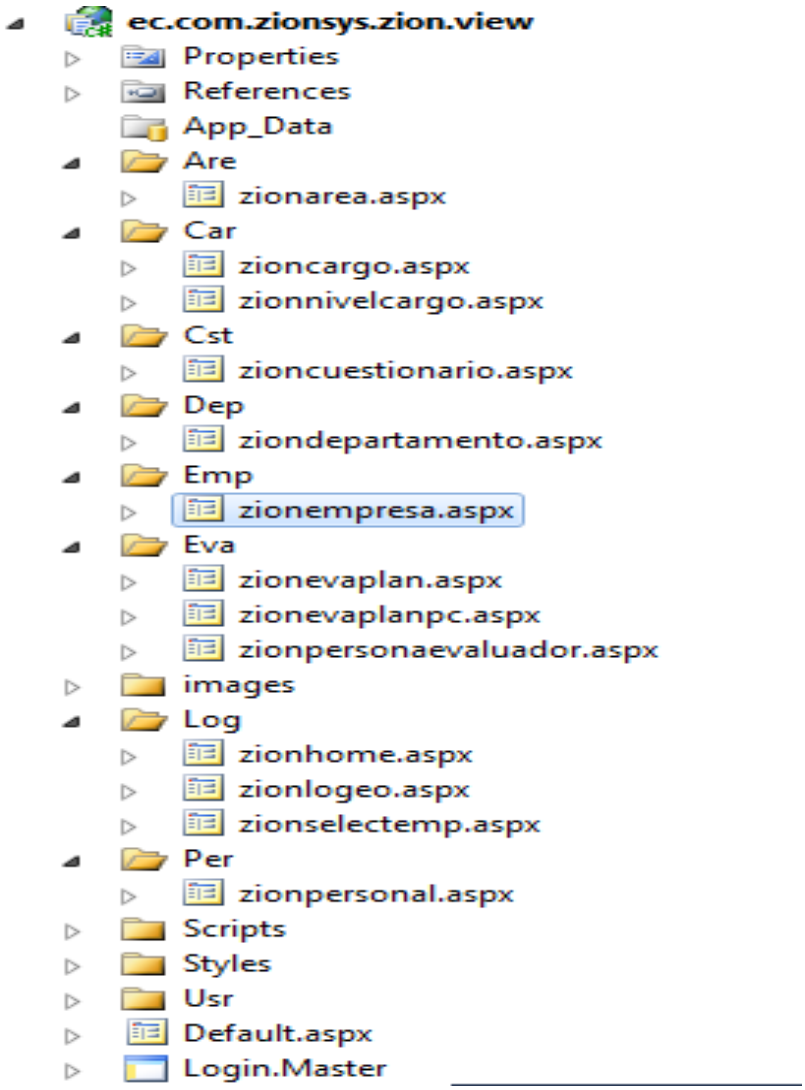
4) Sub Capa Controlador de Capa Lógica



5) Sub Cada Acceso de Capa Lógica



6) Capa Vista



4.2.4.2. Distribución Datos

Dentro de la capa de datos es encuentra también la arquitectura de la Base de Datos, que según la naturaleza del negocio se define: el Diagrama Conceptual y Diagrama Físico de la Base de Datos.

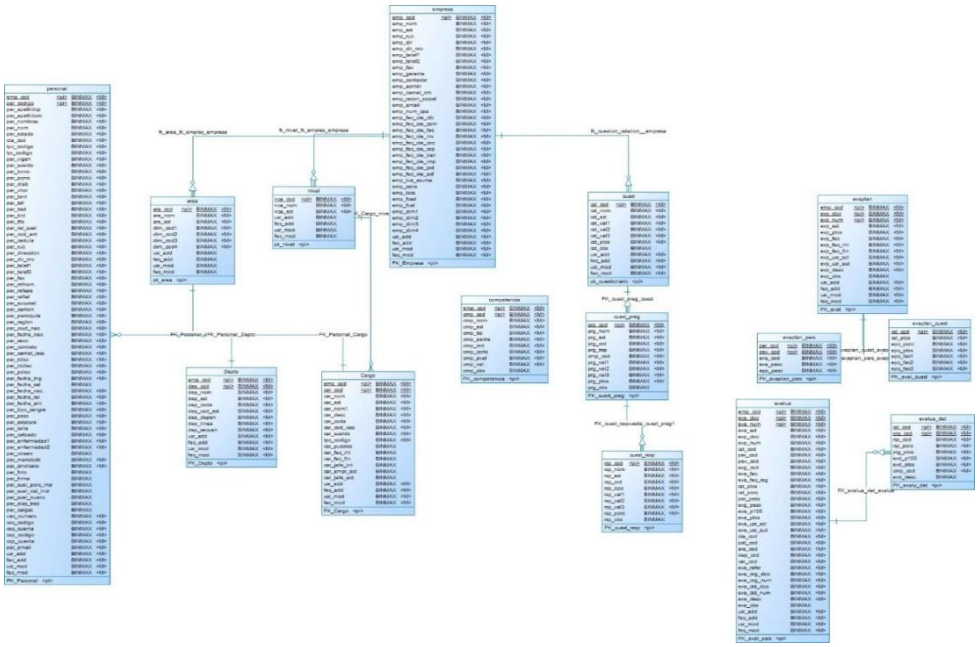


FIGURA 7 Diagrama Conceptual de la Base de Datos

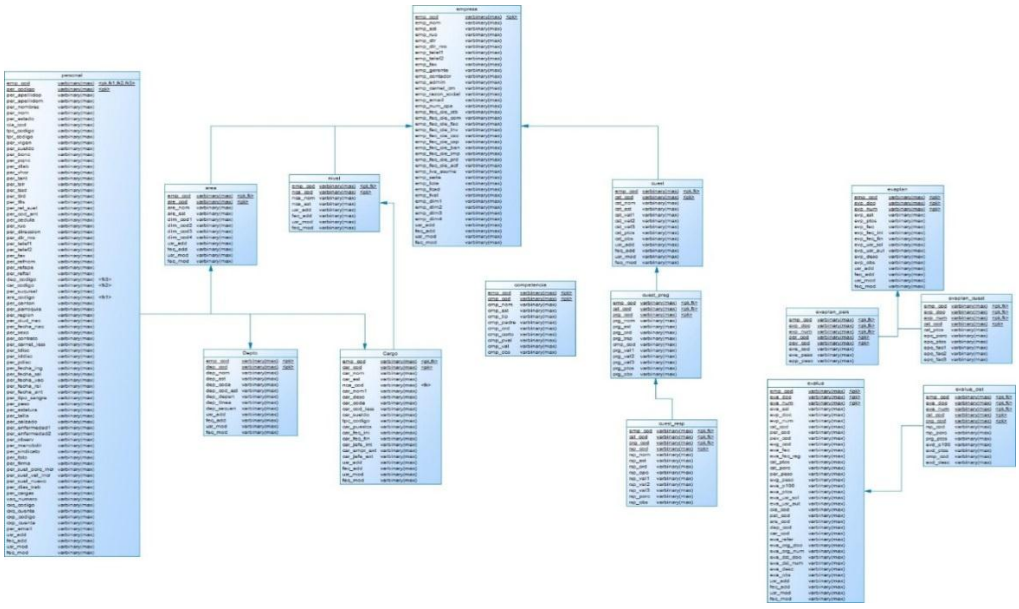


FIGURA 8 Diagrama Físico de la Base de Datos

4.2.4.3. Diagrama de Clases

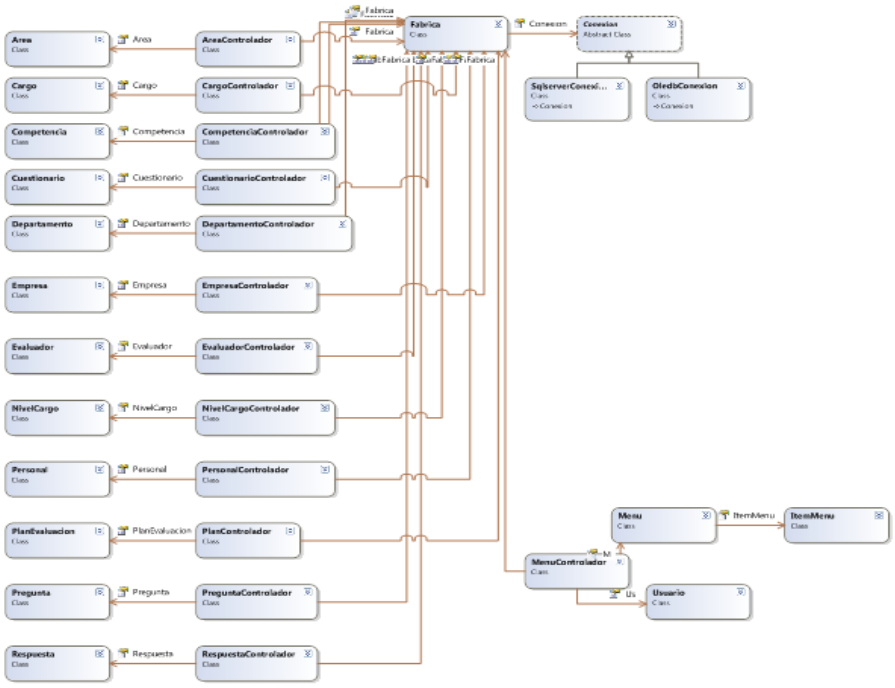


FIGURA 9 Diagrama de Clases

4.3. CONSTRUCCION

4.3.1. Desarrollo

Haciendo mención a los casos de uso, se define el proceso de construcción o desarrollo de los principales procesos según la naturaleza del negocio y el alcance de este proyecto. Los procesos destacados como principales, y de los cuáles se va a exponer su vista y código fuente son:

- Cuestionarios
- Planes de Evaluación

4.3.1.1. Cuestionarios

En la gestión de cuestionarios se presenta una búsqueda en la cual el usuario es capaz de consultar los cuestionarios vigentes mediante los filtros de: Código, Nombre, Estado, Intervalo de Puntos

De los cuestionarios mostrados, se selecciona uno del cual se exponen los datos de:

- Código de la Empresa

- Código del Cuestionario
- Nombre del Cuestionario
- Observación del Cuestionario
- Diferentes valores para operaciones de cuestionarios.
- Puntos que posee el cuestionario en total.
- Estado del cuestionario
- Usuario que añadió el cuestionario
- Fecha de adición del cuestionario
- Usuario de la última modificación del cuestionario
- Fecha de la última modificación del cuestionario

Con estos valores datos visualizados en pantalla, se puede Añadir un nuevo cuestionario, así también como editar uno existente o eliminarlo, además de poder imprimirlo.

Al haber seleccionado un cuestionario, también se muestran sus preguntas en caso de existir, caso contrario se puede de manera añadir una. Tras la selección de una pregunta esta puede ser editada o eliminada, así mismo esta permite visualizar sus respuestas y editarlas, eliminarlas o insertar una nueva.

Los campos visualizados y necesarios para una pregunta son:

- Código del cuestionario
- Código de la pregunta
- Descripción de la pregunta
- Puntos de la pregunta
- Orden de aparición de la pregunta
- Tipo de Respuesta
- Competencia (factor muy importante analizado en el próximo párrafo)
- Observación de la pregunta
- Valores extra de la pregunta para diferentes operaciones

La competencia de la pregunta es un factor muy importante en este proyecto, dado que este es el que determina la calidad de un área en cierta empresa. Es decir, este elemento de competencia es lo que permite analizar

los puntos deseados en la empresa. Estos puntos son definidos por el departamento de Recursos Humanos según la naturaleza del negocio lo requiera. Los puntos exhibidos de las respuestas son:

- Código de la pregunta
- Código de la respuesta
- Descripción de la respuesta
- Orden de aparición de la respuesta
- Literal que identifica una respuesta
- Porcentaje del valor de la pregunta que equivale la respuesta
- Observación de la respuesta
- Diferentes valores de la respuesta para ciertas operaciones
- Estado de la respuesta

Las siguientes figuras 10,11,12 visualizan algunas pantallas del proceso descrito:

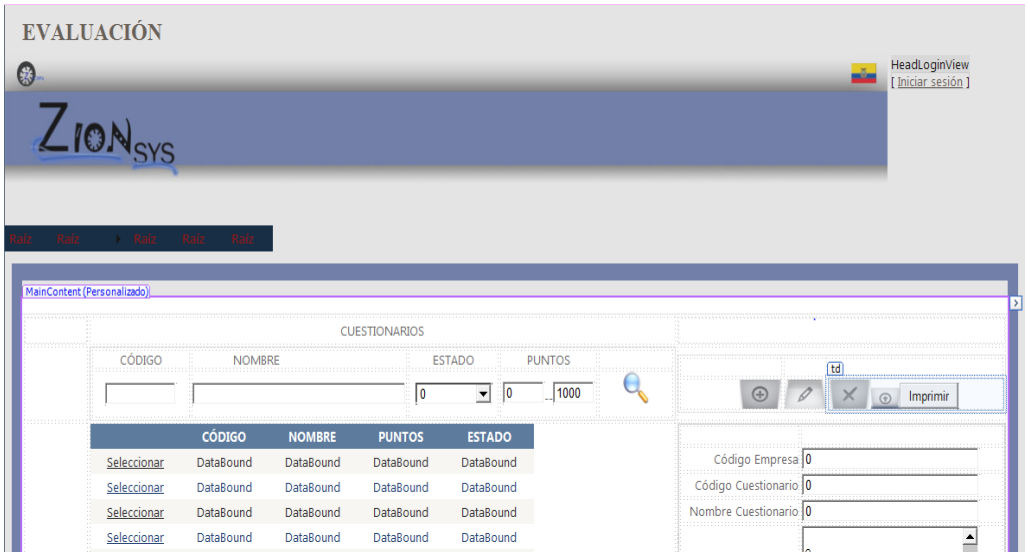


FIGURA 10 Pantalla Selección de Cuestionarios

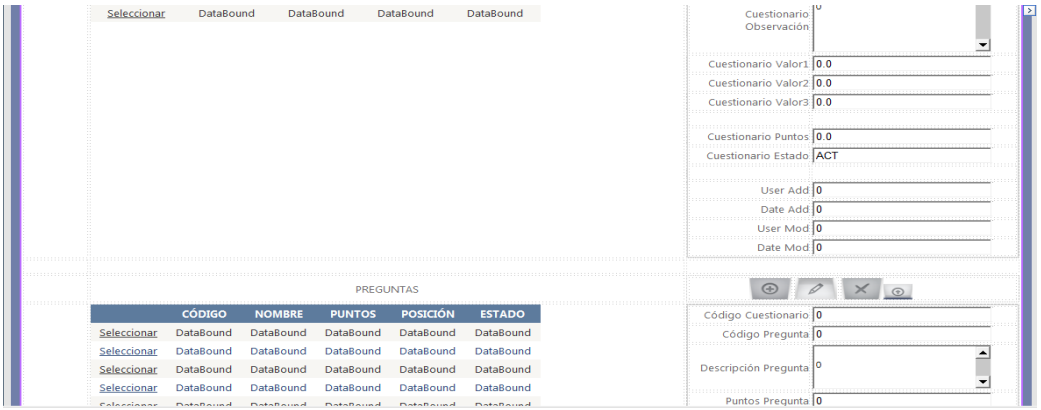


FIGURA 11 Pantalla Actualización Preguntas

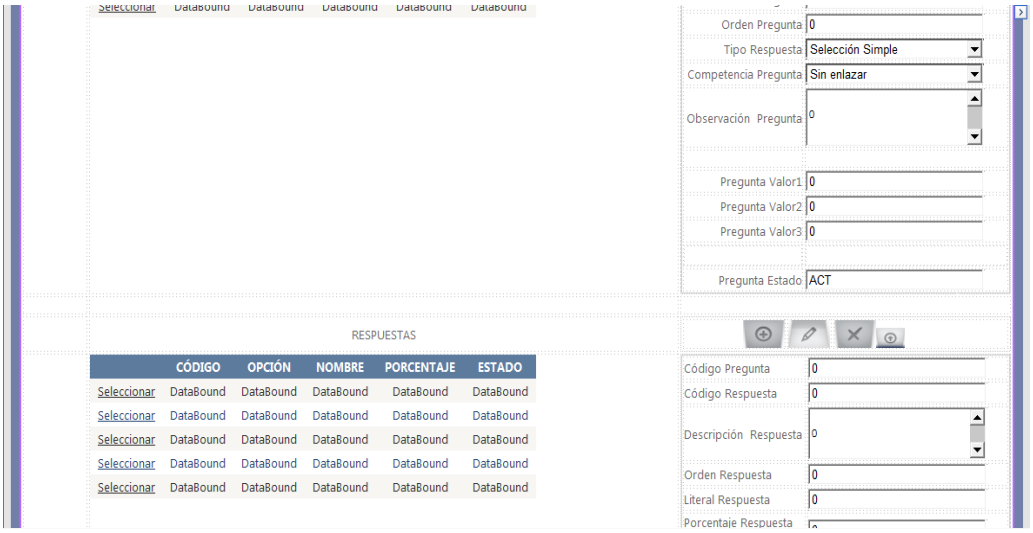


FIGURA 12 Pantalla Actualización Respuestas

La figura 13 muestra un segmento del lenguaje de código asp.net

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Sitio.Master" AutoEventWireup="true"
CodeBehind="zioncuestionario.aspx.cs" Inherits="ec.com.zionsys.zion.view.Cst.zioncuestionario" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
 <style type="text/css">
      .style3
      {
         width: 100%;
      }
      .style5
      {
       width: 676px;
       text-align: center;
      } .....

     .......
            <td>
                </td>
         </tr>
      </table>
      <br />
   </div>
</asp:Content>
```

FIGURA 13 Lenguaje de Marcas Asp.net

La vista de un proceso es fundamental, pero en el fondo esta solo presenta los datos que son realmente procesados en otra capa. La capa Lógica ofrece la capacidad de ordenar el modelo y el controlador que se usa en el proceso como se muestra a través del código fuente a continuación:

Contenedor de información de Cuestionario, Pregunta y Respuesta:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ec.com.zionsys.zion.logic.Negocio
{
 public class Cuestionario
 {
 public string empresaCodigo { get; set; }
 public string cuestionarioCodigo { get; set; }
 public string cuestionarioNombre { get; set; }
 public string cuestionarioEstado { get; set; }
 public string cuestionarioValor1 { get; set; }
 public string cuestionarioValor2 { get; set; }
 public string cuestionarioValor3 { get; set; }
 public string cuestionarioPuntos{ get; set; }
 public string cuestionarioObservacion { get; set; }
 public string cuestionarioUsrAdd { get; set; }
 public string cuestionarioUsrMod { get; set; }
 public DateTime cuestionarioFecAdd { get; set; }
 public DateTime cuestionarioFecMod { get; set; }
 public List<Pregunta> preguntas { get; set; }
 }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ec.com.zionsys.zion.logic.Negocio
{
 public class Pregunta
 {
 public string empresaCodigo { get; set; }
 public string cuestionarioCodigo { get; set; }
 public string preguntaCodigo { get; set; }
```

```
 public string preguntaNombre { get; set; }
 public string preguntaEstado { get; set; }
 public string preguntaOrden { get; set; }
 public string preguntaTipoRespuesta { get; set; }
 public string competenciaCodigo { get; set; }
 public string preguntaValor1{ get; set; }
 public string preguntaValor2 { get; set; }
 public string preguntaValor3 { get; set; }
 public string preguntaPuntos { get; set; }
 public string preguntaObservacion { get; set; }
 }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ec.com.zionsys.zion.logic.Negocio
{
 public class Respuesta
 {
 public string empresaCodigo { get; set; }
 public string cuestionarioCodigo { get; set; }
 public string preguntaCodigo { get; set; }
 public string respuestaCodigo { get; set; }
 public string respuestaNombre { get; set; }
 public string respuestaEstado { get; set; }
 public string respuestaOrden { get; set; }
 public string respuestaOpcion { get; set; }
 public string respuestaValor1 { get; set; }
 public string respuestaValor2 { get; set; }
 public string respuestaValor3 { get; set; }
 public string respuestaPorcentaje { get; set; }
 public string respuestaObservacion{ get; set; }
 }
}
```

Controladores de Cuestionario:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ec.com.zionsys.zion.data.BD;
using ec.com.zionsys.zion.logic.Negocio;
using System.Data;
```

```
namespace ec.com.zionsys.zion.logic.Controladores
{
 public class CuestionarioControlador
 {
 private Fabrica fabrica;
 public CuestionarioControlador()
 {
 fabrica = new Fabrica("Sqlserver", "sa", "Admin123");
 }
 public void InsertarCuestionario(Cuestionario cuestionario)
 {
 Conexion conexion = fabrica.crearConexion();
 string sqlquery = "INSERT INTO [cuest] " +
 "([emp_cod]"+
 ",[cst_cod]"+
 ",[cst_nom]"+
 ",[cst_est]"+
 ",[cst_val1]"+
 ",[cst_val2]"+
 ",[cst_val3]"+
 ",[cst_ptos]"+
 ",[cst_obs]"+
 ",[usr_add]"+
 ",[fec_add]"+
 ",[usr_mod]"+
 ",[fec_mod]) "+
 " VALUES "+
 "('"+cuestionario.empresaCodigo+"'"+
 ",'"+cuestionario.cuestionarioCodigo+"'"+
 ",'"+cuestionario.cuestionarioNombre+"'"+
 ",'"+cuestionario.cuestionarioEstado+"'"+
 ",'" + cuestionario.cuestionarioValor1 + "'" +
 ",'" + cuestionario.cuestionarioValor2 + "'" +
 ",'" + cuestionario.cuestionarioValor3 + "'" +
 ",'" + cuestionario.cuestionarioPuntos + "'" +
 ",'"+cuestionario.cuestionarioObservacion+"'"+
 ",'"+cuestionario.cuestionarioUsrAdd+"'"+
 ",'" + cuestionario.cuestionarioFecAdd.ToString("yyyy/MM/dd hh:mm:ss") + "'" +
 ",'" + cuestionario.cuestionarioUsrMod + "'" +
 ",'" + cuestionario.cuestionarioFecMod.ToString("yyyy/MM/dd hh:mm:ss") + "');";
 conexion.EnviarComando(sqlquery);
 }
 public void EditarCuestionario(Cuestionario cuestionario)
```

```
        {
        Conexion conexion = fabrica.crearConexion();
        string sqlquery = "UPDATE [cuest] SET " +
        "[cst_nom] = '"+cuestionario.cuestionarioNombre+"'" +
        ",[cst_est] = '"+cuestionario.cuestionarioEstado+"'" +
        ",[cst_val1] = '"+cuestionario.cuestionarioValor1+"'" +
        ",[cst_val2] = '"+cuestionario.cuestionarioValor2+"'" +
        ",[cst_val3] = '" + cuestionario.cuestionarioValor3 + "'" +
        ",[cst_ptos] = '" + cuestionario.cuestionarioPuntos + "'" +
        ",[cst_obs] = '" + cuestionario.cuestionarioObservacion + "'" +
        ",[usr_mod] = '" + cuestionario.cuestionarioUsrMod + "'" +
        ",[fec_mod] = '" + cuestionario.cuestionarioFecMod.ToString("yyyy/MM/dd hh:mm:ss") + "'" +
        "WHERE cst_cod = '"+cuestionario.cuestionarioCodigo+"' and emp_cod =
'"+cuestionario.empresaCodigo+"';";
        conexion.EnviarComando(sqlquery);
        }
        public DataSet BuscarPorEmp(string emp_cod)
        {
        Conexion conexion = fabrica.crearConexion();
        DataSet ds = conexion.Consultar("select * from cuest where emp_cod = '" + emp_cod + "';");
        return ds;
        }
        public Cuestionario Buscar(string cod)
        {
        Conexion conexion = fabrica.crearConexion();
        DataSet ds = conexion.Consultar("select * from cuest where cst_cod = '" + cod + "';");
        Cuestionario cuestionarioBuscado = new Cuestionario();
        cuestionarioBuscado.empresaCodigo = ds.Tables["TABLA"].Rows[0]["emp_cod"].ToString();
        cuestionarioBuscado.cuestionarioCodigo =
ds.Tables["TABLA"].Rows[0]["cst_cod"].ToString();
        cuestionarioBuscado.cuestionarioNombre =
ds.Tables["TABLA"].Rows[0]["cst_nom"].ToString();
        cuestionarioBuscado.cuestionarioEstado = ds.Tables["TABLA"].Rows[0]["cst_est"].ToString();
        cuestionarioBuscado.cuestionarioValor1 =
ds.Tables["TABLA"].Rows[0]["cst_val1"].ToString();
        cuestionarioBuscado.cuestionarioValor2 =
ds.Tables["TABLA"].Rows[0]["cst_val2"].ToString();
        cuestionarioBuscado.cuestionarioValor3 =
ds.Tables["TABLA"].Rows[0]["cst_val3"].ToString();
        cuestionarioBuscado.cuestionarioPuntos =
ds.Tables["TABLA"].Rows[0]["cst_ptos"].ToString();
        cuestionarioBuscado.cuestionarioObservacion =
ds.Tables["TABLA"].Rows[0]["cst_obs"].ToString();
```

```
            cuestionarioBuscado.cuestionarioUsrAdd =
ds.Tables["TABLA"].Rows[0]["usr_add"].ToString();
            cuestionarioBuscado.cuestionarioFecAdd =
DateTime.Parse(ds.Tables["TABLA"].Rows[0]["fec_add"].ToString());
            cuestionarioBuscado.cuestionarioUsrMod =
ds.Tables["TABLA"].Rows[0]["usr_mod"].ToString();
            cuestionarioBuscado.cuestionarioFecMod =
DateTime.Parse(ds.Tables["TABLA"].Rows[0]["fec_mod"].ToString());
            return cuestionarioBuscado;
            }
            public Cuestionario Buscar2(string cod, string emp_cod)
            {
            Conexion conexion = fabrica.crearConexion();
            DataSet ds = conexion.Consultar("select * from cuest where cst_cod = '" + cod + "' and
emp_cod = '"+emp_cod+"';");
            Cuestionario cuestionarioBuscado = new Cuestionario();
            cuestionarioBuscado.empresaCodigo = ds.Tables["TABLA"].Rows[0]["emp_cod"].ToString();
            cuestionarioBuscado.cuestionarioCodigo =
ds.Tables["TABLA"].Rows[0]["cst_cod"].ToString();
            cuestionarioBuscado.cuestionarioNombre =
ds.Tables["TABLA"].Rows[0]["cst_nom"].ToString();
            cuestionarioBuscado.cuestionarioEstado = ds.Tables["TABLA"].Rows[0]["cst_est"].ToString();
            cuestionarioBuscado.cuestionarioValor1 =
ds.Tables["TABLA"].Rows[0]["cst_val1"].ToString();
            cuestionarioBuscado.cuestionarioValor2 =
ds.Tables["TABLA"].Rows[0]["cst_val2"].ToString();
            cuestionarioBuscado.cuestionarioValor3 =
ds.Tables["TABLA"].Rows[0]["cst_val3"].ToString();
            cuestionarioBuscado.cuestionarioPuntos =
ds.Tables["TABLA"].Rows[0]["cst_ptos"].ToString();
            cuestionarioBuscado.cuestionarioObservacion =
ds.Tables["TABLA"].Rows[0]["cst_obs"].ToString();
            cuestionarioBuscado.cuestionarioUsrAdd =
ds.Tables["TABLA"].Rows[0]["usr_add"].ToString();
            cuestionarioBuscado.cuestionarioFecAdd =
DateTime.Parse(ds.Tables["TABLA"].Rows[0]["fec_add"].ToString());
            cuestionarioBuscado.cuestionarioUsrMod =
ds.Tables["TABLA"].Rows[0]["usr_mod"].ToString();
            cuestionarioBuscado.cuestionarioFecMod =
DateTime.Parse(ds.Tables["TABLA"].Rows[0]["fec_mod"].ToString());
            return cuestionarioBuscado;
            }
            public void EliminarCuestionario(string cst_cod)
```

```
{
Conexion conexion = fabrica.crearConexion();
string sqlquery = "DELETE FROM [cuest] WHERE cst_cod = '"+cst_cod+"'";
conexion.EnviarComando(sqlquery);
}
public DataSet BusquedaAvanzada(string complemento)
{
Conexion conexion = fabrica.crearConexion();
DataSet ds = conexion.Consultar("select * from cuest " + complemento);
return ds;
}
public DataSet BuscarPorCodyEmp(string cst_cod , string emp_cod)
{
Conexion conexion = fabrica.crearConexion();
DataSet ds = conexion.Consultar("select * from cuest where emp_cod = '" + emp_cod + "'
and cst_cod = '"+cst_cod+"';");
return ds;
}
public void ActualizarPuntos(string puntos, string cst_cod, string emp_cod)
{
Conexion conexion = fabrica.crearConexion();
string sqlquery = "UPDATE [cuest] SET " +
"[cst_ptos] = '" + puntos + "' " +
"WHERE cst_cod = '" + cst_cod + "' and emp_cod = '" + emp_cod + "';";
conexion.EnviarComando(sqlquery);
}}
```

4.3.1.2. Plan de Evaluación

De similar manera, se exhibe la vista del proceso de Plan de Evaluación, no obstante en este caso el código se puede visualizar en el repositorio del proyecto, las algunas pantallas se visualizan en la figuras

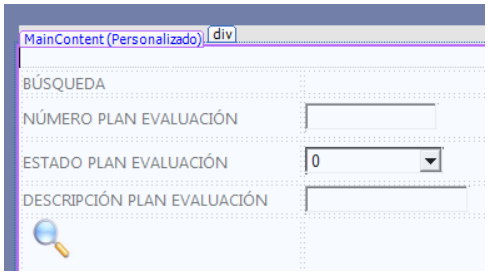


FIGURA 14 Pantalla Búsqueda de Órdenes de Evaluación

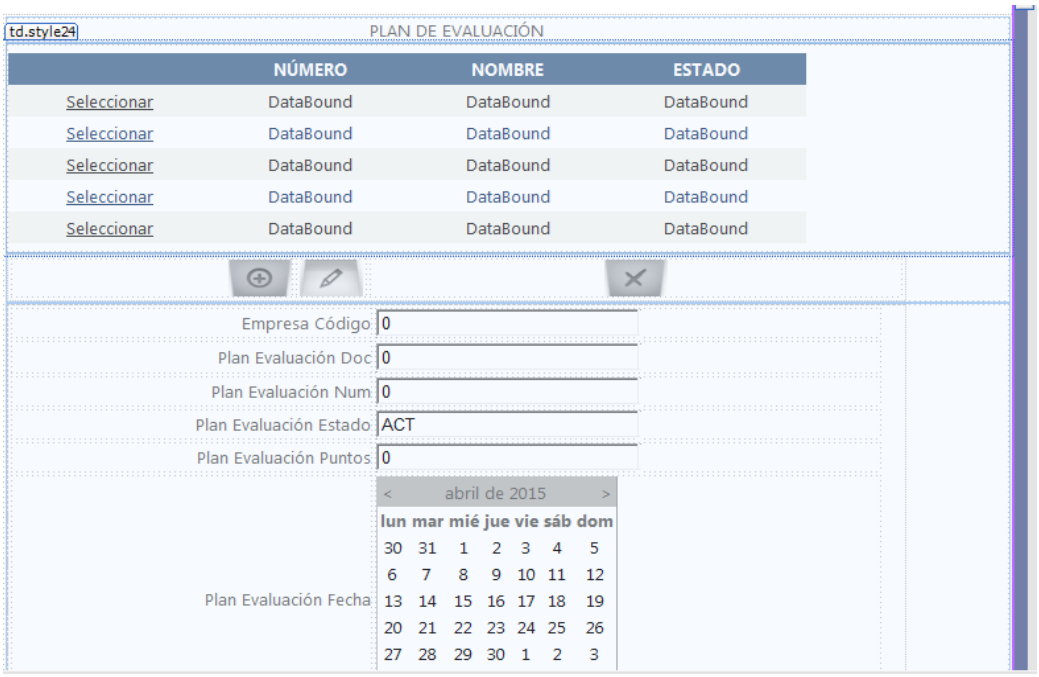


FIGURA 15 Pantalla Planes de Evaluación

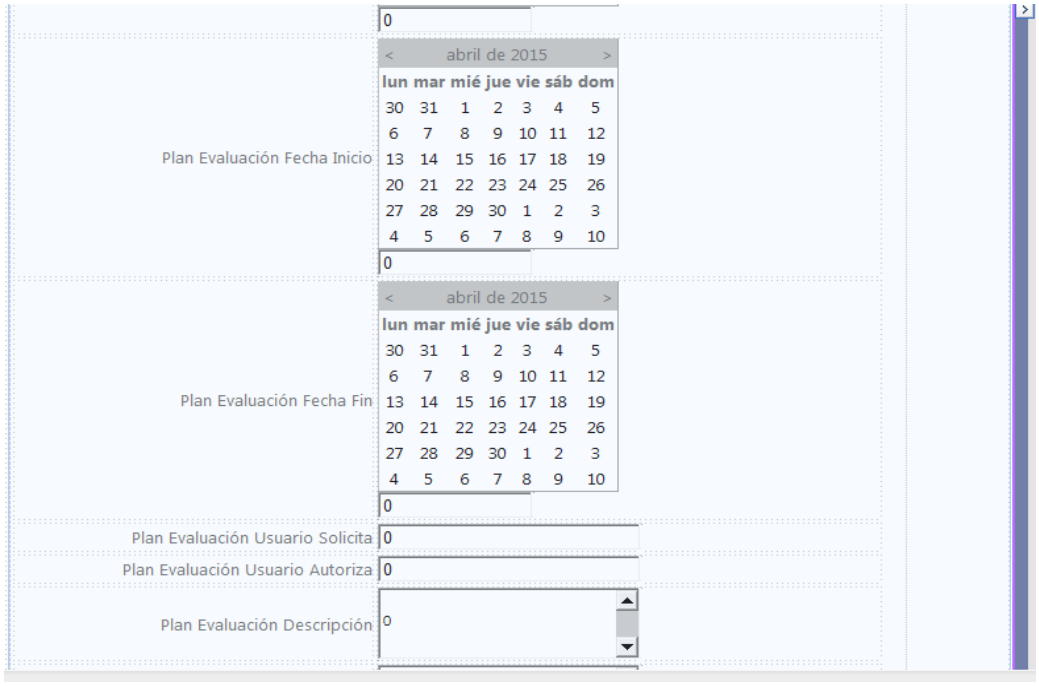


FIGURA 16 Pantalla de Ingreso Datos Plan Evaluación

4.3.1.3. Asignación de Cuestionarios

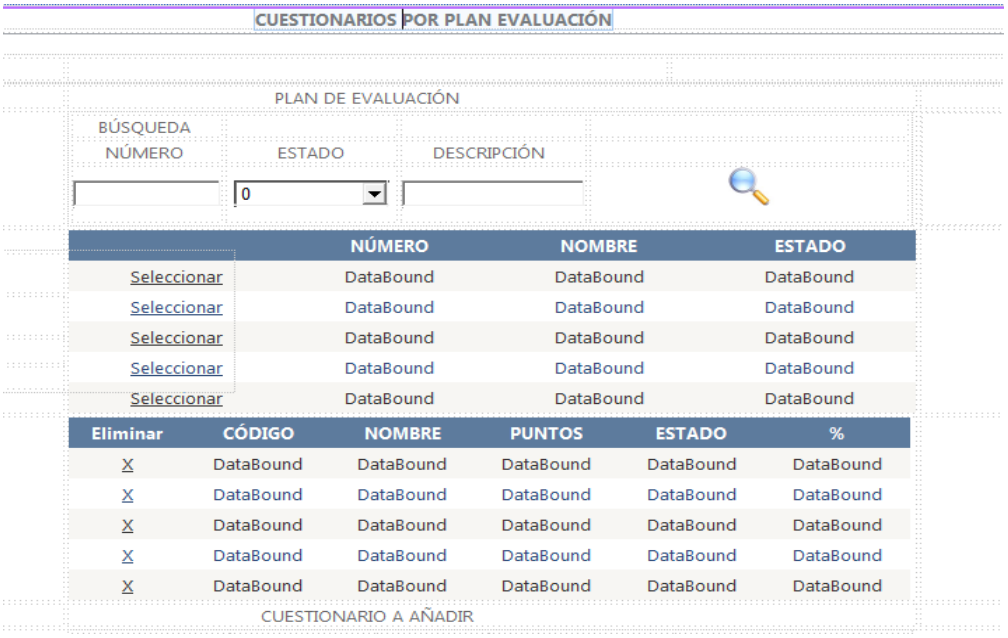


FIGURA 17 Pantalla Cuestionarios por Plan Evaluación

FIGURA 18 Pantalla Valoración Preguntas

4.3.2. Casos de Pruebas

El proceso para realizar estas pruebas fue el de permitir a un representante de empresa Zeuz Sistemas:

- Ingresar diferentes tipos de datos.
- Modificar datos existentes
- Eliminar Datos
- Consultar Datos

En el trascurso del desarrollo y construcción de la aplicación, las pruebas se realizaron sobre las vistas elegidas por el cliente, dado que este consideraba la importancia de solo dos vistas de entre todo el proyecto. Las vistas elegidas por el cliente fueron: Cuestionarios, Plan de Evaluación, como se muestra la tabla 11.

TABLA 11

Resultados de Pruebas con el Cliente

Sec	Fecha	Descripción	Resultado
1	11/04/2015	Ingreso de Cuestionario	Se ingresó correctamente
2	11/04/2015	Modificación de Cuestionario	
3	11/04/2015	Eliminación de Cuestionario	No permitió eliminar porque el cuestionario posee preguntas y respuestas (se deben eliminar estas primero)
4	11/04/2015	Ingreso de Pregunta	Se ingresó correctamente
5	11/04/2015	Modificación de Pregunta	Se modificó correctamente
6	11/04/2015	Eliminación de Pregunta	Se eliminó correctamente
7	11/04/2015	Ingreso de Respuesta	Se ingresó correctamente
8	11/04/2015	Modificación de Respuesta	Se modificó correctamente
9	11/04/2015	Eliminación de Respuesta	Se eliminó correctamente
10	11/04/2015	Ingreso Plan de Evaluación	Se ingresó correctamente
11	11/04/2015	Modificación Plan de Evaluación	Se modificó correctamente

Las pruebas fueron validadas mediante mensajes de aceptación que la aplicación muestra el momento de haber realizado ciertas tareas. Los mensajes expuestos son los que se muestran en la figura 19

FIGURA 19 Mensajes de Confirmación

4.3.3. Planilla de Incidentes

Durante el proceso de desarrollo del proyecto, en especial en la fase de construcción se presentaron ciertos incidentes que dificultaron las tareas del programador, estos incidentes fueron registrados y solucionados, como se muestra en la tabla 12.

TABLA 12

Planilla de Incidentes

	Fecha	Incidente	Solución
1	31/03/15	No se logra conectar la aplicación con la base de datos	Se habilita la comunicación de la Base de Datos a conexiones remotas
2	06/04/2015	Error de compatibilidad de formatos de fecha de la aplicación con la base de datos	Se establece un estándar de fecha compatibles en la aplicación
3	08/04/2015	Las páginas devuelven su scroll al principio de las mismas tras una acción de un control	Se habilita AutoPostBack y se define en el archivo web.config la característica para recordar el nivel del scroll
4	09/04/2015	No aparecen opciones del menú establecido	Se cambió el formato css que usa el control del menú
5	12/04/2015	Erro al instalar ISS en el servidor web	Se reinstala Windows Server 2008

4.4. TRANSICIÓN

4.4.1. Scripts de Despliegue

4.4.1.1. Aplicación

Para poder ejecutar la aplicación, es necesario ubicar ésta dentro un servidor de aplicaciones web. Para este proyecto se utiliza el servidor web

IIS (Internet Information Server). Es necesario ubicar los archivos de la aplicación dentro de los siguientes directorios en el equipo que tenga instalado el servidor web, como se muestra en la figura 20.

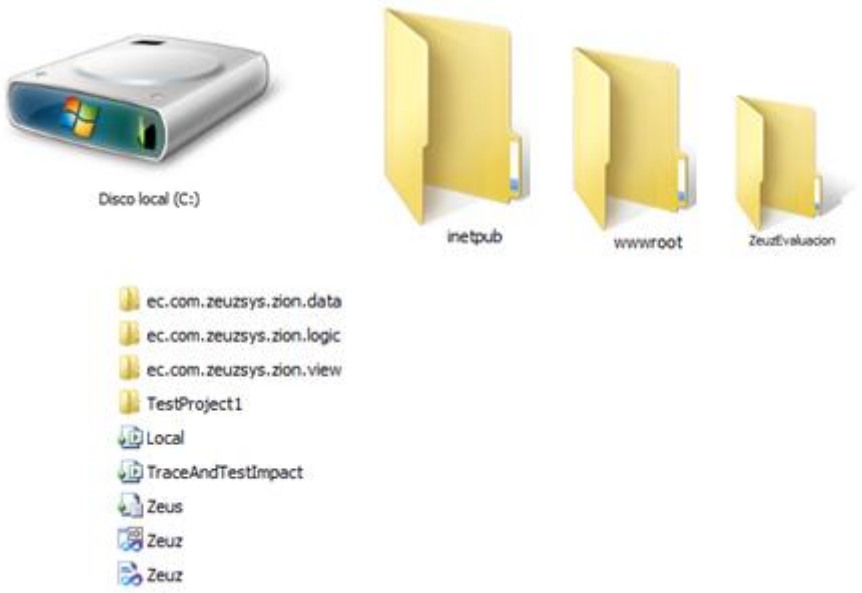


FIGURA 20 Directorios de la Aplicación

Posterior a la ubicación se procede a la configuración del servidor web por parte de un técnico.

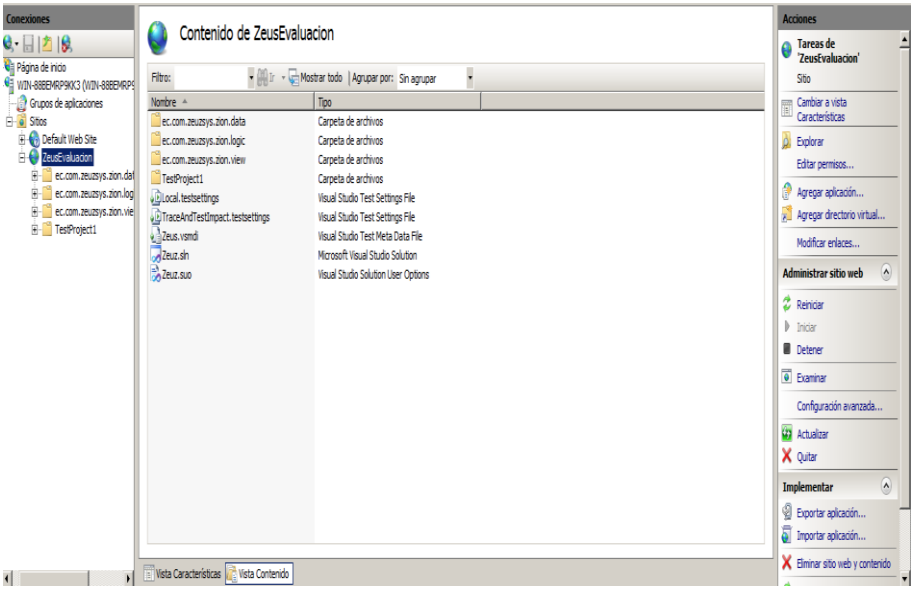


FIGURA 21 Contenido del Proyecto

4.4.1.2. Servidor Web

Es necesario tener instalado el servidor web IIS (Internet Information Server). Para lograr una correcta instalación es conveniente seguir los pasos guiados que ofrece la empresa Microsoft a la cual se puede acceder mediante el siguiente enlace: http://windows.microsoft.com/es-419/windows-vista/install-internet-information-services-iis-7-0

4.4.1.3. Servidor de Datos

Para lograr tener lista la base de datos es necesario correr el script proporcionado en el repositorio del proyecto, pero para eso se debe tener instalado el gestor de base de datos. Para este proyecto se usa el Gestor de Base de Datos Sql Server 2008 R2 el cual se puede instalar siguiendo la guía proporcionada por Microsoft: http://www.microsoft.com/es-es/download/details.aspx?id=30438

4.4.1.4. Reportes

Los reportes son parte fundamental del proyecto, estos son leídos desde la aplicación, pero para eso es necesario tener instalado el software de reportes en el equipo designado como servidor web. En este caso se usa Crystal Reports, el cual se puede instalar siguiendo la guía proporcionada en el enlace que se muestra a continuación: http://www.microsoft.com/es-es/download/details.aspx?id=30438

4.4.2. Repositorio del Proyecto

El repositorio del proyecto consta de los siguientes elementos, como se muestra en la figura 22.

- Versión de Aplicación
- Script de Base de Datos
- Versión de Reportes
- Versión de Documentación
- Versión de Manual de Usuario
- Versión de Imágenes del Proyecto

FIGURA 22 Repositorio del Proyecto

4.4.3. Entrega / Recepción

El presente proyecto se desarrolló y entregó de acuerdo a las fechas establecidas, como lo muestra el Anexo 6 Acta de Entrega Recepción.

CAPÍTULO 5: CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

- Para el desarrollo de software existen varias metodologías que permiten a los pequeños grupos de desarrollo concentrarse en construir software fomentando prácticas de fácil adopción, un entorno ordenado y que los proyectos finalicen exitosamente. Con el análisis y comparación de estas 3 metodologías (RUP, XP, MFS) se define una guía metodológica basada en: fases, disciplinas, roles, artefactos, patrones que contribuyen a la implementación y posterior adaptación del proceso a la realidad de la organización.
- Para llevar a cabo este trabajo se recabó información detallada de: Procesos de Desarrollo, Metodologías Ágiles, Sistemas Operativos, Bases Distribuidas, Paradigma Objetos, Herramientas de Análisis y Diseño y Recursos Humanos basado en Competencias, lo que permitió tener una visión global de toda la problemática y el conocimiento necesario para la definición de los aspecto relevantes y la elaboración del Sistema de Evaluación Basado en Competencias.
- Con el análisis comparativo de las metodologías y la elaboración de las matrices se concluye que las fases y disciplinas permite llevar un mejor control y administración de tiempos de entrega, con la aplicación de roles y artefactos se permite una delegación de responsabilidades y funciones sobre el proyecto, además la definición de patrones permite una mejor fluidez en la comunicación entre el equipo d trabajo y los usuarios.
- Esta definición de permite priorizar a las personas sobre el proceso, pues el mejor control y delegación de responsabilidades permite que la codificación y pruebas con los usuarios se efectúen independientemente por procesos, dejando la respectiva constancia del correcto funcionamiento
- La elaboración del plan de Proyecto, Especificaciones de Requerimientos, Casos de Uso, Bases de Datos y Procesos, permitió definir tiempo de entrega, los cuales se fueron ajustando de acuerdo

las reuniones semanales de seguimiento de proyecto, entre los usuarios y el equipo de desarrollo.

- Las pruebas individuales con los usuarios antes y después de cada proceso, permitieron realizar ajustes a los diseños de bases de datos e intefases de presentación, mediante un mayor involucramiento de usuarios en el desarrollo y logrando una total aceptación del producto final.
- El desarrollo de la aplicación se realizó utilizando herramientas actuales y orientadas a objetos como: Power Designer, Visual Studio 2010, SQL Server 2008, Internet Information Server, logrando mejores tiempos de respuesta y depuración de errores, también la captura de pantallas para la elaboración de los manuales.
- La arquitectura de Aplicación se realizó en capas y subcapas, lo que permitió separar: la funcionalidad, actualización de datos y presentación de información al usuario, minimizando los riesgos y optimizando los tiempos.
- El sistema desarrollado permite la evaluación de recursos Humanos basado en Competencias utilizando la técnica de 360 grados, lo que permitió identificar las falencias en el clima laboral de la organización y la tomas de correctivos.

5.2. RECOMENDACIONES

- Para la utilización de la guía metodológica definida en esta tesis, se recomienda tener claro los conceptos descritos en este documento, sin mayor complicación pues el modelo básico son las metodologías agiles.
- Son necesarias y obligatorias las reuniones semanales de seguimiento del proyecto con los usuarios y el equipo de proyecto, que permitirá realizar ajustes a los diseños y tiempos de respuestas, sugeridos, coordinadas y aceptados por el cliente.

- Es necesario para el desarrollo lograr un equipo de programadores con experiencia por el tiempo de respuesta y básicos por los costos, logrando un punto de equilibrio entre tiempo y costo.
- Se recomienda realizar pruebas obligatorias antes y después en cada proceso, las que permitirán tener aceptaciones parciales parte del cliente.
- Si el proyecto es de gran dimensión, se recomienda trabajar con un equipo multidisciplinario y dividir en módulos funcionales específicos, trabajando aisladamente en cada uno con la lista de características y pruebas por iteración, para poder entregar el software a medida de la necesidad del cliente.
- Se recomienda que el desarrollo de aplicaciones se lo realice en varias capas, al menos 3: Presentación, Negocio y de Datos, de tal manera que el producto final pueda gozar de seguridad, fiabilidad, escalabilidad y portabilidad.
- Se recomienda que el Líder del Proyecto sea un profesional con perfil de formación informática, son sólidos conocimientos en: desarrollo de sistemas, aplicaciones web y herramientas orientadas a objetos.
- Se recomienda la utilización de la guía metodológica definida en el desarrollo de los otros módulos de las aplicaciones de Zeuz Sistemas.

BIBLIOGRAFÍA

Beck, K. (2000). *Extreme Programing Explained, Addison-Wesley XP Series.*

Booch, G. (1998). *Software Arquitecure and UML.* Retrieved from http://www.rational.com/uml

Colusso R, G. J. (2011). *Introducción a las Metodologías Agiles Desarrollo de Software.*

Cota, A. (1994). *Ingenieria de Software, Soluciones Avanzadas.*

CrystalReport_Microsoft. (2014). *Microsoft*. Retrieved 04 15, 2015, from https://msdn.microsoft.com/es-es/library/ms225593(v=vs.90).aspx

Cubeiro J, F. G. (1996). *Las Competencias para Gestión de Recursos Humanos.* Bilbao: Deusto.

Date. (2000). *Bases de Datos Distribuidas.*

FrameWork_Microsoft. (2014). *Microsoft FrameWork*. Retrieved 04 15, 2015, from https://msdn.microsoft.com/es-es/library/zw4w595w(v=vs.110).aspx

Greiff, W. (1994). *Paradigma vs Metodología, El caso de POO.*

IIS_Microsoft. (2014). *https://technet.microsoft.com/es-es/library/cc753433(v=ws.10).aspx*. Retrieved 04 15, 2015, from https://technet.microsoft.com/es-es/library/cc753433(v=ws.10).aspx

Narváez, G. (2015). *TESIS APLICACIÓN DISTRIBUIDA MULTI CAPAS.* Quito.

PowerDesigner_Sysbase. (2014). *http://www.sybaseproducts.com/practices/enterprise-architecture*. Retrieved Enero 15, 2015, from http://www.sybaseproducts.com/practices/enterprise-architecture

Sanchez, I. M. (2004, Junio 4). *http://www.informatizate.net*. Retrieved Enero 15, 2015, from http://www.informatizate.net: http://www.informatizate.net

SQLServer_Microsoft. (2014). *https://technet.microsoft.com/es-es/library/ms165588(v=sql.105).aspx*. Retrieved from https://technet.microsoft.com/es-es/library/ms165588(v=sql.105).aspx

VIsuaStudio_Microsoft. (2014). *https://msdn.microsoft.com/es-es/library/zw4w595w(v=vs.110).aspx*. Retrieved from https://msdn.microsoft.com/es-es/library/zw4w595w(v=vs.110).aspx

Zavala, G. (2000). *Disenio de Sistemas de Información sobre Internet.* Retrieved from http://www.angelfire.com/scifi/jzavalar

BIOGRAFÍA

GUILLERMO GERMÁN NARVÁEZ VARGAS

CI.: 170902618-9

Lugar y fecha de nacimiento: Guayaquil, 09 de Octubre de 1968

FORMACIÓN ACADÉMICA

Primaria: Escuela San Pedro Pascual, Quito

Secundaria: Colegio San Luis Gonzaga (1ro – 5to)
Colegio Estados Unidos de Brasil (6to)

Superior: Carrera de Ingeniería de Sistemas e Informática,
Universidad de las Fuerzas Armadas - Espe, Sangolquí.

Títulos obtenidos: Bachiller Físico Matemático, 1986

EXPERIENICA LABORAL:

AGA DEL ECUADOR, Programador en Clipper

METROPOLITAN TOURING, Analista de Sistemas, Informix

GRUPO MODERNA, Gerente de Sistemas

UNIVERSIDAD FUERZAS ARMADAS - ESPE, Profesor

UNIVERSIDAD TECNOLOGICA ISRAEL, Profesor

ZEUZ SISTEMAS, Gerente General

HOJA DE LEGALIZACIÓN DE FIRMAS

ELABORADO POR

Guillermo Germán Narváez Vargas

DIRECTOR DE LA CARRERA

Ing. Mauricio Campaña

Sangolquí, Mayo 2015