

**UNIVERSIDAD TECNOLÓGICA PRIVADA DE SANTA CRUZ**

**FACULTAD DE CIENCIAS Y TECNOLOGÍA**

**CARRERA: INGENIERÍA DE SISTEMAS**

**SISTEMA DE GESTION COMERCIAL PARA VUITON SRL.**

**Caso de Estudio**

**AUTORES: RAÚL DANIEL BARBA ROJAS**

**JORGE ESTEBAN SANDOVAL CABRERA**

**SANTA CRUZ – BOLIVIA 2017**

Tabla de contenido

[**1** **CAPITULO I: ASPECTOS GENERALES** 11](#_Toc499215633)

[**1.1** **INTRODUCCION** 11](#_Toc499215634)

[**1.2** **DEFINICION DEL PROBLEMA** 11](#_Toc499215635)

[**1.3** **OBJETIVOS** 12](#_Toc499215636)

[**1.3.1** **OBJETIVO GENERAL** 12](#_Toc499215637)

[**1.3.2** **OBJETIVOS ESPECIFICOS** 12](#_Toc499215638)

[**1.4** **METODOLOGIA DE DESARROLLO** 12](#_Toc499215639)

[**2** **CAPITULO II: MARCO TEORICO INGENIERIA DE SOFTWARE** 13](#_Toc499215640)

[**2.1** **Que es y que no es Ingeniera de Software** 13](#_Toc499215641)

[**2.2** **Principios Fundamentales** 13](#_Toc499215642)

[**2.2.1** **Principios que guían el proceso** 13](#_Toc499215643)

[**2.2.2** **Principios que guian la practica** 14](#_Toc499215644)

[**2.3** **Principios que guian toda la actividad estructural** 15](#_Toc499215645)

[**2.3.1** **Principios de comunicación** 16](#_Toc499215646)

[**2.3.2** **Principios de planeacion** 17](#_Toc499215647)

[**2.3.3** **Principios de modelado** 18](#_Toc499215648)

[**2.3.4** **Requerimientos de los principios de modelado** 20](#_Toc499215649)

[**2.3.5** **Principios del modelo de diseño** 21](#_Toc499215650)

[**2.4** **Principios de construccion** 23](#_Toc499215651)

[**2.4.1** **Principios de codificacion** 23](#_Toc499215652)

[**2.4.2** **Principios de prueba** 24](#_Toc499215653)

[**2.4.3** **Principios de despliegue** 25](#_Toc499215654)

[**2.5** **Que es calidad** 26](#_Toc499215655)

[**2.5.1** **Calidad del software** 27](#_Toc499215656)

[**2.5.2** **El dilema de la calidad del software** 28](#_Toc499215657)

[**2.5.3** **Lograr la calidad del software** 29](#_Toc499215658)

[**2.6** **Elementos de aseguramiento de la calidad de software** 30](#_Toc499215659)

[**2.6.1** **Seguridad de Software** 31](#_Toc499215660)

[**2.7** **Administracion del Proyecto** 31](#_Toc499215661)

[**2.7.1** **El espectro administrativo** 32](#_Toc499215662)

[**2.7.2** **Definicion de: El Personal** 32](#_Toc499215663)

[**2.7.3** **Definicion de: El Producto** 34](#_Toc499215664)

[**2.7.4** **Definicion de: El Proceso** 35](#_Toc499215665)

[**2.7.5** **Definicion de: El Proyecto** 35](#_Toc499215666)

[**2.8** **Calendarizacion del Proyecto** 36](#_Toc499215667)

[**2.9** **Metricas del Producto** 36](#_Toc499215668)

[**2.9.1** **Medidas, Metricas e Indicadores** 36](#_Toc499215669)

[**2.9.2** **Principios de Medicion** 37](#_Toc499215670)

[**2.9.3** **Metricas para el Modelo de Requerimientos** 37](#_Toc499215671)

[**2.9.4** **Metrica para el Modelo de Diseño** 38](#_Toc499215672)

[**2.9.5** **Metricas para diseño orientado a objetos** 38](#_Toc499215673)

[**2.9.6** **Metricas para Pruebas** 40](#_Toc499215674)

[**2.9.7** **Definicion de: Metricas para calidad del Software** 41](#_Toc499215675)

[**2.10** **Adminstracion de Riesgo** 43](#_Toc499215676)

[**2.10.1** **Identificacion de Riesgos** 44](#_Toc499215677)

[**2.10.2** **Proyeccion del Riesgo** 45](#_Toc499215678)

[**2.10.3** **Valoracion de impacto de riesgo** 45](#_Toc499215679)

[**2.10.4** **El plan MMMR** 46](#_Toc499215680)

[**3** **CAPITULO III: MARCO TEORICO DE LA METODOLOGIA** 47](#_Toc499215681)

[**3.1** **METODOLOGIA DE DESARROLLO XP** 47](#_Toc499215682)

[**3.1.1** **VALORES** 47](#_Toc499215683)

[**3.1.2** **CARACTERISTICAS FUNDAMENTALES** 49](#_Toc499215684)

[**3.1.3** **ROLES** 50](#_Toc499215685)

[**4** **CAPITULO IV: INGENIERIA DE SOFTWARE DEL PROYECTO** 51](#_Toc499215686)

[**4.1** **PLANIFICACION TEMPORAL** 51](#_Toc499215687)

[**4.2** **ORGANIZACIÓN DEL PERSONAL** 52](#_Toc499215688)

[**4.3** **ESTIMACION** 52](#_Toc499215689)

[**4.3.1** **METRICAS BASADAS EN PUNTOS DE FUNCION** 52](#_Toc499215690)

[**4.3.2** **VALORES DE AJUSTE DE COMPLEJIDAD** 53](#_Toc499215691)

[**4.3.3** **CALCULOS DE PUNTOS DE FUNCION** 54](#_Toc499215692)

[**4.3.4** **CALCULO DEL ESFUERZO** 54](#_Toc499215693)

[**4.3.5** **CALCULO DE NUMERO DE PERSONAS QUE CONFORMAN EL EQUIPO DE DESARROLLO DEL SOFTWARE** 54](#_Toc499215694)

[**4.3.6** **COSTO DEL PERSONAL** 54](#_Toc499215695)

[**4.4** **ANALISIS Y GESTION DEL RIESGO** 55](#_Toc499215696)

[**4.4.1** **ESCALA BASADA EN EL ASPECTO DEL PERSONAL** 55](#_Toc499215697)

[**4.4.2** **ESCALA BASADA EN EL ASPECTO TECNICO** 55](#_Toc499215698)

[**4.4.3** **ESCALA BASADA EN EL ASPECTO DEL NEGOCIO** 56](#_Toc499215699)

[**4.4.4** **PROYECCION DEL RIESGO** 57](#_Toc499215700)

[**4.5** **PRESUPUESTO** 57](#_Toc499215701)

[**4.5.1** **COSTO PARCIAL DE LAS TAREAS DE DESARROLLO DE SOFTWARE** 57](#_Toc499215702)

[**4.5.2** **COSTO PARCIAL DE LA GESTION DEL PROYECTO** 58](#_Toc499215703)

[**4.5.3** **COSTO PARCIAL DE RECURSOS** 58](#_Toc499215704)

[**4.5.4** **COSTO TOTAL DEL PROYECTO** 58](#_Toc499215705)

**SISTEMA DE GESTION COMERCIAL PARA VUITON SRL**

1. **SITUACION PROBLEMÁTICA**

VUITON SRL. Comercializa productos de cuero, es decir, zapatos, carteras, cinturones, billeteras, los cuales son importados de Brasil. La empresa es reconocida a nivel nacional pues cuenta con 1 central y 6 sucursales en la ciudad de Santa Cruz, 2 sucursales en la ciudad de La Paz, 2 sucursales en la ciudad de Cochabamba y hace envíos aéreos a las demás ciudades de Bolivia.

A la fecha, VUITON SRL, cuenta con un Sistema de Gestión Comercial, sin embargo, por su expansión a nivel nacional, el sistema actual no cubre las necesidades de automatización de sus tareas y requiere actualizar todo el Sistema de Gestión Comercial.

1. **LOGICA DE NEGOCIO**

El gerente comercial y el encargado de ventas, asisten 2 veces al año, a la Feria del Cuero en Brasil, para elegir los productos y las marcas que van a comprar, para la correspondiente estación del año. Al regresar a Santa Cruz, se registra en el sistema, los artículos, marcas y nombre de la empresa a la que se realizó el pedido.

Aproximadamente 1 mes después VUITON SRL, recibe un correo de parte de la empresa a la que hicieron el pedido, informando el tiempo de llegada de los artículos a la frontera Brasil-Bolivia vía terrestre, inmediatamente, la empresa aduanera en Bolivia, hace el pago de los impuestos correspondientes, recibe los artículos y los envía a la oficina central en la ciudad de Santa Cruz.

La oficina central, hace la recepción de los artículos, verifica que la cantidad sea exactamente igual al pedido, luego verifican que los artículos sean los que se pidieron y el estado en el que llegaron. Si todo está correcto, el administrador y personal de inventario, realizan la codificación de los productos, generándolo como código de barra, imprimiéndolo en etiquetas adhesivas y pegándolas en cada artículo. El código de barra de cada artículo, debe identificar si es un calzado, tipo de calzado, cartera, el color y tamaño de la misma, billetera, color y tamaño o cinturón, color, tamaño y textura; además debe contener la fecha de ingreso a almacén central.

Realizada la tarea de almacenamiento, se debe distribuir los productos a cada sucursal, registrar qué artículos van a qué sucursal y en qué tipo de transporte se está enviando. En cada una de las sucursales, se debe realizar la recepción de los artículos, leyendo el código de barra para ser registrado automáticamente en su almacén, es necesario verificar que los artículos enviados desde almacén central, son los mismos que llegaron a la sucursal destino.

Con los artículos de la temporada distribuidos y recepcionados por cada sucursal, la encargada de Marketing, promociona a través de las redes sociales y televisión, la nueva colección.

Las clientes interesadas en la nueva temporada, van a la sucursal de su preferencia y eligen lo que más le gusta, en muchas ocasiones, el artículo que le interesa, no está disponible en esa sucursal, entonces, el personal de ventas, verifica en el sistema, en qué otra sucursal tienen el artículo requerido y propone a la cliente, esperar un determinado tiempo para que hagan traer el artículo o en su defecto, le da la dirección de la sucursal para que ella pueda ir hasta allá a hacer la compra.

La compra del artículo, puede ser realizada en tipo de moneda, dólares o bolivianos, con tarjetas de débito o crédito. Para clientes frecuentes, se tiene definido en el sistema un porcentaje de descuenta que va desde 10 al 30%. La facturación de la compra es computarizada como Impuestos Internos dispuso.

La clientela que reside en ciudades donde no tienen sucursales, se conectan mediante internet a las redes sociales donde está la foto del artículo, precio y el código, además de los colores en los que se encuentran disponibles, la persona elige el producto de su preferencia y mediante un correo electrónico a la encargada del área comercial, solicita el artículo deseado, incluyendo el código del artículo, el color, la textura y el tamaño del mismo. Además y principalmente, adjuntan al correo, la nota de depósito que realizaron a la cuenta de la empresa, por el valor de la compra. Una vez comprobado el depósito en la cuenta, la encargada del área comercial, hace el envío del o los productos, a través del tipo de transporte más conveniente, para estos casos, la empresa asume el costo de envío. Esta venta también es facturada.

Para la Gerencia, es importante conocer, cuáles son los artículos más vendidos por mes, qué vendedor concreta más venta y genera más ingresos a la empresa. Los vendedores tienen un salario básico de 1.500 Bs, además reciben el pago por horas extra y un bono por artículo vendido, es decir, si vende un calzado, tiene un bono de 2 $us por cada uno, si vende una cartera, tiene un bono de 7 $us por cada una, si vende billetera o cinturón, tiene un bono de 1 $us por cualquiera de ellos. A fin de cada mes, el sistema debe sacar un reporte por vendedor, con la cantidad de artículos vendidos y cantidad de dinero por esas ventas. El vendedor que realice ventas por mayor cantidad de dinero en el mes, es elegido como El Mejor Vendedor del Mes y recibe un bono de 50 $us.

Los reportes de artículos más vendidos y menos vendidos, son importantes para la gerencia, pues con base en ellos decidirá qué comprar y sobretodo, cómo venderlos para que recupere la inversión.

**ESTRUCTURA DE PRESENTACION DEL TRABAJO**

CARÁTULA

ÍNDICE

ENUNCIADO DEL CASO DE ESTUDIO

I. ASPECTOS GENERALES

* Título del proyecto
* Introducción
* Definición del problema
* Delimitaciones
* Objetivo General
* Objetivos Específicos

II. MARCTO TEORICO INGENIERIA DE SOFTWARE

* Qué es y qué no es Ingeniería de Software
* Principios fundamentales
  + Principios que guían el proceso
  + Principios que guían la práctica
* Principios que guían toda la actividad estructural
  + Principios de comunicación
  + Principios de planeación
  + Principios de modelado
  + Requerimientos de los principios de modelado
  + Principios del modelo de diseño
* Principios de construcción
  + Principios de codificación
    - Principios de preparación.
    - Principios de programación
    - Principios de validación
  + Principios de la prueba
  + Principios de despliegue
* Qué es calidad
  + Calidad del software
  + El dilema de la calidad del software
    - Software “suficientemente bueno”
    - El costo de la calidad
    - Riegos
  + Lograr la calidad del software
    - Métodos de la Ingeniería de Software
    - Técnicas de administración de proyectos
    - Control de calidad
      * Aseguramiento de la calidad
* Elementos de aseguramiento de la calidad de software
  + Seguridad del Software
* Administración del Proyecto
  + El espectro Administrativo
    - Definición de: El Personal
      * Participantes
      * Líderes de Equipo
      * Equipo de Software
    - Definición de: El producto
      * Ámbito del software
      * Descomposición del problema
    - Definición de El proceso
      * Fusión de producto y proceso
      * Descomposición del proceso
    - Definición de: El proyecto
* Calendarización del Proyecto
* Métricas del Producto
  + Medidas, métricas e indicadores
  + Principios de medición
  + Métricas para el Modelo de Requerimientos
    - Métricas basada en funciones
    - Métricas para calidad de la especificación
  + Métricas para el Modelo de Diseño
  + Métricas para diseño orientado a objetos
  + Métricas para Pruebas
    - Métricas para pruebas Orientados a Objetos
    - Métricas para Mantenimiento
  + Definición de: Métricas para Calidad del Software
    - Medición de la calidad
    - Eficiencia en la remoción del defecto
    - Establecimiento de un programa de métricas de software
* Administración del Riesgo
  + Identificación de Riesgos
  + Proyección del Riesgo
  + Valoración de impacto de riesgo
  + El plan MMMR

III. MARCO TEORICO DE LA METODOLOGIA ELEGIDA (PUDS o XP)

IV. INGENIERIA DE SOFTWARE DEL PROYECTO

* Planificación temporal.
* Organización del personal (Estructura del equipo de desarrollo del software)
* Estimación del proyecto (Asumir Productividad = 29,5)
  + Métricas basadas en Punto de función
  + Valores de ajuste de complejidad
  + Cálculo de punto de función
  + Cálculo del Esfuerzo
  + Cálculo de número de personas que conformarán el equipo de desarrollo del software.
  + Costo del personal
  + Costo del Software
* Análisis y Gestión de Riesgos
  + Escala basada en el aspecto del personal
  + Escala basada en el aspecto metodológico
  + Escala basada en el aspecto del negocio
* Gestión de la Calidad
  + Calidad en función a 5 métricas (Corrección, Confiabilidad, Eficiencia, Integridad, Facilidad de Mantenimiento)
  + Normas
* Presupuesto
  + Costo parcial de las tareas de desarrollo de software.
  + Costo parcial de la gestión de proyecto.
  + Costo parcial de recursos
  + Costo parcial de servicios básicos.
  + Costo parcial de material extra
  + COSTO TOTAL DEL PROYECTO
  + Plan de comunicación

**SISTEMA DE GESTION COMERCIAL PARA VUITON SRL.**

# **CAPITULO I: ASPECTOS GENERALES**

## **INTRODUCCION**

En la empresa VUITON SRL surgió la necesidad de actualizar todo su sistema de gestión comercial, ya que debido a su expansión nacional, el sistema actual no cubre las necesidades de automatización de sus tareas.

El proyecto será realizado haciendo uso de la metodología XP. La fecha de inicio del proyecto será el 27 de noviembre de 2017 planeado a realizarse en 5 meses.

## **DEFINICION DEL PROBLEMA**

VUITON SRL. Comercializa productos de cuero, es decir, zapatos, carteras, cinturones, billeteras, los cuales son importados de Brasil. La empresa es reconocida a nivel nacional pues cuenta con 1 central y 6 sucursales en la ciudad de Santa Cruz, 2 sucursales en la ciudad de La Paz, 2 sucursales en la ciudad de Cochabamba y hace envíos aéreos a las demás ciudades de Bolivia.

A la fecha, VUITON SRL, cuenta con un Sistema de Gestión Comercial, sin embargo, por su expansión a nivel nacional, el sistema actual no cubre las necesidades de automatización de sus tareas y requiere actualizar todo el Sistema de Gestión Comercial.

## **OBJETIVOS**

### **OBJETIVO GENERAL**

Desarrollar una actualización al Sistema de gestión comercial de la empresa VUITON SRL. para que cubra la necesidad de automatización de sus tareas, utilizando la metodología XP.

### **OBJETIVOS ESPECIFICOS**

* Analizar la información del proceso de gestión y los requerimientos del software.
* Diseñar el modelo de la Base de Datos.
* Diseñar el software.
* Desarrollar el software.
* Realizar pruebas al software.

## **METODOLOGIA DE DESARROLLO**

Se utilizará la metodología XP para realizar este software, lo que evitará la extensa documentación y un proceso demasiado complejo que conlleve mucho tiempo. En cambio, se podrá realizar el software en ciclos cortos que muestren un verdadero avance, ya que las metodologías agiles se enfocan en la gente y los resultados.

# **CAPITULO II: MARCO TEORICO INGENIERIA DE SOFTWARE**

## **Que es y que no es Ingeniera de Software**

La ingeniería de software es un conjunto amplio de principios, conceptos, métodos y herramientas que deben considerarse al planear y desarrollar software, que tiene como propósito el de desarrollar un software de calidad en el tiempo determinado.

No es ingeniería de software cuando se desarrolla un software sin tomar en cuenta la gestión de calidad, riesgo, tiempo y métricas del producto, ocasionando que el proyecto se entrega más tarde, no cumpla con los requerimientos o que exceda el presupuesto inicial del proyecto entrando a una crisis de software.

## **Principios Fundamentales**

La práctica de la ingeniería de software está guiada por un conjunto de principios fundamentales que ayudan en la aplicación del proceso de software significativo y en la ejecución de método eficaces de ingeniería de software. En el nivel del proceso, los principios fundamentales establecen un fundamento filosófico que guía al equipo de software cuando realiza actividades estructurales y actividades sombrilla, cuando navega por el flujo del proceso y elabora un conjunto de productos del trabajo de la ingeniería de software. En el nivel de la práctica, los principios fundamentales definen un conjunto de valores y reglas que sirven como guía cuando se analiza un problema, se diseña una solución, se implementa y prueba ésta y cuando, al final, se entrega el software a la comunidad de usuarios.

### **Principios que guían el proceso**

Los siguientes principios fundamentales se aplican a la estructura y, por extensión, a todo proceso de software:

**Principio 1. Ser ágil.** Ya sea que el modelo de proceso que se elija sea prescriptivo o ágil, son los principios básicos del desarrollo ágil los que deben gobernar el enfoque.

**Principio 2. En cada etapa, centrarse en la calidad**. La condición de salida para toda actividad, acción y tarea del proceso debe centrarse en la calidad del producto del trabajo que se ha generado.

**Principio 3. Estar listo para adaptar**. Cuando sea necesario, adapte su enfoque a las restricciones impuestas por el problema, la gente y el proyecto en sí.

**Principio 4. Formar un equipo eficaz**. El proceso y práctica de la ingeniería de software son importantes, pero el objetivo son las personas. Forme un equipo con organización propia en el que haya confianza y respeto mutuos.

**Principio 5. Establecer mecanismos para la comunicación y coordinación.** Los proyectos fallan porque la información importante cae en las grietas o porque los participantes no coordinan sus esfuerzos para crear un producto final exitoso.

**Principio 6. Administrar el cambio.** El enfoque puede ser formal o informal, pero deben establecerse mecanismos para administrar la forma en la que los cambios se solicitan, evalúan, aprueban e implementan.

**Principio 7. Evaluar el riesgo.** Son muchas las cosas que pueden salir mal cuando se desarrolla software. Es esencial establecer planes de contingencia.

**Principio 8. Crear productos del trabajo que agreguen valor para otros**. Sólo genere aquellos productos del trabajo que agreguen valor para otras actividades, acciones o tareas del proceso. Todo producto del trabajo que se genere como parte de la práctica de ingeniería de software pasará a alguien más.

### **Principios que guian la practica**

Los siguientes principios fundamentales son vitales para la práctica de la ingeniería de software:

**Principio 1. Divide y vencerás.** Dicho en forma más técnica, el análisis y el diseño siempre deben enfatizar la separación de entidades (SdE). Un problema grande es más fácil de resolver si se divide en un conjunto de elementos (o entidades).

**Principio 2. Entender el uso de la abstracción**. En su parte medular, una abstracción es una simplificación de algún elemento complejo de un sistema usado para comunicar significado en una sola frase. En la práctica de la ingeniería de software, se usan muchos niveles diferentes de abstracción, cada uno de los cuales imparte o implica significado que debe comunicarse. En el trabajo de análisis y diseño, un equipo de software normalmente comienza con modelos que representan niveles elevados de abstracción (por ejemplo, una hoja de cálculo) y poco a poco los refina en niveles más bajos de abstracción (como una columna o la función SUM). El objetivo de una abstracción es eliminar la necesidad de comunicar detalles.

**Principio 3. Buscar la coherencia.** Ya sea que se esté creando un modelo de los requerimientos, se desarrolle un diseño de software, se genere código fuente o se elaboren casos de prueba, el principio de coherencia sugiere que un contexto familiar hace que el software sea más fácil de usar.

**Principio 4. Centrarse en la transferencia de información**. El software tiene que ver con la transferencia de información: de una base de datos a un usuario final, de un sistema heredado a una webapp, de un usuario final a una interfaz gráfica de usuario (GUI, por sus siglas en inglés), de un sistema operativo a una aplicación, de un componente de software a otro… la lista es casi interminable. En todos los casos, la información fluye a través de una interfaz, y como consecuencia hay posibilidades de cometer errores, omisiones o ambigüedades. Este principio implica que debe ponerse atención especial al análisis, diseño, construcción y prueba de las interfaces.

**Principio 5. Construir software que tenga modularidad eficaz**. Cualquier sistema complejo puede dividirse en módulos (componentes), pero la buena práctica de la ingeniería de software demanda más. La modularidad debe ser eficaz. Es decir, cada módulo debe centrarse exclusivamente en un aspecto bien delimitado del sistema: debe ser cohesivo en su función o restringido en el contenido que representa. Además, los módulos deben estar interconectados en forma

**Principio 6. Buscar patrones**. El objetivo de los patrones dentro de la comunidad de software es crear un cúmulo de bibliografía que ayude a los desarrolladores de software a resolver problemas recurrentes que surgen a lo largo del desarrollo. Los patrones ayudan a crear un lenguaje compartido para comunicar perspectiva y experiencia acerca de dichos patrones y sus soluciones.

**Principio 7. Cuando sea posible, representar el problema y su solución desde varias perspectivas diferentes.** Cuando un problema y su solución se estudian desde varias perspectivas distintas, es más probable que se tenga mayor visión y que se detecten los errores y omisiones. Cada uno brinda un punto de vista diferente del problema y de sus requerimientos.

**Principio 8. Tener en mente que alguien dará mantenimiento al software**. El software será corregido en el largo plazo, cuando se descubran sus defectos, se adapte a los cambios de su ambiente y se mejore en el momento en el que los participantes pidan más capacidades.

## **Principios que guian toda la actividad estructural**

En las secciones que siguen se consideran los principios que tienen mucha relevancia para el éxito de cada actividad estructural genérica, definida como parte del proceso de software. Tan sólo son principios fundamentales planteados en un nivel más bajo de abstracción.

### **Principios de comunicación**

Muchos de los siguientes principios se aplican por igual en todas las formas de comunicación que ocurren dentro de un proyecto de software:

**Principio 1. Escuchar**. Trate de centrarse en las palabras del hablante en lugar de formular su respuesta a dichas palabras. Si algo no está claro, pregunte para aclararlo, pero evite las interrupciones constantes. Si una persona habla, nunca parezca usted beligerante en sus palabras o actos.

**Principio 2. Antes de comunicarse, prepararse**. Dedique algún tiempo a entender el problema antes de reunirse con otras personas. Si es necesario, haga algunas investigaciones para entender el vocabulario propio del negocio. Si tiene la responsabilidad de conducir la reunión, prepare una agenda antes de que ésta tenga lugar.

**Principio 3. Alguien debe facilitar la actividad.** Toda reunión de comunicación debe tener un líder que: 1) mantenga la conversación en movimiento hacia una dirección positiva, 2) sea un mediador en cualquier conflicto que ocurra y 3) garantice que se sigan otros principios.

**Principio 4. Es mejor la comunicación cara a cara.** Pero por lo general funciona mejor cuando está presente alguna otra representación de la información relevante.

**Principio 5. Tomar notas y documentar las decisiones**. Las cosas encuentran el modo de caer en las grietas. Alguien que participe en la comunicación debe servir como “secretario “y escribir todos los temas y decisiones importantes.

**Principio 6. Perseguir la colaboración.** La colaboración y el consenso ocurren cuando el conocimiento colectivo de los miembros del equipo se utiliza para describir funciones o características del producto o sistema.

**Principio 7. Permanecer centrado; hacer módulos con la discusión.** Entre más personas participen en cualquier comunicación, más probable es que la conversación salte de un tema a otro. El facilitador debe formar módulos de conversación para abandonar un tema sólo después de que se haya resuelto (sin embargo, considere el principio 9).

**Principio 8. Si algo no está claro, hacer un dibujo.** La comunicación verbal tiene sus límites. Con frecuencia, un esquema o dibujo arroja claridad cuando las palabras no bastan para hacer el trabajo.

**Principio 9. a) Una vez que se acuerde algo, avanzar. b) Si no es posible ponerse de acuerdo en algo, avanzar. c) Si una característica o función no está clara o no puede aclararse en el momento, avanzar.** La comunicación, como cualquier actividad de ingeniería de software, requiere tiempo. En vez de hacer iteraciones sin fin, las personas que participan deben reconocer que hay muchos temas que requieren análisis y que “avanzar” es a veces la mejor forma de tener agilidad en la comunicación.

**Principio 10. La negociación no es un concurso o un juego.** Funciona mejor cuando las dos partes ganan. Hay muchas circunstancias en las que usted y otros participantes deben negociar funciones y características, prioridades y fechas de entrega.

### **Principios de planeacion**

La planeación debe ser tomada con moderación, suficiente para que dé una guía útil al equipo, ni más ni menos. Sin importar el rigor con el que se haga la planeación, siempre se aplican los principios siguientes:

**Principio 1. Entender el alcance del proyecto**. Es imposible usar el mapa si no se sabe a dónde se va. El alcance da un destino al equipo de software.

**Principio 2. Involucrar en la actividad de planeación a los participantes del software.** Los participantes definen las prioridades y establecen las restricciones del proyecto. Para incluir estas realidades, es frecuente que los ingenieros de software deban negociar la orden de entrega, los plazos y otros asuntos relacionados con el proyecto.

**Principio 3. Reconocer que la planeación es iterativa**. Un plan para el proyecto nunca está grabado en piedra. Para cuando el trabajo comience, es muy probable que las cosas hayan cambiado. En consecuencia, el plan deberá ajustarse para incluir dichos cambios.

**Principio 4. Estimar con base en lo que se sabe.** El objetivo de la estimación es obtener un índice del esfuerzo, costo y duración de las tareas, con base en la comprensión que tenga el equipo sobre el trabajo que va a realizar.

**Principio 5. Al definir el plan, tomar en cuenta los riesgos.** Si ha identificado riesgos que tendrían un efecto grande y es muy probable que ocurran, entonces es necesario elaborar planes de contingencia.

**Principio 6. Ser realista. Las personas no trabajan 100% todos los días.** En cualquier comunicación humana hay ruido. Las omisiones y ambigüedad son fenómenos de la vida. Los cambios ocurren. Aun los mejores ingenieros de software cometen errores.

**Principio 7. Ajustar la granularidad cuando se defina el plan.** La granularidad se refiere al nivel de detalle que se adopta cuando se desarrolla un plan. Un plan con “mucha granularidad” proporciona detalles significativos en las tareas para el trabajo que se planea, en incrementos durante un periodo relativamente corto. Un plan con “poca granularidad” da tareas más amplias para el trabajo que se planea, para plazos más largos.

**Principio 8. Definir cómo se trata de asegurar la calidad**. El plan debe identificar la forma en la que el equipo de software busca asegurar la calidad. Si se realizan revisiones técnicas,3 deben programarse. Si durante la construcción va a utilizarse programación por parejas, debe definirse en forma explícita en el plan.

**Principio 9. Describir cómo se busca manejar el cambio**. Aun la mejor planeación puede ser anulada por el cambio sin control. Debe identificarse la forma en la que van a recibirse los cambios a medida que avanza el trabajo de la ingeniería de software.

**Principio 10. Dar seguimiento al plan con frecuencia y hacer los ajustes que se requieran.** Los proyectos de software se atrasan respecto de su programación. Por tanto, tiene sentido evaluar diariamente el avance, en busca de áreas y situaciones problemáticas en las que las actividades programadas no se apeguen al avance real. Cuando se detecten desviaciones, hay que ajustar el plan en consecuencia.

### **Principios de modelado**

Un modelo debe ser capaz de representar la información que el software transforma, la arquitectura y las funciones que permiten que esto ocurra, las características que desean los usuarios y el comportamiento del sistema mientras la transformación tiene lugar. El modelado tiene los siguientes principios:

**Principio 1. El equipo de software tiene como objetivo principal elaborar software, no crear modelos**. Agilidad significa entregar software al cliente de la manera más rápida posible. Los modelos que contribuyan a esto son benéficos, pero deben evitarse aquellos que hagan lento el proceso o que den poca perspectiva.

**Principio 2. Viajar ligero, no crear más modelos de los necesarios.** Todo modelo que se cree debe actualizarse si ocurren cambios. Más importante aún es que todo modelo nuevo exige tiempo, que de otra manera se destinaría a la construcción

**Principio 3. Tratar de producir el modelo más sencillo que describa al problema o al software.** No construya software en demasíam. Al mantener sencillos los modelos, el software resultante también lo será. El resultado es que se tendrá un software fácil de integrar, de probar y de. Además, los modelos sencillos son más fáciles de entender y criticar por parte de los miembros del equipo, lo que da como resultado un formato funcional de retroalimentación que optimiza el resultado final.

**Principio 4. Construir modelos susceptibles al cambio.** Suponga que sus modelos cambiarán, pero vigile que esta suposición no lo haga descuidado. Por ejemplo, como los requerimientos se modificarán, hay una tendencia a ignorar los modelos. ¿Por qué? Porque se sabe que de todos modos cambiarán. El problema con esta actitud es que, sin un modelo razonablemente completo de los requerimientos, se creará un diseño que de manera invariable carecerá de funciones y características importantes.

**Principio 5. Ser capaz de enunciar un propósito explícito para cada modelo que se cree.** Cada vez que cree un modelo, pregúntese por qué lo hace. Si no encuentra una razón sólida para la existencia del modelo, no pierda tiempo en él.

**Principio 6. Adaptar los modelos que se desarrollan al sistema en cuestión.** Tal vez sea necesario adaptar a la aplicación la notación del modelo o las reglas; por ejemplo, una aplicación de juego de video quizá requiera una técnica de modelado distinta que el software incrustado que controla el motor de un automóvil en tiempo real.

**Principio 7. Tratar de construir modelos útiles, pero olvidarse de elaborar modelos perfectos**. Cuando un ingeniero de software construye modelos de requerimientos y diseño, alcanza un punto de rendimientos decrecientes. Es decir, el esfuerzo requerido para terminar por completo el modelo y hacerlo internamente consistente deja de beneficiarse por tener dichas propiedades. El modelado debe hacerse con la mirada puesta en las siguientes etapas de la ingeniería de software.

**Principio 8. No ser dogmático respecto de la sintaxis del modelo.** Si se tiene éxito para comunicar contenido, la representación es secundaria. Aunque cada miembro del equipo de software debe tratar de usar una notación consistente durante el modelado, la característica más importante del modelo es comunicar información que permita la realización de la siguiente tarea de ingeniería. Si un modelo tiene éxito en hacer esto, es perdonable la sintaxis incorrecta.

**Principio 9. Si su instinto dice que un modelo no es el correcto a pesar de que se vea bien en el papel, hay razones para estar preocupado.** Si usted es un ingeniero de software experimentado, confíe en su instinto. El trabajo de software enseña muchas lecciones, algunas en el nivel del inconsciente. Si algo le dice que un modelo de diseño está destinado a fracasar, hay razones para dedicar más tiempo a su estudio o a desarrollar otro distinto.

**Principio 10. Obtener retroalimentación tan pronto como sea posible.** Todo modelo debe ser revisado por los miembros del equipo**.** El objetivo de estas revisiones es obtener retroalimentación para utilizarla a fin de corregir los errores de modelado, cambiar las interpretaciones equivocadas y agregar las características o funciones omitidas inadvertidamente.

### **Requerimientos de los principios de modelado**

Cada método de análisis tiene un punto de vista único. Sin embargo, todos están relacionados por ciertos principios operacionales:

**Principio 1. Debe representarse y entenderse el dominio de información de un problema.** El dominio de información incluye los datos que fluyen hacia el sistema, los datos que fluyen fuera del sistema y los almacenamientos de datos que recaban y organizan objetos persistentes de datos.

**Principio 2. Deben definirse las funciones que realizará el software.** Las funciones del software dan un beneficio directo a los usuarios finales y también brindan apoyo interno para las características que son visibles para aquéllos. Algunas funciones transforman los datos que fluyen hacia el sistema. En otros casos, las funciones activan algún nivel de control sobre el procesamiento interno del software o sobre los elementos externos del sistema.

**Principio 3. Debe representarse el comportamiento del software**. El comportamiento del software de computadora está determinado por su interacción con el ambiente externo. Las entradas que dan los usuarios finales, el control de los datos efectuado por un sistema externo o la vigilancia de datos reunidos en una red son el motivo por el que el software se comporta en una forma específica.

**Principio 4. Los modelos que representen información, función y comportamiento deben dividirse de manera que revelen los detalles en forma estratificada**. El modelado de los requerimientos es el primer paso para resolver un problema de ingeniería de software. Eso permite entender mejor el problema y proporciona una base para la solución. Los problemas complejos son difíciles de resolver por completo. Por esta razón, debe usarse la estrategia de divide y vencerás. Este concepto se llama partición o separación de entidades, y es una estrategia clave en el modelado de requerimientos.

**Principio 5. El trabajo de análisis debe avanzar de la información esencial hacia la implementación en detalle.** El modelado de requerimientos comienza con la descripción del problema desde la perspectiva del usuario final. Se describe la “esencia” del problema sin considerar la forma en la que se implementará la solución.

### **Principios del modelo de diseño**

El modelo de diseño aplica los siguientes principios:

**Principio 1. El diseño debe poderse rastrear hasta el modelo de requerimientos.** El modelo de requerimientos describe el dominio de información del problema, las funciones visibles para el usuario, el comportamiento del sistema y un conjunto de clases de requerimientos que agrupa los objetos del negocio con los métodos que les dan servicio. El modelo de diseño traduce esta información en una arquitectura, un conjunto de subsistemas que implementan las funciones principales y un conjunto de componentes que son la realización de las clases de requerimientos.

**Principio 2. Siempre tomar en cuenta la arquitectura del sistema que se va a construir.** La arquitectura del software es el esqueleto del sistema que se va a construir. Afecta interfaces, estructuras de datos, flujo de control y comportamiento del programa, así como la manera en la que se realizarán las pruebas, la susceptibilidad del sistema resultante a recibir mantenimiento y mucho más.

**Principio 3. El diseño de los datos es tan importante como el de las funciones de procesamiento.** El diseño de los datos es un elemento esencial del diseño de la arquitectura. La forma en la que los objetos de datos se elaboran dentro del diseño no puede dejarse al azar. Un diseño de datos bien estructurado ayuda a simplificar el flujo del programa, hace más fácil el diseño e implementación de componentes de software y más eficiente el procesamiento conjunto.

**Principio 4. Las interfaces deben diseñarse con cuidado.** La manera en la que los datos fluyen entre los componentes de un sistema tiene mucho que ver con la eficiencia del procesamiento, la propagación del error y la simplicidad del diseño. Una interfaz bien diseñada hace que la integración sea más fácil y ayuda a quien la somete a prueba a validar las funciones componentes.

**Principio 5. El diseño de la interfaz de usuario debe ajustarse a las necesidades del usuario final.** Sin embargo, en todo caso debe resaltar la facilidad de uso. La interfaz de usuario es la manifestación visible del software. No importa cuán sofisticadas sean sus funciones internas, ni lo incluyentes que sean sus estructuras de datos, ni lo bien diseñada que esté su arquitectura, un mal diseño de la interfaz con frecuencia conduce a la percepción de que el software es “malo”.

**Principio 6. El diseño en el nivel de componentes debe tener independencia funcional.** La independencia funcional es una medida de la “mentalidad única” de un componente de software. La funcionalidad que entrega un componente debe ser cohesiva, es decir, debe centrarse en una y sólo una función o subfunción.

**Principio 7. Los componentes deben estar acoplados con holgura entre sí y con el ambiente externo.** El acoplamiento se logra de muchos modos: con una interfaz de componente, con mensajería, por medio de datos globales, etc. A medida que se incrementa el nivel de acoplamiento, también aumenta la probabilidad de propagación del error y disminuye la facilidad general de dar mantenimiento al software. Entonces, el acoplamiento de componentes debe mantenerse tan bajo como sea razonable.

**Principio 8. Las representaciones del diseño deben entenderse con facilidad.** El propósito del diseño es comunicar información a los profesionales que generarán el código, a los que probarán el software y a otros que le darán mantenimiento en el futuro. Si el diseño es difícil de entender, no servirá como medio de comunicación eficaz.

**Principio 9. El diseño debe desarrollarse en forma iterativa. El diseñador debe buscar más sencillez en cada iteración.** Igual que ocurre con casi todas las actividades creativas, el diseño ocurre de manera iterativa. Las primeras iteraciones sirven para mejorar el diseño y corregir errores, pero las posteriores deben buscar un diseño tan sencillo como sea posible.

## **Principios de construccion**

La actividad de construcción incluye un conjunto de tareas de codificación y pruebas que lleva a un software operativo listo para entregarse al cliente o usuario final. En el trabajo de ingeniería de software moderna, la codificación puede ser:

La creación directa de lenguaje de programación en código fuente (por ejemplo, Java),

La generación automática de código fuente que usa una representación intermedia parecida al diseño del componente que se va a construir.

La generación automática de código ejecutable que utiliza un “lenguaje de programación de cuarta generación” (por ejemplo, Visual C++).

Las pruebas dirigen su atención inicial al componente, y con frecuencia se denomina prueba unitaria. Otros niveles de pruebas incluyen

De integración (realizadas mientras el sistema está en construcción),

De validación, que evalúan si los requerimientos se han satisfecho para todo el sistema (o incremento de software).

De aceptación, que efectúa el cliente en un esfuerzo por utilizar todas las características y funciones requeridas.

Los siguientes principios y conceptos son aplicables a la codificación y prueba:

### **Principios de codificacion**

Los principios que guían el trabajo de codificación se relacionan de cerca con el estilo, lenguajes y métodos de programación. Sin embargo, puede enunciarse cierto número de principios fundamentales:

#### **Principios de preparacion**

Antes de escribir una sola línea de código, asegúrese de:

Entender el problema que se trata de resolver.

Comprender los principios y conceptos básicos del diseño.

Elegir un lenguaje de programación que satisfaga las necesidades del software que se va a elaborar y el ambiente en el que operará.

Seleccionar un ambiente de programación que disponga de herramientas que hagan más fácil su trabajo.

Crear un conjunto de pruebas unitarias que se aplicarán una vez que se haya terminado el componente a codificar.

#### **Principios de programacion**

Cuando comience a escribir código, asegúrese de:

* Restringir sus algoritmos por medio del uso de programación estructurada
* Tomar en consideración el uso de programación por parejas.
* Seleccionar estructuras de datos que satisfagan las necesidades del diseño.
* Entender la arquitectura del software y crear interfaces que son congruentes con ella.
* Mantener la lógica condicional tan sencilla como sea posible.
* Crear lazos anidados en forma tal que se puedan probar con facilidad.
* Seleccionar nombres significativos para las variables y seguir otros estándares locales de codificación.
* Escribir código que se documente a sí mismo.
* Crear una imagen visual (por ejemplo, líneas con sangría y en blanco) que ayude a entender.

#### **Principios de validacion**

Una vez que haya terminado su primer intento de codificación, asegúrese de:

• Realizar el recorrido del código cuando sea apropiado.

• Llevar a cabo pruebas unitarias y corregir los errores que se detecten.

• Rediseñar el código.

### **Principios de prueba**

En un libro clásico sobre las pruebas de software, Glen Myers enuncia algunas reglas que sirven bien como objetivos de prueba:

* La prueba es el proceso que ejecuta un programa con objeto de encontrar un error.
* Un buen caso de prueba es el que tiene alta probabilidad de encontrar un error que no se ha detectado hasta el momento.
* Una prueba exitosa es la que descubre un error no detectado hasta el momento.

El objetivo es diseñar pruebas que detecten de manera sistemática diferentes clases de errores, y hacerlo con el mínimo tiempo y esfuerzo. Pero las pruebas no pueden demostrar la inexistencia de errores y defectos; sólo demuestran que hay errores y defectos. Es importante recordar esto cuando se efectúe una prueba, la cual sigue los siguientes principios:

* **Principio 1. Todas las pruebas deben poder rastrearse hasta los requerimientos del cliente.** El objetivo de las pruebas de software es descubrir errores. Entonces, los defectos más severos son aquellos que hacen que el programa no cumpla sus requerimientos.
* **Principio 2. Las pruebas deben planearse mucho antes de que den comienzo.** La planeación de las pruebas comienza tan pronto como se termina el modelo de requerimientos. La definición detallada de casos de prueba principia apenas se ha concluido el modelo de diseño. Por tanto, todas las pruebas pueden planearse y diseñarse antes de generar cualquier código.
* **Principio 3. El principio de Pareto se aplica a las pruebas de software.** En este contexto, el principio de Pareto implica que 80% de todos los errores no detectados durante las pruebas se relacionan con 20% de todos los componentes de programas. Por supuesto, el problema es aislar los componentes sospechosos y probarlos a fondo.
* **Principio 4. Las pruebas deben comenzar “en lo pequeño” y avanzar hacia “lo grande”.** Las primeras pruebas planeadas y ejecutadas por lo general se centran en componentes individuales. Conforme avanzan las pruebas, la atención cambia en un intento por encontrar errores en grupos integrados de componentes y, en última instancia, en todo el sistema.
* **Principio 5. No son posibles las pruebas exhaustivas.** Hasta para un programa de tamaño moderado, el número de permutaciones de las rutas es demasiado grande. Por esta razón, durante una prueba es imposible ejecutar todas las combinaciones de rutas. Sin embargo, es posible cubrir en forma adecuada la lógica del programa y asegurar que se han probado todas las condiciones en el nivel de componentes.

### **Principios de despliegue**

La entrega de un incremento de software representa un punto de referencia importante para cualquier proyecto de software. Cuando el equipo se prepara para entregar un incremento, deben seguirse ciertos principios clave:

* **Principio 1. Deben manejarse las expectativas de los clientes.** Con demasiada frecuencia, el cliente espera más de lo que el equipo ha prometido entregar, y la desilusión llega de inmediato. Esto da como resultado que la retroalimentación no sea productiva y arruine la moral del equipo. El ingeniero de software debe tener cuidado con el envío de mensajes conflictivos al cliente (por ejemplo, prometer más de lo que puede entregarse de manera razonable en el plazo previsto, o entregar más de lo que se prometió en un incremento de software y para el siguiente entregar menos).
* **Principio 2. Debe ensamblarse y probarse el paquete completo que se entregará.** Debe ensamblarse en un CD-ROM u otro medio (incluso descargas desde web) todo el software ejecutable, archivos de datos de apoyo, documentos de ayuda y otra información relevante, para después hacer una prueba beta exhaustiva con usuarios reales. Todos los scripts de instalación y otras características de operación deben ejecutarse por completo en tantas configuraciones diferentes de cómputo como sea posible (por ejemplo, hardware, sistemas operativos, equipos periféricos, configuraciones de red, etcétera).
* **Principio 3. Antes de entregar el software, debe establecerse un régimen de apoyo**. Un usuario final espera respuesta e información exacta cuando surja una pregunta o problema. Si el apoyo es ad hoc, o, peor aún, no existe, el cliente quedará insatisfecho de inmediato. El apoyo debe planearse, los materiales respectivos deben prepararse y los mecanismos apropiados de registro deben establecerse a fin de que el equipo de software realice una evaluación categórica de las clases de apoyo solicitado.
* **Principio 4. Se deben proporcionar a los usuarios finales materiales de aprendizaje apropiados.** El equipo de software entrega algo más que el software en sí. Deben desarrollarse materiales de capacitación apropiados; es necesario proveer lineamientos para solución de problemas y, cuando sea necesario, debe publicarse.
* **Principio 5. El software defectuoso debe corregirse primero y después entregarse.** Cuando el tiempo apremia, algunas organizaciones de software entregan incrementos de baja calidad con la advertencia de que los errores “se corregirán en la siguiente entrega”. Esto es un error. Hay un adagio en el negocio del software que dice así: “Los clientes olvidarán pronto que entregaste un producto de alta calidad, pero nunca olvidarán los problemas que les causó un producto de mala calidad. El software se los recuerda cada día.”

## **Que es calidad**

En un nivel algo pragmático, David Garvin, de Harvard Business School, sugiere que “la calidad es un concepto complejo y de facetas múltiples” que puede describirse desde cinco diferentes puntos de vista. El punto de vista trascendental dice que la calidad es algo que se reconoce de inmediato, pero que no es posible definir explícitamente. El punto de vista del usuario concibe la calidad en términos de las metas específicas del usuario final. Si un producto las satisface, tiene calidad. El punto de vista del fabricante la define en términos de las especificaciones originales del producto. Si éste las cumple, tiene calidad. El punto de vista del producto sugiere que la calidad tiene que ver con las características inherentes (funciones y características) de un producto. Por último, el punto de vista basado en el valor la mide de acuerdo con lo que un cliente está dispuesto a pagar por un producto. En realidad, la calidad incluye todo esto y más.

La calidad se refiere a la capacidad que posee un objeto para satisfacer necesidades implícitas o explícitas según un parámetro, un cumplimiento de requisitos de cualidad.

### **Calidad del software**

En el sentido más general se define como: Proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan.

**Factores de la calidad de McCall**

McCall, Richards y Walters proponen una clasificación útil de los factores que afectan la calidad del software. se centran en tres aspectos importantes del producto de software: sus características operativas, su capacidad de ser modificado y su adaptabilidad a nuevos ambientes. McCall, hace las descripciones siguientes:

**Corrección**. Grado en el que un programa satisface sus especificaciones y en el que cumple con los objetivos de la misión del cliente.

**Confiabilidad.** Grado en el que se espera que un programa cumpla con su función y con la precisión requerida

**Eficiencia.** Cantidad de recursos de cómputo y de código requeridos por un programa para llevar a cabo su función.

**Integridad.** Grado en el que es posible controlar el acceso de personas no autorizadas al software o a los datos.

**Usabilidad**. Esfuerzo que se requiere para aprender, operar, preparar las entradas e interpretar las salidas de un programa.

**Facilidad de recibir mantenimiento**. Esfuerzo requerido para detectar y corregir un error en un programa.

**Flexibilidad.** Esfuerzo necesario para modificar un programa que ya opera.

**Susceptibilidad de someterse a pruebas.** Esfuerzo que se requiere para probar un programa a fin de garantizar que realiza la función que se pretende.

**Portabilidad.** Esfuerzo que se necesita para transferir el programa de un ambiente de sistema de hardware o software a otro.

**Reusabilidad.** Grado en el que un programa (o partes de uno) pueden volverse a utilizar en otras aplicaciones.

**Interoperabilidad.** Esfuerzo requerido para acoplar un sistema con otro.

### **El dilema de la calidad del software**

Si produce un sistema de software de mala calidad, usted pierde porque nadie lo querrá comprar. Por otro lado, si dedica un tiempo infinito, demasiado esfuerzo y enormes sumas de dinero para obtener un elemento perfecto de software, entonces tomará tanto tiempo terminarlo y será tan caro de producir que de todos modos quedará fuera del negocio. En cualquier caso, habrá perdido la ventana de mercado, o simplemente habrá agotado sus recursos.

Es correcto afirmar que los ingenieros de software deben tratar de producir sistemas de alta calidad. Es mejor aplicar buenas prácticas al intento de lograrlo. Pero la situación descrita por Meyer proviene de la vida real y representa un dilema incluso para las mejores organizaciones de ingeniería de software.

#### **Software “suficientemente bueno”**

Exactamente, ¿qué significa “suficientemente bueno”? El software suficientemente bueno contiene las funciones y características de alta calidad que desean los usuarios, pero al mismo tiempo tiene otras más oscuras y especializadas que contienen errores conocidos. El vendedor de software espera que la gran mayoría de usuarios finales perdone los errores gracias a que estén muy contentos con la funcionalidad de la aplicación.

#### **El costo de la calidad**

El costo de la calidad incluye todos los costos en los que se incurre al buscar la calidad o al realizar actividades relacionadas con ella y los costos posteriores de la falta de calidad. Para entender estos costos, una organización debe contar con unidades de medición que provean el fundamento del costo actual de la calidad, que identifiquen las oportunidades para reducir dichos costos y que den una base normalizada de comparación. El costo de la calidad puede dividirse en los costos que están asociados con la prevención, la evaluación y la falla.

#### **Riesgos**

“La gente basa su trabajo, confort, seguridad, entretenimiento, decisiones y su propia vida, en software de cómputo. Más vale que esté bien hecho”. La implicación es que el software de mala calidad aumenta los riesgos tanto para el desarrollador como para el usuario final. En la subsección anterior se analizó uno de dichos riesgos (el costo). Pero lo perjudicial de las aplicaciones mal diseñadas e implementadas no siempre se mide en dólares y tiempo. La mala calidad conlleva riesgos, algunos muy serios.

### **Lograr la calidad del software**

La calidad del software no sólo se ve. Es el resultado de la buena administración del proyecto y de una correcta práctica de la ingeniería de software. La administración y práctica se aplican en el contexto de cuatro actividades principales que ayudan al equipo de software a lograr una alta calidad en éste:

#### **Metodos de la Ingenieria de Software**

Si espera construir software de alta calidad, debe entender el problema que se quiere resolver. También debe ser capaz de crear un diseño que esté de acuerdo con el problema y que al mismo tiempo tenga características que lleven al software a las dimensiones y factores de calidad.

#### **Tecnicas de administracion de proyectos**

Las implicaciones son claras: si un gerente de proyecto usa estimaciones para verificar que las fechas pueden cumplirse, se comprenden las dependencias de las actividades programadas y el equipo resiste la tentación de usar atajos, la planeación del riesgo se lleva a cabo de manera que los problemas no alienten el caos, entonces la calidad del software se verá influida de manera positiva.

#### **Control de calidad**

El control de calidad incluye un conjunto de acciones de ingeniería de software que ayudan a asegurar que todo producto del trabajo cumpla sus metas de calidad. Los modelos se revisan para garantizar que están completos y que son consistentes. El código se inspecciona con objeto de descubrir y corregir errores antes de que comiencen las pruebas. Se aplica una serie de etapas de prueba para detectar los errores en procesamiento lógico, manipulación de datos y comunicación con la interfaz.

#### **Aseguramiento de la calidad**

El aseguramiento de la calidad establece la infraestructura de apoyo a los métodos sólidos de la ingeniería de software, la administración racional de proyectos y las acciones de control de calidad, todo de importancia crucial si se trata de elaborar software de alta calidad. Además, el aseguramiento de la calidad consiste en un conjunto de funciones de auditoría y reportes para evaluar la eficacia y completitud de las acciones de control de calidad. La meta del aseguramiento de la calidad es proveer al equipo administrativo y técnico los datos necesarios para mantenerlo informado sobre la calidad del producto, con lo que obtiene perspectiva y confianza en que las acciones necesarias para lograr la calidad del producto funcionan.

## **Elementos de aseguramiento de la calidad de software**

El aseguramiento de la calidad del software incluye un rango amplio de preocupaciones y actividades que se centran en la administración de la calidad del software. Éstas se resumen como sigue:

**Estándares.** El IEEE, ISO y otras organizaciones que establecen estándares han producido una amplia variedad de ellos para ingeniería de software y documentos relacionados. Los estándares los adopta de manera voluntaria una organización de software o los impone el cliente u otros participantes

**Revisiones y auditorías.** Las revisiones técnicas son una actividad del control de calidad que realizan ingenieros de software para otros ingenieros de software. Su objetivo es detectar errores. Las auditorías son un tipo de revisión efectuada por personal de ACS con objeto de garantizar que se sigan los lineamientos de calidad en el trabajo de la ingeniería de software

**Pruebas.** Las pruebas del software son una función del control de calidad que tiene un objetivo principal: detectar errores. El trabajo del ACS es garantizar que las pruebas se planeen en forma apropiada y que se realicen con eficiencia, de modo que la probabilidad de que logren su objetivo principal sea máxima.

**Colección y análisis de los errores**. La única manera de mejorar es medir cómo se está haciendo algo. El ACS reúne y analiza errores y datos acerca de los defectos para entender mejor cómo se cometen los errores y qué actividades de la ingeniería de software son más apropiadas para eliminarlos.

**Administración del cambio.** El cambio es uno de los aspectos que más irrumpe en cualquier proyecto de software. Si no se administra en forma adecuada, lleva a la confusión y ésta casi siempre genera mala calidad. El ACS asegura que se hayan instituido prácticas adecuadas de administración del cambio.

**Educación.** Toda organización de software quiere mejorar sus prácticas de ingeniería de software. Un contribuyente clave de la mejora es la educación de los ingenieros de software, de sus gerentes y de otros participantes. La organización de ACS lleva el liderazgo en la mejora del proceso de softwarey es clave para proponer y patrocinar programas educativos.

**Administración de los proveedores.** Son tres las categorías de software que se adquieren a proveedores externos: paquetes contenidos en una caja (por ejemplo, Office, de Microsoft); un shell personalizado, que da una estructura básica, tipo esqueleto, que se adapta de manera única a las necesidades del comprador; y software contratado, que se diseña y construye especialmente a partir de especificaciones provistas por la organización cliente. El trabajo de la organización de ACS es garantizar que se obtenga software de alta calidad a partir de las sugerencias de prácticas específicas de calidad que el proveedor debe seguir y de la incorporación de cláusulas de calidad como parte de cualquier contrato con un proveedor externo.

**Administración de la seguridad.** Con el aumento de los delitos cibernéticos y de las nuevas regulaciones gubernamentales respecto de la privacidad, toda organización de software debe instituir políticas para proteger los datos en todos los niveles, establecer cortafuegos de protección para las webapps y asegurar que el software no va a ser vulnerado internamente. El ACS garantiza que, para lograr la seguridad del software, se utilicen el proceso y la tecnología apropiados.

**Seguridad.** Debido a que el software casi siempre es un componente crucial de los sistemas humanos, la consecuencia de defectos ocultos puede ser catastrófica. El ACS es responsable de evaluar el efecto de las fallas del software y de dar los pasos que se requieren para disminuir el riesgo.

**Administración de riesgos.** Aunque el análisis y la mitigación de riesgos es asunto de los ingenieros de software, la organización del ACS garantiza que las actividades de administración de riesgos se efectúen en forma apropiada y que se establezcan planes de contingencia relacionados con los riesgos.

### **Seguridad de Software**

La seguridad del software es una actividad del aseguramiento del software que se centra en la identificación y evaluación de los peligros potenciales que podrían afectarlo negativamente y que podrían ocasionar que falle todo el sistema. Si los peligros se identifican al principio del proceso del software, las características de su diseño se especifican de modo que los eliminen o controlen.

## **Administracion del Proyecto**

La administración del proyecto involucra planificación, monitoreo y control del personal, procesos y acciones que ocurren conforme el software evoluciona desde un concepto preliminar hasta su despliegue operativo completo.

Un ingeniero del software administra sus actividades cotidianas, planifica, monitorea y controla las tareas técnicas. Los gerentes de proyecto planifican, monitorean y controlan el trabajo de un equipo de ingenieros de software. Los gerentes ejecutivos coordinan la interfaz entre la empresa y los profesionales del software.

Construir software de computadora es una labor compleja, particularmente si involucra a muchas personas que trabajan durante un tiempo relativamente largo. Por eso es necesario administrar los proyectos de software.

### **El espectro administrativo**

La administración efectiva de un proyecto de software se enfoca en las cuatro P: personal, producto, proceso y proyecto. El orden no es arbitrario. El gerente que olvida que el trabajo de la ingeniería del software es una empresa intensamente humana nunca triunfará en la administración del proyecto.

### **Definicion de: El Personal**

El “factor humano” es tan importante que el Software Engineering Institute desarrolló un Modelo de madurez de capacidades del personal (People-CMM, por sus siglas en inglés), en reconocimiento al hecho de que “toda organización requiere mejorar continuamente su habilidad para atraer, desarrollar, motivar, organizar y conservar la fuerza de trabajo necesaria a fin de lograr sus objetivos empresariales estratégicos”

El People-CMM define las siguientes áreas prácticas clave para el personal de software: plantilla, comunicación y coordinación, ambiente de trabajo, desempeño administrativo, capacitación, compensación, análisis y desarrollo de competencias, desarrollo profesional, desarrollo de grupo de trabajo y desarrollo de equipo/cultura, entre otros.

#### **Participantes**

El proceso de software (y todo proyecto de software) está poblado de participantes, quienes pueden organizarse en alguna de las siguientes áreas:

* **Gerentes ejecutivos**, quienes definen los temas empresariales que con frecuencia tienen una influencia significativa sobre el proyecto.
* **Gerentes de proyecto** (técnicos), quienes deben planificar, motivar, organizar y controlar a los profesionales que hacen el trabajo de software.
* **Profesionales** que aportan las habilidades técnicas que se necesitan para someter a ingeniería un producto o aplicación.
* **Clientes** que especifican los requerimientos para el software que se va a fabricar, así como otros participantes que tienen un interés periférico en el resultado.
* **Usuarios finales**, quienes interactúan con el software una vez que se libera para su uso productivo.

#### **Lideres de Equipo**

La administración del proyecto es una actividad que implica mucho trato con la gente; por esta razón, los profesionales competentes tienen con frecuencia pobre desempeño como líderes de equipo.

En un excelente libro acerca del liderazgo técnico, Jerry Weinberg sugiere un modelo MOI de liderazgo:

* **Motivación**. Habilidad para alentar (mediante “empuje o jalón”) al personal técnico a producirá su máxima capacidad.
* **Organización**. Habilidad para moldear los procesos existentes (o inventar nuevos) que permitirán que el concepto inicial se traduzca en un producto final.
* **Ideas o innovación.** Habilidad para alentar a las personas a crear y sentirse creativas, aun cuando deban trabajar dentro de fronteras establecidas para un producto o aplicación de software particular.

Otra visión de las características que definen a un gerente de proyecto eficaz enfatiza cuatro rasgos clave:

**Resolución de problemas.** Un gerente de proyecto de software eficaz puede diagnosticar los conflictos técnicos y organizativos que son más relevantes, estructura sistemáticamente una solución o motiva adecuadamente a otros profesionales para desarrollarla, aplica lecciones aprendidas de proyectos pasados a situaciones nuevas y sigue siendo suficientemente flexible para cambiar de dirección si los intentos por resolver el problema son infructuosos.

**Identidad administrativa.** Un buen gerente de proyecto debe hacerse cargo del mismo. Debe tener la confianza para asumir el control cuando sea necesario y asegurarse de permitir que el buen personal técnico siga sus instintos.

**Logro.** Un gerente competente debe recompensar la iniciativa y el logro para optimizar la productividad de un equipo de proyecto. Debe demostrar mediante sus acciones que no se castigará del correr riesgos de manera controlada.

**Influencia y construcción del equipo.** Un gerente de proyecto eficaz debe poder “leer” a la gente; debe poder comprender las señales verbales y no verbales, y reaccionar ante las necesidades de las personas que envían estas señales. El gerente debe permanecer bajo control en situaciones de alto estrés.

#### **Equipo de Software**

La “mejor” estructura de equipo depende del estilo administrativo de la organización, del número de personas que formarán el equipo y de sus niveles de habilidad, así como de la dificultad global del problema. Mantei describe siete factores de proyecto que deben considerarse cuando se planee la estructura de los equipos de ingeniería de software:

* Dificultad del problema que se va a resolver.
* “Tamaño” del programa resultante en líneas de código o puntos de función.
* Tiempo que el equipo permanecerá unido (vida del equipo).
* Grado en el que puede dividirse en módulos el problema.
* Calidad y confiabilidad requeridas por el sistema que se va a construir.
* Rigidez de la fecha de entrega.
* Grado de sociabilidad (comunicación) requerido para el proyecto.

### **Definicion de: El Producto**

Antes de poder planear un proyecto, deben establecerse los objetivos y el ámbito del producto, considerarse soluciones alternativas e identificar las restricciones técnicas y administrativas. Sin esta información, es imposible definir estimaciones razonables (y precisas) del costo, una valoración efectiva del riesgo, una descomposición realista de las tareas del proyecto y un calendario de proyecto manejable que proporcione en cada momento un indicio significativo del progreso.

#### **Ambito del software**

La primera actividad en la administración del proyecto de software es determinar el ámbito del software, que se define al responder las siguientes preguntas:

**Contexto**. ¿Cómo encaja en un sistema, producto o contexto empresarial más grande el software que se va a construir y qué restricciones se imponen como resultado del contexto?

**Objetivos de información.** ¿Qué objetos de datos visibles para el cliente se producen como salida del software? ¿Qué objetos de datos se requieren como entrada?

**Función y desempeño.** ¿Qué función realiza el software para transformar los datos de entrada en salida? ¿Existe alguna característica de desempeño especial que deba abordarse?

#### **Descomposicion del problema**

La descomposición del problema, en ocasiones llamada división o elaboración del problema, es una actividad que se asienta en el centro del análisis de requerimientos del software. Durante la actividad de determinación del ámbito, no se hacen intentos por descomponer completamente el problema. En vez de ello, la descomposición se aplica en dos áreas principales: 1) la funcionalidad y el contenido que deben entregarse y 2) el proceso que se usará para entregarlo.

### **Definicion de: El Proceso**

Un proceso de software proporciona el marco conceptual desde el cual puede establecerse un plan completo para el desarrollo de software. Un pequeño número de actividades de marco conceptual se aplica a todos los proyectos de software, sin importar su tamaño o complejidad. Algunos conjuntos de diferentes tareas (tareas, hitos, productos operativos y puntos de aseguramiento de calidad) permiten que las actividades del marco conceptual se adapten a las características del proyecto de software y a los requerimientos del equipo del proyecto.

#### **Fusion de producto y proceso**

La planificación del proyecto comienza con la fusión de producto y proceso. Cada función que se va a someter a ingeniería por parte del equipo debe pasar a través del conjunto de actividades de marco conceptual que defina la organización de software.

#### **Descomposicion del proceso**

Un equipo de software debe tener un grado significativo de flexibilidad al elegir el modelo de proceso de software que es mejor para el proyecto y las tareas de la ingeniería de software que pueblen el modelo de proceso una vez elegido. Un proyecto relativamente pequeño que sea similar a esfuerzos anteriores puede lograrse mejor al usar el enfoque secuencial lineal. Si la fecha límite es tan apretada como para que toda la funcionalidad no pueda entregarse razonablemente, puede ser mejor una estrategia incremental. De igual modo, los proyectos con otras características (por ejemplo, requerimientos de incertidumbre, tecnología innovadora, clientes difíciles, significativo potencial de reuso) conducirán a la selección de otros modelos de proceso

### **Definicion de: El Proyecto**

Los proyectos de software se planean y controlan debido a una razón principal: es la única forma conocida para manejar la complejidad. Para evitar el fracaso del proyecto, un gerente de proyecto de software y los ingenieros de software que construyan el producto deben evitar un conjunto de señales de advertencia comunes, entender los factores de éxito cruciales que conducen a una buena administración del proyecto y desarrollar un enfoque de sentido común para planificar, monitorear y controlar el proyecto.

## **Calendarizacion del Proyecto**

La calendarización del proyecto de software es una acción que distribuye el esfuerzo estimado a través de la duración planificada del proyecto, asignando el esfuerzo a tareas específicas de ingeniería del software. Sin embargo, es importante observar que el calendario evoluciona con el tiempo. Durante las primeras etapas de la planificación del proyecto se desarrolla un calendario macroscópico. Este tipo de calendario identifica las principales actividades de marco conceptual de proceso y las funciones de producto a las cuales se aplican. Conforme el proyecto avanza, cada entrada en el calendario macroscópico se desglosa en un calendario detallado. Aquí, acciones y tareas de software específicas (requeridas para lograr una actividad) se identifican y calendarizan. La calendarización para proyectos de ingeniería de software puede verse desde dos perspectivas más bien diferentes. En la primera, ya se estableció una fecha final (e irrevocable) para liberar un sistema basado en computadora. La organización de software se restringe para distribuir el esfuerzo dentro del marco temporal prescrito. La segunda visión de la calendarización del software supone que se han discutido límites cronológicos burdos, pero que la fecha final la establece la organización de ingeniería del software. El esfuerzo se distribuye para hacer mejor uso de los recursos y se define una fecha final después de un cuidadoso análisis del software.

## **Metricas del Producto**

Las métricas de producto ayudan a los ingenieros de software a obtener comprensión acerca del diseño y la construcción del software que elaboran, al enfocarse en atributos mensurables específicos de los productos de trabajo de la ingeniería del software. Siempre habrá un elemento cualitativo en la creación del software para computadoras. El problema es que la valoración cualitativa tal vez no sea suficiente. Se necesitan criterios objetivos que ayuden a guiar el diseño de datos, arquitectura, interfaces y componentes. Cuando se prueban, es necesaria la guía cuantitativa que ayuda en la selección de los casos de prueba y de sus objetivos. Las métricas de producto proporcionan una base desde donde el análisis, el diseño, la codificación y las pruebas pueden realizarse de manera más objetiva y valorarse de modo más cuantitativo.

### **Medidas, Metricas e Indicadores**

En el contexto de la ingeniería del software, una medida proporciona un indicio cuantitativo de la extensión, cantidad, dimensión, capacidad o tamaño de algún atributo de un producto o proceso. La medición es el acto de determinar una medida. El IEEE Standard Glosary of Software Engineering Terminology define métrica como “una medida cuantitativa del grado en el que un sistema, componente o proceso posee un atributo determinado”.

Un ingeniero de software recolecta medidas y desarrolla métricas de modo que se obtengan indicadores. Un indicador es una métrica o combinación de métricas que proporcionan comprensión acerca del proceso de software, el proyecto de software o el producto en sí.

### **Principios de Medicion**

El proceso de medición se puede caracterizarse mediante cinco actividades:

* Formulación. La derivación de medidas y métricas de software apropiadas para la representación del software que se está construyendo.
* Recolección. Mecanismo que se usa para acumular datos requeridos para derivar las métricas formuladas.
* Análisis. El cálculo de métricas y la aplicación de herramientas matemáticas.
* Interpretación. Evaluación de las métricas resultantes para comprender la calidad de la representación.
* Retroalimentación. Recomendaciones derivadas de la interpretación de las métricas del producto, transmitidas al equipo de software.

### **Metricas para el Modelo de Requerimientos**

El trabajo técnico en la ingeniería del software comienza con la creación del modelo de requerimientos. En esta etapa se derivan los requerimientos y se establece un cimiento para el diseño. Por tanto, son deseables métricas de producto que proporcionen comprensión acerca de la calidad del modelo de análisis.

#### **Metricas basada en Funciones**

La métrica de punto de función (PF) puede usarse de manera efectiva como medio para medir la funcionalidad que entra a un sistema.

* **Número de entradas externas (EE).** Cada entrada externa se origina de un usuario o se transmite desde otra aplicación, y proporciona distintos datos orientados a aplicación o información de control. Con frecuencia, las entradas se usan para actualizar archivos lógicos internos (ALI). Las entradas deben distinguirse de las consultas, que se cuentan por separado.
* **Número de salidas externas (SE).** Cada salida externa es datos derivados dentro de la aplicación que ofrecen información al usuario. En este contexto, salida externa se refiere a reportes, pantallas, mensajes de error, etc. Los ítems de datos individuales dentro de un reporte no se cuentan por separado.
* **Número de consultas externas (CE).** Una consulta externa se define como una entrada en línea que da como resultado la generación de alguna respuesta de software inmediata en la forma de una salida en línea (con frecuencia recuperada de un ALI).
* **Número de archivos lógicos internos** (ALI). Cada archivo lógico interno es un agrupamiento lógico de datos que reside dentro de la frontera de la aplicación y se mantiene mediante entradas externas.
* **Número de archivos de interfaz externos** (AIE). Cada archivo de interfaz externo es un agrupamiento lógico de datos que reside fuera de la aplicación, pero que proporciona información que puede usar la aplicación.

#### **Metricas para calidad de las especificacion**

Para valorar la calidad del modelo de requerimientos y la correspondiente especificación de requerimientos es necesario ver: especificidad (falta de ambigüedad), completitud, corrección, comprensibilidad, verificabilidad, consistencia interna y externa, factibilidad, concisión, rastreabilidad, modificabilidad, precisión y reusabilidad. Además, los autores observan que las especificaciones de alta calidad se almacenan electrónicamente, son ejecutables o al menos interpretables, se anotan mediante importancia relativa, son estables, tienen versión, se organizan, cuentan con referencia cruzada y se especifican en el nivel correcto de detalle.

### **Metrica para el Modelo de Diseño**

Las métricas de diseño para software de computadora, al igual que todas las demás métricas de software, no son perfectas. El debate continúa acerca de su eficacia y sobre la forma en la que deben aplicarse. Muchos expertos argumentan que se requiere más experimentación antes de poder usar las medidas de diseño, aunque el diseño sin medición es una alternativa inaceptable.

### **Metricas para diseño orientado a objetos**

En un tratamiento detallado de las métricas de software para sistemas OO, Whitmire describe nueve características distintas y mensurables de un diseño OO:

**Tamaño.** El tamaño se define en función de cuatro visiones: población, volumen, longitud y funcionalidad. La población se mide al realizar un conteo estático de entidades OO, tales como clases u operaciones. Las medidas de volumen son idénticas a las medidas de población, pero se recolectan de manera dinámica: en un instante de tiempo determinado. La longitud es una medida de una cadena de elementos de diseño interconectados, Las métricas de funcionalidad proporcionan un indicio indirecto del valor entregado al cliente por una aplicación OO.

**Complejidad.** Como el tamaño, existen muchas visiones diferentes de la complejidad del software. Whitmire ve la complejidad en términos de características estructurales al examinar cómo se relacionan mutuamente las clases de un diseño OO.

**Acoplamiento.** Las conexiones físicas entre elementos del diseño OO (por ejemplo, el número de colaboraciones entre clases o el de mensajes que pasan entre los objetos) representan el acoplamiento dentro de un sistema OO.

**Suficiencia**. Whitmire define suficiencia como “el grado en el que una abstracción posee las características requeridas de él o en el que un componente de diseño posee características en su abstracción, desde el punto de vista de la aplicación actual”. Dicho de otra forma, se pregunta: “¿Qué propiedades debe poseer esta abstracción (clase) para serme útil?”.

**Completitud.** La única diferencia entre completitud y suficiencia es “el conjunto de características contra las cuales se compara la abstracción o el componente de diseño”. La suficiencia compara la abstracción desde el punto de vista de la aplicación actual. La completitud considera múltiples puntos de vista, y plantea la pregunta: “¿qué propiedades se requieren para representar por completo al objeto de dominio problema?”. Puesto que el criterio para completitud considera diferentes puntos de vista, tiene una implicación indirecta en el grado en el que puede reutilizarse la abstracción o el componente de diseño.

**Cohesión.** Como su contraparte en software convencional, un componente OO debe diseñarse de manera que tenga todas las operaciones funcionando en conjunto para lograr un solo propósito bien definido. La cohesividad de una clase se determina al examinar el grado en el que “el conjunto de propiedades que posee es parte del problema o dominio de diseño”.

**Primitivismo**. Una característica que es similar a la simplicidad, el primitivismo, es el grado en el que una operación es atómica, es decir, la operación no puede construirse a partir de una secuencia de otras operaciones contenidas dentro de una clase. Una clase que muestra un alto grado de primitivismo encapsula sólo operaciones primitivas.

**Similitud.** El grado en el que dos o más clases son similares en términos de su estructura, función, comportamiento o propósito se indica mediante esta medida.

**Volatilidad.** Como se menciona muchas veces en este libro, los cambios en el diseño pueden ocurrir cuando se modifican los requerimientos o cuando ocurren modificaciones en otras partes de una aplicación, lo que da como resultado la adaptación obligatoria del componente de diseño en cuestión.

### **Metricas para Pruebas**

Aunque se ha escrito mucho acerca de las métricas de software para pruebas, la mayoría de las métricas proponen enfocarse en el proceso de las pruebas, no en las características técnicas de las pruebas en sí. En general, los examinadores deben apoyarse en las métricas de análisis, diseño y código para guiarlos en el diseño y la ejecución de los casos de prueba.

#### **Metricas para pruebas Orientados a objetos**

Las métricas del diseño OO proporcionan un indicio de la calidad del diseño. También ofrecen un indicio general de la cantidad de esfuerzo de prueba requerido para ejercitar un sistema OO. Binder sugiere un amplio arreglo de métricas de diseño que tienen influencia directa sobre la “comprobabilidad” de un sistema OO. Las métricas consideran aspectos de encapsulación y herencia.

* **Falta de cohesión en métodos (FCOM).**15 Mientras más alto sea el valor de la FCOM, mas estados deben ponerse a prueba para garantizar que los métodos no generan efectos colaterales.
* **Porcentaje público y protegido (PPP).** Los atributos públicos se heredan de otras clases y, por tanto, son visibles para dichas clases. Los atributos protegidos son accesibles a los métodos en las subclases. Esta métrica indica el porcentaje de los atributos de clase que son públicos o protegidos. Valores altos de PPP aumentan la probabilidad de efectos colaterales entre las clases porque los atributos públicos y protegidos conducen a alto potencial para acoplamiento.16 Las pruebas deben diseñarse para garantizar el descubrimiento de tales efectos colaterales.
* **Acceso público a miembros de datos (APD).** Esta métrica indica el número de clases (o métodos) que pueden acceder a otros atributos de clase, una violación de la encapsulación. Valores altos de APD conducen al potencial de efectos colaterales entre clases. Las pruebas deben diseñarse para garantizar el descubrimiento de tales efectos colaterales.
* **Número de clases raíz (NCR).** Esta métrica es un conteo de las distintas jerarquías de clase que se describen en el modelo de diseno. Deben desarrollarse las suites de prueba para cada clase raíz y la correspondiente jerarquía de clase. Conforme el NCR aumenta, también aumenta el esfuerzo de prueba.
* **Fan-in (FIN).** Cuando se usa en el contexto OO, el fan-in (abanico de entrada) en la jerarquía de herencia es un indicio de herencia múltiple. FIN > 1 indica que una clase hereda sus atributos y operaciones de más de una clase raíz. FIN > 1 debe evitarse cuando sea posible.
* **Número de hijos (NDH) y profundidad del árbol de herencia (PAH).**17 Como se mencionó en el capítulo 19, los métodos de superclase tendrán que volverse a probar para cada subclase.

#### **Metricas para mantenimiento**

Todas las métricas de software presentadas en este capítulo pueden usarse para el desarrollo de nuevo software y para el mantenimiento del software existente. Sin embargo, se han propuesto métricas diseñadas explícitamente para actividades de mantenimiento. Para ello, se determina la siguiente información:

Mt = número de módulos en la liberación actual

Fc = número de módulos en la liberación actual que cambiaron

Fa = número de módulos en la liberación actual que se agregaron

Fd = número de módulos de la liberación anterior que se borraron en la liberación actual

IMS = Indice de madurez de Software, proporciona un índice de estabilidad del software

El índice de madurez del software se calcula de la forma siguiente:

Conforme el IMS tiende a 1.0, el producto comienza a estabilizarse. El IMS también puede usarse como una métrica para planificar actividades de mantenimiento de software. El tiempo medio para producir una liberación de un producto de software puede correlacionarse con el IMS, y es posible desarrollar modelos empíricos para esfuerzo de mantenimiento.

### **Definicion de: Metricas para calidad del Software**

La meta dominante de la ingeniería del software es producir un sistema, aplicación o producto de alta calidad dentro de un marco temporal que satisfaga una necesidad de mercado. Para lograr esta meta, deben aplicarse métodos efectivos acoplados con herramientas modernas dentro del contexto de un proceso de software maduro. Además, un buen ingeniero de software (y los buenos gerentes de ingeniería del software) deben medir si la alta calidad es realizable. La calidad de un sistema, aplicación o producto sólo es tan buena como los requerimientos que describen el problema, el diseño que modela la solución, el código que conduce a un programa ejecutable y las pruebas que ejercitan el software para descubrir errores.

#### **Medicion de la calidad**

Aunque existen muchas medidas de calidad del software,8 la exactitud, capacidad de mantenimiento, integridad y usabilidad proporcionan útiles indicadores para el equipo del proyecto. Gilb sugiere definiciones y medidas para cada una:

**Exactitud**. Un programa debe operar correctamente o proporcionará poco valor a sus usuarios. La exactitud es el grado en el cual el software realiza la función requerida. La medida más común de la exactitud son los defectos por KLOC, donde un defecto se define como una falta verificada de acuerdo con los requerimientos. Cuando se considera la calidad global de un producto de software, los defectos son aquellos problemas reportados por un usuario del programa después de que el programa se liberó para su uso general. Con propósitos de valoración de calidad, los defectos se cuentan sobre un periodo estándar, por lo general un año.

**Capacidad de mantenimiento.** El mantenimiento y soporte del software representan más esfuerzo que cualquiera otra actividad de ingeniería del software. La capacidad de mantenimiento es la facilidad con la que un programa puede corregirse si se encuentra un error, la facilidad con que se adapta si su entorno cambia o de mejorar si el cliente quiere un cambio en requerimientos. No hay forma de medir directamente la capacidad de mantenimiento; por tanto, deben usarse medidas indirectas. Una métrica simple orientada a tiempo es el tiempo medio al cambio (TMC), el tiempo que tarda en analizarse la petición de cambio, diseñar una modificación adecuada, implementar el cambio, probarlo y distribuirlo a todos los usuarios.

**Integridad**. La integridad del software se ha vuelto cada vez más importante en la era de los ciberterroristas y hackers. Este atributo mide la habilidad de un sistema para resistir ataques (tanto accidentales como intencionales) a su seguridad. Los ataques pueden hacerse en los tres componentes de software: programas, datos y documentación. Para medir la integridad, deben definirse dos atributos adicionales: amenaza y seguridad. Amenaza es la probabilidad (que puede estimarse o derivarse de evidencia empírica) de que un ataque de un tipo específico ocurrirá dentro de un tiempo dado. Seguridad es la probabilidad (que puede estimarse o derivarse de evidencia empírica) de que el ataque de un tipo específico se repelerá

**Usabilidad.** Si un programa no es fácil de usar, con frecuencia está condenado al fracaso, incluso si las funciones que realiza son valiosas. La usabilidad es un intento por cuantificar la facilidad de uso y puede medirse en términos de las características que se presentaron en el capítulo 11.

#### **Eficiencia en la remocion del defecto**

Una métrica de calidad que proporciona beneficio tanto en el nivel del proyecto como en el del proceso es la eficiencia de remoción del defecto (ERD). En esencia, la ERD es una medida de la habilidad de filtrado de las acciones de aseguramiento y control de la calidad según se aplican a lo largo de todas las actividades del marco conceptual del proceso. Cuando se considera para un proyecto como un todo, la ERD se define en la forma siguiente:

Donde E es el número de errores que se encontraron antes de entregar el software al usuario final y D es el número de defectos que se encontraron después de la entrega.

#### **Establecimiento de un programa de metricas de software**

El Software Engineering Institute (SEI) desarrolló un manual muy completo para establecer un programa de métrica de software “dirigido hacia la meta”. El manual sugiere los siguientes pasos:

1. Identificar las metas empresariales.

2. Identificar lo que se quiere conocer o aprender.

3. Identificar las submetas.

4. Identificar las entidades y atributos relacionados con las submetas.

5. Formalizar las metas de medición.

6. Identificar preguntas cuantificables y los indicadores relacionados que se usarán para ayudar a lograr las metas de medición.

7. Identificar los elementos de datos que se recopilarán para construir los indicadores que ayuden a responder las preguntas.

8. Definir las medidas que se van a usar y hacer operativas estas definiciones.

9. Identificar las acciones que se tomarán para implementar las medidas.

10. Preparar un plan para la implantación de las medidas.

## **Adminstracion de Riesgo**

El análisis y la administración del riesgo son acciones que ayudan al equipo de software a entender y manejar la incertidumbre. Muchos problemas pueden plagar un proyecto de software. Un riesgo es un problema potencial: puede ocurrir, puede no ocurrir. Pero, sin importar el resultado, realmente es una buena idea identificarlo, valorar su probabilidad de ocurrencia, estimar su impacto y establecer un plan de contingencia para el caso de que el problema realmente ocurra.

El software es una empresa difícil. Muchas cosas pueden salir mal y, francamente, muchas con frecuencia lo hacen. Por esta razón es que estar preparado, comprender los riesgos y tomar medidas proactivas para evitarlos o manejarlos son elementos clave de una buena administración de proyecto de software.

### **Identificacion de Riesgos**

La identificación de riesgos es un intento sistemático por especificar amenazas al plan del proyecto (estimaciones, calendario, carga de recursos, etc.). Al identificar los riesgos conocidos y predecibles, el gerente de proyecto da un primer paso para evitarlos cuando es posible y para controlarlos cuando es necesario.

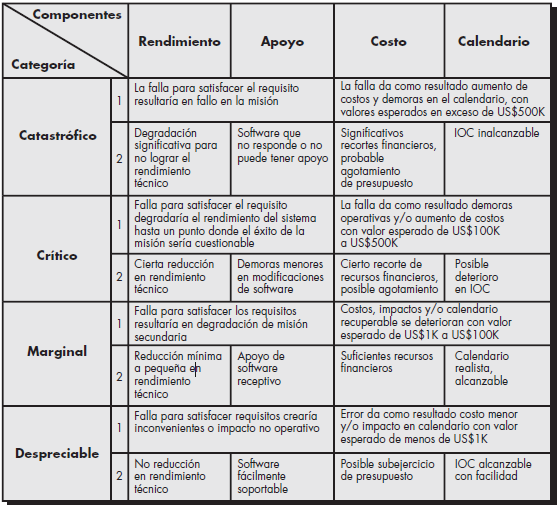
Un método para identificar riesgos es crear una lista de verificación de ítem de riesgo. La lista de verificación puede usarse para identificación del riesgo y así enfocarse sobre algún subconjunto de riesgos conocidos y predecibles en las siguientes subcategorías genéricas:

* Tamaño del producto: riesgos asociados con el tamaño global del software que se va a construir o a modificar.
* Impacto empresarial: riesgos asociados con restricciones impuestas por la administración o por el mercado.
* Características de los participantes: riesgos asociados con la sofisticación de los participantes y con la habilidad de los desarrolladores para comunicarse con los participantes en forma oportuna.
* Definición del proceso: riesgos asociados con el grado en el que se definió el proceso de software y la manera como se sigue por parte de la organización desarrolladora.
* Entorno de desarrollo: riesgos asociados con la disponibilidad y calidad de las herramientas por usar para construir el producto.
* Tecnología por construir: riesgos asociados con la complejidad del sistema que se va a construir y con lo “novedoso” de la tecnología que se incluye en el sistema.
* Tamaño y experiencia del personal: riesgos asociados con la experiencia técnica y de proyecto global de los ingenieros de software que harán el trabajo.

### **Proyeccion del Riesgo**

La proyección del riesgo, también llamada estimación del riesgo, intenta calificar cada riesgo en dos formas: 1) la posibilidad o probabilidad de que el riesgo sea real y 2) las consecuencias de los problemas asociados con el riesgo, en caso de que ocurra. Usted trabaja junto con otros gerentes y personal técnico para realizar cuatro pasos de proyección de riesgo:

1. Establecer una escala que refleje la probabilidad percibida de un riesgo.
2. Delinear las consecuencias del riesgo.
3. Estimar el impacto del riesgo sobre el proyecto y el producto.
4. Valorar la precisión global de la proyección del riesgo de modo que no habrá malos entendidos.



### **Valoracion de impacto de riesgo**

Tres factores afectan las probables consecuencias si ocurre un riesgo: su naturaleza, su ámbito y su temporización. La naturaleza del riesgo indica los problemas probables si ocurre.

El ámbito de un riesgo combina la severidad (¿cuán serio es?) con su distribución global (¿cuánto del proyecto se afectará o cuántos participantes se dañarán?). Finalmente, la temporización de un riesgo considera cuándo y por cuánto tiempo se sentirá el impacto. En la mayoría de los casos se quiere que las “malas noticias” ocurran tan pronto como sea posible, pero en algunos, mientras más se demoren, mejor.

puede aplicar los siguientes pasos para determinar las consecuencias globales de un riesgo: 1) determine la probabilidad promedio del valor de ocurrencia para cada componente de riesgo; 2) determine el impacto para cada componente con base en los criterios mostrados, y 3) complete la tabla de riesgos y analice los resultados como se describe en las secciones anteriores.

La exposición al riesgo global, ER, se determina usando la siguiente relación:



donde P es la probabilidad de ocurrencia para un riesgo y C es el costo para el proyecto si ocurre el riesgo.

### **El plan MMMR**

En el plan de proyecto del software puede incluirse una estrategia de administración del riesgo, o los pasos de administración del riesgo pueden organizarse en un plan de mitigación, monitoreo y manejo de riesgo (MMMR) por separado. El plan MMMR documenta todo el trabajo realizado como parte del análisis de riesgos y el gerente del proyecto lo usa como parte del plan de proyecto global.

Una vez documentada la MMMR y comenzado el proyecto, inician los pasos de mitigación y monitoreo del riesgo. Como ya se estudió, la mitigación del riesgo es una actividad que busca evitar el problema. El monitoreo del riesgo es una actividad de seguimiento del proyecto con tres objetivos principales: 1) valorar si los riesgos predichos en efecto ocurren, 2) asegurar que los pasos para evitar el riesgo definidos para un riesgo determinado se aplican de manera correcta y 3) recopilar información que pueda usarse para futuros análisis de riesgos. En muchos casos, el problema que ocurre durante un proyecto puede monitorearse en más de un riesgo. Otra actividad del monitoreo de riesgos es intentar asignar orígenes (cuál riesgo causó cuál problema a lo largo del proyecto).

# **CAPITULO III: MARCO TEORICO DE LA METODOLOGIA**

## **METODOLOGIA DE DESARROLLO XP**

La programación extrema o *eXtreme Programming* es una metodología de desarrollo de la ingeniería de software formulada por Kent Beck. Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.

Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

### **VALORES**

Los valores de la programación extrema son: simplicidad, comunicación, retroalimentación (*feedback*), coraje y respeto.

Los cinco valores se detallan a continuación:

* **Simplicidad:** es la base de la programación extrema. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento. Un diseño complejo del código junto a sucesivas modificaciones por parte de diferentes desarrolladores hace que la complejidad aumente exponencialmente.

Para mantener la simplicidad es necesaria la refactorización del código, ésta es la manera de mantener el código simple a medida que crece.

También se aplica la simplicidad en la documentación, de esta manera el código debe comentarse en su justa medida, intentando eso sí que el código esté autodocumentado. Para ello se deben elegir adecuadamente los nombres de las variables, métodos y clases. Los nombres largos no decrementan la eficiencia del código ni el tiempo de desarrollo gracias a las herramientas de autocompletado y refactorización que existen actualmente.

Aplicando la simplicidad junto con la autoría colectiva del código y la programación por parejas se asegura que cuanto más grande se haga el proyecto, todo el equipo conocerá más y mejor el sistema completo.

* **Comunicación:** se realiza de diferentes formas. Para los programadores el código comunica mejor cuanto más simple sea. Si el código es complejo hay que esforzarse para hacerlo inteligible. El código autodocumentado es más fiable que los comentarios ya que éstos últimos pronto quedan desfasados con el código a medida que es modificado. Debe comentarse sólo aquello que no va a variar, por ejemplo el objetivo de una clase o la funcionalidad de un método.

Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de cómo utilizar su funcionalidad. Los programadores se comunican constantemente gracias a la programación por parejas. La comunicación con el cliente es fluida ya que el cliente forma parte del equipo de desarrollo. El cliente decide qué características tienen prioridad y siempre debe estar disponible para solucionar dudas.

* **Retroalimentación (*feedback*):** Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real.

Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante.

Considérense los problemas que derivan de tener ciclos muy largos. Meses de trabajo pueden tirarse por la borda debido a cambios en los criterios del cliente o malentendidos por parte del equipo de desarrollo. El código también es una fuente de retroalimentación gracias a las herramientas de desarrollo. Por ejemplo, las pruebas unitarias informan sobre el estado de salud del código. Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.

* **Coraje o valentía:** Muchas de las prácticas implican valentía. Una de ellas es siempre diseñar y programar para hoy y no para mañana. Esto es un esfuerzo para evitar empantanarse en el diseño y requerir demasiado tiempo y trabajo para implementar el resto del proyecto. La valentía le permite a los desarrolladores que se sientan cómodos con reconstruir su código cuando sea necesario. Esto significa revisar el sistema existente y modificarlo si con ello los cambios futuros se implementaran más fácilmente. Otro ejemplo de valentía es saber cuándo desechar un código: valentía para quitar código fuente obsoleto, sin importar cuanto esfuerzo y tiempo se invirtió en crear ese código. Además, valentía significa persistencia: un programador puede permanecer sin avanzar en un problema complejo por un día entero, y luego lo resolverá rápidamente al día siguiente, sólo si es persistente.
* **Respeto**: se manifiesta de varias formas. Los miembros del equipo se respetan los unos a otros, porque los programadores no pueden realizar cambios que hacen que las pruebas existentes fallen o que demore el trabajo de sus compañeros. Los miembros respetan su trabajo porque siempre están luchando por la alta calidad en el producto y buscando el diseño óptimo o más eficiente para la solución a través de la refactorización del código. Los miembros del equipo respetan el trabajo del resto no haciendo menos a otros, una mejor autoestima en el equipo eleva su ritmo de producción.

### **CARACTERISTICAS FUNDAMENTALES**

Las características fundamentales del método son:

* **Desarrollo iterativo e incremental**: pequeñas mejoras, unas tras otras.
* **Pruebas unitarias continuas**, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación
* **Programación en parejas**: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. La mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata.
* Frecuente **integración del equipo de programación con el cliente** o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
* **Corrección de todos los errores** antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
* **Refactorización del código**, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
* **Propiedad del código compartida**: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
* **Simplicidad en el código**: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

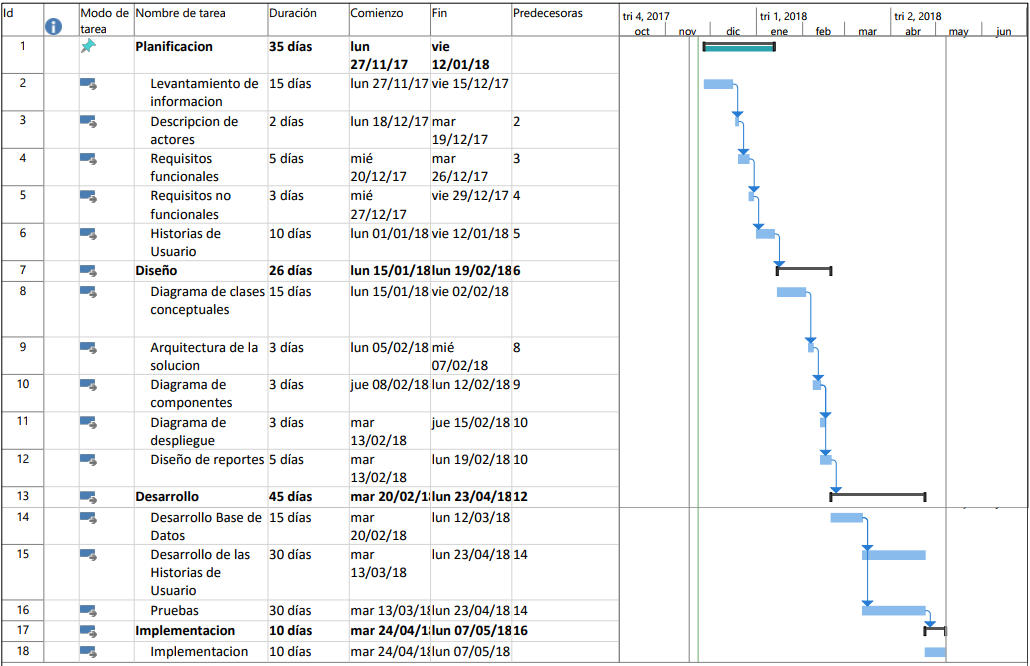
La simplicidad y la comunicación son extraordinariamente complementarias. Con más comunicación resulta más fácil identificar qué se debe y qué no se debe hacer. Cuanto más simple es el sistema, menos tendrá que comunicar sobre éste, lo que lleva a una comunicación más completa, especialmente si se puede reducir el equipo de programadores.

### **ROLES**

* **Programador:** Escribe las pruebas unitarias y produce el código del sistema. Es la esencia del equipo.
* **Cliente:** Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar el mayor valor de negocio.
* **Tester**: Ayuda al cliente a escribir las pruebas funcionales. Ejecuta pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
* **Tracker:** Es el encargado de seguimiento. Proporciona realimentación al equipo. Debe verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones.
* **Entrenador (coach):** Responsable del proceso global. Guía a los miembros del equipo para seguir el proceso correctamente.
* **Consultor:** Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Ayuda al equipo a resolver un problema específico. Además este tiene que investigar según los requerimientos.
* **Gestor (Big boss):** Es el dueño de la tienda y el vínculo entre clientes y programadores. Su labor esencial es la coordinación.

# **CAPITULO IV: INGENIERIA DE SOFTWARE DEL PROYECTO**

## **PLANIFICACION TEMPORAL**



## **ORGANIZACIÓN DEL PERSONAL**

El proyecto será realizado por un equipo conformado por: el jefe principal que será el director del proyecto. 1 jefe por cada área. Y su equipo correspondiente, la organización del equipo será descentralizado controlado.

El equipo del de desarrollo será el siguiente:

* Jefe de proyecto
* Jefe de área de desarrollo
* Analistas.
* Desarrolladores.
* QA’s tester.

Cada miembro del proyecto tendrá un diferente monto de salario, el cual equivale a una media entre 500$ y 1000$ para los jefes, desarrolladores, analistas y testers.

## **ESTIMACION**

### **METRICAS BASADAS EN PUNTOS DE FUNCION**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parámetro de Medición** | **Cuenta** | **FACTOR DE PONDERACION** | | | **SUBTOTAL** |
| **Simple** | **Medio** | **Complejo** |
| Nro. Entradas | 18 | **3** | **4** | **6** | 72 |
| Nro. De salidas | 16 | **4** | **5** | **7** | 112 |
| Nro. De consultas | 12 | **3** | **4** | **6** | 48 |
| Nro. De archivos | 15 | **7** | **10** | **15** | 105 |
| Nro. De Interfaces externas | 2 | **5** | **7** | **10** | 20 |
| **Cuenta Total** |  | | | | **357** |

### **VALORES DE AJUSTE DE COMPLEJIDAD**

Responder a cada una de las siguientes 14 preguntas y asignarles un valor entre 0 y 5.

0= NO INFLUENCIA

1= INCIDENTAL

2= MODERADO

3= MEDIO

4= SIGNIFICATIVO

5= ESENCIAL

|  |  |
| --- | --- |
| 1.        ¿Requiere el sistema copias de seguridad y de recuperación fiables? | 5 |
| 2.        ¿Requiere comunicación de datos? | 4 |
| 3.        ¿Existen funciones de procesamiento distribuido? | 3 |
| 4.        ¿Es crítico el rendimiento? | 5 |
| 5.        ¿Se ejecutará el sistema en un entorno operativo existente y fuertemente utilizado? | 5 |
| 6.        ¿Requiere entrada de datos interactiva? | 5 |
| 7.        ¿Requiere la entrada de datos interactiva que las transacciones de entrada se lleven a cabo sobre múltiples pantallas u operaciones? | 4 |
| 8.        ¿Se actualizan los archivos maestros de forma interactiva? | 3 |
| 9.        ¿Son complejas las entradas, las salidas, los archivos o las peticiones? | 3 |
| 10.      ¿Es complejo el procesamiento interno? | 4 |
| 11.      ¿Se ha diseñado el código para ser reutilizable? | 5 |
| 12.      ¿Están incluidas en el diseño la conversión y la instalación? | 3 |
| 13.      ¿Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones? | 5 |
| 14 ¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario? | 4 |
| La Sumatoria | 58 |

### **CALCULOS DE PUNTOS DE FUNCION**

El cálculo de los puntos de función del proyecto se realizará a partir de la siguiente formula:

*PF=CT\*(0.65+0.01\*Fi)*

En donde el Puntos de Función (PF) es igual a Cuenta Total (CT) por 0.65 + 0.01 por el Valor del ajuste de complejidad.

**PF=357\*(0.65+0.01\*58)**

**PF=357\*(1.23)**

**PF=439.11**

### **CALCULO DEL ESFUERZO**

**Esfuerzo =** PF / Productividad

**Esfuerzo =** 439.11 / 29.5

**Esfuerzo = 14.89**

### **CALCULO DE NUMERO DE PERSONAS QUE CONFORMAN EL EQUIPO DE DESARROLLO DEL SOFTWARE**

**Duración =** 5 Meses

**Personas =** Esfuerzo / Duración

**Personas =** 14.89 / 5

**Personas =** 2.98 **≈ 3**

### **COSTO DEL PERSONAL**

**Costo =** ((((Personas \* 2.1$hrs) \* Horas) \* Días) \* Duración)

**Costo =** ((((3 \* 2.1) \* 8) \*30) \*5)

**Costo = 7560$**

## **ANALISIS Y GESTION DEL RIESGO**

### **ESCALA BASADA EN EL ASPECTO DEL PERSONAL**

**Preguntas:**

* ¿Tiene el personal un conjunto de habilidades adecuadas?
* ¿Se dispone de la gente suficiente?
* ¿Hay personal a medio tiempo?
* ¿Han recibido los miembros del personal el entrenamiento adecuado?

**Respuesta:**

1. Si, el personal tiene las capacidades intelectuales que son respaldadas por certificados correspondientes al área de trabajo para la creación del software.
2. Si, la cantidad del personal es exacto para cubrir las exigencias que requiere el desarrollo del programa.
3. No, cada una de las personas tiene un contrato de trabajo en donde ellos deberán trabajar las horas correspondientes.
4. Si, se realizó una preparación previa, de la forma la cual se realizará el proyecto, siguiendo la metodología de desarrollo ágil.

### **ESCALA BASADA EN EL ASPECTO TECNICO**

**Preguntas:**

* ¿Podrá existir un fallo de diseño en la Base de Datos?
* ¿El servidor puede dejar de funcionar?
* ¿Podría haber cambios en los requerimientos?
* ¿Puede funcionar en un estado de error?

**Respuesta:**

1. No, el análisis y diseño de los modelos relacionales con la herramienta UML determinan un desarrollo de la Base de Datos con muy poco margen de error.
2. No, es muy poco probable ya que es un servidor nuevo.
3. Si, podría añadirse nuevas funcionalidades en el futuro, lo cual requiera cambios de hardware o software.
4. Si, el sistema puede seguir funcionando a pesar de presentar un error, sin embargo, las acciones previas estarán en espera.

### **ESCALA BASADA EN EL ASPECTO DEL NEGOCIO**

**Preguntas:**

* ¿Podrá existir un fallo en la lógica del negocio?
* ¿Los casos de uso son funcionales?
* ¿Podría haber cambios en la funcionalidad del negocio?

**Respuesta:**

1. No, la captura de requerimientos se realiza de manera que no exista una falla en la lógica de negocio del software.
2. Si, cada uno representa una funcionalidad dentro del software, algunos de manera independiente
3. No, una vez que se definan los requisitos funcionales, estos no deben ser cambias, no antes de la etapa de desarrollo.

### **PROYECCION DEL RIESGO**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NOMBRE | Nº RIESGO | PRIORIDAD | PROBABILIDAD | IMPACTO | PASOS |
| Habilidades del personal | **1** | **4** | **0,3** | **60%** | Capacitar al personal para realizar el proyecto |
| Cantidad del personal | **2** | **2** | **0,5** | **80%** | 3000 $ para contratar a nuevos programadores |
| Medio Tiempo | **3** | **10** | **0,8** | **10%** | Delegar los trabajos en grupo |
| Preparación del personal | **4** | **7** | **0,8** | **10%** | Enseñar a cada el proceso ya definido del proyecto |
| Diseño BD | **5** | **1** | **0,5** | **85%** | Modificar nuevamente todo el diseño |
| Fallo en el servidor | **6** | **3** | **0,1** | **70%** | 1000$ para los repuestos del mismo |
| Cambio de requerimientos | **7** | **9** | **0,7** | **10%** | 200$ para acomodar nuestro software con nuevas licencias |
| Funcionalidad en Error | **8** | **5** | **0.4** | **40%** | Realizar un parche correctivo a dicho error |

## **PRESUPUESTO**

### **COSTO PARCIAL DE LAS TAREAS DE DESARROLLO DE SOFTWARE**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Fase | Etapa | Costo(usd) |
| 1 | Codificación | Desarrollo Base de Datos | 1600 |
| 2 | Codificación | Desarrollo de Historias de Usuario | 4200 |
| 3 | Implementación | Implementación | 300 |
| 4 | Implementación | Soporte | 250 |
|  | **TOTAL** |  | **6350** |

### **COSTO PARCIAL DE LA GESTION DEL PROYECTO**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Fase | Etapa | Costo(usd) |
| 1 | Planificación | Levantamiento de información | 800 |
| 2 | Planificación | Historias de Usuarios | 800 |
| 3 | Diseño | Diseños | 2000 |
|  | **TOTAL** |  | **3600** |

### **COSTO PARCIAL DE RECURSOS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Cantidad | Descripción | Marca/Empresa | c/u  (usd) | Total (usd) |
| 1 | 2 | Office 2016 | Microsoft | 90 | 180 |
| 2 | 5 | Win10 pro | Microsoft | 100 | 500 |
| 3 | 3 | Visual Studio 2015 pro | Microsoft | 150 | 450 |
| 4 | 1 | SQL Server 2014 Standard | Microsoft | 100 | 100 |
| 5 | 5 | Laptops | Dell | 600 | 3000 |
|  | **TOTAL** |  |  |  | **4230** |

### **COSTO TOTAL DEL PROYECTO**

|  |  |
| --- | --- |
| Descripción | Costo Total (usd) |
| Costo parcial de las tareas de desarrollo de software | 6350 |
| Costo parcial de la gestión de proyectos | 3600 |
| Costo parcial de recursos | 4230 |
| Costo al Personal | 12600 |
| Costo Total del Proyecto | **26780** |

# **CAPITULO V: INGENIERIA DEL PROYECTO**

## **MODELO DE REQUISITOS**

### **REQUISITOS FUNCIONALES**

|  |  |  |
| --- | --- | --- |
| # | Requerimiento | Atributos de Calidad |
| RF1 | Página de Inicio | * Inicio de sesión de usuario * Mostrar el logo de la empresa y una pequeña descripción * Un menú debe tener las siguientes opciones: * Inicio * Ventas * Compras * Inventario * Productos * Pedidos * Empleados * Clientes * Sucursales * Proveedores * Cerrar Sesión |
| RF2 | Gestión de Ventas | * El sistema permitirá la búsqueda, registro, modificación e eliminación de docentes. * El código es autogenerado. |
| RF3 | Gestión de Compras | * El sistema permitirá la búsqueda, registro, modificación e eliminación de tribunales. * El código es autogenerado. |
| RF4 | Gestión de Empleados | * El sistema permitirá la búsqueda, registro, modificación e eliminación de modalidades. * El código es autogenerado. |
| RF5 | Gestión de Inventario | * El sistema permitirá la búsqueda, registro, modificación e eliminación de trabajos finales de grado. * El código es autogenerado. |
| RF6 | Gestión de Productos | * El sistema permitirá la búsqueda, registro, modificación e eliminación de estudiantes. * El código es autogenerado. |
| RF7 | Gestión de Clientes | * El sistema permitirá la búsqueda, registro, modificación e eliminación de especialidades. * El código es autogenerado. |
| RF8 | Gestión de Proveedores | * El sistema permitirá la búsqueda, registro, modificación e eliminación de estados. * El código es autogenerado. |
| RF9 | Gestión de Sucursales | * El sistema permitirá la búsqueda, registro, modificación e eliminación de estados. * El código es autogenerado. |
| RF10 | Gestión de Pedidos | * El sistema permitirá la búsqueda, registro, modificación e eliminación de estados. * El código es autogenerado. |
| RF11 | Reportes de ventas, compras, productos, estado de inventario, pedidos | * El sistema permitirá la visualización de estos reportes. |

### **REQUISITOS NO FUNCIONALES**

* **Requerimientos de Hardware:**
  + Procesador Intel(R) Core(TM) i3 CPU @ 2.60GHz.
  + Memoria RAM 8 Gb para mejor rendimiento.
  + Mínimo 1GB de disco duro para el software, se recomienda más espacio a medida que crezca los datos de la base de datos.
* **Requerimientos de Software**
  + Visual Studio 2015 Pro
  + SQL Server 2014 Standard.
  + La aplicación debe desarrollarse en C#, utilizando el motor de base de datos SQL Server.
  + Durante el registro de cualquier tipo de dato se pedirá al usuario la confirmación de la acción.
  + El sistema mostrara mensajes confirmando una acción o avisando de un error.
  + Todos los campos de los formularios deberán ser validados antes de realizar una acción (Ejemplo: Guardar, Modificar y eliminar) en la base de datos.
  + Copia de seguridad automática de la base de datos todas las noches en horarios de 10:00pm.
  + Modificación y actualización de datos a través de formularios.
  + Se avisará del error mediante un mensaje.
  + La aplicación deberá tener acceso por niveles para los diferentes roles.
  + Caducidad de contraseña.
  + Bitácora de las operaciones hechas sobre las tablas de la base de datos.

### **DESCRIPCION DE LOS ACTORES**

|  |  |  |
| --- | --- | --- |
| CODIGO | ACTOR | DESCRIPCION |
| EV | Encargado de Ventas | Registra nuevo proveedor, modifica proveedor, elimina proveedor, busca proveedor, registra productos, modifica productos, elimina productos, busca productos, registra solicitud de compra, modifica solicitud de compra, elimina solicitud de compra, busca solicitud de compra, registra pedido, elimina pedido, busca pedido, modifica pedido. |
| PV | Personal de Ventas | Registra venta, modifica venta, elimina venta, busca venta, registra solicitud de venta, modifica solicitud de venta, elimina solicitud de venta, busca solicitud de venta, busca inventario, busca sucursal, registra cliente, modifica cliente, busca cliente, elimina cliente. |
| PI | Personal de Inventario | Registra recepción, modifica recepción, elimina recepción, busca recepción, busca solicitud de compra, busca pedido, registra inventario, modifica inventario, elimina inventario, busca inventario, registra sucursal, modifica sucursal, elimina sucursal, busca sucursal. |
| G | Gerencia | Registra empleado, modifica empleado, elimina empleado, busca empleado, busca venta, busca solicitud de venta, busca inventario. |

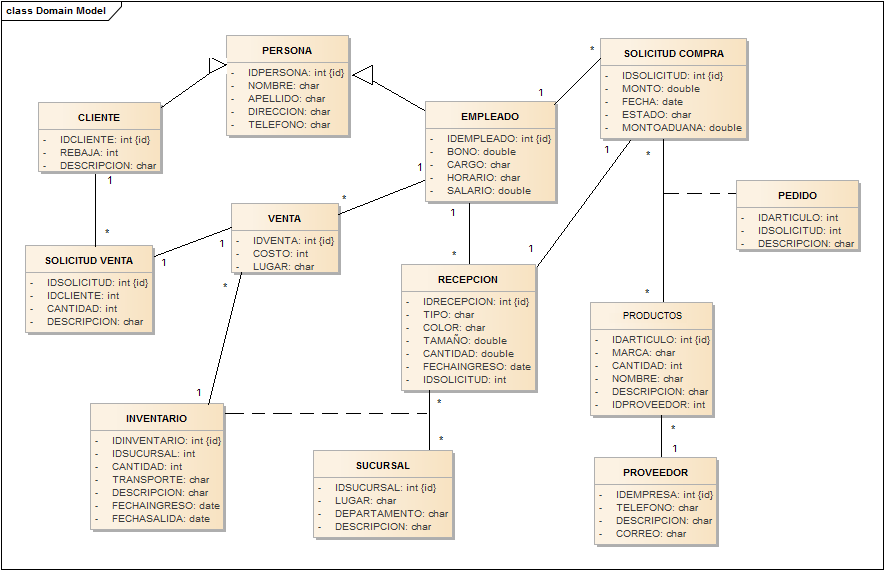
## **MODELO DE ANALISIS**

### **HISTORIAS DE USUARIO**

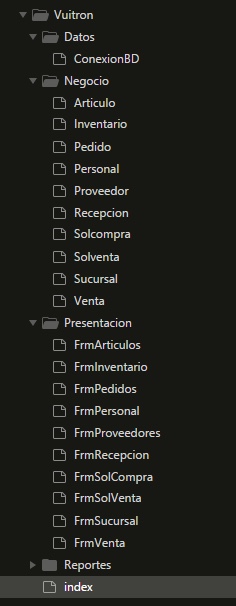
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nro. de historia** | | **Titulo** | | |
| 001 | *Gestión de Ventas* | | | |
| **Descripción** | | | | |
| Permite el registro de los datos necesarios para la gestión de Ventas. | | | | |
| **Prioridad** | | | Alta | |
| **Criterios de aceptación** | | | | |
| * El código de Venta se debe generar de manera automática, y debe ser único. * Debe mostrar la sucursal donde será realizada la venta. * Debe permitir buscar a un cliente para asignarlo a la venta. * Debe permitir buscar y seleccionar los productos del inventario. * Se debe permitir modificar el estado de la venta * Se debe agregar la fecha de registro de la venta. * Se debe crear una interfaz de búsqueda de Venta, para su posterior modificación si esta no fue facturada. | | | | |
| Responsables | | | | Raul Barba, Esteban Sandoval |
| **Adjunto** | | | | |
|  | | | | |

## **MODELO DE DISEÑO**

### **DIAGRAMA DE CLASES CONCEPTUALES**



### **ARQUITECTURA DE LA SOLUCION**



### **DISEÑO DE PANTALLA DE REPORTE**

**Reporte de ventas por empleado**

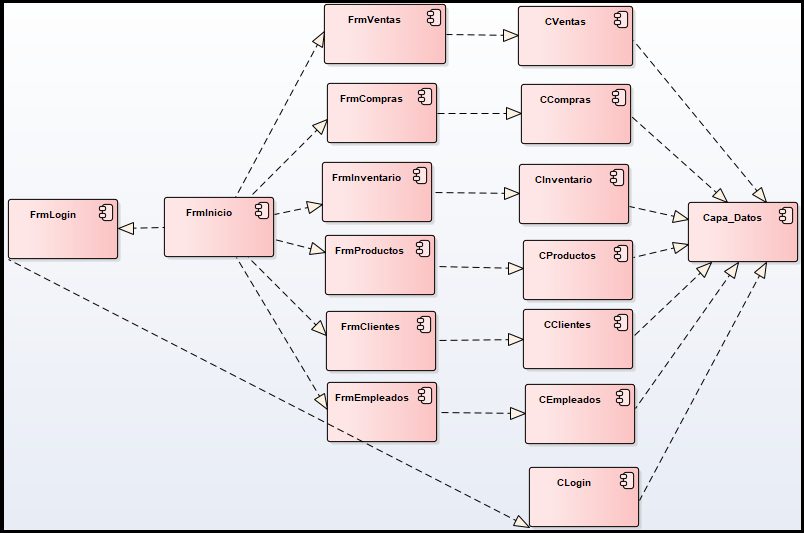
**Parámetros:**

**Fecha inicio: 01/01/2017**

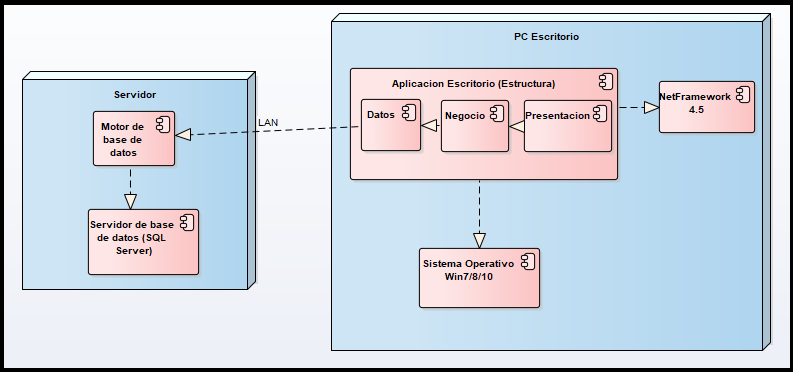
**Fecha fin: 30/06/2017**

****

### **DIAGRAMA DE COMPONENTES**



### **DIAGRAMA DE DESPLIEGUE**



# **CONCLUSIONES**

* Se realizó el diseño del sistema de gestión comercial para que cubra las necesidades de automatización.
* Se hizo uso de la ingeniería de software para la planificación y desarrollo del sistema.
* Se logró hacer una captura y análisis apropiado de los requerimientos.
* Se diseñó los diagramas necesarios para la funcionalidad del sistema
* Se desarrolló 1 caso de uso utilizando la metodología XP.

# **BIBIOGRAFIA**

* *Ingeniería del Software. Un Enfoque Practico – Roger Pressman – 7ma Edición*
* *Guia MAAP – Ing de software UTEPSA*
* Scrum y XP desde las trincheras - Henrik Kniberg *- 2007*