Gestión de Proyectos de Software

Se estima que un setenta por ciento de los fracasos expresados en atrasos y en la no estimación de costos reales asociados en los proyectos IT se debe a la no incorporación de prácticas esenciales y estandarización en los procesos de Software. No llevando a la práctica las acciones correctas, no tomando decisiones en el momento oportuno, y no ofreciendo compromiso. Uno de los mayores retos a los que se enfrentan las empresas hoy en día, es hacer que estas prácticas esenciales sean de su estrategia el centro de todas sus actividades. De esta forma la mayoría de las empresas IT mueren al cabo de 5 años. Este libro pretende entregar desde el punto de vista práctico y esencial las practicas necesarias, mediciones y elementos de calidad en todo proyecto IT, de tal forma poder obtener un mejoramiento inmediato en sus actividades. Sin lugar a dudas este libro proporciona a los académicos y alumnos una panorámica actual y completa sobre la ingeniería de software, combinando el rigor científico practico.

Las empresas que sobrevivan en el mercado del siglo XXI deben implementar el software como un elemento que permita generar estrategias de diferenciación en sus procesos de negocio. Con el objetivo de ser más competitivos, algunas organizaciones del software están implantando la dirección disciplinada de los procesos usados para el desarrollo y mantenimiento del software. A través de la mejora de sus procesos, estas organizaciones han estado obteniendo la mejora necesaria de la calidad de sus productos y resultados buenos en sus negocios.

Una de las mayores dificultades con las que se enfrentan aquellos que comienzan y trabajan en proyectos IT es la gran diversidad de conceptos que resulta necesario incorporar para adentrarse en el tema y el hecho de que coexisten a su vez un gran número propuestas. Este libro permite implementar en una organización una estrategia global de desarrollo de proyectos de IT desde los requerimientos hasta la mantención, enfocado a los paradigmas necesarios, gestión del recurso humano, métricas, SQA, medición y estimación de costos.

Este libro se presenta en forma clara y concreta los elementos necesarios para llevar a una organización a un nivel superior sus procesos de software. Sin duda permitirá a los interesados en la ingeniería de software aumentar en bases sólidas los conocimientos sobre las técnicas y metodologías necesarias para tener éxito en sus proyectos IT.

BREVE CV

Alejandro Bedini González, ingeniero y master en Ingeniería Informática con especialización Ingeniería de Software en la USM. Con Postgrado en Management Financiero de la Universidad de Buenos Aires. Es Coordinador local y difusor del proyecto SPICE (ISO/IEC 15504) modelo para determinar las capacidades y mejoras en los procesos de software. Ha publicado, dictado conferencias y efectuado asesorías a empresas en Argentina, Bolivia, Colombia, Chile, Ecuador, Escocia, España, Irlanda, Italia, Perú y Venezuela. En el ámbito de la calidad de software y ebusiness. Es Fundador de SPIN-Ecuador y cofundador de SPIN-Perú, SPIN-Bolivia, SPIN-Colombia y SPIN-Mendoza Argentina. Es miembro del STAFF de la compañía SYNSPACE, que entre otras actividades se encarga de los proyectos IT y área de gestión conocimiento de la agencia aeroespacial europea y de importantes empresas comerciales. Profesor de MBA en Bolivia, Ecuador y Chile.

Prologo

Circulan 31 mil millones de correos electrónicos cada día.
2006: esta cifra llegará a 60.000 Millones

El conocimiento se duplica cada siete u ocho años.

En los últimos 30 años se ha publicado más que desde los comienzos de la humanidad (se realizan sobre 2000 publicaciones diarias). El 90% de los científicos que han existido están vivos hoy.

Las empresas que desarrollan software su ciclo de vida son 5 años.
Actualmente se invierte más en software que en hardware. Solo 5% de los proyectos de software cumplen con lo estimado inicialmente.
Este libro reúne todo lo necesario para enfrentar un proyecto exigente actual en el contexto de la ingeniería de software desde el primer contacto con el cliente hasta la implantación. Presenta las mejores prácticas a seguir para asegurar un proyecto exitoso en la gestión en un proyecto de software. El libro incorpora con precisión los aspectos de calidad más exigentes y actuales, así como casos para comprender la aplicación concreta de la teoría a la práctica cotidiana.

Es el producto de más de 15 años de experiencia en la formulación, dirección y control de proyectos de los autores.

Ante este escenario, en el arte de gestionar un proyecto donde todas las variables presentes están sujetas a cambios de todo tipo, el profesional debe desde el principio desarrollar una estrategia de trabajo que le permita establecer un espiral virtuoso de excelencia. Es por ello que ponemos a disposición de los profesionales hispano parlantes este material para uso como texto guía o de consulta para su aplicación en cursos universitarios.

Este libro será un excelente soporte para ser una especie sobreviviente

Lautaro Guerra G., Alejandro Bedini G.

Índice

Introducción

La paradoja Tecnológica

El avance tecnológico diariamente remece a la sociedad, ya que este implica un cambio fundamental en la manera de operar en todo orden de actividades dentro de la organización, ya sea en el área estratégica u operacional.

Hoy en día importa que el crecimiento del mercado sea mayor que la disminución de los precios[1]. El desafío está en como ser competitivo en un mundo en el cual la tecnología es virtualmente gratis.

Es necesario hacer una nueva definición de los valores económicos. El valor hoy en día está en establecer una relación de largo plazo con el cliente, aún cuando esto signifique esfuerzos por parte de la empresa, como por ejemplo: regalar la primera generación de productos.

Algunos casos que evidencian lo anterior son los siguientes:

- La TV satelital digital es un ejemplo de estrategia del futuro; el grupo de empresas liderado por GM, hace 5 años planea y diseña los componentes necesarios, antes que los precios fueran adecuados. RCA, socio del grupo, ofrece la antena de plato pequeña y el decodificador por US$ 699: a Junio de 1994, 400.000 casas en USA ya la tenían.

- En India, los piratas roban la señal satelital de News Corp. Star TV y ganan revendiendo sus programas por cable. El dueño, R. Murdock dice "Los entrepreneurs están ampliando el mercado potencial de Star TV". Esto le permite subir las tarifas del avisaje.

Las nuevas reglas del juego

- **Los productos son valiosos si son más baratos.** El nicho para los productos caros es cada vez menor, los compradores buscan bajos precios y gran volumen. Ejemplo de esto es lo sucedido con Compaq Computer Corporation. En 1982 IBM acababa de poner en el mercado el primer computador de la historia, Compaq pensó en poner un computador al alcance del usuario a través de tiendas comerciales, y de esta manera competir con precios más bajos. En 1983 ya vendía 53.000 computadores.
- **Hace dinero regalando**. Se regala el producto y los accesorios los accesorios indispensables se venden. La venta de productos de alta tecnología se asemeja al caso de la hoja de afeitar, regala la máquina y gana con la venta de la hoja. Ejemplo de esto es el caso del Software Mosaic para Internet, sus creadores lo entregaban gratis pero vendían el upgrade como Netscape.

1 Tanto el aumento del mercado como la disminución de los precios se comportan de forma exponencial.

- **Los equipos ganan.** La complejidad de los dispositivos electrónicos, la TV satelital y en general de productos de alta tecnología requiere de la colaboración que antes sólo existía en la construcción de aviones, barcos o viajes a la luna.
- **Boutique masiva.** Usar técnicas de manufactura que permitan hacer único cada producto que sale de la línea. Como ejemplo de esto Dell Inc., quien ha logrado desarrollar una estrategia enfocada a dar a cada cliente una respuesta personalizada a sus necesidades, es decir, los clientes obtienen justamente lo que quieren.
- **Acelerar y derrochar.** La eficiencia en la ingeniería de un producto es buena, pero contar con recursos computacionales y de comunicación y aumentar la velocidad de llegar al mercado es esencial y por ende es la mejor y más eficaz ruta.

Capitulo 1

Proyectos de Software

1.1 Definición de Proyectos Informáticos

Un *proyecto* es una asociación de esfuerzos, limitado en el tiempo, con un objetivo definido, que requiere del acuerdo de un conjunto de especialidades y recursos. También puede definirse como una organización temporal con el fin de lograr un propósito específico. Cuando los objetivos de un proyecto son alcanzados se entiende que el proyecto está completo.

La gran variedad de elementos que intervienen en un proyecto, hacen que éste sea único; Pese a ello, es posible aplicar técnicas y métodos comunes para asistir su gestión.

Los proyectos informáticos obedecen a esta definición, pero además se caracterizan por el impacto directo e indirecto que provocan en toda la organización, la casi inevitable existencia de relaciones con otros proyectos informáticos, el estar altamente propensos a sufrir de obsolescencia, especialmente tecnológica y la intensa participación de recurso humano de distintas áreas durante su desarrollo.

Para la definición de proyectos informáticos se ha hecho un esfuerzo en identificar y estandarizar las etapas que lo conforman. Basándose en metodologías bien definidas, se han desarrollado herramientas computacionales que permiten asistir su gestión en forma automatizada.

1.1.1 Elementos de Definición de un Proyecto

Según la definición de proyectos, es posible representarlo en el eje del tiempo con la duración requerida para lograr el objetivo establecido, comenzando en un instante hasta finalizar en el momento T, donde el período T representa la duración esperada del proyecto.

Al definir un proyecto es necesario tener claridad sobre los puntos que se definen a continuación:

- ✓ *Cliente:* Persona a quien va dirigido el resultado del proyecto, generalmente ellos presentan un problema que requiere solución.
- ✓ *Usuarios:* Persona que utilizará el sistema o parte de él.
- ✓ *Inicio:* Momento en que es expresada la necesidad específica en el cliente.
- ✓ *Término:* Momento en que se cumple el resultado definido tanto en costo, oportunidad, calidad o desempeño técnico.
- ✓ *Costo:* Recurso o insumo entrante al proyecto, expresado generalmente en dinero.
- ✓ *Tiempo:* Recurso que origina una secuencia y luego un programa, es transformable en costo. Se incorpora al proyecto en dos dimensiones: la duración del esfuerzo y el momento en que éste se realiza.

- ✓ *Desempeño Técnico:* Característica de los resultados expresados a través de un prototipo, gráfico, índices y funcionamiento fiable en términos de los objetivos intermedios y del objetivo final.
- ✓ *Jefe del Proyecto:* Persona responsable del proyecto. Encargado de la dirección del proyecto, su planificación y el control de todos los costos, recursos, programas y de la satisfacción del cliente.

1.1.2 La Gestión de Proyectos

La gestión de proyectos es un proceso continuo. Este proceso requiere de una estrategia global, apoyada por herramientas de trabajo que incrementen la productividad. El propósito de planificar y controlar es proveer una propuesta uniforme para el desarrollo y la administración de los proyectos. Los planes deben apoyar los niveles estratégicos, tácticos y operacionales de las organizaciones con el fin de alcanzar las metas corporativas de largo, mediano y corto plazo.

A través del ciclo de vida de un proyecto, se conforman dos categorías de actividades a realizar y que se encuentran directamente relacionadas: las *actividades de gestión* y las *actividades de desarrollo del sistema.*

Las actividades de gestión son aquellas relacionadas con la administración de las organizaciones, personas, sistemas y procedimientos comprometidos en el proceso de planificación y construcción del sistema. La planificación del proyecto, junto con las actividades de control, es iterada para cada fase del proyecto y proveen de la estrategia de administración con la cual las actividades de desarrollo del sistema son estimadas, programadas y ejecutadas.

Las actividades de desarrollo del sistema se centran en el desarrollo mismo. Las metodologías de desarrollo están típicamente organizadas en distintas fases, agrupadas en áreas funcionales de estudio, diseño y construcción, basadas en una estructura de partición del trabajo.

La administración y planificación de proyectos requiere de la integración de dos modelos implícitos de trabajo, usualmente no reconocidos: *el modelo de administración y el modelo de desarrollo.*

1.1.3 El Modelo de Administración

El modelo de administración identifica las relacionas entre la administración misma y los procesos de planificación y control.

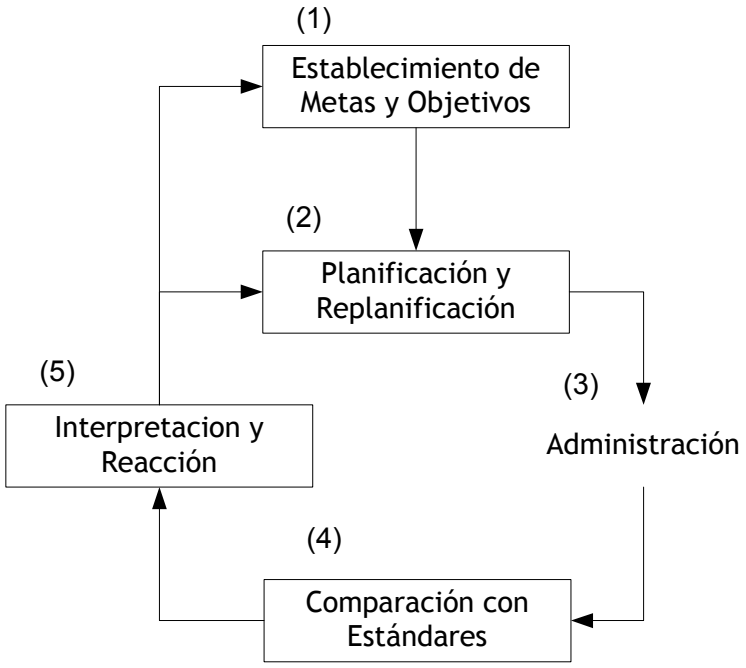


Figura 1.1: Modelo de Administración

El modelo de administración incluye a organizaciones, sistemas y personas. Los administradores de proyectos de software son responsables por la planificación del desarrollo, supervisión de las tareas y aseguramiento de que el trabajo es realizado *de acuerdo a los estándares, a tiempo y dentro del presupuesto.* Una buena administración no garantiza el éxito, pero una mala administración generalmente conlleva a un producto de Software terminado tarde, que excede el costo estimado y caro de mantener.

La administración de proyectos informáticos se diferencia de los proyectos tradicionales en que:

- ✓ El producto es intangible. El administrador de proyectos (o Jefe de proyecto) depende de la documentación disponible para revisar el progreso.
- ✓ No se tiene aún comprensión acabada del proceso de desarrollo de Software. Los modelos que se utilizan son sólo una simplificación para ayudar a la comprensión.
- ✓ Los proyectos grandes son únicos en su tipo. La experiencia histórica casi no existe para grandes proyectos.

Actividades de Administración.

Las actividades de administración son:
- ✓ Generación de la propuesta.
- ✓ Estimación de costos.
- ✓ Planificación y creación de itinerario.
- ✓ Monitoreo y revisiones.
- ✓ Selección y evaluación de personal.
- ✓ Informes y presentaciones.

La planificación y el control del proyecto son una parte integral del proceso de administración. Una planificación eficaz puede tomar desde un 15% hasta un 25% del total del esfuerzo del proyecto. La planificación ocurre continuamente durante el proyecto, desde la concepción inicial hasta la producción final. Su principal objetivo es lograr la construcción dentro de un programa y presupuesto progresivamente refinados, para lo cual se realizan las siguientes funciones:
- ✓ Asignación de Recursos.
- ✓ Estimación y Planificación detallada del Proyecto.
- ✓ Revisiones del Proyecto.
- ✓ Control de Calidad.

1.1.4 Fases y Revisiones Administrativas

Los proyectos bien definidos están compuestos por fases, cada una de las cuales tienen objetivos específicos y salidas mensurables. Las revisiones administrativas conducen el proyecto, ya que en ellas se debe decidir si:
- ✓ Detener y posponer el proyecto.
- ✓ Cambiar el ámbito, objetivos y restricciones (y sí fuera necesario, repetir toda o parte de la fase en cuestión).
- ✓ Aprobar los puntos de calidad / Hitos.
- ✓ Pasar a la siguiente fase.

Una de estas decisiones debe ser tomada durante cada sesión de revisión.
Cada sesión debiera incluir a todos los participantes del proyecto: operadores de computadoras, auditores, equipo técnico, personal de desarrollo y el usuario final.
El modelo de administración se sustenta en organizaciones y procesos. Si éstos son utilizados apropiadamente, es posible incrementar la productividad de los administradores y del personal que compone las organizaciones.

Organizaciones

Las formas de organizar un proyecto son:
- ✓ Grupos de dirección del proyecto.
- ✓ Grupo de evaluación especial de las tareas.
- ✓ Grupos de proyectos.

Procesos

Algunos de los procesos en un proyecto son:

- ✓ Análisis y administración de la cartera de proyectos.
- ✓ Control de cambios (diseño y producción).
- ✓ Evaluación y determinación del tamaño de los proyectos.
- ✓ Análisis y administración del riesgo.
- ✓ Autorización del proyecto.
- ✓ Análisis costo / beneficio, Cálculo de Tasa Interna de Retorno, Flujo de Caja, Valor Presente Neto.
- ✓ Evaluación de la Calidad (SQA)

1.1.5 La cartera de aplicaciones

Una Cartera de Aplicaciones es un inventario de todos los proyectos planificados y actuales, incluyendo todos los tipos de proyectos, es decir:

- ✓ Nuevos desarrollos (utilizando herramientas tradicionales o prototipos).
- ✓ Mejoras.
- ✓ Soporte a producción.
- ✓ Mantenimiento.
- ✓ Instalaciones de paquetes.

Esta cartera debe ser administrada eficientemente para alcanzar las metas establecidas por la organización en el ámbito ejecutivo. Los administradores clave, asignados por el nivel ejecutivo, conforman el grupo de administradores de la cartera (GAC). El procedimiento de autorización del proyecto provee de información decisional detallada que permite al GAC dirigir el contenido de esta cartera.
Los cambios en la cartera se originan a partir de cambios en el negocio o a medida que las distintas etapas del trabajo se van complementando. Normalmente se agregan nuevos recursos, o bien cambian las prioridades para dar cumplimiento a las necesidades surgidas por los cambios.

1.1.6 Organización para proyectos de software

Dentro de las organizaciones se pueden identificar estructuras formales e informales. Una Estructura Organizacional típica es aquella que identifica los niveles jerárquicos estratégicos, tácticos y operacionales.
Independiente de su tipo, una estructura Organizacional debe ser lo suficientemente flexible como para permitir un apropiado flujo de información a los distintos niveles, facilitando los requerimientos de reportes, comunicación y toma de decisiones.
Los equipos de trabajo deben ser pequeños, con no más de ocho personas en total. Si el proyecto es muy grande, éste deberá ser dividido en subsistemas, teniendo especial cuidado en definir adecuadamente la interfaz entre ellos.

Los beneficios de trabajar con un equipo pequeño son el que permite:

- ✓ Definir estándares de calidad.
- ✓ Que los miembros del equipo trabajen en conjunto.
- ✓ Programar las tareas sin afectar el ego.

- ✓ Que todos puedan conocer el trabajo del otro.
- ✓ La comunicación es más rápida y eficiente. Se crean canales de comunicación simples.
- ✓ Se establecen lazos afectivos que fortalecen la creatividad.

1.2 Características del Desarrollo de Software

1.2.1 Desarrollo de sistemas como un proceso industrial

El desarrollo de sistemas de software es una industria relativamente joven que no ha alcanzado el nivel de madurez encontrado en ramas industriales más tradicionales. Consecuentemente, los productos desarrollados mediante el uso de la tecnología de software, a menudo sufren de la carencia de prácticas establecidas. Esta falta de experiencia le da importancia al desarrollo de software, hasta ahora ubicado dentro de los métodos y procesos creativos usados inicialmente en la construcción de sistemas basados en la computadora, característica que se encuentra en casi todos los métodos de ingeniería de software y herramientas relacionadas que han sido creados para apoyar la realización de estos sistemas[2].
¿Cómo hacemos para proveer a esta industria de los métodos que nos permitan tratar con los aspectos prácticos de una visión más global de sus productos?
Una analogía útil
La construcción es una de las ramas industriales existentes más maduras, sus orígenes se remontan a los comienzos de la vida civilizada. Como todos usamos algún tipo de construcción y se está acostumbrado a sus propiedades, la analogía entre esta y el desarrollo de software nos proporcionará un útil común denominador. Examinando brevemente sus propiedades generales, se estará capacitado para entender la necesidad de que existan propiedades equivalentes en la industria del software.

Con el fin de proveer de una lógica a todas las fases de la construcción, es esencial que una filosofía bien establecida sirva como guía para el trabajo de todos los componentes en las diversas actividades de un proyecto de construcción. La filosofía se comprende concretamente bajo la forma de una arquitectura con actividades relacionadas que establecen la forma en que se debe actuar, tal cual se muestra en la Figura 1.2.

[2] Sistemas se refiere a la integración de hardware y software.

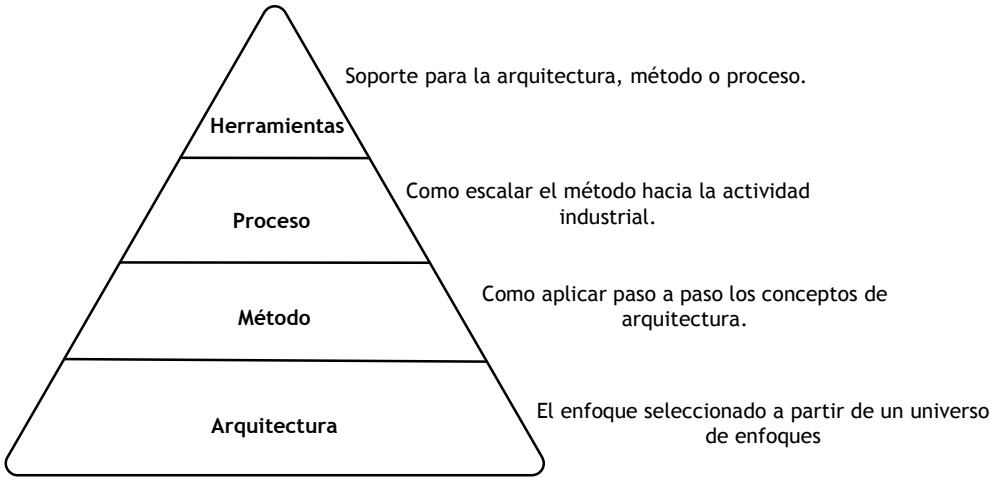


Figura 1.2: Componente de una filosofía empresarial

La *arquitectura* de un enfoque de construcción, se entiende como la fundación o base de técnicas y conceptos seleccionados de un universo de bases potenciales, que define las estructuras características de todas las construcciones diseñadas usando dicho enfoque.

El *método* hace explícito los procedimientos, paso a paso, que deben ser seguidos en la aplicación de la arquitectura a los proyectos.

El *proceso* permite que el método sea escalado de modo que pueda ser aplicado en gran escala a proyectos con muchas partes y actividades que interactúan entre sí.

Las *herramientas* dan soporte a todos los aspectos de la empresa y, explícitamente, a las actividades de arquitectura, método y proceso.

Algunas propiedades de los conceptos método y proceso son:

- Un método es más básico y es descrito igual que un proyecto, mediante la descomposición de sus distintas actividades. Un proyecto termina cuando se ha completado la última actividad y el producto (o construcción) ha sido puesto en operación.
- Un proceso, por otra parte, dura tanto como dura el producto y describe como las distintas actividades interactúan durante toda la vida del producto.

Es importante no confundir la "arquitectura" que está detrás de un método con la arquitectura de un producto en particular, el cual puede ser entendido mediante la aplicación de la arquitectura. Estas últimas representan instancias empleando la filosofía de empresa como muestra la Figura 1.3, de donde se deriva que una arquitectura puede usarse para varias construcciones y varias arquitecturas pueden usarse para una construcción específica.

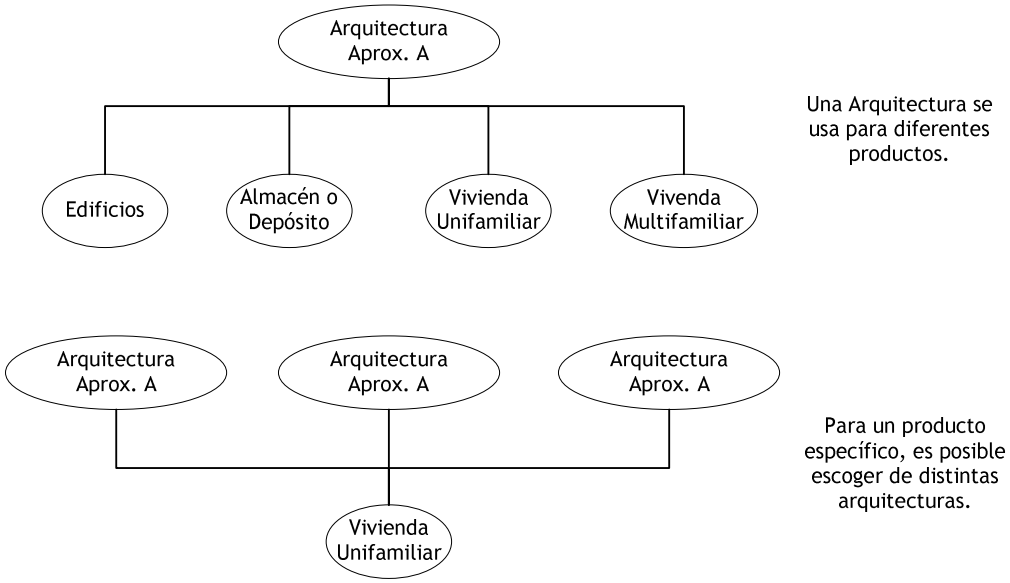


Figura 1.3: Arquitecturas

Para cada enfoque posible se pueden definir varios métodos diferentes, cada uno describe cómo trabajar con los constructores en que se basa la arquitectura, lo que lleva a una definición de procedimientos paso a paso donde por ejemplo se usa una combinación apropiada de componentes y bloques de construcción.

Los métodos deben ser escalados y relacionados a otras actividades, guiando a varios procesos posibles por cada método definido. Estos procesos pueden ser apoyados por diferentes herramientas. Consideraremos ahora como se da apoyo a las diferentes actividades de la construcción de edificios. El modelo que introdujimos es aplicado durante cada actividad de la construcción, tal como se muestra en la Figura 1.4.

Estas actividades son el diseño creativo, construcción y soporte a largo plazo, para las que se requiere de interfaces bien definidas que permitan una transición adecuada entre fases

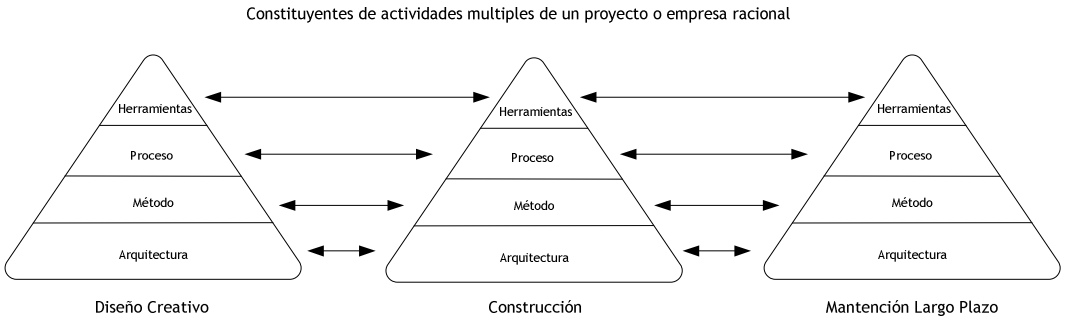


Figura 1.4: Diseño creativo, construcción y mantención.

Para cada actividad existe una filosofía (punto de vista y conceptos asociados) a partir de la cual se deriva una arquitectura particular, un método, un proceso y herramientas para la fase.

Los detalles de una actividad están directamente asociados a factores de la actividad anterior, y siempre que sea posible se debe aplicar trazabilidad que permita volver a factores relevantes cuando existan problemas.

El Diseño creativo

La transformación a partir de un conjunto de requerimientos y nociones vagas de lo que se desea, a un plan estructural del producto y a un plan de acción para su desarrollo, son actividades creativas. Los requerimientos para construir una casa por ejemplo se expresan en términos funcionales y en términos de un plano que sigue estándares de construcción especificados, dichos estándares de construcción están basados en antiguas tradiciones acerca de lo que constituye una buena casa. Con respecto a obtener y determinar que constituye un buen producto de software, existe aún un largo camino por andar.

Durante la planificación de construcción de una casa hecha a la medida, los planos de construcción y de arquitectura pueden ser la única base para examinar la construcción antes de su producción. En algunas ocasiones se utiliza un modelo de escala, sin embargo, cuando se va a construir una serie de casas, donde todas las casas tienen la misma arquitectura básica, se construye un modelo a escala y una o más casas prototipos. Los prototipos permiten a los potenciales compradores evaluar la funcionalidad de la casa en términos de sus necesidades y como un medio para ver errores y mejorar la arquitectura básica.

En la creación de construcciones modernas se está dando además una atención importante a los enfoques que explotan el uso de grandes bloques de construcción basados en la ensambladura de módulos y componentes. Esta práctica hace que la construcción a gran escala sea más económica y se asegure calidad y seguridad en el producto final.

El diseño creativo coincide con el enfoque de arquitectura y sigue los métodos y procesos paso a paso con el apoyo de herramientas, para convertir los requerimientos en un plan de arquitectura viable para el proyecto, incluyendo, cuando se requiera, la creación prototipos.

La construcción

La primera actividad en la construcción es ir de lo más abstracto hasta lo más concreto. Después de que se ha llegado a un plan suficientemente concreto se produce la construcción. Por esto la producción es la última fase de la construcción. La cantidad de gente involucrada hasta el punto de la construcción aún en proyectos grandes es bastante pequeña en comparación con el número de personas involucradas en la producción.

La producción es el resultado de la manufactura de planes de construcción más abstractos y de planes de construcción más detallados. Más aún la actividad de producción pueden tomar ventaja de cualquier modelo relacionado y/o prototipos que puedan haber sido desarrollados. Aquí podemos diferenciar entre las casas hechas a la medida y aquellas diseñadas para construcción masiva. En las primeras, la construcción es llevada a cabo por artesanos especialistas en sus áreas, en el segundo caso, se necesita gente menos experta pero que efectúen su trabajo de una forma más eficiente.

La responsabilidad para proyectos a gran escala se deja a menudo en manos del empresario quien toma la responsabilidad de la producción de acuerdo a la documentación. A partir de esto el empresario desarrolla el plan de construcción detallado, define los procesos y entrega una lista de servicios o subcontratistas que tomarán las responsabilidades dentro del proyecto. Con el fin de usar a los subcontratistas en forma eficaz, se torna vital el uso de normas, estándares, bloques y componentes de construcción. Una vez más encontramos la importancia de los métodos, procesos y herramientas que definen y documentan explícitamente los procedimientos que deben seguir el empresario y los subcontratistas.

Soporte de largo plazo

Los proyectos de construcción deben tener en cuenta que el producto existirá por largo tiempo, por lo que el enfoque de arquitectura de esta fase debe considerar los requerimientos del ciclo de vida para mantenimiento, alteración y extensión. En la industria del software, debido a la inherente flexibilidad de la alteración, es esencial una filosofía que contenga una arquitectura que permita el soporte a largo plazo.

En resumen, durante todas las actividades, desde los requerimientos originales del producto, a través de las actividades de diseño creativo hasta la construcción, producción y soporte de largo plazo, la documentación es un aspecto vital para una actividad industrial racional (coherente), la cual debe ser mantenida al día en cuanto a alteraciones, variaciones, experiencias, reusabilidad de tecnología, durante toda las fases del proyecto. En esta área las herramientas asistidas por computadora hacen su principal contribución a

su propia rama y a todas las ramas de la industria. La industria del software sin embargo debe aprender de las tradiciones de otras ramas de la industria respecto al contenido de información y administración de la documentación apropiada.

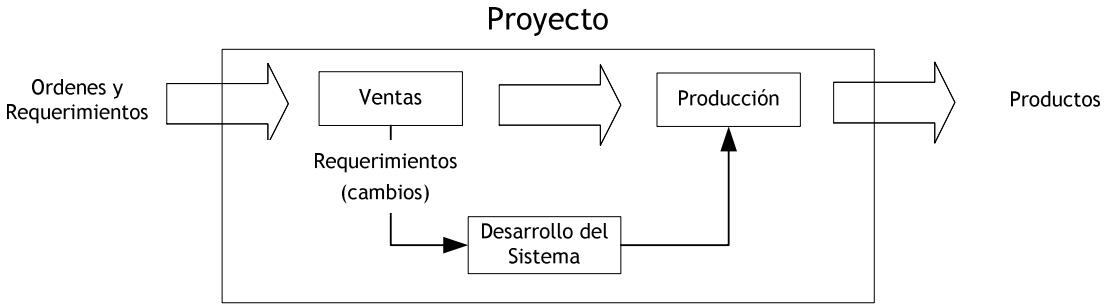
La habilidad de reutilizar tecnología que ha evolucionado a través de proyectos es una parte esencial de la rentabilidad para los involucrados en producción masiva. Los bloques de construcción que han sido identificados y explotados deben estar bien documentados y entendidos de tal forma de que puedan ser usados en nuevos proyectos. La industria del software ha comenzado a ver la importancia de los componentes durante los 80, sin embargo en un contexto amplio, la madurez asociada con la identificación y explotación de bloques de construcción útiles no ha evolucionado.
Desde nuestra caracterización del proceso de construcción basado en una arquitectura, método, procesos y herramientas, podemos hacer las siguientes observaciones respecto a los resultados de un proceso de escalamiento y derivar analogías directas con la industria del software a partir de estas observaciones:

- ✓ El proceso debe producir un resultado previsible, independiente de los individuos que realicen el trabajo.
- ✓ El volumen de salida no afecta el proceso.
- ✓ Debe ser posible asignar partes del proceso a distintos fabricantes / subcontratistas.
- ✓ Debe ser posible hacer uso de bloques de construcción y componentes predefinidos.
- ✓ Debe ser posible planificar y calcular el proceso con gran precisión.
- ✓ Cada persona entrenada para una operación, debe realizarla de una manera similar.

1.2.2 Desarrollo de sistemas como un parte de una actividad mayor

El desarrollo de un sistema es parte de una actividad mayor que, a menudo, tiene como objetivo el desarrollo de un producto en que el software es parte integral, el producto consiste en los servicios que el Departamento de informática ofrece al resto de la Empresa.

La actividad como un todo, contiene al menos dos procesos colaterales al desarrollo del sistema: ventas y producción, entre los que se produce el principal flujo de actividad, ver Figura 1.5.



El Desarrollo de Sistema, normalmente es una parte integrada del proyecto

Figura 1.5: Desarrollo del sistema, integrada.

El departamento de ventas ordena configuraciones de producto para entregar a los clientes, y formula los requerimientos para nuevos productos. Una orden debe ser formulada de modo que se pueda identificar inmediatamente la configuración del producto final. El departamento de producción entrega un sistema completo al cliente.

Debe ser posible además formular una orden en términos comprensibles para el cliente, sin la ayuda del departamento de desarrollo de sistemas. Así, no debiera existir participación de programadores en el proceso de producción; sólo personas especializadas en duplicar productos, ensamblar y reconfigurar sistemas y probarlos antes de la entrega.

El desarrollo de nuevos servicios, se inicia como resultado de nuevas demandas del cliente, convenidas con el departamento de ventas. De nuevo, se requiere de una terminología comprensible para el cliente, por lo que la participación del departamento de desarrollo es minimizada en el contacto con el cliente.

Los productos se definen como un conjunto de paquetes de servicios de funcionalidad o, paquetes de servicio. La comunicación entre los subprocesos se realiza en términos de paquetes de servicios. Estos se diseñan de modo que puedan ser usados en diferentes productos, siendo así posible construir un gran número de aplicaciones a partir de un conjunto de paquetes, ver Figura 1.6.

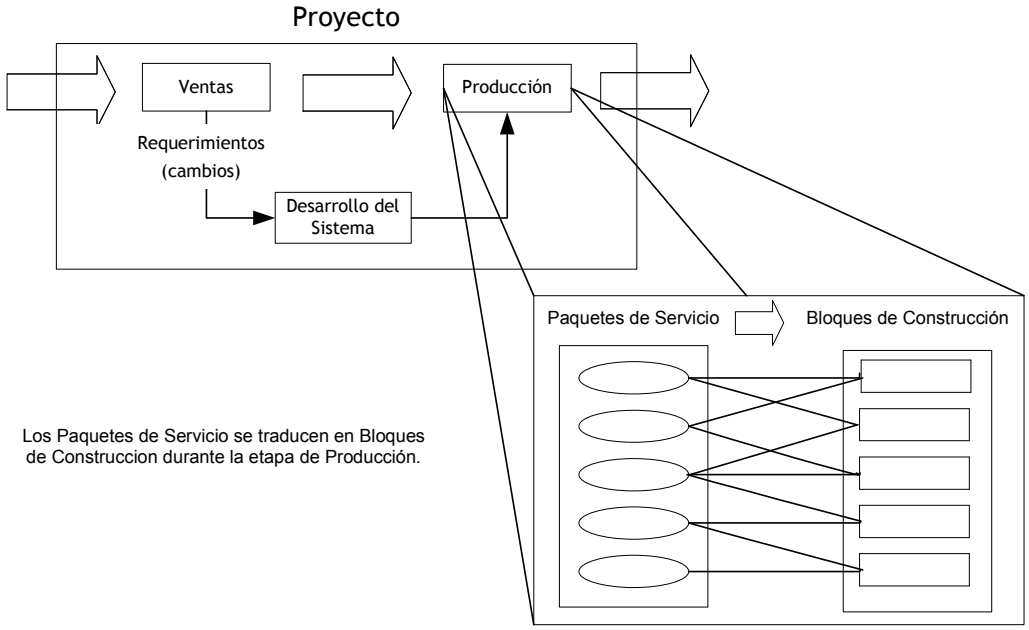


Figura 1.6: Servicios de la empresa

Las órdenes del departamento de ventas forman la base para la producción. Cada entrega a un cliente consiste en una configuración de un número de paquetes de servicios que en conjunto proveen la funcionalidad solicitada.

Desarrollo del Sistema

El desarrollo del sistema puede ser visualizado como un proceso de producción de modelos descriptivos en todos los niveles: análisis, diseño, construcción y prueba. Los primeros modelos son más abstractos (se enfocan a las cualidades externas del sistema), mientras que los últimos son detallados e instruccionales en el sentido que describen cómo se debe construir el sistema y su funcionalidad.

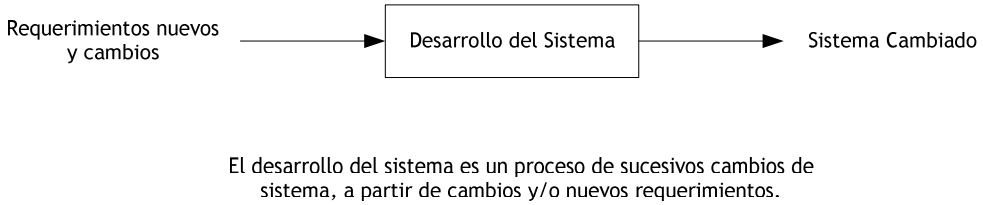


Figura 1.7: Desarrollo de sistemas

El objetivo es dividir el complicado desarrollo de un gran sistema, en actividades que permitan a varios diseñadores actuar en paralelo.

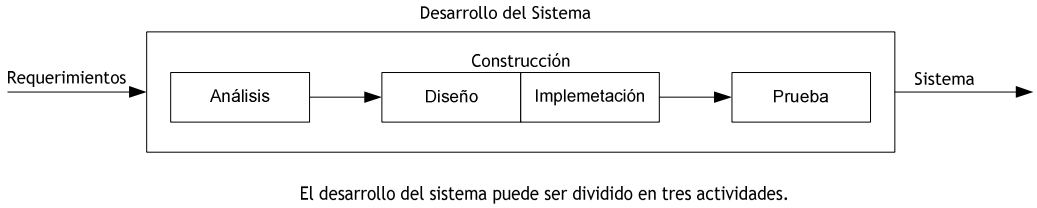


Figura 1.8: Actividades en el desarrollo de sistemas

- ✓ *Análisis:* Una especificación orientada a la aplicación (modelo de análisis) es desarrollada para especificar lo que el sistema ofrecerá al usuario, define el comportamiento del sistema en condiciones ideales e independientes de cualquier ambiente de construcción.

- ✓ *Construcción:* Las condiciones idealizadas del análisis son gradualmente reemplazadas por los requerimientos del ambiente de construcción. Esta fase define cómo el modelo de análisis orientado a la aplicación será transformado con la ayuda de software de sistema, sistemas administradores de bases de datos e interfaces con el usuario. Las actividades de la construcción son el diseño y la implementación. Las actividades de diseño formalizan el modelo de análisis en términos del ambiente de la aplicación y especifica los componentes o bloques constitutivos. La implementación consiste en programar (codificar) los bloques.

- ✓ *Prueba:* Se verifica la correcta construcción de los paquetes de servicio del modelo de análisis y el rendimiento del sistema. Se realiza en varios niveles, desde funcionalidades específicas hasta el sistema completo. La visión del proyecto se eleva a la del producto.

Transición del Análisis a la Construcción

El análisis debe ser independiente del ambiente de construcción. Los cambios en los requerimientos de construcción no deben afectar el resultado del análisis, aún cuando se reemplacen partes importantes del sistema, como por ejemplo el administrador de la base de datos, ante esto el modelo no debe ser afectado.

El modelo de análisis debe ser orientado a la aplicación. El trabajo se realiza en condiciones ideales, no se consideran aspectos técnicos como: memoria, rendimiento, tolerancia a fallas, etc.

El modelo de análisis debe describir los elementos de la aplicación sobre la base de conceptos como paquetes de servicio. Así, la estructura de la construcción refleja la estructura del problema.

El modelo de análisis no debe ser muy elaborado, ya que parte de este trabajo debe ser adaptado a la construcción (de otro modo será más difícil).

Entrada y Salida de un Desarrollo de Sistema.

La especificación de requerimientos es el principal punto de inicio del desarrollo del sistema, los que se derivan del ambiente al que debe servir el sistema.

El resultado de cualquier desarrollo de sistema es un conjunto de descripciones, que funcionan como base para la producción (departamento de producción) y descripción del producto (departamento de ventas). Documentos y productos constituyen la descripción del sistema. Todos los productos y subproductos se describen mediante un conjunto de documentos.

Los resultados de este desarrollo, son usados por otras actividades de la empresa, tanto como lo son los productos entregados a los clientes.
Los usuarios directos del sistema, a menudo trabajan para servir a usuarios indirectos.

Un sistema es desarrollado basado en una orden de trabajo de un cliente, esto genera la primera dificultad en el desarrollo del producto al tener ambas partes visiones distintas del negocio, esto dio origen al desarrollo participativo. El éxito del sistema depende fuertemente de cuan bien se han formulado y detallado las especificaciones, y llevado a un modelo formal. El departamento que desarrolla productos de software es sólo una de las partes interesadas en el desarrollo del sistema, otros departamentos como ventas, producción o servicios también influenciarán el producto.

Capítulo 2

Planificación de Proyectos de Software

La Planificación es un proceso que comienza con una misión, metas y objetivos que deben lograrse. Desarrolla planes, procedimientos, establece una organización y asigna recursos y responsabilidades con el propósito de alcanzar los objetivos propuestos. El resultado principal de la planificación es el Plan del Proyecto.

2.1 Objetivos de la Planificación de Proyectos de Software

El principal objetivo de la planificación en proyectos de desarrollo de software es ordenar el qué hacer durante el proyecto y asignar adecuadamente los recursos y tareas para cumplir los objetivos propuestos.
En general se planifica para:

- ✓ Organizar el qué hacer del proceso de desarrollo de software.
- ✓ Minimizar tiempo y costos involucrados.
- ✓ Maximizar el uso de recursos disponibles.
- ✓ Establecer hitos del proyecto.
- ✓ Medir el avance.
- ✓ Mejorar la comunicación.
- ✓ Obtener soporte técnico, de gerencia y político.

La planificación es una tarea que se desarrolla al inicio del proyecto pero rige el resto de las fases. Una buena planificación inicial ayudará a que las metas propuestas se cumplan y que los eventuales inconvenientes sean abordados de mejor forma.

2.2 Principios y consideraciones para la Planificación

Todas las organizaciones planifican, pero por lo general no se realiza de la manera adecuada, muchas veces la planificación se realiza de manera informal cuando debiera ser formal. La planificación formal es aquella que es:

- ✓ Documentada.
- ✓ Uniforme y regularmente aplicada.
- ✓ Con resultados concretos, distribuidos, entendidos y comprometidos por la organización.

En una planificación formal deben quedar claramente identificados los planes, procedimientos, la organización, la asignación de recursos y las responsabilidades.
El proceso de planificación produce idealmente un conjunto de planes, clasificados como esenciales y de soporte.
Los planes esenciales son aquellos que se consideran imprescindibles en cada proyecto, dentro de estos están: Plan de Proyecto, Plan de Pruebas y Plan de Instalación.
Los planes de soporte no siempre son necesarios, entre ellos están: Plan de Entrenamiento, Plan de Control de Cambios.

La planificación es un proceso continuo, no es un esfuerzo que se realiza una única vez en el proyecto. Si los mecanismos de control identifican algún problema, probablemente los planes deberán ajustarse a esta nueva situación.
La planificación es un proceso de toma de decisiones. No se toman decisiones futuras, sino más bien, se evalúa el impacto futuro de decisiones actuales. A medida que se planifica se decide lo que debería hacerse y lo que no. Debe comprometer a aquellos individuos que poseen la habilidad de poner en marcha las cosas, obteniendo resultados concretos.
Al planificar no se intenta eliminar el riesgo, con o sin planificación existen circunstancias que pueden atentar contra el éxito de un proyecto, la planificación no puede prevenirlos, pero puede ayudar a reducir su impacto y a controlar el riesgo.
La planificación de proyectos requiere soporte de la administración y de otras áreas organizacionales. Todo el esfuerzo puede frustrarse cuando no se cuenta con este soporte.

2.3 Ciclo de Planificación de Proyectos de Desarrollo de Software

El ciclo de planificación de proyectos de Desarrollo de Software, comienza con los requerimientos iniciales y tiene las siguientes etapas:

Negociación de Compromisos

El jefe de proyecto y el cliente y/o usuario negocian los compromisos mutuos, los cuales se establecen sobre la base de los requerimientos del producto de software y objetivos del proyecto.

Descomposición de Requerimientos

El producto de software se divide en elementos claves denominados Estructuras de División del Trabajo (EDTo WBS). Una EDT es un organigrama jerárquico donde se establecen las distintas partes de un producto de software. Representa una jerarquía de componentes o bien de procesos. La jerarquía de componentes identifica cada uno de los componentes del software y la manera en que éstos se relacionan. La jerarquía de procesos representa las actividades de trabajo requeridas para desarrollar el software y sus interrelaciones. Si se usa este tipo de EDT se deben considerar las fases, actividades y tareas estándares definidas por la organización y también las tareas especiales del proyecto.

Estimación del Tamaño de un producto de Software

Una vez establecido el estándar de medición (Líneas de Código, Puntos de Función, Puntos Objetos), se utiliza la EDT de componentes para estimar el tamaño de cada componente del software. El tamaño total del producto de software se obtiene al sumar los valores estimados para cada componente y al ajustar la estimación de acuerdo a la información histórica de la organización, si es necesario.

Estimación de Recursos

El tamaño del producto de software sirve de base para estimar esfuerzo (Persona-Mes, Hombres-Hora), tiempo y costo de desarrollo. Los modelos empíricos de estimación de costos de software cumplen éste propósito. La estimación de recursos puede hacerse en el ámbito de proyecto, de fases y de actividades y tareas.

Desarrollo de Itinerario del Proyecto

El itinerario del proyecto se confecciona distribuyendo el esfuerzo estimado dentro del marco de tiempo establecido. El itinerario debe considerar los hitos del proyecto.

Término de fase y/o actividades.

El término de cada fase o actividades se establece formalmente y define un hito o un producto.

Generación y entrega de productos.

En ciertas partes de itinerario es necesario que la actividad de generar el producto sea explicita. Generalmente en proyectos de SW el producto es un informe.

Puntos de control o Hitos del proyecto

El itinerario y las estimaciones resultantes se comparan con las necesidades iniciales, si éstos se ajustan, los compromisos pueden ser hechos y el trabajo puede proceder. Generalmente los costos son muy altos y el itinerario demasiado largo, en este caso se requiere volver a la negociación de compromisos y replanificar, si es necesario.

La existencia de una base de datos que registre información histórica de los proyectos de Desarrollo de Software de una organización, permite contar con factores de ajuste para estimaciones futuras, mejorando progresivamente el proceso de planificación.

2.4 Plan del Proyecto de Desarrollo de Software.

Como se menciono anteriormente, el Plan del Proyecto es el resultado principal del proceso de planificación. Este plan existe sólo cuando está documentado, distribuido, entendido y comprometido.

El contenido básico del plan del proyecto se indica en la Figura 2.1

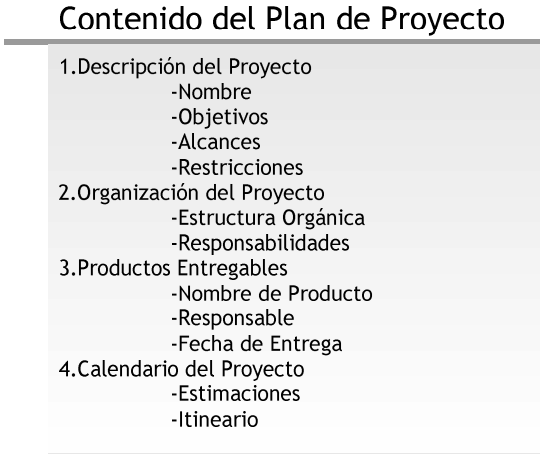


Figura 2.1: Contenido del Plan de Proyecto

La Descripción del Proyecto proporciona las características generales de éste. La Organización refleja la forma en que el grupo de proyecto ha sido estructurado para llevar a cabo el trabajo y los responsables de las funciones clave. Los Productos a Entregar incluyen los documentos u otro tipo de

producto, con compromiso de entrega al usuario o a otros grupos de trabajo interno del proyecto, así como los responsables de la entrega. El Calendario comprende tanto las estimaciones realizadas para confeccionar y justificar el itinerario del proyecto, como éste mismo.

2.4.1 ¿Para qué se usa el plan del proyecto?

Los proyectos de Desarrollo de Software involucran a diversos participantes y cada uno de ellos da un uso distinto al plan del proyecto, ver Tabla 2.1.

Jefe Proyecto	◆ Organizar el qué hacer del proyecto. ◆ Prever problemas. ◆ Identificar problemas. ◆ Conocer el estado del proyecto. ◆ Medir el avance. ◆ Obtener soporte técnico, gerencial y político.
Desarrolladores	◆ Conocer objetivos, alcances y restricciones del proyecto. ◆ Conocer la relación de actividades y tareas. ◆ Determinar los esfuerzos individuales
Usuario	◆ Medir el progreso y utilización de recursos. ◆ Conocer los hitos del proyecto. ◆ Recordar los compromisos establecidos

Tabla 2.1: Uso del Plan del Proyecto.

2.5 Fallas en la Planificación.

Los problemas más comunes que enfrentan hoy en día los proyectos de desarrollo de software, tales como: retrasos en la entrega del producto final, aumento de los costos de desarrollo y mantenimiento y escasa calidad del software, se deben principalmente a una mala o escasa planificación.
Las principales causas de las fallas en la planificación de proyectos de Desarrollo de Software son las siguientes:

- ✓ Inadecuada definición del proyecto.
- ✓ Comprensión errónea del problema.
- ✓ Desconocimiento o inexperiencia de cómo planificar.
- ✓ Incumplimiento del ciclo de planificación.
- ✓ Escasa negociación de compromisos con el usuario al inicio del proyecto
- ✓ Definición incompleta de los requerimientos.
- ✓ Estimaciones optimistas.
- ✓ Supuestos y restricciones del proyecto inválidos o no verificados.
- ✓ Aplicación errónea o no-utilización de la información histórica de la organización.
- ✓ Mala administración del proyecto.
- ✓ Fallas en el uso de los planes.
- ✓ Carencia de control de cambios.
- ✓ Escasa motivación.
- ✓ Estilo erróneo de liderazgo.
- ✓ Carencia de control y gestión.
- ✓ Organización errónea del grupo de trabajo.

2.6 Especificación de Requerimientos

La especificación es el resultado del proceso de planificación y puede ser vista como un proceso de representación. La ejecución del plan concluye en la instanciación de un producto o proceso en particular. La especificación del producto describe la visión externa del producto. La especificación del proceso describe cómo realizar un determinado proceso.
La especificación de requerimientos es una descripción detallada y precisa de la funcionalidad del sistema teniendo en cuenta las restricciones del mismo.
Generalmente, la especificación de requerimientos sirve como base para el contrato entre los desarrolladores y el cliente.
Como ejemplo, la especificación del diseño de Software. Esta puede contener:

- ✓ La especificación del tipo de producto de entrada (producto de los requerimientos), incluyendo una sintaxis formal y descripción semántica para el documento de requerimientos (ANSI / IEEE-Std-830).
- ✓ La especificación del tipo de producto de salida (producto del diseño), incluyendo una sintaxis formal y la descripción semántica para el documento de diseño.
- ✓ La especificación del tipo de proceso (proceso de diseño), incluyendo una pauta para el uso de una técnica de diseño específica, como Diseño estructurado o Diseño Orientado a Objetos (DOO).

Esta fase trata de aclarar qué es lo que un sistema debe de hacer. Describe la función y el rendimiento del sistema y las restricciones que gobernarán su desarrollo. También describe la información (control y datos) que sirve de entrada y salida al sistema. Es evidente que, en esta etapa, el sistema objetivo está sujeto a muchos cambios antes de que sea realmente implantado.

Para la especificación Balzer y Goldman proponen ocho principios[6] para una buena especificación:

Principio 1: *Separar funcionalidad de implementación.*
Las especificaciones deben describir que se desea realizar, no cómo se va a realizar.

Principio 2: *Se necesita utilizar un lenguaje de especificación de sistemas orientado a procesos.*
Si se considera un entorno dinámico, donde los cambios afectan al comportamiento de algunas entidades, entonces los sistemas no pueden ser representados formalmente. Por lo tanto, se puede usar una descripción orientada al proceso, en la cual la especificación se obtiene mediante un modelo de comportamiento deseado en términos de respuestas funcionales ante distintos estímulos del entorno. Ejemplo, Sistemas Empotrados.

Principio 3: *Debe abarcar el sistema del cual el software es un componente.*

Un sistema resulta de la interacción de sus componentes. Sólo dentro del contexto del sistema completo y de la interacción entre sus partes puede ser definido el comportamiento de un componente específico.

Principio 4: *Debe abarcar el entorno en el cual opera el sistema.*
Se debe especificar el entorno en el cual el sistema opera e interactúa. Es necesario reconocer que el propio entorno es un sistema compuesto de objetos que interactúan, pasivos y activos. Con ello se permite especificar la "interfaz" del sistema.

Principio 5: *Debe ser un modelo cognitivo.*
La especificación debe ser un modelo cognitivo, en vez de un modelo de diseño o de implementación, debe describir un sistema tal como es percibido por su comunidad de usuarios y los objetivos que manipula deben corresponderse con objetivos reales de dicho dominio.

Principio 6: *La especificación debe ser operacional.*
La especificación debe ser completa y lo suficientemente formal para que pueda usarse para determinar si una implementación propuesta satisface la especificación, en casos de prueba elegidos arbitrariamente.

Principio 7: *La especificación debe ser tolerable con la incompletitud y ampliable.*
La especificación nunca esta totalmente completa ya que el entorno en el que existe es demasiado complejo para esto, por lo que la especificación es una abstracción de alguna situación real o imaginada. Las herramientas de análisis que apoyan y prueban la especificación deben ser capaces de tratar con la incompletitud. Esto debilita el análisis.

Principio 8: *Debe ser localizada y débilmente acoplada*
Los principios anteriores tratan con la especificación como una entidad estática. Aún cuando la especificación debe servir como base al diseño y la implementación, no es un objeto estático sino más bien dinámico ya que sufre considerables modificaciones. Estas modificaciones se presentan tres actividades: formulación, desarrollo, y mantenimiento.

2.6.1 Terminología y modelo de referencia para el ciclo de vida del Software.

Las especificaciones del proceso y los productos se crean para, se usan en, son afectadas por y son modificadas durante cada fase en particular. Las fases de acuerdo al modelo adoptado son: necesidades del Software, requerimientos del cliente, requerimientos del diseñador, diseño del Software y construcción. Se pueden incluir otras fases: verificación y validación, integración, mantenimiento y entrenamiento.

El modelo de referencia es el que se muestra en la Figura 2.2, en la columna de la izquierda. Aquí se distinguen los siguientes tipos de productos:

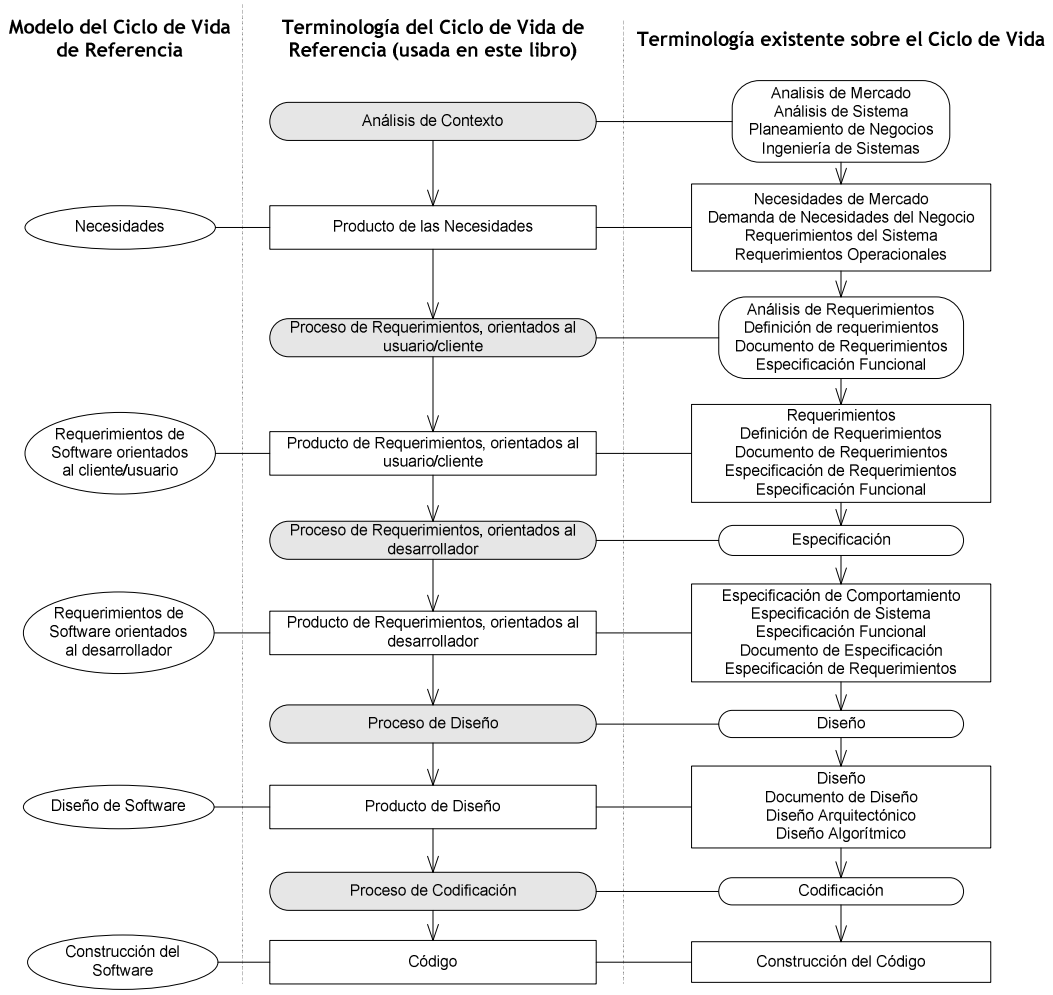


Figura 2.2: Terminología y Modelo del Ciclo de Vida de Referencia

Etapa	Tipo de Pregunta
Necesidades del Software	¿Qué demanda existe? ¿Qué necesidades debe satisfacer el producto de software?
Requerimientos del software orientados al usuario/cliente	¿Qué características, funcionales o no, debe tener el producto para satisfacer las necesidades desde el punto de vista del cliente?
Requerimientos del software orientados al desarrollador	¿Qué características, funcionales o no, debe tener el producto para satisfacer las necesidades desde el punto de vista del desarrollador de Software?
Diseño del software	¿Cómo construir el producto para comportarse en la forma descrita en los requerimientos orientados al desarrollador?
Codificación	¿Cómo construir realmente este producto en una máquina y qué tecnología en particular se usará?

Tabla 2.2: Preguntas para cada etapa del Ciclo de Vida Estándar

2.6.2 De la especificación del Software

Propósito y Contexto.

La especificación describe todas las características importantes de un producto o proceso particular en algún formato. Las características deseables, así como el formato adecuado para representarlas, están determinadas por el propósito y contexto del tipo dentro del proyecto de desarrollo de Software.

Perspectiva del producto.

Las especificaciones están dirigidas a crear los productos del ciclo de vida (que son los productos del proyecto) a satisfacción del usuario. Se caracterizan los tipos de productos por aplicación y requerimientos de calidad.

Tipo de Aplicación.

El tipo de aplicación tiene un fuerte impacto en los productos y procesos que necesitan ser especificados. Las posibles clasificaciones son basadas en las características de los flujos de control del Software (secuencial, concurrente, tiempo real), o basadas en la aplicación (comercial, sistemas, control de procesos, científica, integrada).

Requerimientos de calidad.

Estas características tienen impactos en los aspectos que deben ser especificados y en sus atributos. Algunas características posibles son: confiabilidad, correctitud, tolerancia a fallas, mantenibilidad, portabilidad, amigabilidad, disponibilidad.

Perspectiva del proceso.

Se caracteriza en términos del modelo de ciclo de vida y las fases individuales. Los modelos del ciclo de vida de un proyecto se analizan con mayor detalle en el Anexo B.

2.6.3 Procesos y productos del ciclo de vida.

Comunicación.

La existencia de las especificaciones permite que los miembros del equipo alcancen un consenso acerca de sus roles, haciendo explícitos los objetivos, contexto y procedimientos del proyecto. Son un medio para enseñar y entrenar al personal.

Creación de productos.

Muchas de las tareas de un proyecto están orientadas a crear, en una forma trazable, instancias de un tipo de producto en otro (por ejemplo, un producto del diseño a partir de los requerimientos del Diseñador). Las especificaciones explícitas para los tipos de productos y los procesos creativos, ayudan a guiar y controlar las tareas. Si todas las especificaciones son completas y formales, el producto deseado puede ser creado automáticamente.

Modificación de productos y procesos.

Los proyectos de Software deben adaptarse oportunamente a los cambios. Un cambio o aumento de requerimientos durante el mantenimiento requiere modificar productos existentes, cambiando o no las especificaciones del producto. El cambiar el proyecto o las características ambientales, ya sea agregando nuevo personal o introduciendo nueva tecnología, requiere modificar los procesos existentes y posiblemente las especificaciones subyacentes. La existencia de especificaciones explícitas del producto y el proceso, permite incorporar cambios en forma sistemática.

Productos de validación y verificación.

El propósito de la V&V es demostrar que algún producto del ciclo de vida, el código por ejemplo, es consistente con el producto del ciclo de vida de tipo diferente, el producto del Diseño por ejemplo. Este tipo de chequeo cruzado entre productos se facilita si existen especificaciones explícitas.

Certificación de cumplimiento a lo planificado.

La certificación de calidad del Software (SQA) se refiere a la certeza de que el desarrollo de Software se realiza de acuerdo al plan. Mucho del trabajo de SQA es comparar los productos de Software y los procesos con sus especificaciones. Es importante tener en cuenta los estándares.

Perspectiva del personal.
Las especificaciones se crean o son usadas por diferentes audiencias que juegan diversos roles en el proyecto. Aunque algunas especificaciones están dirigidas para ser usadas por máquinas, las personas deben comprenderlas. Las personas que participan en un proyecto de software son:

- ✓ Cliente.
- ✓ Usuario final.
- ✓ Sub contratista.
- ✓ Analista de requerimientos.
- ✓ Ingeniero de especificación.
- ✓ Diseñador.
- ✓ Codificador.
- ✓ Personal de Verificación y Validación.
- ✓ Personal de SQA (Software Quality Assurance).
- ✓ Personal de SCM (Software Configuration Management)
- ✓ Personal de Mantenimiento
- ✓ Gerente.

Es por esto que los documentos deben ser elaborados con tal claridad, que toda persona que lo lea y analice, debe entender los mismos requerimientos. En otras palabras la documentación debe ser clara, completa y precisa.

Contenido.

El contenido se caracteriza por aquellos aspectos y atributos necesarios para el producto o proceso.

Aspectos especiales.

Se requieren algunas definiciones previas para caracterizar el contenido tanto del producto como del proceso.

- ✓ *Dinámica.* Característica relativa al uso. En un proceso pueden ser capturadas durante su ejecución, por ejemplo, el conjunto de decisiones tomadas por el diseñador o datos históricos sobre la cantidad de tiempo requerida para el diseño en proyectos pasados.
- ✓ *Estática.* Características de un objeto relativo a su representación. Las características estáticas de un proceso tienen que hacerse durante su especificación (los pasos en un proceso de Diseño) Los aspectos estáticos en un producto se describen en el producto en sí y en sus especificaciones (estructura de datos o estructuras de control algorítmicas).
- ✓ *Funcionales.* Características de un objeto de cualquier tipo relativo a sus requerimientos de funcionamiento. (funciones como almacenar y recuperar).
- ✓ *No funcionales.* Se identifican al analizar cómo son provistos los servicios por el objeto (una de las funciones del producto debe ser provista en un tiempo menor que t, el producto del diseño debe ser

obtenido para un determinado proceso dentro de un lapso de tiempo y presupuesto dado).

- ✓ *Externa.* Característica de un objeto visto como caja negra.
- ✓ *Interna.* Característica de un objeto visto como caja blanca.

Se utilizarán estas definiciones para caracterizar y explicar aspectos del producto o proceso que se desean establecer en las especificaciones.

- ✓ *Comportamiento (externo, dinámico):* La respuesta externa observable del producto o proceso frente a un estímulo estando en uso real. Puede incluir estados externos observables, salidas o condiciones de borde en la validez de entradas y estados.
- ✓ *Interfaces (externa, estática):* La estructura de la frontera entre el producto o proceso y su ambiente.
- ✓ *Flujo (interno, dinámico):* La dinámica interna de un producto o proceso en uso. Esto puede incluir los flujos de control, data, e información entre las unidades estructurales del producto o proceso.
- ✓ *Estructura (interna, estática):* La organización de un producto o proceso en partes interactuantes. Incluye la descomposición en unidades básicas. La estructura de datos, algoítmico, o de arquitectura, a sí como las interfaces internas entre superestructuras, es de interés.

Atributos.

En general cada uno de los aspectos anteriores puede ser representado en una variedad de formas. El propósito y contexto del producto o tipo de proceso de interés, requieren una forma adecuada para mostrar ciertos atributos.
Por ejemplo, si el flujo de datos de un producto de diseño necesita ser validado, se puede especificar que su representación necesita exhibir los atributos de consistente y ejecutable.

Representación.

Ciertos aspectos del Software deben ser representados de manera de exhibir los atributos deseados. El formato de representación usado se basa en modelos y lenguajes. Los modelos permiten la formulación de aspectos de interés. Los lenguajes permiten un reflejo fiel de esos modelos en una forma tal que muestra los atributos deseados.
Algunos modelos son:

- ✓ Modelos funcionales.
  Input - output, algebraicos, axiomáticos.
- ✓ Modelos de estado finito.
  Cartas de estado.
- ✓ Modelos de estímulo repuesta.
- ✓ Modelos de redes de Petri.
- ✓ Modelos de estructura de datos.
- ✓ Modelos de flujo de información.
- ✓ Modelos de estructura de datos.
- ✓ Modelos entidad relación.
- ✓ Modelos relacionales.

Lenguajes.

Se distinguen entre Formales, semiformales e informales. Hay diferentes paradigmas de lenguajes: imperativo, declarativo u orientado a los datos. La elección del lenguaje se acota con el problema, las habilidades del equipo de desarrollo y los requerimientos específicos del cliente.

Soporte.

Es necesario tener un soporte eficaz para crear las especificaciones.

2.6.4 Lenguaje Unificado de Modelamiento (UML)

Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Su aplicación está en entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir.
UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. Las herramientas pueden ofrecer generadores de código de UML para una gran variedad de lenguaje de programación, así como construir modelos por ingeniería inversa a partir de programas existentes.

Los objetivos de UML se pueden resumir en:

- ✓ Ser un lenguaje de modelado de propósito general que pueden usar todos los modeladores. No tiene propietario y está basado en el común acuerdo de gran parte de la comunidad informática.
- ✓ No pretende ser un método de desarrollo completo. No incluye un proceso de desarrollo paso a paso. UML incluye todos los conceptos que se consideran necesarios para utilizar un proceso moderno iterativo, basado en construir una sólida arquitectura para resolver requisitos dirigidos por casos de uso.
- ✓ Ser tan simple como sea posible pero manteniendo la capacidad de modelar toda la gama de sistemas que se necesita construir. UML necesita ser lo suficientemente expresivo para manejar todos los conceptos que se originan en un sistema moderno, tales como la concurrencia y distribución, así como también los mecanismos de la ingeniería de software, como son la encapsulación y componentes.
- ✓ Debe ser un lenguaje universal, como cualquier lenguaje de propósito general.
- ✓ Imponer un estándar mundial.

La Arquitectura es de cuatro capas, definida a fin de cumplir con la especificación Meta Object Facility del OMG[4] :

- ✓ *Meta-metamodelo:* define el lenguaje para especificar metamodelos.
- ✓ *Metamodelo:* define el lenguaje para especificar modelos.
- ✓ *Modelo:* define el lenguaje para describir un dominio de información.
- ✓ *Objetos de usuario:* define un dominio de información específico.

Representación de UML

UML se encuentra diseñado para dibujar los esquemas en superficies bidimensionales, considerando que algunas formas corresponden a proyecciones tridimensionales en el plano bidimensional.
Hay cuatro clases de construcciones gráficas que se usan en la notación de UML: íconos, símbolos bidimensionales, rutas y cadenas.

Un **icono** es una figura gráfica con un tamaño y forma fijos. Pueden aparecer dentro de símbolos de área, como terminales en las rutas o como símbolos independientes que puedan o no conectar con las rutas.

Los **símbolos** de dos dimensiones tienen alto y ancho variables, y pueden ampliarse para permitir otras cosas tales como listas de cadenas o de otros símbolos. Muchos de ellos están divididos en compartimientos similares o de tipos diferentes.

Las **rutas** se conectan con los símbolos, el arrastrar o suprimir uno de ellos afecta a su contenido y las rutas conectadas. Una ruta es una secuencia de segmentos de recta o de curva que se unen en sus puntos finales.

Las **cadenas** presentan varias clases de información en una forma "no analizada", UML asume que cada uso de una cadena en la notación tiene una sintaxis por la cual pueda ser analizada la información del modelo. Las cadenas pueden existir como el contenido de un cuadro, como elementos en las listas, como etiquetas unidas a los símbolos o a las rutas, o como elementos independientes en un diagrama.

Sobre la base de las definiciones anteriores, UML contempla una serie de diagramas que ayudan al desarrollo. La tabla siguiente, señala los diagramas de acuerdo al área particular.

AREA	VISTA	DIAGRAMA	CONCEPTOS PRINCIPALES
Estructural	Vista Estática	Diagrama de Clases	Clase, asociación, generalización, dependencia, realización, interfaz.
	Vista de Casos de Uso	Diagramas de Casos de Uso	Casos de Uso Caso de Uso, actor, asociación, extensión, generalización.
	Vista de Implementación	Diagrama de Componentes	Componente, interfaz, dependencia, realización.
	Vista de Despliegue	Diagrama de Despliegue	Nodo, componente, dependencia, localización.
Dinámica	Vista de Estados de Màquina	Diagramas de Estado	Estado, evento, transición, acción.
	Vista de Actividad	Diagramas de Actividad	Estado, actividad, transición, determinación, división, unión.
	Vista de Interacción	Diagramas de Secuencia	Interacción, objeto, mensaje, activación.
		Diagramas de Colaboración	Colaboración, interacción, rol de colaboración, mensaje.
Gestión de Modelo	Vista de Gestión de Modelo	Diagramas de Clases	Paquetes, subsistema, modelo.
Extensión de UML	Todas	Todos	Restricción, estereotipo, valores, etiquetados.

Tabla 2.3: Diagrama UML

2.7 Metodología de Desarrollo de Productos de Software

La metodología de desarrollo a estudiar, está definida en términos de dos procesos que cubren la realización de proyectos:

- ✓ Proceso de Definición del Proyecto
- ✓ Proceso de Desarrollo de Software

El proceso de Definición del Proyecto comprende las actividades de la Ingeniería de Sistemas necesarias para establecer las bases del proyecto a realizar. *El proceso de Desarrollo de Software* está enmarcado dentro de un modelo de desarrollo (Modelo Espiral, por ejemplo) que resulta ser altamente dinámico y flexible, ya que facilita el desarrollo de diversos tipos de productos de software, permite la incorporación de técnicas modernas de ingeniería de software, no presenta rigidez frente a los cambios y puede utilizarse para desarrollar un producto de software completo o por componentes individuales.

2.7.1 Proceso de Definición del Proyecto

El objetivo del proceso de definición del proyecto está orientado a establecer las bases del proyecto. Está compuesto por tres actividades:

✓ Identificación de Necesidades

Objetivo: Identificar y estudiar las necesidades del usuario a fin de establecer el origen del problema, objetivos, alcances y restricciones del sistema computacional a construir.

Tareas	Establecer el Origen del Problema. Establecer los Objetivos del Sistema Computacional. Establecer los Alcances del Sistema Computacional. Establecer las Restricciones del Sistema Computacional

✓ Desarrollo de Alternativas de Configuración del Sistema Computacional

Tareas	Estudiar los Elementos del Sistema Computacional. Integrar los Elementos del Sistema Computacional.
Consideraciones	Esta actividad puede causar la redefinición de alcances y restricciones establecidos en la actividad de Identificación de Necesidades

Objetivo: Establecer las posibles configuraciones del sistema computacional concordantes con los objetivos, alcances y restricciones establecidos en la actividad anterior.

✓ Estudio de Prefactibilidad del Proyecto

Objetivo: Evaluar la factibilidad de la implantación de las alternativas de configuración del sistema computacional propuestas.

Tareas	Evaluar la Factibilidad Legal de las Configuraciones propuestas. Evaluar la Factibilidad Técnica de las Configuraciones propuestas. Evaluar la Factibilidad Operativa de las Configuraciones propuestas. Evaluar la Factibilidad Económica de las Configuraciones propuestas.
Contenido	Documento Prefactibilidad del Proyecto

2.7.2 Proceso de Desarrollo de Software

El objetivo de este proceso es transformar las necesidades del usuario en un producto de software aprobado y certificado para su operación. El proceso de desarrollo está compuesto por las siguientes fases:

✓ Definición de Requerimientos

Objetivo	Identificar, analizar y documentar los requerimientos del producto de software a desarrollar.
Actividades	Identificación de Requerimientos. Documentación de Requerimientos.
Productos	Documento de Especificación de Requerimientos. Inicio Documento Manual de Usuario

- ✓ Diseño

Objetivo	Modelar los datos involucrados en el producto de software, diseñar la estructura y especificar proceduralmente los componentes de éste.
Actividades	Diseño Preliminar. Diseño Detallado. Diseño de la Base de Datos
Consideraciones	Si se utiliza Análisis Estructurado, este incluye la construcción de Diagramas de Flujo de Datos (DFD), Diccionario de Datos (DD) y Miniespecificaciones. Otra técnica para modelar el problema es construir el Modelo de Datos y el Diccionario de Datos. Se recomienda construir el Diagrama de Estructura del Diseño Estructurado. Esta actividad es obligatoria para el desarrollo de productos orientados a Bases de Datos. Si se ha utilizado la Técnica de Análisis Estructurado para modelar el problema, es necesario considerar además: ◆ Los archivos del DFD corresponden directamente a entidades en la termino- logía de modelado de datos. ◆ Los flujos de datos pueden contener entidades o atributos de entidades. ◆ El Diccionario de Datos contendrá, ya sea como archivos o Flujos de Datos, a las entidades representadas en el Modelo de Datos.
Producto	Documento de Diseño. Documento Especificaciones de Codificación.

- ✓ Codificación

Objetivo	Probar individual e integradamente los componentes del producto de software, para encontrar errores y comprobar la satisfacción de los requerimientos.
Actividades	Prueba Unitaria. Prueba de Integración Prueba de Aceptación.
Producto	Documento de Prueba. Documento Guía de Operación. Documento Manual de Usuario. Documento Manual del Producto de Software.

- ✓ Pruebas

Objetivo	Instalar el producto de software en su ambiente operacional.
Actividades	Preparación del Ambiente Operacional. Traspaso a Producción. Capacitación y Entrenamiento al Usuario. Formalización del Término del Proyecto.
Producto	Formulario de Instalación. Nota Interna de Término de Proyecto.

- ✓ Instalación

Objetivo	Instalar el producto de software en su ambiente operacional.
Actividades	Preparación del Ambiente Operacional. Traspaso a Producción. Capacitación y Entrenamiento al Usuario. Formalización del Término del Proyecto.
Producto	Formulario de Instalación. Nota Interna de Término de Proyecto.

2.8 Herramientas de Apoyo al Proceso de desarrollo

La Tabla 2.4 muestra un resumen de las etapas y las herramientas que las apoyan:

PLANIFICACIÓN	✓ **Administración de Riesgos**
ANÁLISIS	✓ Análisis Estructurado. ✓ Prototipo. ✓ Desarrollo Automatizado.
DISEÑO	✓ Diseño Estructurado. ✓ Diseño Orientado a Objetos. ✓ Prototipo. ✓ Desarrollo automatizado.
PROGRAMACIÓN	✓ Programación Estructurada. ✓ Programación orientada a objetos. ✓ Prototipo. ✓ Desarrollo Automatizado.

Tabla 2.4: Etapas y Herramientas

Capítulo 3

Estructura Orgánica en Proyectos de Software

La organización del grupo de proyecto es otra de las funciones de la Administración de Proyectos. Existen tres tipos de estructura orgánica para los proyectos de desarrollo de software:

- ✓ Formato de Proyecto
- ✓ Formato Funcional
- ✓ Formato Matricial

A medida se pasa de la organización jerárquica tradicional hacia tipos de organización por proyectos y en redes, la idea de puestos de trabajo fijos también empieza a modificarse. La misión y responsabilidad fluctúan de acuerdo con el encargo de cada momento.

Estamos dando paso cada vez más al *puesto basado en objetivos concretos,* por lo que las responsabilidades se modifican también de acuerdo con este cambio. Todos los puestos tienen por objeto satisfacer las necesidades del cliente, con el énfasis puesto en sus "necesidades".

El resultado para la organización es que las relaciones se basan menos en líneas jerárquicas funcionales predeterminadas, y más en la necesidad de la interacción para conseguir metas concretas. *Los individuos adquieren mayor autonomía* en su funcionamiento y la autoridad se basa en sus competencias y capacidad de aprendizaje, en lugar de basarse en responsabilidades previas.

Esto da lugar a *organizaciones por proyectos* y estructuras más transversales, que aseguran la coherencia de los planes de acción. La excelencia es el motor de la jerarquía informal. Las organizaciones que se basan en polos de excelencia son el resultado de nuestra ambición de ser el número uno en todo lo que hacemos. Pueden ser formales o informales, reales o virtuales. Pueden materializarse en la forma de Centros de Servicio Compartidos que alojan centros de excelencia y crean sinergias.

Los Factores a considerar en la selección de la Estructura Organizacional del Proyecto de Desarrollo de Software son:

- ✓ Tamaño del producto a desarrollar.
- ✓ Número de proyectos. Pocos proyectos grandes (mayores de 20 personas), muchos proyectos pequeños (menores de 10 personas).
- ✓ Alcance del desarrollo. Tipo de actividades distintas que se realizan en un instante dado.
- ✓ Ambiente. La estructura organizacional debe reconocer y ser capaz de operar en la cultura organizacional de la empresa en la cual esta inserta.

- ✓ Limitaciones físicas. Proyecto es desarrollado localmente o en diversos lugares distantes entre sí.
- ✓ Cultura organizacional. Cual es el estilo de gestión de la organización. Con que estructura se siente más cómodo el ejecutivo superior.

3.1 Formato de Proyecto.

En el formato de proyecto el grupo de trabajo esta formado por desarrolladores que llevan a cabo el proyecto de principio a fin. Realizan las tareas involucradas en las fases de Definición de Requerimientos, Diseño, Codificación y Prueba, además de las revisiones del producto y la documentación. Algunos miembros del equipo de desarrollo pueden permanecer durante la Instalación y Mantenimiento, mientras otros participan en nuevos proyectos, sin dejar de lado la responsabilidad del mantenimiento del producto de software entregado.

Organización por proyecto.

Es efectiva cuando los proyectos son pequeños y cada proyecto tiene una sola ubicación. Por lo menos el 70% de los recursos necesarios están bajo el control directo del Jefe de Proyecto, quien cumple los roles de Jefe Técnico y Jefe Administrativo.

En junio de 2001 se publicaron los resultados de un estudio del IDIED donde se identificaron y analizaron 24 empresas basadas en la innovación o EBIs (definidas como empresas que crean soluciones tecnológicas) de los sectores del *software* y las aplicaciones electrónicas. Recientemente se publicó un segundo trabajo, continuación del anterior, donde se analizan en detalle las estrategias de innovación y las formas de organización de ocho de esas empresas, que representan el 84,6% de la facturación, el 73% de las exportaciones y el 79% del empleo del grupo original. El estudio brindó resultados interesantes para la gestión de empresas, no sólo las basadas en la innovación, y para la política pública.
Las empresas entrevistadas presentan un proceso de mutación en marcha en la industria que refleja en este sector tendencias más amplias y generales en la administración de empresas. En particular, se encontraron dos evidencias que abren la puerta para futuras investigaciones:

- ✓ *Las empresas se encuentran en un proceso de cambio de paradigma de organización.* Esta mutación obedece, por un lado, a los cambios que se verifican en el entorno y a las adaptaciones de las empresas al mismo, y por otro, a la evolución de las empresas y la sofisticación de sus procesos de gestión.
- ✓ *El modo (las rutinas), a través del cual los clientes participan en el proceso de desarrollo de los productos está evolucionando.* Este cambio incluye el uso intensivo de la experimentación y el trabajo con usuarios avanzados. Esta metodología permite reducir costos derivados de errores comerciales y de diseño y maximiza las posibilidades comerciales del producto.

Las evidencias encontradas permiten extraer las siguientes premisas:

I. De la definición por productos a la definición por competencias

Las empresas han dejado progresivamente de definirse en función de los productos que hacen y a hacerlo en función de las habilidades o competencias centrales que les permiten desarrollar esos productos. Este enfoque desarrollado por primera vez por Hamel y Prahalad (1990) ha dado paso a una variación sobre el mismo tema.
En particular, las empresas reconocen explícita o implícitamente que sus competencias básicas no son las tecnológicas sino, crecientemente, las asociadas a habilidades organizacionales:

- ✓ Identificación de oportunidades de negocio.
- ✓ Diseño de productos (soluciones).
- ✓ Gestión de proyectos.

II. De la organización funcional a la organización por proyectos

Es natural que esta forma de organización de la producción tenga su correlato en un esquema organizacional que pivotee, no ya en las funciones básicas de la empresa (administración, finanzas, producción, comercialización), sino en los proyectos que se transforman en la forma básica de organización de las actividades de la empresa.

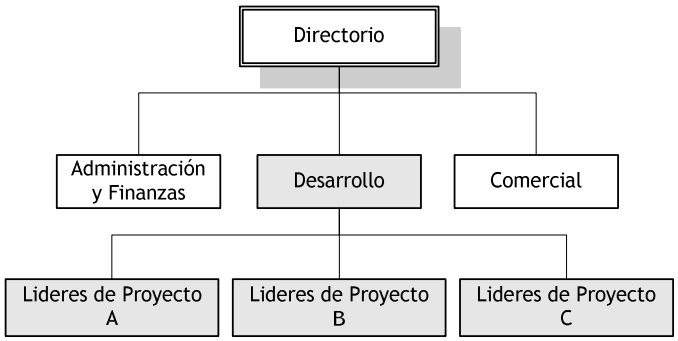


Figura 3.1: El cambio en el organigrama

En la Figura 3.1, se muestra el organigrama de estas empresas, se aprecia una cúpula gerencial y de esta la división por líderes de proyecto. Cada líder de proyecto cuenta con un número relativamente elevado de pequeñas estructuras organizadas alrededor de ellos.

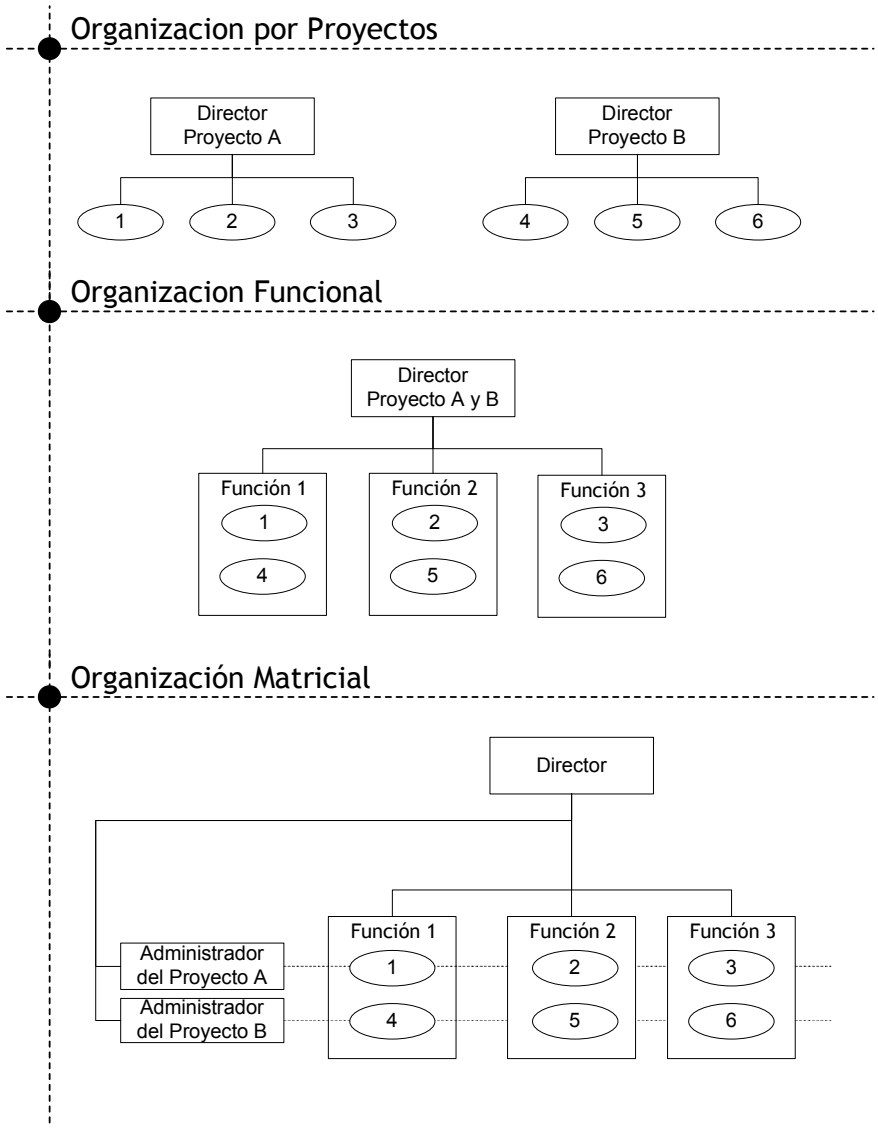


Figura 3.2: Organización del Proyecto

Ventajas:

- ✓ Las decisiones técnicas y administrativas se hacen en los niveles más bajos, permitiendo rapidez y control efectivo.
- ✓ La autoridad impersonal, minimiza las interfaces y define claramente las responsabilidades.
- ✓ Motivación es alta durante el período de desarrollo.

Desventajas:

- ✓ Alta gerencia no ve el desarrollo de los proyectos.
- ✓ No se logra economía de escala en los recursos críticos (personal especializado).
- ✓ Entrenamiento es alto.
- ✓ Desplazamiento de personal de un proyecto a otro es difícil.
- ✓ Inhibe la estandarización.

3.2 Formato Funcional.

En este esquema, un grupo distinto de desarrolladores lleva a cabo cada fase del proyecto, los productos pasan de un equipo a otro conforme el software va evolucionando. De esta forma, un equipo de planificación y análisis desarrolla el Estudio de Prefactibilidad y el Plan del Proyecto, entregando la documentación correspondiente al equipo de Definición del Producto, quien realiza la Definición de Requerimientos, el producto de esta etapa es entregado al equipo de diseño, el cual se encarga de generar el documento de diseño que utilizará el grupo de codificación, este se encarga de codificar y depurar el software que luego debe se revisado por el equipo de prueba. Finalmente, el grupo de control de calidad certifica la calidad del producto; se forma un equipo independiente de mantenimiento para el resto de la vida útil del producto. Una variación del formato funcional, comprende tres equipos: uno de análisis, otro de diseño y codificación, y un tercero de prueba y mantenimiento. Los miembros de los equipos pueden rotar periódicamente para contribuir al desarrollo profesional y evitar la especialización. El formato funcional requiere una alta comunicación entre equipos, pero permite que la documentación sea mas clara.

Organización funcional

Rara vez es usada en proyectos de Software, su problema es que todas las decisiones que cruzan las fronteras funcionales dependen de un individuo.

El principio fundamental de este tipo de organización es el *Staff*.
Este tipo de organización se sustituyó en la organización lineal por la funcional en la que cada operario pasa a reportar, no solo a su jefe superior, sino a varios, pero cada uno en su especialidad.

El Staff es el resultado de la organización lineal y funcional, en esta organización existen órganos de decisión en la asesoría.

Los órganos de línea se caracterizan por la autoridad lineal y el principio escalar, mientras que los órganos staff prestan asesoría a servicios especializados.

En la fusión de la estructura lineal con la funcional, predomina la estructura lineal. Cada órgano reporta a un solo y único órgano superior; Principio de autoridad. Pero cada órgano recibe asesoría y servicio especializado de los diversos órganos de *staff*.

Ventajas:
- ✓ Administración fuerte y control centralizado.
- ✓ Se puede reforzar e implantar fácilmente estándares.
- ✓ Personal esta asociado a una unidad.
- ✓ Se adapta fácilmente a las decisiones de largo plazo.

Desventajas:

- ✓ La resolución de las decisiones la realiza una sola autoridad para todos los proyectos.
- ✓ Limita la creación de generalistas, tiende a la especialización.
- ✓ El control de los proyectos es bajo.

3.3 Formato Matricial.

En organizaciones matriciales, las funciones de Desarrollo, Soporte Técnico, Control de Calidad y Mantenimiento, tienen su propia administración y un equipo de gente dedicada exclusivamente a dicha función. Cada grupo funcional participa en todo proyecto; por ejemplo los miembros del equipo de desarrollo pertenecen organizacionalmente a esa función, pero trabajan bajo la supervisión de un jefe de proyecto en particular. De la misma manera, el personal de control de calidad pertenece a esa función, pero trabaja en uno o más proyectos bajo la supervisión del jefe de proyecto correspondiente. En las organizaciones matriciales cada quien tiene por lo menos dos jefes, la ambigüedad provocada por esto es el costo de tener un proyecto mas controlado.

Por otra parte, el personal asignado a un proyecto, puede integrarse con facilidad cuando sea necesario y liberarse cuando se requiera. En una organización bien administrada, la carga de trabajo es balanceada de tal manera que los individuos que regresan a sus funciones se asignan a otros proyectos, o pueden permanecer en su unidad funcional, en entrenamiento o en la adquisición de nuevas habilidades.

Organización Matricial.

Esta estructura busca optimizar la organización. Su mayor desventaja es que no hay un responsable por el éxito de un proyecto. En este esquema el Gerente funcional decide como hacer el trabajo. Suministra los recursos para el desarrollo. El gerente de proyectos decide que hacer, contrata los recursos económicos. La esencia es la combinación de patrones funcionales y de proyectos o de productos en la misma organización. Esta clase de organización se encuentra con frecuencia en la organización, en la industria aeroespacial, en mercadotecnia, entre otros.

La Organización Matricial se usa debido a que las compañías y los consumidores se han interesado en los resultados finales, han surgido presiones para establecer la responsabilidad de garantizar dichos resultados.

Problemas de la Organización Matricial.

- ✓ Existe una situación de conflicto entre los gerentes funcionales y los de proyecto.
- ✓ El conflicto y la ambigüedad de papeles pueden provocar stress a los gerentes así como a los miembros del equipo.
- ✓ En las organizaciones matriciales también suelen ser fuentes de problemas, el desequilibrio de autoridad y el poder.
- ✓ Requiere de muchas reuniones que consumen tiempo.

Normas para ser eficaz la Organización Matricial.

- ✓ Definir los objetivos del proyecto.
- ✓ Clasificar roles, autoridad y responsabilidad de los administradores y miembros del equipo.
- ✓ Equilibrar el poder en los gerentes funcionales y de proyecto.
- ✓ Seleccionar para el proyecto un administrador con dotes de liderazgo.
- ✓ Poner en práctica controles apropiados de costos, tiempo y calidad.
- ✓ Recompensar en forma justa a los gerentes y miembros del equipo.

Organización Matricial del Equipo de Proyecto.
Relación Jefe De Proyecto - Jefe Unidad Funcional

Como se ha comentado, una de las funciones más relevantes de la dirección de proyectos es a de integrar en el equipo de proyecto a especialistas procedentes de otras áreas o de otras empresas, responsabilidad que debe ser asumida por el jefe de proyecto. La mayor dificultad deriva de que se rompe uno de los principios de gestión clásicos, como es el principio de unidad de dirección. Es decir, un empleado de una unidad funcional que es asignado temporalmente a un proyecto pasa a tener dos jefes: su jefe jerárquico de la unidad funcional, del cual depende formal y habitualmente, y el jefe de proyecto, a las órdenes del cual trabaja sólo en el ámbito del proyecto.

Ambos jefes deben colaborar en ciertos aspectos del proyecto, y en particular en el nombramiento de los diferentes técnicos que intervendrán en el proyecto. Si el director funcional es el que posee los recursos y conoce la valía personal y forma de trabajar de los mismos, es evidente que será la persona más adecuada para proporcionar las personas que intervendrán en el proyecto. Pero si el jefe de proyecto ha de conseguir sus objetivos poniendo en juego los recursos aportados al proyecto, deberá velar porque esos recursos sean idóneos en calidad y cantidad, no pudiendo en caso contrario responsabilizarse de la consecución de los objetivos.

En la Figura 3.3 se presenta un diagrama con la organización matricial de un proyecto donde queda reflejada esta interdependencia entre los recursos asignados y las unidades funcionales.

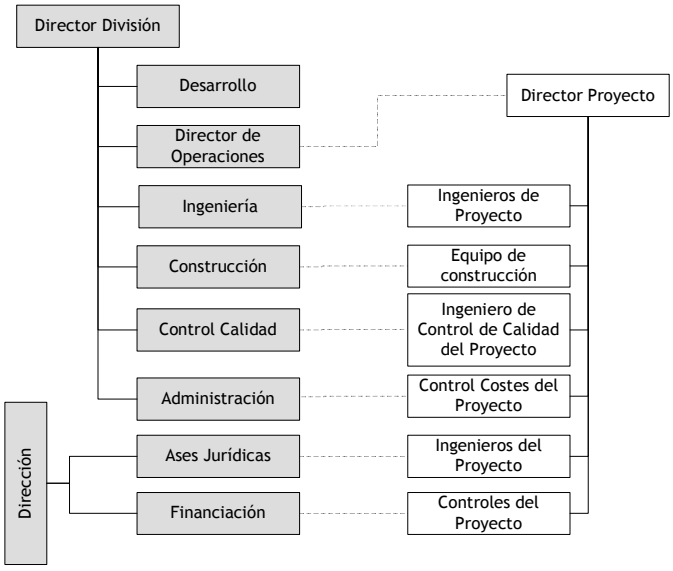


Figura 3.3: Organización Matricial de un Proyecto

La malla como se muestra en la figura representa un análisis bi-dimensional del comportamiento gerencial: preocupación por las personas y preocupación por la producción. La escala es de 0 a 10 siendo 10 lo máximo.

3.4 La malla organizacional.

La eficiencia organizacional es la habilidad intrínseca de una organización para generar software de calidad en el mínimo de tiempo con el mínimo de recursos. Una vez que la eficiencia de la organización ha sido determinada se puede resolver el problema de estimar el esfuerzo de desarrollo.

La organización afecta la productividad de software en tres formas: la estructura usada para organizar, los sistemas usados para planificar y controlar el desarrollo de software y las técnicas de administración usadas, ver Figura 3.4.

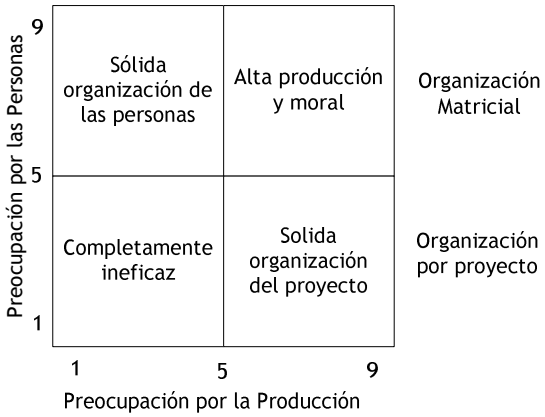


Figura 3.4: Matriz de Administración de una Organización Funcional

A través del rediseño de la estructura organizacional, ayudamos a las organizaciones públicas y privadas a responder mejor ante las necesidades de sus clientes y de la comunidad.

Partiendo de las necesidades de los clientes, se identifican los procesos clave y se dirigen los esfuerzos de la organización hacia una integración interfuncional. El resultado, en la mayoría de las organizaciones, es el "aplanamiento" de la estructura. Esto incrementa la capacidad de coordinación y reduce los niveles jerárquicos.

Las organizaciones que operan un cambio en su estructura, basado en el enfoque de los procesos, obtienen las siguientes ventajas:

- ✓ Mayor calidad en menor tiempo y al menor coste.
- ✓ Más capacidad de respuesta al cambio de las necesidades y expectativas del cliente.
- ✓ Mejor posicionamiento ante el constante cambio en las oportunidades y amenazas del mercado.
- ✓ Despliegue del conocimiento existente en la organización para resolver problemas y añadir valor.

Por otra parte, ayudamos a la organización a definir su misión, visión y valores, en línea con la estrategia, y contribuimos a hacer efectivo el cambio cultural inherente a la reestructuración organizacional, diseñando y desarrollando programas de comunicación interna.

3.5 Perfil de un Analista y TFEA

El analista es el encargado de generar una especificación que satisfaga completamente los objetivos de la gestión para el desarrollo de SW.
El analista idealmente debe tener las siguientes características:

- ✓ Habilidad para comprender conceptos abstractos, reorganizarlos en divisiones lógicas y sintetizar soluciones basadas en cada división;
- ✓ Habilidad para entresacar hechos importantes de fuentes conflictivas o confusas;
- ✓ Habilidad para comprender entornos de usuario/cliente;
- ✓ Un gran bagaje técnico;
- ✓ Conocimientos básicos en otras disciplinas: administración, economía, organización.

Existen numerosas técnicas aplicables a la especificación, una de ellas es la Técnica para Facilitar la Especificación de la Aplicación (TFEA), que comprende la creación de un equipo mixto de clientes y personas encargadas de la identificación del problema, proponer elementos de solución, evaluación de enfoques y la especificación de un conjunto preliminar de requisitos. Sus directrices básicas son:

- ✓ Reunión en lugar neutral donde asisten técnicos y clientes.
- ✓ Establecimientos de reglas para la preparación y participación.
- ✓ Agenda formal que cubra puntos importantes, e informal para que se desarrolle un flujo de ideas.

- ✓ Elección de un “facilitador”.
- ✓ Se utiliza un mecanismo de definición (horas de trabajo, diagramas, pizarras o tableros).

Capítulo 4

Mediciones en Producto y Proceso de Software

4.1 ¿Por qué medir?

"Medimos para mejorar"
Las mejoras en el proceso de desarrollo de software y sistemas de calidad no pueden ser evaluadas sin un esfuerzo efectivo de medición. Cada organización desea mejorar sus procesos de desarrollo de software debido a que existe un tangible beneficio con la construcción de un mejor software.

A continuación se enumeran las siguientes necesidades de medición:

- ✓ Mejoras en la calidad y productividad.
- ✓ Planificación y estimación de proyectos con alguna precisión.
- ✓ Disposición del personal adecuado, bien utilizado y motivado.
- ✓ Existencia de una adecuada estructura organizacional.
- ✓ Uso de técnicas y herramientas efectivas para el proceso.
- ✓ Obtención de un espacio físico y ambiente de trabajo óptimo.

4.2 ¿Qué es una medición?

Una medición es simplemente una representación desde el mundo real y empírico a una representación matemática, donde puede ser más fácilmente entendible en atributos de entidad y las relaciones entre las otras entidades. El problema real es interpretar el comportamiento matemático y juzgar que significa en el mundo real.

En definitiva, las mediciones nos entregan una descripción cuantitativa de los procesos, productos y recursos claves permitiéndonos entender su comportamiento y resultado.

Los aspectos esenciales de la medición son:

- ✓ *Datos duros*, son cuantificables con poca o sin subjetividad (esfuerzo, volumen documentación, errores detectados, etc.
- ✓ *Datos blandos*, presentan un grado de subjetividad (habilidad y experiencia, presiones de tiempo, satisfacción del cliente, cooperación del cliente, etc.
- ✓ *Datos normalizados*, son usados con propósito comparativo (LOC, PF, CC, PO)

Existen diferentes escalas de dimensionamiento, en las cuales cada una captura mayor información que su predecesora. Ellas son:

- ✓ *Escala Nominal*, la cual ordena ítems por categoría. Un ejemplo de esta escala de medición es cuando catalogamos un lenguaje de programación: C++, Java, entre otros.
- ✓ *Escala Ordinal*, la cual ordena ítems. Un ejemplo es cuando se asigna una severidad a la falla encontrada como menor, mayor, catastrófica.
- ✓ *Escala de Intervalos*, la cual define una distancia desde un punto a otro. Este tipo de escala entrega cálculos no disponibles en la escala ordinal, como el cálculo del significado. Lamentablemente no existe el punto cero absoluto y las relaciones no tienen sentido. Por ello hay que tener cuidado cuando se hacen comparaciones. Por ejemplo en la escala de temperatura Celcius y Fahrenheit, no podemos decir que 30º Celcius es el doble de calor que 15º F.
- ✓ *Escala de Proporción,* esta escala es la que entrega mayor información y flexibilidad, debido a que incorpora el cero absoluto. Mediciones como LOC y número de defectos son mediciones de proporción.

Se podrá decir que una medición es válida si presenta la siguiente condición de representación: si éste captura en el mundo matemático el comportamiento que percibimos en el mundo empírico. Por ejemplo, debemos demostrar que H es una medición de altitud, y si A es más alto que B, entonces H(A) es más alto que H(B). Pero esta prueba debe ser empírica y esto a menudo es difícil de demostrar.

En general se desea medir los siguientes aspectos en Ingeniería de Software:

- ✓ Procesos o tareas a ejecutar (modelado, diseño, prueba).
- ✓ Productos entregados durante el proceso (documentación de diseño, código fuente, registro de pruebas).
- ✓ Recursos que permiten realizar el proceso (personal, computadoras, dinero).

4.3 Atributos internos y externos.

Cada una de estas entidades puede ser medida definiendo sus atributos internos o externos. Un atributo interno es medido directamente desde la entidad. Por ejemplo, una medida interna del código fuente es el tamaño medido por las líneas de código. Un atributo externo es una medida de la entidad con relación a una necesidad externa definida por el ambiente en el cual es desarrollada o utilizada. Por ejemplo, la mantenibilidad del código fuente (medición externa), representa la habilidad de una entidad específica de un producto (código fuente) de alcanzar los requerimientos de acomodarse a los cambios fácilmente.

Ejemplos de estas relaciones se muestran en las tablas siguientes:

PRODUCTO		
Entidades	**Atributos Internos**	**Atributos Externos**
Especificaciones	Tamaño, re-uso, modularidad, redundancia, funcionalidad, correctitud	Comprensibilidad, mantenibilidad
Diseño	Tamaño, re-uso, modularidad, acoplamiento, cohesividad, funcionalidad	Calidad, complejidad, mantenibilidad
Codificación	Tamaño, re uso, modularidad, acoplamiento, funcionalidad, complejidad algorítmico, estructuración	Confiabilidad, Usabilidad, mantenibilidad
Datos de prueba	Tamaño, nivel de cobertura	Calidad
PROCESOS		
Entidades	**Atributos Internos**	**Atributos Externos**
Especificación de requerimientos	Tiempo, esfuerzo, número de cambios en los requerimientos	Calidad, costo, estabilidad
Diseño detallado	Tiempo, esfuerzo, número de especificaciones erróneas detectadas	Costo, costo - eficacia
Pruebas	Tiempo, esfuerzo, número y errores encontrados	Costo, costo - eficacia, estabilidad
Vista de Despliegue	Diagrama de Despliegue	Nodo, componente, dependencia, localización.
RECURSOS		
Entidades	**Atributos Internos**	**Atributos Externos**
Personal	Años de experiencia, tasa de trabajo	Productividad, experiencia, inteligencia.
Equipos	Tamaño, nivel de comunicación, estructuración	Productividad, calidad.
Software	Tamaño, costo	Usabilidad, Confiabilidad
Hardware	Precio, velocidad, tamaño de la memoria	Confiabilidad
Oficinas	Tamaño, temperatura, luz	Confort, calidad

Tabla 4.1: Atributos internos y externos.

Atributo	Definición
Acoplamiento	Grado de fuerza de la inter-conexión entre los componentes del sistema.
Calidad	Grado de excelencia.
Cohesividad	Grado en que las entidades dependen o no de las otras entidades.
Correctitud	Grado en el que un entidad satisface las especificaciones y cumple los objetivos del usuario.
Complejidad	Cantidad de diversos elementos que lo componen.
Complejidad algorítmica	Grado de relativa dificultad computacional de las funciones.
Costo	Cuanto cuesta conseguir un objetivo determinado.(Monetariamente)
Costo-eficiencia	Grado en que el costo esta relacionado con la eficencia obtenida.
Confiabilidad	Grado con el que una entidad realiza su función con una precisión requerida.
Confort	Grado de comodidad.
Eficiencia	Cantidad de recursos y código requeridos por una entidad para realizar una función.
Esfuerzo	Nivel de empleo de recursos en la consecución de algún fin.
Estabilidad	Grado de permanencia y equilibrio ante efectos externos.
Estructuración	Grado en que las partes de un todo están en una estructura determinada.
Experiencia	Nivel de práctica o conocimiento de algún tema.
Funcionalidad	Cuan Práctico, eficaz, utilitario es el resultado obtenido.
Inteligencia	Capacidad de la persona sobre el conjunto de funciones que tienen por objeto el conocimiento (sensación, asociación, memoria, imaginación, entendimiento, razón, conciencia).
Mantenibilidad	Esfuerzo requerido para localizar y corregir un error en un programa en funcionamiento.
Nivel de cobertura	Grado en que abarca o contempla cierto ámbito solicitado (en este caso datos de Prueba).
Nivel de comunicación	Grado de comunicación existente entre los distintos desarrolladores.
Productividad	Grado de producción por unidad de trabajo.
Tasa de trabajo	Medida del trabajo que realiza.
Usabilidad	Esfuerzo necesario para aprender, operar, preparar entradas e interpretar la salida de un programa.
Re-uso	Grado en que una entidad se puede utilizar en otras actividades.
Otros Atributos	
Flexibilidad	Esfuerzo requerido para modificar un programa en funcionamiento.
Facilidad de prueba	Esfuerzo requerido para probar un programa (para garantizar que realiza la función deseada)
Interoperatividad	Esfuerzo requerido para aclopar
Portabilidad	Esfuerzo requerido para transferir un programa de una configuración hardware o entorno de software a otro.

Tabla 4.2: Definiciones de los atributos

En general, las mediciones que se realizan son pocas y simples; para el proceso éstas corresponden a costo y esfuerzo incurridos a lo largo del tiempo necesario en el desarrollo hasta el fin del proyecto; para el producto, líneas de código o puntos de función producidos, páginas de documentación escritas, velocidad de ejecución de los programas, tamaño de memoria requerida y defectos reportados en un período dado; para los recursos, cantidad de personal involucrado, tamaño, experiencia del equipo de desarrollo, costo y disponibilidad de las herramientas utilizadas. De las estadísticas que generan los proyectos realizados se puede obtener algunos indicadores o métricas de calidad y productividad:

Productividad **= KLOC / PM**

KLOC son las Kilo Líneas de Código fuente entregada y PM son las Personas Mes destinados al proyecto.

Calidad **= defectos / KLOC**
Costo **= $ / LOC**
Documentación **= páginas de documentación / KLOC**

En general las mediciones asociadas a las LOC están en permanente discusión, ya que son dependientes del lenguaje de programación, no son fáciles de utilizar en lenguajes no procedurales y su uso en estimaciones requiere de mucho detalle en los requerimientos.

4.4 Atributos de las técnicas de estimación.

Una forma de estimar es utilizar métricas basadas en mediciones pasadas para hacer las estimaciones. También es útil dividir el proyecto en pequeñas unidades o partes para facilitar la estimación. El proceso de gestión de proyecto de software comienza con un conjunto de actividades que, globalmente, se denomina planificación del proyecto. La primera de estas actividades es la estimación. La estimación es una actividad importante que no debe llevarse a cabo de forma descuidada.

Existen técnicas útiles para la estimación de costos, tiempos y recursos. La Planificación es parte fundamental de la planificación y como la planificación del proyecto sirve como guía para una buena ingeniería de software, no es en absoluto aconsejable embarcarse sin esta.

La estimación de costos, tiempos y recursos para el esfuerzo de desarrollo de Software requiere experiencia, acceso a una buena información histórica y coraje para confiar en medidas cuantitativas cuando todo lo que existe son datos cualitativos.
Existen tres puntos que caracterizan al proyecto a estimar:

- ✓ La ***complejidad del proyecto***, que tiene un gran efecto sobre la incertidumbre que es inherente a la planificación. La complejidad, sin embargo, es una medida relativa que se ve afectada por la familiaridad

con anteriores esfuerzos. Para un equipo de desarrollo de SOFTWARE que sólo haya desarrollado aplicaciones no interactivas, una aplicación de tiempo real parece "excesivamente compleja".

- ✓ El ***tamaño del proyecto***, que es otro factor importante y puede afectar a la precisión y eficacia de las estimaciones. A medida que aumente el tamaño o la interdependencia entre distintos elementos del SOFTWARE, éste tiende a crecer rápidamente.
- ✓ El ***grado de estructuración del proyecto*****,** que también tiene efecto sobre el riesgo de la estimación. En este contexto, la estructuración se refiere a la facilidad con la que las funciones pueden ser compartimentalizadas y la naturaleza jerárquica de la información que debe ser procesada.

El riesgo se mide por el grado de incertidumbre de las estimaciones cuantitativas establecidas para los recursos, los costos y las agendas. El planificador y el cliente deben tener claro que cualquier cambio en los requerimientos del software significa inestabilidad en el costo y en los tiempos.
Un análisis de los requisitos del *software* proporciona la información necesaria para las estimaciones, pero el análisis suele durar semanas o meses. Las estimaciones son necesarias "ahora". Estas estimaciones se hacen sin marco de tiempo limitado, al principio de un proyecto de *software* y deben ser actualizadas regularmente.
Una de las actividades importantes en este aspecto son las reuniones iniciales que se mantienen entre el cliente y el analista. Aunque en un comienzo no se encuentre por dónde empezar e incluso se parezcan mucho más a una conversación entre dos adolescentes, de ésta nacen las bases para un buen proyecto.

En dicha reunión se realiza un conjunto de preguntas, las primeras son:

- ✓ ¿Quién está detrás de la petición para este trabajo?.
- ✓ ¿Quién usará esta solución?.
- ✓ ¿Cuál será el beneficio económico de la solución?.
- ✓ ¿Hay otro origen para la solución?.

El siguiente conjunto de preguntas que se realizan, está orientadas a un mejor conocimiento del problema y del cliente y son las siguientes:

- ✓ ¿Cómo caracterizará el cliente un buen rendimiento que será generado por una solución exitosa?
- ✓ ¿A cuál(es) problema(s) se dirigirá esta solución?
- ✓ ¿Puede describir o mostrar el ambiente en el cual será usada la solución?
- ✓ ¿Hay fuerzas o resultados de rendimiento especiales que afectan la forma en la cual la solución es abortada?

El conjunto final de preguntas se enfoca hacia la eficacia del encuentro:

- ✓ ¿Es Usted la persona correcta para responderlas?
- ✓ ¿Es Usted un contestador oficial?
- ✓ ¿Son mis preguntas relevantes para el problema que tiene?
- ✓ ¿Estoy haciendo muchas preguntas?, ¿Hay algo más que deba preguntarle?

Este conjunto de preguntas buscan lograr una comunicación más expedita, lo cual es esencial para establecer la envergadura del proyecto. Esto es recomendable sólo en su inicio, luego se puede complementar con elementos de la solución de los problemas, negociación y especificación.

4.5 Estimación de costos en el Software.

En todo proyecto existe un presupuesto asignado, el cual debe ser controlado y respetado. Para realizar esto es necesario planificar adecuadamente el qué hacer, por lo cual la primera actividad a realizar es la de estimar el costo del desarrollo, lo que se realiza simultáneamente con la itineración.

Para estimar recursos, costo y tiempo en un proyecto de software se necesita experiencia, acceso a información histórica y hacer uso de métricas cuantitativas cuando existen los datos para ello. Las estimaciones tienen asociado en forma inherente, un grado de riesgo e incertidumbre que incide en la probabilidad de éxito, en la estimación y por ende en el resultado.

Los componentes principales de costos son:

- ✓ Hardware.
- ✓ Entrenamiento.
- ✓ Esfuerzo.

Existen siete técnicas posibles para estimar los costos del Software

- ✓ *Modelos algorítmicos.* Se utiliza un modelo basado en información histórica que relaciona información histórica de costo con alguna métrica del proyecto.
- ✓ *Juicio experto.* El costo es obtenido por consenso de expertos en el desarrollo. La experiencia debe ser en las tecnologías y aplicación a desarrollar.
- ✓ *Estimación por analogía.* Se basa en el desarrollo previo de proyectos similares.
- ✓ *Ley de Parkinson.* El trabajo se expande hasta llenar todo el tiempo disponible.
- ✓ *Precio para ganar.* El costo se estima según el presupuesto disponible.
- ✓ *Estimación top down.* El costo se estima considerando la funcionalidad total del producto y cómo ésta es provista por las subfunciones interactuantes. El costo se basa en las funciones lógicas.
- ✓ *Estimación bottom up.* Se estima el costo de cada componente para luego agregarse en un costo total.

Los factores que afectan a la estimación de un proyecto son básicamente dos: la complejidad del proyecto y el tamaño del mismo.
Al principio, el costo del software constituía un pequeño porcentaje del costo total de los sistemas basados en computadora. Un error considerable en las estimaciones del costo del software tenía relativamente poco impacto. Hoy en día, el software es el elemento más caro de la mayoría de los sistemas

informáticos. Un gran error en la estimación del costo puede ser lo que marque la diferencia entre beneficios y pérdidas. Sobrepasarse en el costo puede ser desastroso para el equipo de desarrollo

Desde un punto de vista ideal, se deben aplicar conjuntamente todas las técnicas apropiadas, usando cada una de ellas como comprobación de las otras. Las técnicas de descomposición utilizan un enfoque de «divide y vencerás» para la estimación del proyecto de software.

Dentro de la mayor parte de las organizaciones, la estimación de costos se basa en las experiencias pasadas. Los datos históricos se usan para identificar los factores de costo y determinar la importancia relativa de los diversos factores dentro de la organización. Lo anterior, por supuesto, significa que los datos de costos y productividad de los proyectos actuales deben ser centralizados y almacenados para un empleo posterior.

La *estimación jerárquica hacia abajo* se enfoca primero a los costos del nivel del sistema, así como a los costos de manejo de la configuración, del control de calidad, de la integración del sistema, del entrenamiento y de las publicaciones de documentación. Los costos del personal relacionado se estiman mediante el examen del costo de proyectos anteriores que resulten similares.

En la *estimación jerárquica hacia arriba*, primero se estima el costo del desarrollo de cada módulo o subsistema; tales costos se integran para obtener un costo total. Esta técnica tiene la ventaja de enfocarse directamente a los costos del sistema, pero se corre el riesgo de despreciar diversos factores técnicos relacionados con algunos módulos que se desarrollarán. La técnica subraya los costos asociados con el desarrollo independiente de cada módulo o componente individual del sistema, aunque puede fallar al no considerar los costos del manejo de la configuración o del control de calidad.

En la práctica, ambas técnicas deben desarrollarse y compararse para que iterativamente se eliminen las diferencias obtenidas.

En la Tabla 4.3 se muestran algunas de las variables para medir costos en proyectos TI.

POSIBLES COSTOS DE UN SISTEMA DE INFORMACIÓN	
Costos de elaboración	✓ **Costos de consulta** ✓ **Costos de alquiler o de compra de los equipos** ✓ **Costos de modificación del emplazamiento de los equipos (aire acondicionado, seguridad, etc.).** ✓ **Costos del capital** ✓ **Costos de gestión y de personal**
Costos de puesta en marcha	✓ **Costos del software del sistema operativo** ✓ **Costos de la instalación de los equipos de comunicaciones (líneas telefónicas y de datos)** ✓ **Costos del personal para la puesta en marcha** ✓ **Costos de contratación de personal y alquileres** ✓ **Costos de interrupción en el resto de la organización** ✓ **Costos de gestión necesarios para dirigir la actividad inicial**
Costos relacionados con el proyecto	✓ **Costos de adquisición del software de aplicación** ✓ **Costos de las modificaciones del software para que encajen con los sistemas locales** ✓ **Costos de personal, gastos generales, etc. del desarrollo de aplicaciones internas** ✓ **Costos de entrenamiento del personal en el uso de las aplicaciones** ✓ **Costos de recogida de información y procedimiento de instalación** ✓ **Costos de preparación de la documentación** ✓ **Costos de la supervisión del desarrollo**
Costos del proceso	✓ **Costos de mantenimiento del sistema (hardware, software e instalaciones)** ✓ **Costos de suministros (electricidad, teléfono, etc.)** ✓ **Costos de depreciación del hardware** ✓ **Costos del personal involucrado en la gestión de los sistemas de información, operación y actividades de planificación**

Tabla 4.3: Variables para medir costos en proyectos TI

4.6 Estimación de recursos

Recursos Humanos: Determinar el personal necesario, la posición dentro de la organización y la especialidad requerida. El número de personas sólo puede ser determinado después de hacer una estimación de esfuerzo.

Recursos de Hardware: Durante la planificación del proyecto de software, se deben considerar básicamente tres categorías de HW; el sistema de desarrollo, la máquina objetivo y los demás elementos de HW del nuevo sistema. Sistemas de desarrollo, es el HW donde se desarrolla el proyecto, máquina objetivo es el HW que utilizará nuestro cliente y el sistema, lo demás serán los periféricos asociados.

Recursos de Software: Se utilizan distintas herramientas que ayudan en el desarrollo del nuevo sistema, estas herramientas tienen diferentes propósitos como: gestión de proyectos, soporte, análisis y diseño, programación, mantenimiento, entre otros.

Capítulo 5

Estimación en Proyectos de Software

La estimación del costo y del esfuerzo del software nunca será una ciencia exacta. Son demasiadas las variables humanas, técnicas, de entorno, políticas que pueden afectar al costo final del software y al esfuerzo aplicado para desarrollarlo. Sin embargo, la estimación del proyecto de software puede dejar de ser un oscuro arte para convertirse en una serie de pasos sistemáticos que proporcionen estimaciones con un grado de riesgo aceptable.

Para realizar estimaciones seguras de costos y esfuerzos tenemos varias opciones posibles:

- ✓ Utilizar "técnicas de descomposición" relativamente sencillas para generar las estimaciones de costo y de esfuerzo del proyecto.
- ✓ Desarrollar un modelo empírico para el cálculo de costos y esfuerzos.
- ✓ Adquirir una o varias herramientas automáticas de estimación.

Dejar las estimaciones para más adelante o retrasarlas no es una opción ya que estas se necesitan de antemano. Las tres opciones restantes son métodos viables para la estimación del proyecto. Las técnicas de descomposición utilizan un enfoque divide y vencerás. Los modelos empíricos son utilizables como complemento de las técnicas de descomposición donde cada modelo se basa en la experiencia (datos históricos), por ultimo las herramientas automáticas de estimación ponen en ejecución una o varias técnicas de descomposición o modelos empíricos.

5.1 Técnicas de Descomposición.

Los proyectos son constantemente utilizados dentro de la organización, esto, porque constituyen el principal medio de crecimiento para ésta. Generalmente, los proyectos han sido utilizados como un instrumento de "acción" para manejar grandes inversiones en cualquier área de la organización.

Por esto se busca una forma práctica para estimar su esfuerzo y tiempo de ejecución, para así minimizar la inversión.

Dado que la estimación del esfuerzo de un proyecto de software no es una ciencia exacta, existen demasiadas variables humanas y técnicas influyendo y afectando al producto final.

Se trabajará sobre la base de Técnicas de Descomposición, de esta manera se divide el problema en módulos pequeños más manejables que permitan

definir una estimación de tiempo, de cantidad de personas necesarias para llevar a cabo el proyecto propuesto. La estimación de proyectos de software es una forma de resolución de problemas y en la mayoría de los casos, el problema a resolver (esto es desarrollar estimaciones de costo y de esfuerzo para un proyecto de software), es demasiado complejo, por ello se debe separar en un conjunto de pequeños problemas mas manejables.

5.2 Estimación de Líneas de Código (LDC) y Puntos de Función (PF).

Las LDC y los PF se describen como medidas básicas desde donde se calculan métricas de productividad. Los datos de LDC y PF se utilizan de dos formas durante la estimación del proyecto de software:

- ✓ Como una variable de estimación que se utiliza para "dimensionar"cada elemento del software.
- ✓ Como métricas de línea base recopiladas de proyectos anteriores y utilizadas junto con variables de estimación para desarrollar proyecciones de costo y de esfuerzo.

En un comienzo el proyecto se disgrega en pequeñas subfunciones que pueden ser estimadas individualmente , ya sea en LDC o PF para cada función. Cuando se utiliza LDC como variable de estimación, la descomposición funcional es absolutamente necesaria.

También debe tenerse en cuenta que mientras las LDC se estiman directamente, los PF se determinan indirectamente mediante la estimación de numero de entradas, salidas, archivos de datos, consultas e interfaces, así como también de catorce valores de ajuste de complejidad. Independientemente de la variable de estimación que use el planificador del proyecto, normalmente proporciona un rango de valores para cada función descompuesta.

¿Serán correctas las estimaciones? La única respuesta razonable a esta pregunta es "No podemos asegurarlo". Cualquier técnica de estimación, no importa como sea de sofisticada, tiene que ser comprobada utilizando otro método. Incluso entonces, deberán prevalecer la experiencia y el sentido común.

¿Qué ocurre cuando la concordancia entre las estimaciones es pobre? Para responder a esta pregunta se debe reevaluar la información que se ha utilizado para hacer las estimaciones. Muchas divergencias entre estimaciones se deben a menudo, a una de dos causas:

- ✓ No se entiende adecuadamente el ámbito del proyecto o ha sido malinterpretado por el planificador.
- ✓ Los datos de productividad utilizados en la técnica LDC, son inadecuados para esa aplicación, están obsoletos (no reflejan con precisión la organización de desarrollo de software) o se han aplicado mal.

El planificador debe determinar las causas de la divergencia y reconciliar las estimaciones.

5.2.1 Líneas de Código (LDC) v/s Puntos de Función (PF).

LDC y PF son técnicas de estimación distintas, a pesar de que ambas tienen varias características en común. La estimación comienza con un enfoque limitado para el ámbito del software y desde esta sentencia intenta descomponer el software en funciones que se pueden estimar, individualmente.

Para cada función entonces se estima las LDC y el PF (la variable de estimación). De forma alternativa, se puede seleccionar otro componente para dimensionar clases u objetos, cambios, o procesos de gestión en los que puede tener impacto.

Las métricas de productividad (p. ej.: LDC/pm o PF/pm) se aplican entonces para la variable de estimación adecuada y se extrae el costo o el esfuerzo de la función. Las estimaciones de función se combinan para producir una estimación global del proyecto entero. En general, el dominio del proyecto debería calcular las medias de LDC/pm o PF/pm. Es decir, los proyectos se deberían agrupar por tamaño de equipo, área de aplicación, complejidad y otros parámetros relevantes. Entonces se deberían calcular las medias del dominio local. Cuando se estima un proyecto nuevo, primero se debería asignar aun dominio, y a continuación utilizar la media del dominio adecuado para la productividad al generar la estimación.

Las técnicas de estimación de LDC y PF difieren en el nivel de detalle que se requiere para la descomposición y el objetivo de la partición. Cuando se utiliza LDC como variable de estimación, la descomposición es absolutamente esencial y a menudo se toman para considerables niveles de detalle. Cuanto más grande sea el grado de particionamiento, más probable será que pueda desarrollar estimaciones más exactas.

Para estimaciones de PF, la descomposición funciona de diferente manera. En lugar de centrarse en la función, se estiman cada una de las características del dominio de información entradas, salidas, archivos de datos, peticiones, e interfaces extremas y los catorce valores de ajuste de la complejidad. Las estimaciones resultantes se utilizan para derivar un valor de PF que se pueda unir a datos pasados y utilizar para generar una estimación.

Las métricas usadas para estimar el tamaño del producto de software deben ser razonablemente fáciles de usar en etapas tempranas del proyecto y fácilmente mensurables una vez que el trabajo ha finalizado.

Las comparaciones subsecuentes de las estimaciones iniciales con el tamaño del producto de software actual, proveen retroalimentación a los planificadores acerca de cómo hacer estimaciones más precisas en futuros proyectos.

En la actualidad las métricas más usadas para dimensionar un producto de software son las Líneas de Código (LDC) y los Puntos de Función (PF). Ambos poseen ventajas y desventajas que serán discutidas a continuación.

Las LDC miden en forma directa el tamaño del producto de software. Se calculan simplemente contando las instrucciones de código fuente de cada componente del producto de software excluyendo, generalmente, los comentarios y blancos. Antes de adoptar esta métrica como estándar, la organización debe definirla en forma exhaustiva. Esta definición debe respetarse, ya que podría atentarse contra la integridad de los datos de la base de datos de los proyectos de la organización.
Las ventajas y desventajas de las LDC se sintetiza en la Tabla 5.1:

Ventajas	Desventajas
✓ Fácil de calcular. ✓ Base de cálculo de modelos de estimación de costos de software existentes. ✓ Existencia de literatura al respecto. ✓ Fácil de automatizar	✓ Dependencia del lenguaje de programación. ✓ Difícil de estimar en etapas tempranas del proyecto. ✓ Difícil de calcular en lenguajes de programación no procedurales.

Tabla 5.1: Ventajas y desventajas de LDC

Algunos datos sobre KLOC, PF y costos.

- ✓ La mayoría de los sistemas de defensa tienen 300,000 puntos de función (aprox. 27 millones de líneas de código).
- ✓ F4-G a finales de los 70 tenía 28,500 líneas de código.
- ✓ El F-15C de los 80 tenía 507,000 líneas de código.
- ✓ El F-22 tiene más de 1.6 millones líneas de código ejecutándose a bordo.
- ✓ El Euro y Y2K son los problemas de negocio más caros en la historia humana. Se estimó que el problema Y2K afectaría globalmente a 36,000,000 aplicaciones de software. Los sistemas del software muy grandes con 100,000 punto de función puede costar más que construir un estadio del fútbol techado, ó 50 rascacielos.[5]
- ✓ Windows 98 tiene 87000 puntos de función.
- ✓ Windows NT 5.0 o WINDOWS 2000, la estimación pública fue de 35,000,000 líneas de código.
- ✓ Microsoft Word aprox. 5,000 PF.
- ✓ Windows Xp mayor que 100,000 PF.
- ✓ Linux aprox. 50,000 PF.

5.3 Modelos para las Estimaciones.

5.3.1 Modelo COCOMO. (COCOMO 81)

COCOMO, el Modelo Constructivo de Costos, fue desarrollado por Barry Boehm en 1981.

El modelo asume que los requerimientos son relativamente estables y que el proyecto será administrado por el cliente y el desarrollador. El modelo entrega un orden de magnitud de los costos del Software. Utiliza como datos el tamaño estimado del proyecto y el tipo de producto a desarrollar.

Tipos de proyectos

Los modelos COCOMO están definidos para tres modos de desarrollo:

- ✓ *Proyectos Orgánicos:* El equipo de desarrollo es pequeño y experimentado, en un ambiente familiar y con aplicaciones conocidas. En general considera proyectos de no más de 50.000 LDC.
- ✓ *Proyectos Semiconectados* (Semilibres): El equipo de desarrollo está formado por personal experimentado y novato con algo de experiencia en la tecnología y la aplicación. Suelen tener interacciones con otros sistemas, y presentan un tamaño de no más de 300.000 LDC.
- ✓ *Proyectos Integrados* (Empotrados): Tiene un gran equipo de desarrollo, pero en general es poco experimentado en el tema dado que se trata de proyectos casi únicos. Corresponden a proyectos que presentan un fuerte acoplamiento entre el Hardware, Software y los procedimientos operacionales. La modificación a los requerimientos no es práctica y los costos de validación son altos.

Jerarquía de Modelos COCOMO

La jerarquía de modelos COCOMO viene dada por el nivel de detalle empleado en su utilización:

- ✓ *Modelo COCOMO básico*: es un modelo univariable estático que calcula el esfuerzo y el tiempo del desarrollo de software en función del tamaño del programa, expresado en Líneas de Código (LDC) estimadas. Adecuado para hacer estimaciones de forma rápida, aunque sin gran precisión.
- ✓ *Modelo COCOMO intermedio:* es un modelo univariable estático que calcula el esfuerzo del desarrollo de software en función del tamaño del programa y un conjunto de "conductores de coste", que incluyen la evaluación subjetiva del producto, del hardware, del personal y de los atributos del proyecto. Estos conductores del coste se consideran como términos de impacto agregado al esfuerzo total del proyecto.
- ✓ *Modelo COCOMO avanzado* (o detallado): es un modelo que incorpora todas las características de la versión intermedia y lleva a cabo una evaluación del impacto de los conductores de coste en cada fase (análisis, diseño, etc.) del proceso de ingeniería de software.

Cabe destacar que las estimaciones relacionadas con el esfuerzo se expresan en hombre-mes (tiempo que requeriría una sola persona para desarrollar el sistema), considerando que la dedicación de una persona es de 152 horas al mes.

5.3.2 Modelo COCOMO Básico

Las ecuaciones básicas del modelo se muestran en la Tabla 5.2.

	ORGANICO	SEMIDESCONECTADO	INTEGRADO
Esfuerzo Estimado [PM]	$E_D = 2{,}4 \times (KLDC)^{1{,}05}$	$E_D = 3{,}0 \times (KLDC)^{1{,}12}$	$E_D = 3{,}6 \times (KLDC)^{1{,}20}$
Tiempo de desarrollo [Meses]	$T_D = 2{,}5 \times (E_D)^{0{,}38}$	$T_D = 2{,}5 \times (E_D)^{0{,}35}$	$T_D = 2{,}5 \times (E_D)^{0{,}32}$
Productividad	$\mathrm{Pr} = \frac{LDC}{E_D}$		
N° medio de personas	$P = \frac{E_D}{T_D}$		
Esfuerzo de mantenimiento [PM]	TCA (Tráfico de Cambio Anual): porción de instrucciones fuente que sufren algún cambio durante un año, bien sea por adición o modificación. $E_M = TCA \cdot E_D$ Y por lo tanto, el número medio de personas a tiempo completo dedicadas a mantenimiento durante 12 meses sería: $P_M = \frac{E_M}{12}$		

Tabla 5.2: Ecuaciones Cocomo Básico

La Tabla 5.3 muestra los perfiles de proyectos estándares para Modo Orgánico:

TAMAÑO (Líneas)	ESFUERZO ESTIMADO (PM)	PRODUCTIVIDAD (LDC/PM)	TIEMPO DE DESARROLLO (Meses)	PERSONAL (Personas)
Pequeño 2 [KLDC]	5.0	400	4.0	1.1
Intermedio 8 [KLDC]	21.3	376	8.0	2.7
Medio 32 [KLDC]	91.0	352	14.0	6.5
Grande 128 [KLDC]	392.0	327	24.0	16.0

Tabla 5.3: Perfiles de Proyecto

Donde el tamaño viene dado el KS, que corresponde a miles de líneas de código fuente.

Cabe mencionar que el número de personas empleadas en el proyecto a lo largo de su desarrollo no es uniforme, el número medio de personas obtenido no representa una cifra muy significativa, es decir, debe considerarse sólo como una aproximación. Putnam (1978) establece que la forma óptima para asignar personal en el proyecto es siguiendo una curva Rayleigh, ver Figura 5.1. La experiencia demuestra que no es conveniente partir con mucha gente al inicio de éste.

De cualquier modo sería más conveniente analizar el personal requerido, en cada etapa de desarrollo del proyecto, el cual también se comporta como una curva Rayleigh para cada etapa como lo muestra la Figura 5.1 [13]:

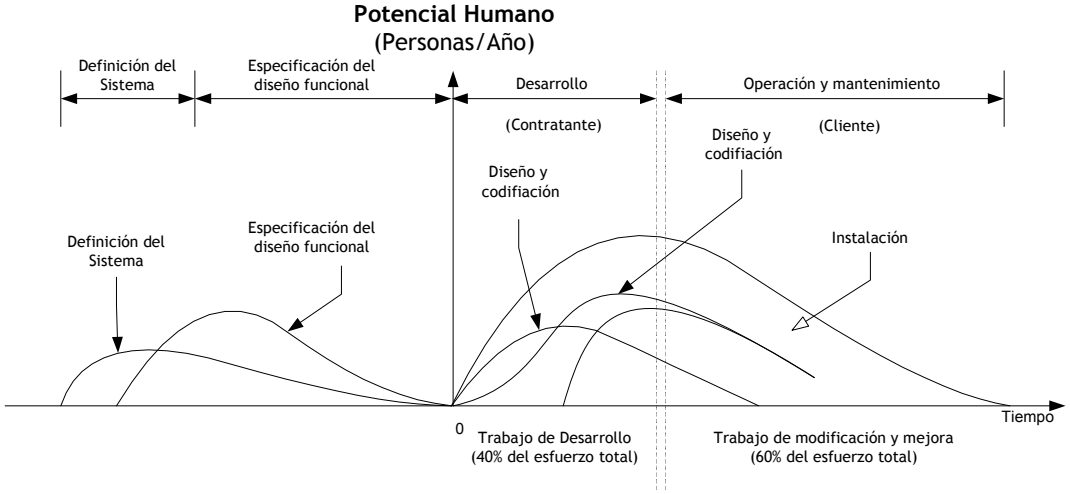


Figura 5.1: Curva de Rayleigh-Norden de esfuerzo de desarrollo

En general el modelo de estimación nos entrega un orden de magnitud de la cantidad de personal requerido para el desarrollo del proyecto, para esto, utiliza en forma implícita la productividad estimada supuestamente obtenida de datos de proyectos existentes. En el caso de proyectos orgánicos, la productividad es de 352 [LDC / h-m], lo que resulta en 16 [instrucciones / h-d]. Para los proyectos integrados la productividad implícita es de 105 [LDC / h-m], lo que resulta en cerca de 4 [instrucciones / h-d].

5.3.3 Modelo COCOMO Intermedio.

Este modelo es una versión ampliada del modelo Básico, en la que se presenta una mayor precisión en las estimaciones, manteniendo prácticamente la misma sencillez del anterior modelo. Esta mayor precisión viene dada por la incorporación de 15 factores que reflejan la influencia de ciertos elementos sobre el coste del software. Estos 15 factores se agruparon en cuatro grandes grupos: Atributos del Producto, del Computador, del Personal y del Proyecto. Cada uno de estos 15 atributos tiene asociado un factor multiplicador para estimar el efecto de éste sobre el esfuerzo nominal. En la Tabla 5.4 se muestran las ecuaciones del Esfuerzo Nominal.

	ORGANICO	SEMIDESCONECTADO	INTEGRADO
Ecuaciones del esfuerzo nominal [PM]	$E_N = 3{,}2 \times (KLDC)^{1{,}05}$	$E_N = 3{,}0 \times (KLDC)^{1{,}12}$	$E_D = 2{,}8 \times (KLDC)^{1{,}20}$

Tabla 5.4: Ecuaciones de Esfuerzo

Una vez obtenido el esfuerzo nominal, este debe multiplicarse por el producto de la ponderación de los 15 atributos, ver Tabla 5.5.

ATRIBUTOS	VALOR					
	Muy bajo	Bajo	Nominal	Alto	Muy alto	Extra alto
Atributos del Producto						
Fiabilidad	0.75	0.88	1.00	1.15	1.40	
Tamaño de la BD		0.94	1.00	1.08	1.16	
Complejidad	0.70	0.85	1.00	1.15	1.30	
Atributos de Computador						
Restricciones de tiempo de ejecución			1.00	1.11	1.30	1.68
Restricciones de memoria virtual			1.00	1.06	1.21	1.56
Volatilidad de la máquina Virtual		0.87	1.00	1.15	1.30	
Tiempo de respuesta		0.87	1.00	1.07	1.15	
Atributos del Personal						
Capacidad de análisis	1.46	1.19	1.00	0.86	0.71	
Experiencia en la aplicación	1.29	1.13	1.00	0.91	0.82	
Capacidad de los programadores	1.42	1.17	1.00	0.86	0.70	
Experiencia en la máquina virtual	1.21	1.10	1.00	0.90		
Experiencia en el lenguaje	1.14	1.07	1.00	0.95		
Atributos del Proyecto						
Técnicas de programación	1.24	1.10	1.00	0.91	0.82	
Utilización de herramientas de software	1.24	1.10	1.00	0.91	0.83	
Restricciones de tiempo de desarrollo	1.23	1.08	1.00	1.04	1.10	

Tabla 5.5: Ponderaciones para los 15 atributos de Cocomo Intermedio

La descripción de cada uno de ellos se muestra a continuación:

✓ Atributos del Producto

- *Fiabilidad del Software:* Indica las posibles consecuencias para el usuario en el caso que todavía existan defectos en el producto. Una puntuación Muy Baja indica que solamente hace falta eliminar los

defectos sin ninguna otra consecuencia mientras que una puntuación Muy Alta implica pérdidas de vidas humanas.

- *Tamaño de la Base de Datos:* Un valor Bajo para este atributo significa que el tamaño de la base de datos (en Bytes) es menor que 10*LDC. El valor Nominal indica un tamaño entre 10 y 100 veces las LDC; un valor Muy Alto significa que la base de datos es mayor que 1000 veces las LDC.
- *Complejidad:* El código de complejidad Baja usa operaciones de E/S, estructuras de datos simples y codificación fácil. La complejidad Nominal implica algún procesamiento de E/S, E/S de múltiples archivos, uso de rutinas de biblioteca y comunicación ínter módulos. Complejidad Muy Alta implica el uso de código recursivo, manejo de archivos complejos, procesamiento paralelo y administración de datos complicada.

✓ Atributos del Computador

- *Restricciones de tiempo de ejecución:* Siempre será más exigente para un programador escribir un programa que tiene una restricción en el tiempo de ejecución. Esta puntuación se expresa en el porcentaje de tiempo de ejecución disponible. Es Nominal cuando el porcentaje es el 50%, y Extremadamente Alto cuando la restricción es del 95%.
- *Restricciones de memoria virtual:* Se espera que un cierto porcentaje del almacenamiento principal sea utilizado por el programa. El esfuerzo de programación se incrementa si el programa tiene que correr en un volumen menor del almacenamiento principal. El esfuerzo extra de Nominal se presenta cuando la reducción del almacenamiento principal es del 50% y Extremadamente Alto indica una reducción del 95%.
- *Volatilidad de la máquina virtual:* Durante el desarrollo del software la máquina virtual (combinación de hardware y software) en la que el programa se va a desarrollar puede sufrir algunos cambios.
- *Tiempo de respuesta:* Cuanto menor sea el tiempo de respuesta requerido, más alto será el esfuerzo humano.

✓ Atributos del Personal

- *Capacidad de análisis:* La capacidad del grupo de analistas, en términos de habilidad de análisis, eficiencia y capacidad para cooperar tiene un impacto significativo en el esfuerzo humano. Cuanto más capaz sea el grupo, menos esfuerzo será necesario.
- *Experiencia en aplicación:* La experiencia del grupo en una aplicación similar tiene una gran influencia en el esfuerzo. Los tiempos de experiencia para los distintos niveles se muestran en la Tabla 5.6.

Muy bajo	Menos de 4 meses de experiencia media
Bajo	1 año de experiencia media
Nominal	3 años de experiencia media
Alto	6 años de experiencia media
Muy alto	Más de 12 años , o re-implementacion de un sub-sistema

Tabla 5.6: Tiempos de experiencia

- *Capacidad de los programadores:* La cuantificación es similar al caso de la capacidad de análisis, pero en este caso relacionado con los programadores. Se aplica a los programadores como grupo, pero no a los programadores individuales.
- *Experiencia con la máquina virtual:* Cuanto mayor sea la experiencia del grupo de programación con el procesador, menor será el esfuerzo necesario. Los niveles de variación de este factor se muestran en la Tabla 5.7.

Muy bajo	Menos de 1 mes de experiencia media
Bajo	4 meses
Nominal	1 año
Alto	Más de 3 años
Muy alto	Más de 12 años , o re-implementacion de un sub-sistema

Tabla 5.7 Niveles de variación

- *Experiencia en el lenguaje:* Un grupo de programadores con amplia experiencia en un lenguaje determinado programará de una manera mucho más segura, generando un menor número de defectos y de requerimientos humanos. La Tabla 5.8 muestra lo que refleja cada factor de ponderación considerado.

Muy bajo	Menos de 1 mes de experiencia media
Bajo	4 meses
Nominal	1 año
Alto	Más de 3 años

Tabla 5.8 Factor de ponderación

✓ Atributos del Proyecto

Técnicas de Programación: En este punto se consideran los niveles que se muestran en la Tabla 5.9.

Muy bajo	No se utilizan prácticas modernas de programación
Bajo	Son experimentales de algunas prácticas
Nominal	Experiencia razonable en el uso de algunas prácticas
Alto	Experiencia razonable en gran parte de las prácticas
Muy alto	Uso habitual de prácticas

Tabla 5.9: Niveles de programación

- *Utilización de herramientas de software*: El uso adecuado de herramientas de software es un multiplicador de la productividad. La puntuación varía desde Muy Bajo cuando sólo se utilizan herramientas básicas, a Muy Alto cuando se utilizan herramientas específicas.
- *Restricciones de tiempo de desarrollo:* Se refiere al plazo que requiere menor esfuerzo humano. Cualquier apresuramiento (Muy Bajo) o retraso (Muy Alto) demandará más esfuerzo.

Cabe destacar que para que este modelo sea útil deberá ser sintonizado o ajustado a las necesidades de la organización usando datos históricos. Lamentablemente es difícil que esto ocurra en la práctica.

Finalmente, el esfuerzo total corresponde a:

$$E_D = E_N \times \prod F_i$$

Si se observan los valores para el modelo Intermedio, se puede ver que los coeficientes escalares (3,2; 3,0; 2,8) decrecen a medida que se incrementa la complejidad del modo de desarrollo, al contrario que en el modelo Básico. Esta diferencia indujo a Conte y otros a presentar un conjunto de ecuaciones alternativas, cuyo comportamiento, según sus autores, mejora el de las ecuaciones originales del modelo Intermedio, ver Tabla 5.10.

	ORGANICO	SEMIDESCONECTADO	INTEGRADO
Ecuaciones del esfuerzo nominal [PM]	$E_N = 2{,}6 \times (KLDC)^{1{,}08}$	$E_N = 2{,}9 \times (KLDC)^{1{,}12}$	$E_D = 2{,}9 \times (KLDC)^{1{,}20}$

Tabla 5.10: Mejoras de las Ecuaciones del esfuerzo

5.3.4 Modelo COCOMO avanzado.

Este modelo presenta principalmente dos mejoras respecto al anterior modelo:

- ✓ Los factores correspondientes a los atributos son sensibles a la fase sobre la que se realizan las estimaciones, puesto que aspectos tales como experiencia en la aplicación, utilización de herramientas software, entre otros, tienen mayor influencia en unas fases que en otras.
- ✓ Establece una jerarquía de tres niveles de productos, de forma que:
  - Los aspectos que presentan gran variación a bajo nivel, se consideran a nivel módulo.
  - Los que presentan pocas variaciones a nivel de subsistema.
  - Los restantes se consideran a nivel sistema.

5.3.5 COCOMO 2

Al modelo inicial de COCOMO (COCOMO 81) le siguió una actualización para el lenguaje ADA en 1987, desde ese entonces el desarrollo de nuevos ciclos de vida, ocasionados por la evolución del desarrollo de software ha incrementado la dificultad de estas estimaciones. COCOMO 2[9] está adaptado a los ciclos de vida de los modelos de desarrollo de software actuales, dado que es posible de aplicar a aquellas nuevas prácticas no tradicionales de software como desarrollo rápido de aplicaciones, aplicaciones no secuenciales, reusabilidad del software, reingeniería, programación orientada a objetos, entre otras.

COCOMO 2 consiste en realidad en tres diferentes modelos:

- ✓ El Modelo de Composición de la Aplicación.

Utilizable para proyectos construidos en base a herramientas de construcción GUI modernas. Basado en nuevos puntos de objeto.

- ✓ El Modelo de Diseño Temprano (Pre-Arquitectura)

El cual se puede utilizar para conseguir estimaciones robustas de los costos y duración de un proyecto antes de determinar completamente su arquitectura. Este modelo utiliza un pequeño conjunto de parámetros de costo, y nuevas ecuaciones de estimación. Basado en Puntos de Función no Ajustados o KLDC.

- ✓ El Modelo Post-Arquitectura.

Este es el modelo COCOMO 2 más detallado. Puede ser utilizado después de haber desarrollado la arquitectura global del proyecto. Utiliza nuevos parámetros de costo, nuevas reglas de conteo de líneas y nuevas ecuaciones. Este modelo corresponde al esfuerzo de desarrollo estimado una vez que se ha fijado la arquitectura del sistema. Este modelo 'base' puede ajustarse para:

- ✓ Estimaciones más tempranas, correspondiente al Modelo de diseño temprano (Pre-arquitectura).
- ✓ Mantenimiento.
- ✓ Estimación de número de defectos esperados.

Los siguientes son los pasos a seguir:

1. *Estimación de puntos de función:* COCOMO 2 juega con la posible adopción de un modelo de puntos de función y uno de puntos de aplicación.
2. *Conversión de puntos de función a KLDC:* En COCOMO 2 se abandona definitivamente la idea de medir el tamaño del código en líneas físicas y se utilizan instrucciones o líneas lógicas de código fuente. Para medir a posteriori el número de líneas lógicas, Boehm pone a la disposición el programa CodeCount y las reglas sobre las cuales se basa tal programa.

Para efectos de Java, por ejemplo, cada punto de función corresponde a 53 líneas lógicas de código fuente. Este factor depende de cada lenguaje de programación como se muestra en la Tabla 5.11.

LENGUAJE	NIVEL	LDC/PF
C	2.5	128
Decision Support default	9	36
Ansi Basic	5	64
C++	6	53
Java	6	53
SQL	25	13
PL/I	4	80
Th Generation default	16	20
Ansi Cobol 74	3	107
Visual 4.0	11	29
Visual Basic 1	7	46
Visual Basic 2	7.5	43
Visual Basic 3	8	40
Visual Basic 4	9	36
Visual Basic 5	11	29
Visual Basic DOS	8	40
Visual C++	9.5	34
Basic Assembly	1	320

Tabla 5.11: Factores para los lenguajes de Programación

3. Aplicación de las fórmulas básicas de esfuerzo, tiempo calendario y personal requerido:

La fórmula básica de esfuerzo (PM):

$$ESFUERZO = 2{,}94 \times EAF \times (KLDC)^{E}$$

donde **EAF** corresponde al producto de los factores de ajuste de costo derivados de los parámetros de costo. Los cuales se presentan en la Tabla 5.12.

ATRIBUTOS	VALOR					
	Muy bajo	Bajo	Nominal	Alto	Muy alto	Extra alto
Atributos del Producto						
Fiabilidad	0.82	0.92	1.00	1.10	1.26	
Tamaño de la BD		0.90	1.00	1.14	1.28	
Complejidad	0.73	0.87	1.00	1.17	1.34	1.74
Reusabilidad		0.95	1.00	1.07	1.15	1.24
Necesidades de documentación	0.81	0.91	1.00	1.11	1.23	
Atributos de Computador						
Restricciones de tiempo de ejecución			1.00	1.11	1.29	1.63
Restricciones de memoria virtual			1.00	1.05	1.17	1.46
Volatilidad de la platafoma		0.87	1.00	1.15	1.30	
Atributos del Personal						
Capacidad de análisis	1.42	1.19	1.00	0.85	0.71	
Experiencia en la aplicación	1.22	1.10	1.00	0.88	0.81	
Capacidad de los programadores	1.43	1.15	1.00	0.88	0.76	
Experiencia en lenguaje y herramientas	1.20	1.09	1.00	0.91	0.84	
Experiencia en la plataforma	1.19	1.09	1.00	0.91	0.85	
Continuidad del personal	1.29	1.12	1.00	0.90	0.81	
Atributos del Proyecto						
Desarrollo Multisitio	1.22	1.09	1.00	0.93	0.86	0.80
Uso de herramientas de software	1.17	1.09	1.00	0.90	0.78	
Restricciones de tiempo de desarrollo	1.43	1.14	1.00	1.00	1.00	

Tabla 5.12: Ponderaciones Modelo Post-Arquitectura

E es el exponente no lineal, calculado según:

$$E = 0{,}91 + 0{,}01 \times \sum F_i$$

donde los **Fi** corresponden a los siguientes 5 parámetros de costo:

- ✓ PREC: Desarrollos previos similares. Ver Tabla 5.13.

0.00	Nuevo desarrollo es idéntico a previos
1.24	Es muy parecido
2.48	Bastante parecido
3.72	Aspectos novedosos
4.96	Muy diferente
6.2	Totalmente diferente

Tabla 5.13: PREC

- ✓ FLEX: Flexibilidad del desarrollo (por ejemplo grado de acuerdo con requerimientos pre-establecidos o con interfaces externas pre-existentes), ver Tabla 5.14.

0.00	Metas son generales
1.24	Cierto Acuerdo
2.48	Acuerdo general
3.72	Cierta flexibilidad
4.96	Flexibilidad Ocasional
6.2	Riguroso

Tabla 5.14: FLEX

- ✓ RESL: Manejo de riesgos y arquitectura, ver Tabla 5.15.

0.00	El plan identifica todos los riesgos críticos y establece hitos para resolverlos. El calendario y presupuesto toma en cuenta riesgos. La arquitectura puede tomarse hasta el 40% del esfuerzo de desarrollo. Se posee herramientas para resolver/mitigar riesgos y verificar la especificación de la arquitectura. Se presenta muy poca incertidumbre en la interfaz con el usuario, tecnología y desempeño y los riesgos no son críticos.
1.41	El plan identifica la mayoría de los riesgos críticos y establece hitos para resolverlos. El calendario y presupuesto toma en cuenta la mayoría de los riesgos. La arquitectura puede tomarse hasta el 33% del esfuerzo de desarrollo. Se posee herramientas para resolver/mitigar mayoría de los riesgos y verificar la especificación de la arquitectura. Se presenta poca incertidumbre en misión, interfaz con el usuario, tecnología y desempeño y los riesgos no son críticos
2.83	El plan identifica muchos de los riesgos críticos y establece hitos para resolverlos. El calendario y presupuestos generalmente toman en cuenta los riesgos. La arquitectura puede tomarse hasta el 25% del esfuerzo de desarrollo. Se posee herramientas regularmente para resolver/mitigar riesgos y verificar la especificación de la arquitectura. Existe algo de incertidumbre en misión, interfaz con usuario, tecnología, desempeño y no más de un riesgo crítico.
4.24	El plan identifica algunos de los riesgos críticos y establece hitos para resolverlos. El calendario y presupuesto toma en cuenta algunos de los riesgos. La arquitectura puede tomarse hasta el 17% del esfuerzo de desarrollo: Hay problemas con la disponibilidad del arquitecto. Se posee algo de herramientas para resolver/mitigar riesgos y verificar especificación de la arquitectura. Se presenta una considerable incertidumbre en misión, interfaz con usuario, tecnología y desempeño. Se vislumbran entre 2-4 riesgos críticos.
5.65	El plan identifica pocos riesgos críticos y establece hitos para resolverlos, calendario y presupuesto toma en cuenta pocos riesgos. La arquitectura puede tomarse hasta el 10% del esfuerzo de desarrollo, hay problemas con la disponibilidad del arquitecto (disponibilidad menor al 40%). Se posee pocas herramientas para resolver/mitigar riesgos y verificar la especificación de la arquitectura Se presenta una significativa incertidumbre en misión, interfaz con usuario, tecnología y desempeño. Se presentan entre 5-10 riesgos críticos
7.07	El plan no identifica los riesgos críticos. El calendario y presupuesto no toman en cuenta los riesgos, la arquitectura puede tomarse hasta el 5% del esfuerzo de desarrollo, hay problemas con la disponibilidad del arquitecto (disponibilidad menor al 20%). Herramientas no disponibles para resolver/mitigar riesgos y verificar especificación de la arquitectura. Se presenta una extrema incertidumbre en misión, interfaz con usuario, tecnología y desempeño. Más de 10 riesgos críticos.

Tabla 5.15: RESL

- ✓ TEAM: Cohesión del equipo de desarrollo, ver Tabla 5.16.

0.00	Interacciones fluidas, objetivos y culturas de accionistas totalmente consistentes, total habilidad y disponibilidad de accionistas para acomodar objetivos de otros accionistas, dilatada experiencia previa operando como equipo, visión y compromisos 100% compartidos.
1.10	Interacciones altamente cooperativas, objetivos y culturas de accionistas fuertemente consistentes, fuerte habilidad y disponibilidad de accionistas para acomodar objetivos de otros accionistas, considerable experiencia previa operando como equipo, visión y compromisos considerablemente compartidos.
2.19	Interacciones principalmente cooperativas, objetivos y culturas de accionistas considerablemente consistentes, considerable habilidad y disponibilidad de accionistas para acomodar objetivos de otros accionistas, mediana experiencia previa operando como equipo, visión y compromisos medianamente compartidos.
3.29	Interacciones básicas cooperativas, objetivos y culturas de accionistas básicamente consistentes, habilidad y disponibilidad básica de accionistas para acomodar objetivos de otros accionistas, poca experiencia previa operando como equipo, visión y compromisos poco compartidos.
4.38	Algunas interacciones difíciles, objetivos y culturas de accionistas algo consistentes, algo habilidad y disponibilidad de accionistas para acomodar objetivos de otros accionistas, poca experiencia previa operando como equipo, visión y compromisos poco compartidos.
5.48	Interacciones difíciles, objetivos y culturas de accionistas poco consistentes, poca habilidad y disponibilidad de accionistas para acomodar objetivos de otros accionistas, nada de experiencia previa operando como equipo, visión y compromisos nada compartidos.

Tabla 5.16: TEAM

- ✓ EPML: Nivel de madurez estimada, en relación al modelo de madurez de software, ver Tabla 5.17.

0.00	Nivel 5
1.56	Nivel 4
3.12	Nivel 3
4.68	Nivel 2
6.24	Nivel 1 Superior
7.80	Nivel 1 Inferior

Tabla 5.17: EPML

Para calcular el tiempo de desarrollo en Meses se utiliza la siguiente ecuación:

$$Tiempo = 3{,}67 \times Esfuerzo^{(0{,}28+0{,}002\times\sum F_i)}$$

Para estimar el número de personas requeridas para el desarrollo, utilizamos la siguiente aproximación:

$$Personas = Esfuerzo / Tiempo$$

En la Tabla 5.18 se muestra un resumen de los factores considerados por COCOMO en sus diferentes versiones.

		Cocomo 2: Post-arquitectura	Cocomo 2: Pre-arquitectura	Cocomo 87	Cocomo 81. Cocomo 85
Factores de Personal					
ACAP	Capacidad de Análisis	✓		✓	✓
APEX AEXP	Experiencia en la aplicación	✓		✓	✓
PCAP	Capacidad del Programador	✓		✓	✓
LEXP	Experiencia en el Lenguaje de Programación			✓	✓
VEXP	Experiencia con la MV			✓	✓
PERS	Capacidad del personal		✓		
PREX	Experiencia del personal		✓		
PCON	Continuidad del personal	✓			
PLEX PEXP	Experiencia en la plataforma	✓			
LTEX	Experiencia en Lenguaje y Herramientas	✓			
Factores del Producto					
RELY	Precisión del Software requerido	✓		✓	✓
DATA	Tamaño de la Base de Datos	✓		✓	✓
CPLX	Complejidad del producto de Software	✓		✓	✓
RUSE	Reusabilidad requerida	✓	✓	✓	
DOCU	Necesidades del documentación del ciclo de vida	✓			
RCPX	Complejidad y precisión del producto		✓		
Factores del Computador					
TIME	Restricción del Tiempo de Ejecución	✓		✓	✓
STOR	Restricción de Memoria	✓		✓	✓
TURN	Restricción del Tiempo de Respuesta			✓	✓
VIRT	Volatilidad del la máquina virtual				✓
VMVH	Volatilidad de la MV:host			✓	
VMVT	Volatilidad de la MV:target			✓	
PVOL	Volatilidad de la plataforma	✓			
PDIF	Dificultad de la plataforma		✓		
Factores del Proyecto					
TOOL	Uso de Herramientas de Software	✓		✓	✓
MODP	Práticas de programación modernas			✓	✓
SCED	Planificación del desarrollo	✓	✓	✓	✓
SECU	Seguridad de la aplicación			✓	
SITE	Desarrollo multisitio	✓			
FCIL	Facilidades		✓		

Tabla 5.18: Factores según versión de Cocomo

Estimación Pre-Arquitectura

COCOMO 2 sugiere que muchos de los 17 multiplicadores de esfuerzo no pueden estimarse en el diseño temprano, por lo que los reduce a 7 multiplicadores:

RCPX: Fiabilidad y Complejidad del producto.
RUSE: Nivel de reusabilidad del desarrollo.
PDIF: Dificultad de uso de la plataforma.
PERS: Capacidad del personal de desarrollo.
PREX: Experiencia del personal de desarrollo.
FCIL: Facilidades de desarrollo.
SCED: exigencias sobre el calendario.

Ajustes por Mantenimiento

El modelo COCOMO 2.0 trata la reutilización del software usando un modelo de estimación no lineal. Esto incluye una estimación de la cantidad de software a ser adaptado, ASLOC, y tres parámetros según el grado de modificación:

DM: porcentaje de diseño modificado.
CM: porcentaje de código modificado.
IM: porcentaje del esfuerzo original que supone integrar el software reutilizable.

El incremento que supone la comprensión del software (SU) se obtiene de la Tabla 5.19; y se expresa cuantitativamente como un porcentaje.

	MUY BAJO	BAJO	NOMINAL	ALTO	MUY ALTO
Estructura	Poca cohesión, acoplamiento alto, código "espagueti"	Cohesión moderada, acoplamiento alto	Razonablemente bien estructurado; algunas áreas fallas	Cohesión alta, acoplamiento bajo	Modularidad fuerte, información encapsulada, abstracciones de datos y de control
Correlación entre la estructura del programa y la del mundo de la aplicación	No hay correlación	Algo de correlación	Correlación moderada	Buena correlación	Alta correlación
Documentación	Oscura, confusa, obsoleta o incompleta	Código parcialmente comentado, algo de documentación útil adicional	Moderadamente completa y clara	Buena, con algunas fallas	Excelente, bien organizada, incluye justificaciones de diseño
SU	50	40	30	20	10

Tabla 5.19: Incremento de SU

El otro incremento no lineal sobre la reusabilidad tiene que ver con el grado de valoración y asimilación (AA) necesarias para determinar si un cierto módulo software completamente reutilizable es apropiado para la aplicación, e integrar su descripción dentro de la descripción global del producto. La Tabla 5.20 proporciona una escala para este porcentaje.

0	No requiere esfuerzo
2	Búsqueda básica de módulos y documentación
4	Hay que hacer algunas pruebas y evaluación de módulos
6	Hay que hacer considerables pruebas
8	Hay que hacer pruebas extensivas

Tabla 5.20: Porcentajes (reutilización)

Finalmente la ecuación usada para determinar esta reusabilidad es:

$$ESLOC = ASLOC \times \frac{(AA+SU+0,4xDM+0,3xCM+0,3xIM)}{100}$$

Lo que nos entrega una estimación de las líneas de código necesarias para el cambio.

Estimación del número esperado de defectos

La familia de modelos de COCOMO 2 incluye COQUALMO (COcomo QUAlity MOdels) conformado por dos modelos, un modelo de inyección de defectos y un modelo de remoción de defectos.

La salida del modelo de inyección de defectos es el número de defectos no triviales, lo que incluye:

- ✓ *Defectos críticos:* causan caída del sistema o pérdida irrecuperable de datos o pone en riesgo a la gente.
- ✓ *Defectos graves y moderados:* impiden que ejecuten correcta o apropiadamente funciones críticas del sistema.

Estos modelos están bajo desarrollo. Según una primera aproximación, las tasas de inyección base de defectos no triviales son:

- ✓ 10 defectos de requerimientos por KLDC.
- ✓ 20 defectos de diseño por KLDC .
- ✓ 30 defectos de codificación por KLDC.

No entraremos en detalle en el modelo de remoción de defectos. Basta con indicar que la tasa de remoción de defectos se ve afectado por inspecciones, herramientas de análisis y pruebas (de la ejecución). COQUALMO estima que la tasa residual de defectos puede variar entre 1.57 y 60 defectos por KLDC. Las pruebas contribuyen a remover entre 23 y 88% de los defectos.

Estimación de esfuerzo COCOMO 81 V/S COCOMO 2

Un estudio inicial determina que el tamaño del producto estará alrededor de 64000 LDC. Por el tamaño del producto se presume que se deben utilizar las ecuaciones de modo semiconectado de COCOMO 81.

A partir de la aplicación de las ecuaciones se obtienen resultados como los que se muestran en la Tabla 5.21.

	Cocomo 81 (básico)	Cocomo 2
Esfuerzo [PM]	316	219
Tiempo de desarrollo [Meses]	10.7	16.5
Esfuerzo [PM]	29	13

Tabla 5.21: Resultados de Cocomo

Estas estimaciones se realizaron manteniendo los conductores de costo de forma nominal, lo que implica que el esfuerzo estimado por COCOMO 81 básico corresponde también a lo entregado por COCOMO 81 intermedio.

Es posible observar que ambas técnicas generan resultados similares en cuanto a estimación de esfuerzo, pero COCOMO 2 resulta más confiable, en el sentido que estima un mayor tiempo de desarrollo, pero manteniendo un equipo de trabajo relativamente pequeño comparado con COCOMO 81.

La Figura 5. 2 y 5.3 muestran las estimaciones de esfuerzo en [PM] y Personal entregadas por ambos métodos para distintos tamaños de proyectos en LDC. En este caso es posible observar que ambos métodos presentan resultados muy similares para proyectos de pequeño tamaño, al ir aumentado el tamaño del proyecto, los resultados se vuelven diferentes.

En relación al Tiempo de desarrollo, ver Figura 5.4, se observa que COCOMO 2 recomienda un mayor tiempo de desarrollo que COCOMO 81, esto conlleva a un menor tamaño de equipo de desarrollo, lo que resulta bastante lógico considerando la gran utilización de tiempo en comunicación que se presenta al utilizar equipos de trabajo muy grandes.

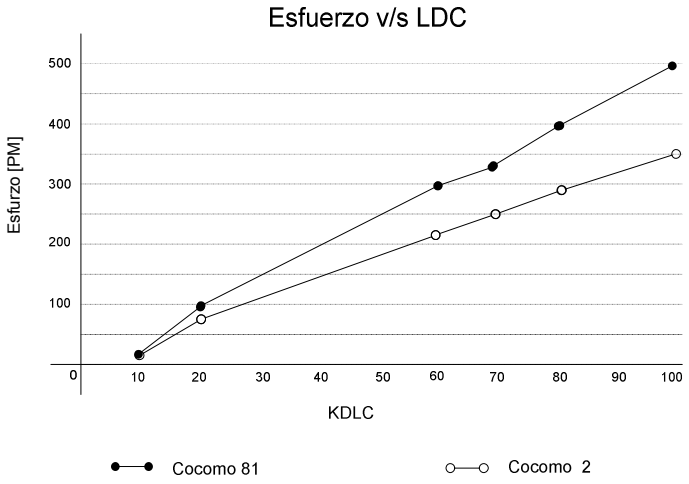


Figura 5.2: Esfuerzo v/s LDC

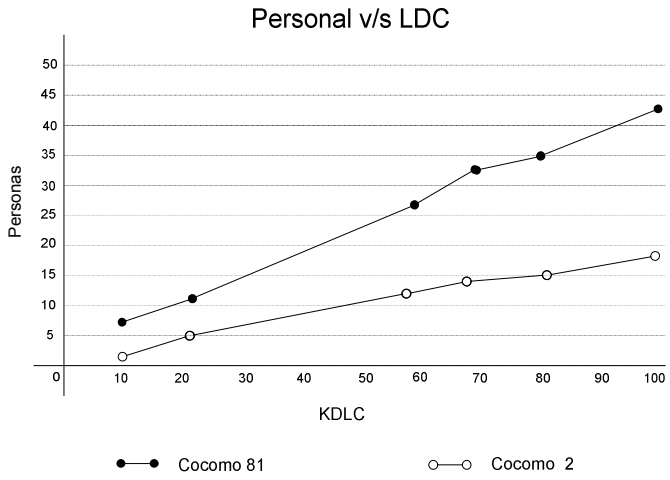


Figura 5.3: Personal v/s LDC

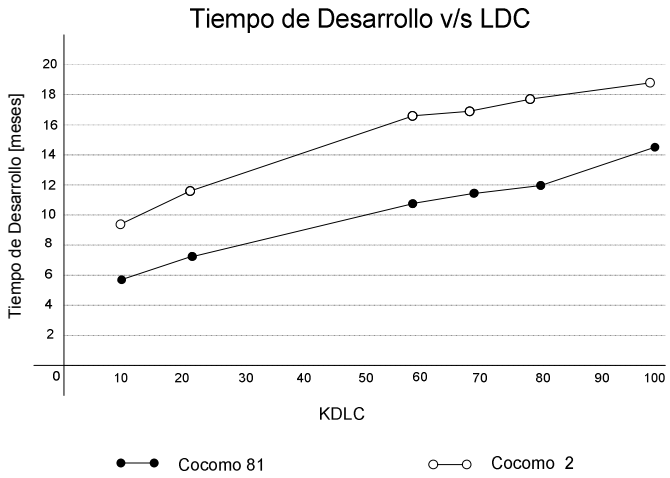


Figura 5.4: Tiempo de Desarrollo v/s LDC

5.3.6 Modelo Puntos de Función.

Los Puntos de Función[7] son una métrica para establecer el tamaño y la complejidad de los sistemas informáticos basada en la cantidad de funcionalidad requerida y entregada a los usuarios. También, se dice que los Puntos de Función miden el tamaño lógico o funcional de los proyectos o aplicaciones

El análisis por puntos de función es un método para cuantificar el tamaño y la complejidad de un sistema software en términos de las funciones de usuario que este desarrolla (o desarrollará). Esto hace que la medida sea independiente del lenguaje o herramienta utilizada en el desarrollo del proyecto.

El análisis por puntos de función está diseñado para medir aplicaciones de negocios; no es apropiado para otro tipo de aplicaciones como aplicaciones técnicas o científicas. Esas aplicaciones generalmente median con algoritmos complejos que el método de puntos de función no está diseñado para manejar.

La metodología fue desarrollada por Allan J. Albrecht, como una métrica del tamaño de un sistema de información automatizado. La estimación del tamaño es necesario para la medición de productividad, en las actividades de mantención de sistemas y al estimar el esfuerzo necesario para realizar estas actividades .

Aspectos relevantes:

- ✓ No depende de la tecnología, son parámetros normalizados; lo que permite poder comparar dichos resultados.
- ✓ Se pueden calcular muy tempranamente, luego de la especificación de requerimientos, lo que facilita poder estimar el dimensionamiento de los proyectos en estudio.

Se caracterizan por:

- ✓ Ser un método independiente de las herramientas de análisis, diseño y programación, debido a que se preocupa de la complejidad de las funciones a implementar, además independientes del lenguaje, herramientas o metodologías utilizadas en la implementación; por ejemplo, no tienen que considerar lenguajes de programación, sistemas de administración de bases de datos, hardware, o cualquier otra tecnología de procesamiento de datos.
- ✓ Requerir de una descomposición funcional del proyecto de software a realizar, en términos tales que se detectan todas las piezas elementales que componen el producto final.
- ✓ La estimación de la "Cantidad de Puntos de Función" de las funciones medidas, se realiza contando la cantidad de entradas, salidas, archivos, consultas e interfaces que utiliza. A mayor cantidad, mayor es el "peso de complejidad" que se le asignará.
- ✓ Ajustar la estimación del esfuerzo requerido, determinando la presencia de ciertos elementos que dificultan el desarrollo del proyecto.
- ✓ Pueden ser estimados a partir de la especificación de requisitos o especificaciones de diseño, haciendo posible de este modo la estimación del esfuerzo de desarrollo en etapas tempranas del mismo. Como los puntos de función están íntimamente relacionados con la declaración de requisitos, cualquier modificación a ésta, puede ser reflejada sin mayor dificultad en una re estimación.
- ✓ Estar basados en una visión externa del usuario del sistema, los usuarios no técnicos del software poseen un mejor entendimiento de lo que los puntos de función están midiendo. El método resuelve muchas de las inconsistencias que aparecen cuando se utiliza líneas de código como métrica del tamaño del software.

Existen varias versiones de este método. El método original se presenta en 1979 y tiene los siguientes problemas:

- ✓ La complejidad es totalmente subjetiva.
- ✓ El rango se presume como insuficiente para reflejar diferencias de complejidad.

Luego en 1984 se obtiene la revisión del método el cual se explica más adelante.

Los elementos principales del método son:
Los puntos de función, se asocian al producto de software tal como son vistos por el usuario, es decir:

- ✓ *Entradas Externas* (o número de entradas de usuario)**:** Es un proceso elemental en el cual los datos cruzan los límites de afuera hacia adentro, pueden venir de una pantalla de entrada de datos o de otra aplicación, son usados para mantener uno o más archivos lógicos internos. Los datos pueden ser de información de control o del negocio, si son datos de información de control, no se tiene que actualizar un archivo lógico interno.
- ✓ *Salidas Externas* (o número de salidas de usuario)**:** Es un proceso elemental en el cual los datos derivados cruzan los límites de adentro hacia afuera. Los datos crean reportes o archivos de salida enviados a otras aplicaciones. Estos reportes o archivos son creados de uno o más archivos lógicos internos o archivos de interfaz externas. También pueden ser pantallas o mensajes de error que proporcionan información.
- ✓ *Consultas Externas* (o número de peticiones de usuario)**:** Es un proceso elemental con componentes de entrada y de salida que resultan de la adquisición de datos de uno o más archivos lógicos internos o archivos de interfaz externas. El proceso de entrada no actualiza ningún archivo lógico interno, y el proceso de salida no contiene datos derivados.
- ✓ *Archivos internos lógicos* (números de archivos): Es un grupo de datos definidos por el usuario que están relacionados lógicamente, que residen en su totalidad dentro de los límites de la aplicación y que son mantenidos a través de entradas externas.
- ✓ *Archivos de interfaz externo:* Es un grupo de datos definidos por el usuario que están relacionados lógicamente y que solo son usados para propósitos de referencia. Los datos residen enteramente fuera de la aplicación y son mantenidos por otra aplicación. El archivo de interfaz externo es un archivo lógico interno para otra aplicación.

Primero se trabaja con los Puntos de Función no Ajustados (PFNA), los que serán obtenidos al ponderar las: Entradas, Salidas, Consultas, Archivos internos y Archivos de interfaz externo, por un factor asociado a la complejidad de su procesamiento, tales factores se muestran en la Tabla 5.22.

Componentes	Simple	Promedio	Complejo
Entradas	3	4	6
Salidas	4	5	7
Consultas	3	4	6
Archivos	7	10	15
Interfaces	5	7	10

Tabla 5.22: Factores PF

Los pesos que definen la complejidad de procesamiento se justifican como el valor relativo de la función al usuario - cliente[3]. La indicación de Simple, Promedio, Complejo; reflejan en forma teórica y estimada el esfuerzo requerido de procesamiento para las respectivas componentes a ser consideradas, las que se describen en forma global a continuación:

Simple: Factores humanos no afectan, existen poco nivel de transformaciones, no existen consideraciones significativas de rendimiento entre otras.

- ✓ *Promedio:* Los ítems mencionados anteriormente inciden de manera intermedia en el procesamiento de cada componente.
- ✓ *Complejo:* Están afectando factores humanos, muchas transformaciones, existen consideraciones significativas de rendimiento, entre otras.

Se definirán la complejidad de procedimientos para cada una de las componentes:

- ✓ Entradas Externas, Salidas Externas y Consultas Externas:

  - *Complejidad Simple:* Tiene pocos elementos de datos, tiene pocas referencias a tipos de archivos internos, tiene pequeño o ninguna intervención del factor humano en el diseño.
  - *Complejidad Compleja:* Tiene muchos elementos de datos, tiene muchas referencias a tipos de archivos internos, y tiene una cantidad significante de consideraciones del factor humanas en el diseño.
  - *Complejidad Promedio:* El promedio está entre la Simple y Compleja.

- ✓ Archivos internos lógicos y Archivos de interfaz externo:

  - *Complejidad Simple:* Los archivos tienen pocos registros.
  - *Complejidad Compleja:* Los archivos tienen muchos registros.
  - *Complejidad Promedio:* El promedio está entre la Simple y Compleja.

[3] Dato recogido de estadísticas en la revista IT METRICS STRATEGIES

Al multiplicar el número de componentes por su factor correspondiente y sumando todos los totales se obtienen los Puntos de Función No Ajustados (PFNA).

Finalmente, Los *puntos de función* (PF) se obtienen al ponderar los puntos de función no ajustados (PFNA) por un Factor de Complejidad Técnica (FCT):

$$PF = PFNA * FCT$$

Este se encuentra asociado a 14 características de la aplicación, ver Tabla 5.23.

	CARACTERISTICAS	DESCRIPCION
1	Comunicación de datos	¿Cuántas herramientas de comunicación hay para ayudar en la transferencia o intercambio de información de la aplicación o sistema?
2	Procesamiento de datos distribuidos	¿Cómo son manejados los datos distribuidos y las funciones de procesamiento?
3	Nivel de ejecución	¿El tiempo de respuesta o el nivel de eficiencia es requerido por el usuario?
4	Configuración más usada	¿Qué tanto se usa la plataforma de hardware en donde la aplicación se va a ejecutar?
5	Nivel de transacciones	¿Qué tan frecuentemente se ejecutan las transacciones al día, semana, mes?
6	Captura de datos En Línea	.¿Qué porcentaje de información se captura En Línea?
7	Eficiencia del usuario final	¿Se diseñó la aplicación pensando en la eficiencia del usuario final?
8	Actualización En Línea	¿Cuántos archivos lógicos internos se actualizan en transacciones En Línea?
9	Procesamiento complejo	¿La aplicación tiene mucho procesamiento lógico o matemático?
10	Reusabilidad	¿La aplicación se desarrolló para cumplir una o muchas necesidades del usuario?
11	Facilidad de Instalación	¿Qué tan difíciles son la conversión y la instalación?
12	Facilidad de Operación	¿Qué tan efectivos y/o automatizados son los procedimientos de inicio, respaldo y recuperación?
13	Múltiples Sitios	¿La aplicación se diseñó, desarrolló y soportó específicamente para ser instalada en múltiples sitios para varias organizaciones?
14	Facilidad de mantenimiento	¿La aplicación se diseñó, desarrolló y soportó específicamente para facilitar el mantenimiento?

Tabla 5.23: Factor de Complejidad

El factor de Complejidad (FCT) está basado en características generales del sistema que evalúan la funcionalidad general de la aplicación que se está midiendo. Cada característica tiene asociada una descripción que ayuda a determinar su nivel de influencia en una escala de cero (sin influencia) a cinco (influencia fuerte).

Cada una de estas características aporta un grado de influencia, este aporte está tabulado en valores que van desde 0 a 5 y que reflejan lo que se muestra en la Tabla 5.24.

GRADO DE INFLUENCIA	PESOS
Sin influencia	1
Accidental	2
Moderado	3
Medio	4
Significativo	5
Influencia fuerte	6

Tabla 5.24: Grados de Influencias para PF

Pasos para el cálculo de los Puntos de Función

Paso 1: *Definir Entradas, Salidas, Consultas, Archivos lógicos internos y Archivos de interfaz externo*. Asociar a cada ítem de los grupos mencionados la identificación de *simple*, *promedio* o *complejo* según corresponda a las condiciones de desarrollo, observadas en ellos.

Por ejemplo:

COMPONENTE	SIMPLE	PROMEDIO	COM-PLEJAS	TOTAL
Entradas	2	3	1	6
Salidas	2	2	0	4
Consultas	...	...	...	...
Archivos	...	...	...	...
Interfaz	...	...	...	...

Tabla 5.25: Contabilización de Componentes

Paso 2: Obtener los Puntos de Función No ajustados, el que estará dado por:

$$PFNA = \sum_{i=1}^{5} PF_i \quad ,donde$$

$$PF_i = Cantidad\ de\ veces\ en\ que\ se\ obtendra\ el\ item \times factor(o\ peso)_i\ donde\ i = 1\ a\ 5$$

Por ejemplo:

COMPONENTE	SIMPLE	factor	PROMEDIO	factor	COM-PLEJAS	factor	$\sum$
Entradas	2	x 3	3	x 4	1	x 6	24
Salidas	2	x 4	2	x 5	0	x 7	18
Consultas	...	x 3	...	x 4	...	x 6	...
Archivos	...	x 7	...	x 10	...	x 15	...
Interfaz	...	x 5	...	x 7	...	x 10	...
						PFNA	42

Tabla 5.26: Cálculo de PFNA

Paso 3: Obtener el Factor de Complejidad Técnica, el que estará dado por:

$$FCT = 0.65 + 0.01 \times \sum_{i=1}^{14} C_i$$

Paso 4: Finalmente para obtener el valor de Puntos de Función, estará dado por:

$$PF = PFNA \times FCT$$

5.3.7 Puntos característicos (Features Points)

En 1986 la compañía Software Productivity Research desarrolló un método experimental que simplificaba el cálculo de los tradicionales PF, por lo tanto, reduciendo el esfuerzo para llevarlo a cabo. Este nuevo método agrega tres nuevos objetivos a los anteriores:

- ✓ Calcular PF con facilidad y rapidez.
- ✓ Predecir tamaño de código fuente para ciertos lenguajes.
- ✓ Retroajustar PF para software existente.

Se adicionó un nuevo componente a los cinco ya existentes en la versión de 1984: algoritmos (complejidad algorítmica), que podría ser calculado mediante Complejidad Ciclomática.

Lo anterior y el orden de los demás componentes quedan expresados en la Tabla 5.27.

COMPONENTE	PESO
Entradas	4
Salidas	5
Consultas	4
Archivos lógicos	7
Archivos de Interfaz	7
Algoritmos	3

Tabla 5.27: Componentes v/s Peso empírico

Los 14 Factores de Complejidad Tecnológica fueron reemplazados por tres factores importantes que tienen el efecto de incrementar o decrecer el valor obtenido en los PFNA a $\pm 40\%$. Estos son:

- ✓ Complejidad del Problema.
- ✓ Complejidad de los Datos.
- ✓ Complejidad del Código.

Estos factores se ordenan de la siguiente manera:

- ✓ Complejidad del Problema

1. Algoritmos y cálculos simples
2. Mayoría de situación 1
3. Algoritmos y cálculos promedio
4. Mayoría de situación 2
5. Algoritmos y cálculos complejos

- ✓ Complejidad de los Datos

1. Datos simples con pocas variables y baja complejidad
2. Numerosas variables con relaciones simples
3. Múltiples archivos, campos e interacciones de datos
4. Estructura de archivos e interacción de datos complejos (BD)
5. Estructura de archivos e interacción de datos muy complejos (BDOO)

- ✓ Complejidad del Código

1. No procedural
2. Estructurado
3. Bien estructurado, con módulos reusables
4. Estructura adecuada, pero con un grado de complejidad
5. Estructura adecuada, pero con un muy alto grado de complejidad

Una vez ordenados estos tres factores se suman y se asocian con un valor en una tabla determinada, con ello se obtiene el factor de ajuste. Posteriormente el cálculo continua como los PF tradicionales.

5.3.8 Modelo Algorítmico de Costos de Software.

Modelo Algorítmico de Costos de Software (MACOS)[10], usa de base la métrica Puntos de Función y un Factor de Ajuste (FA) asociado a características del producto y del ambiente de desarrollo, para calcular el Esfuerzo Ajustado Estimado, EAE (medido en Horas - Hombre) y el Tiempo de Desarrollo Estimado, TDESE (medido en Horas). El Factor de Ajuste del modelo MACOS considera la incidencia de 32 factores asociados a aspectos técnicos del diseño y construcción del producto de software, así como al entorno en el que se realiza el proyecto. Estos factores se clasifican en: difíciles de cuantificar y cuantificables.

Factores difíciles de cuantificar

INTUSU : Calidad de las interfaces con el usuario.
ESPREQ : Calidad de la especificación de requerimientos.
RECONT : Existencia de un plan de recursos y control del proyecto.
PLAPRO : Existencia de un plan detallado del proyecto.
TESTVV : Existencia de un plan detallado de pruebas, de verificación y validación.
AMBTRA : Calidad del ambiente de trabajo.
SEGPRI : Restricciones de seguridad y privacidad existentes.
DOCEXI : Calidad de la documentación.
CONPER : Calidad de la administración del proyecto.

Factores cuantificables, perceptibles para el usuario del producto

VOLTRA : Volumen de transacciones a realizar
TAMADB : Tamaño de la base de datos.

DATARC : Características de captura, validación, actualización y almacenamiento de datos.
FACOPE : Facilidad de operación
FAFPLCA : Facilidad y flexibilidad de cambio.
RENOPE : Rendimiento requerido para la operación del producto de software.
CONPRO : Confiabilidad del producto de software entregado.
RESTPO : Restricciones en el tiempo de desarrollo

- ✓ Del ambiente usuario

MODOP : Modo de desarrollo y operación del producto de software.
EXUSUA : Experiencia del usuario en la aplicación y en el procesamiento de datos.

Factores cuantificables, perceptibles para el desarrollador:

- ✓ Del personal de desarrollo

CAPAN : Capacidad de los analistas.
EXPANA : Experiencia de los analistas.
CAPRO : Capacidad de los programadores.
EXPMV : Experiencia del grupo en la plataforma de desarrollo (configuración de hardware y software) necesaria para el desarrollo del producto de software.
EXPLEN : Experiencia en el lenguaje
COMPE : Continuidad del personal

- ✓ Del computador

RAPCPU : Restricciones del almacenamiento principal y de uso de CPU durante el desarrollo.
TRESPOC : Tiempo de respuesta promedio del computador.
VOLAMV : Volatilidad de la plataforma de desarrollo.

- ✓ Del proyecto

HERRAD : Herramientas de ayuda utilizadas.
PMOPRO : Prácticas modernas de programación utilizadas.

- ✓ Del producto

COMPLP : Complejidad del producto de software.
FAICMA : El diseño del producto de software contempla su facilidad de instalación, conversión y mantenimiento.

Para cuantificar estos factores se asigna a cada uno un valor denominado grado de influencia (GI), que varía entre -5 y +5, el cual indica la incidencia del factor en los costos del proyecto, donde el signo más significa aumento del esfuerzo y el signo menos significa disminución del esfuerzo.

Grado de Influencia	Valor
Sin influencia	0
Accidental	+/- 1
Moderado	+/- 2
Medio	+/- 3
Significatuvo	+/- 4
Esencial	+/- 5

Tabla 5.28: Grados de Influencia

El modelo MACOS sigue los siguientes pasos:

Paso 1: Calcular los Puntos de Función No Ajustados (PFNA), esto es, contar por categoría las entradas, salidas, consultas, archivos internos e interfaces con otros productos de software. Asignar a cada una de ellas, una de las siguientes ponderaciones: Simple, Promedio o Compleja.

PF_i = $cuenta_i$ * ponderación; i=1,…, 5

PFNA = suma(PF_i)

Paso 2: Calcular el FA de la siguiente manera:

A cada uno de los 32 factores que componen el Factor de Ajuste (FA), asignar uno de los grados de influencia (GI), según su incidencia en los costos del proyecto.

SGI = suma(GI_i), i=1,...,32

FA = 1 + (SGI/400)

- ✓ *Calcular el Esfuerzo Ajustado Estimado*:

EAE = 1,04 * $(PFNA)^{1,22}$ * FA [horas-hombre]

- ✓ *Calcular el Tiempo de Desarrollo estimado:*

Contar el personal de tiempo completo disponible para el desarrollo del proyecto (NPTC).

TDESE = EAE/NPTC [horas]

Según los antecedentes del proyecto y la productividad del personal que se asignará.

5.3.9 Modelo de Estimación para Proyectos Cliente/Servidor

En la actualidad para los proyectos del tipo Cliente/Servidor, la aplicación de los métodos para estimar esfuerzo se basa en la experiencia adquirida en proyectos no basados en esta arquitectura, es más se trata de tomar las distintas partes del proyecto como subsistemas más pequeños, con el fin de que con la estimación de ellos se obtuviere la estimación del proyecto completo.

Cabe destacar que muchas de las estimaciones utilizadas para el desarrollo de métodos empíricos, son obtenidas utilizando la experiencia de los analistas en

otros proyectos, por lo que estas estimaciones no tienen un respaldo numérico (viéndolo desde un punto de vista más tangible) de los resultados obtenidos.

El modelo que se plantea para calcular la estimación de esfuerzo de los proyectos Cliente/Servidor, se basa en el método de puntos de función. Para aplicarlo a proyectos Cliente/Servidor se debe tener en cuenta los siguientes puntos al calcular los puntos de función:

- ✓ En la componente de los archivos de interface externos, la ponderación se verá afectada por la complejidad de la creación de la base de datos del sistema. Por esto se deberá dar un peso de promedio o complejo a este componente, dependiendo del tamaño de la base de datos.
- ✓ En la componente de los archivos internos lógicos, se tomaran de todas las bases de datos (que intercalan con el sistema), como unidades separadas, es decir, cada tabla de la base de datos (o archivo de la base de datos) se tomará como un archivo interno lógico.
- ✓ En cuanto al calculo del factor de complejidad para los puntos de función, se tendrá mucho cuidado a todas aquellas características relacionadas a la comunicación entre las bases de datos, ya que serán las que influirán en un alto nivel durante la ejecución del sistema.

El modelo considera los puntos de función (medidos) del sistema y el número de archivos (o el número de tablas que contiene la base de datos) (medidos) del sistema, como se verá más adelante. Con estas dos variables independientes se podrá estimar el esfuerzo del desarrollo del sistema.

Pasos para aplicar el modelo a un proyecto Cliente/Servidor

Se presenta a continuación los pasos a seguir para estimar el esfuerzo de desarrollo de un proyecto Cliente/Servidor:

Paso 1: Calcular los Puntos de Función del proyecto

Definir en forma global para el proyecto: *Entradas*, *Salidas*, *Consultas*, *Archivos lógicos internos* y *Archivos de interfase externos*. Asociar a cada ítems de los grupos mencionados la identificación de *simple*, *promedio* o *complejo* según corresponda a las condiciones de desarrollo, observadas en ellos.
Por ejemplo:

COMPONENTE	SIMPLE	PROMEDIO	COMPLEJAS	TOTAL
Entradas	2	3	1	6
Salidas	2	2	0	4
Consultas	...	...	...	...
Archivos	...	...	...	...
Interfaz	...	...	...	...

Tabla 5.29: Contabilización de Componentes

Paso 2: Obtener los Puntos de Función No ajustados ($PFNA$), el que estará dado por:

Por ejemplo: Suponiendo que se tiene los mismos datos que el caso anterior.

$$PFNA = \sum_{i=1}^{5} PF_i \quad ,donde$$

$$PF_i = Cantidad\ de\ veces\ en\ que\ se\ obtendra\ el\ item \times\ factor(o\ peso)_i\ donde\ i = 1\ a\ 5$$

COMPONENTE	SIMPLE	factor	PROMEDIO	factor	COM-PLEJAS	factor	$\sum$
Entradas	2	x 3	3	x 4	1	x 6	24
Salidas	2	x 4	2	x 5	0	x 7	18
Consultas	...	x 3	...	x 4	...	x 6	...
Archivos	...	x 7	...	x 10	...	x 15	...
Interfaz	...	x 5	...	x 7	...	x 10	...
						PFNA	42

Tabla 5.30: Cálculo PFNA

Paso 3: Obtener el Factor de Complejidad Técnica, el que estará dado por:

$$FCT = 0.65 + 0.01 \times \sum_{i=1}^{14} C_i$$

Paso 4: Finalmente para obtener el valor de Puntos de Función (pf), estará dado por:

$$PF = PFNA \times FCT$$

Por lo tanto se obtienen los ***Puntos de Función*** en [PF] del proyecto.

Paso 5: Aplicación del Modelo para Estimación de Proyectos C/S [8]

$$EsfuerzoEstimado = 11.453 + 0.031 \cdot X + 0.112 \cdot Y \ \ [PM]$$

donde:

$$X = Puntos\ de\ Función\ Medidos\ del\ Sistema \begin{cases} X \geq 185 \\ X \leq 729 \end{cases}$$

$$Y = Nº\ de\ Archivos \begin{cases} Y \geq 1 \\ Y \leq 70 \end{cases}$$

Paso 6: Presentación de resultados

En estos momentos hemos encontrado el Esfuerzo Estimado del proyecto en estudio en unidades [PM] (Persona–Mes, es decir, el tiempo en meses que se tarda una persona en realizar el proyecto entero solo), dependiendo del número de personas a trabajar en el proyecto se va disminuyendo el tiempo de desarrollo del proyecto.

5.3.10 Modelo de estimación para aplicaciones Intranet/Internet

La siguiente técnica[3] es una variación del método de estimación de tamaño y esfuerzo basado en los Puntos Función, para proyectos de software con reutilización de componentes objetuales.

En este modelo de estimación, se modifica la técnica de Puntos de Función para destacar la importancia del concepto de reutilización en el desarrollo de aplicaciones Intranet/Internet.

Método de punto de Función incorporando la reusabilidad en el desarrollo de software

Antes de revisar las fases de la metodología es necesario introducir algunos datos que serán de utilidad durante los cálculos.

Para analizar el impacto de la reusabilidad en los proyectos se definen los siguientes factores que influyen directamente en la reutilización. Estos factores se han clasificado en tres categorías y se muestran en la Tabla 5.31.

Interfaz Usuario		
1	Formularios de Datos	Interfaces que permiten el ingreso de datos.
2	Validaciones de datos de entrada por campo	Verificar que cada dato ingresado sea coherente a su tipo.
3	Mensajes de errores en ingreso de datos	Alertas explicativas a algún error cometido al ingresar un dato.
Reglas de Negocio		
4	Clases Genéricas	Entidad que permite ser instanciada por un objeto determinado.
5	Clase de tipo gestores	Clase que permite emitir consultas básicas sobre una determinada clase.
6	COTS	Componentes reutilizables disponibles.
Almacenamiento de Datos		
7	Funciones de rutinas de validaciones	Procedimientos empleados en rutinas especificas para validaciones.
8	Script de tablas y procedimientos almacenados	Generación de la estructura de datos que sustenta un sistema.
9	Modulos de procesos especificos	Funciones que operan sobre determinadas clases.

Tabla 5.31: Factores de MACOS

Para los factores anteriores es necesario considerar las siguientes ponderaciones, estas se utilizan para llevar a cabo el ajuste de las estimaciones.

Factor		Ponderación
Interfaz Usuario		
1	Formularios de Datos	3
2	Validaciones de datos de entrada por campo	2
3	Mensajes de errores en ingreso de datos	3
Reglas de Negocio		
4	Clases Genéricas	5
5	Clase de tipo gestores	6
6	COTS	4
Almacenamiento de Datos		
7	Funciones de rutinas de validaciones	3
8	Script de tablas y procedimientos almacenados	2
9	Modulos de procesos especificos	4

Tabla 5.32: Factor de Reutilización

Las Fases de la Metodología son:

Fase 1: Analizar los requerimientos funcionales del sistema a desarrollar.
Fase 2: Aplicar la técnica actual de punto de función, para los requerimientos establecidos en el punto anterior.
Fase 3: Identificar los factores candidatos para definir los componentes reusables a utilizar en la estimación mostrada en la tabla de Ponderación para los factores. A cada uno de estos factores se les debe estimar el porcentaje de reutilización a partir de las ocurrencias, como muestra en la Tabla 5.33.

	Ocurrencia	Ocurrencia	
Nº Factor	Reutilización	Total	% Reutilización
1	x	y	x/y
...			
9			

Tabla 5.33: Cálculo del % de reutilización

Fase 4: Finalmente predecir el impacto de los componentes reusables en el nuevo desarrollo. A través de la Tabla 5.34 de ponderaciones presentada, encontrar el factor de reutilización.

Nº Factor	% Reutilización	Ponderación	% Reutilización
1	A	3	3*A
2	B	2	2*B
3	C	3	3*C
4	D	5	5*D
5	E	6	6*E
6	F	4	4*F
7	G	3	3*G
8	H	2	2*H
9	I	4	4*I
			Sumatoria Total

Tabla 5.34: Cálculo del factor de reutilización

Fase 5: Analizar el resultado obtenido en la fase 2, para combinarlos con los del paso anterior y obtener la nueva estimación.

Pf reuso = Pf teorico* (1- factor de reuso)

Con lo que se obtiene el esfuerzo en PF corregido por la reutilización.

5.3.11 Consideraciones en la utilización de los modelos

En resumen, estos modelos son útiles en la medida que sea posible encontrar y utilizar métricas globales con algún valor estadístico, esto es difícil por los siguientes factores:

- ✓ Muchos lenguajes diferentes
- ✓ Inconsistencias en las mediciones de las LOC y los PF
- ✓ Muchas áreas de aplicación muy distintas
- ✓ No hay consistencia en los estándares de calidad de una métrica de productividad en particular

De publicaciones recientes (1993, 1994) a modo de muestra, aparece la productividad en un gran rango como, se muestra en la Tabla 5.35.

AREA DE APLICACION	INTEGRADO
Sistemas de información	800-3200
Sistemas de aplicación	400-1000
Sistemas integrados de tiempo real	100-600
Sistemas críticos	30-400

Tabla 5.35: Productividad (LOC/HM)

Estos valores pueden ser superados si se utilizan buenas herramientas CASE para generar el código, se integra a los sistemas código reutilizable y se tiene experiencia en el área. Se ha publicado información acerca del rango de calidad en los productos de software, como se muestra en la Tabla 5.36.

AREA DE APLICACION	DEFECTOS/LOC
Sistemas de información	5-30
Sistemas de aplicación	0.8-10
Sistemas integrados de tiempo real	0.5-5
Sistemas críticos	0.01-0.1

Tabla 5.36: Rango de calidad del producto

Para obtener una visión más completa de estas cifras, en la Tabla 5.37 se comparan las prácticas de desarrollo de software en USA y en Japón:

Aspectos cuantitativos

AREA DE APLICACION	JAPON	USA	AREA DE APLICACION
Rotación de personal	<1%	10%	anual
Presupuesto entrenamiento	6-8%	1-2%	del presupuesto total
Administración de la calidad	10%	2%	del costo total
Productividad del desarrollo	20-50	10-25	LOC / día
Certificación de Calidad	>10%	2%	usa grupo SQA independiente
Productividad del mantenimiento	100	25-50	LOC / día
Tasa defectos	0.008	1-30	defectos/KLOC

Tabla 5.37: Desarrollo de software en USA y en Japón

Aspectos cualitativos de las prácticas de desarrollo en Japón.

- ✓ Espacios de desarrollo abiertos con 70 a 120 personas desde administradores a programadores en el mismo lugar físico, sin espacios individuales y 2 a 3 programadores por terminal.
- ✓ Poco énfasis en uso de tecnología de punta o estado del arte en Ingeniería de Software. Muchas horas de trabajo, sobretiempo pagado y pocas secretarias.

Se ha concluido que la diferencia en las cifras se debe a:

- ✓ Enfoque en pocas tecnologías probadas, trabajando con ellas hasta optimizarlas a su trabajo.
- ✓ Enfoque en la calidad y confiabilidad más que en productividad.
- ✓ Enfoque en gestión y el personal más que en la tecnología.
- ✓ Uso meticuloso y rutinario de métricas que son recolectadas, analizadas y corregidas.

5.4 Estimación de Esfuerzo

El esfuerzo de desarrollo de un proyecto significa básicamente cuantas personas se van a ocupar en el desarrollo de un proyecto, y dependiendo del número de personas se puede reducir el tiempo de desarrollo.

El esfuerzo de un proyecto se mide en unidades de:

Persona - Año = [PA]
Persona - Mes = [PM]
Persona – Día = [PD]

Para los cálculos se tendrá en consideración las siguientes conversiones:

- ✓ *1 año posee 12 meses de trabajo.*
- ✓ *1 año posee 240 días de trabajo.*
- ✓ *1 mes posee 20 días de trabajo.*

Por lo tanto se tiene las conversiones:

1 [PA] (Persona – Año) = 2 [PM] (Persona – Mes)
1 [PA] (Persona – Año) = 240 [PD] (Persona – Día)
1 [PM] (Persona – Mes) = 20 [PD] (Persona – Día)

Para calcular el esfuerzo de desarrollo de los proyectos, se tomarán en cuenta los siguientes tipos de esfuerzo:

✓ *Esfuerzo Medido:* Aquel que representa el tiempo real de término del proyecto, que se obtiene por los datos recogidos del proyecto en estudio. Viene dado por el total de personas involucradas por el tiempo total del desarrollo del proyecto.

$$Esfuerzo\ Medido = N^{o}\,de\,Personas \times Meses\,de\,Duración \quad [\mathrm{PM}]$$

✓ *Esfuerzo Estándar:* Aquel que representa un esfuerzo de desarrollo tomada de una productividad constante. Se tomará como dato para su cálculo, la productividad mundial[4]. Se obtienen los siguientes valores de Productividad para el cálculo del esfuerzo Estándar de desarrollo de proyectos, ver Tabla 5.38.

Productividad por Puntos de Función		
	[PF/PA]	[PF/PM]
Mundial	92.5	7.70833
USA	88.17	7.34750
Canada	111	9.25

Tabla 5.38: Productividad por PF

Se utilizará la Productividad Mundial, como una productividad estándar, que represente el estándar de desarrollo de proyectos de software.

[4] Datos recogidos de estadísticas en la revista IT METRICS STRATEGIES

Por lo tanto el esfuerzo se calculara mediante la fórmula:

$$Esfuerzo\,Estándar = \frac{Puntos\ de\ Función\ \ [PF]}{\mathrm{Pr}oductividad\ \ [PF/PM]}\ [\mathrm{PM}]$$

Finalmente como resultado se obtendrá el esfuerzo de desarrollo en las unidades de Persona-Mes [PM], lo que equivale el tiempo necesario en meses de trabajo para una persona, para el proyecto entero. Dependiendo del caso se puede transformar éste dato a esfuerzo por Persona-Año [PA] o Persona-Día [PD], todas estas conversiones utilizando los datos anteriormente mencionados.

Existen estudios que relacionan las líneas de código y los puntos de función, aunque ello es dependiente del lenguaje de programación.

Algunos resultados se muestran en la Tabla 5.39.

Lenguaje de programación	LOC/PF (promedio)
Assembler	300
C	120
COBOL	100
FORTRAN	100
PASCAL	90
Ada	70
LOO	30
4GL	20
Generador de código	15

Tabla 5.39: Lenguajes v/s LOC/PF

Uso de Puntos de función para la estimación del costo del proyecto, la programación y el esfuerzo.

La estimación exitosa usando Puntos de Función se basa en varias técnicas de estimación: Top - Down, Analogía y Consejos de Expertos. La estimación Top - Down es una técnica de estimación que calcula el programa entero, costo y esfuerzo usando parámetros amplios. Los parámetros amplios y las comparaciones están basados en datos históricos usando técnicas estimativas de Analogía. El Consejo de Expertos se obtiene de expertos con experiencia en proyectos similares o experiencia en le uso de Puntos de Función. La comparación de proyectos con otros similares es una actividad crítica para lograr una estimación exitosa. Cuando se evalúan los proyectos similares, se debe considerar lo siguiente:

- ✓ Tipo de plataforma de hardware: mainframe, cliente/servidor, pc, entre otros.
- ✓ Tipo de lenguaje: COBOL, C, entre otros.
- ✓ Tipo de proyectos: software de sistemas, software de aplicación, entre otros.
- ✓ Tipo de sistema operativo: MVS, Windows, UNIX, Linux.

Una vez que los proyectos han sido determinados, se deben obtener los siguientes datos:

- ✓ Medida histórica de entrega (horas por PF) de proyectos similares.
- ✓ Programas históricos (duración de programas por PF) de proyectos similares.
- ✓ Costos históricos (dinero por PF).

Una vez que el tamaño del proyecto se ha determinado por PF, el estimado de horas, costos y programa se puede calcular. Los cálculos se deben hacer con datos de proyectos similares como se indicó anteriormente.

Por ejemplo, se determina que el tamaño actual es de 500 PF y la medida de entrega de un proyecto similar es de U$10 por PF, entonces el costo total esperado para el proyecto sería:

$$U\$10(\$/PF)\times 500FPs = U\$5000$$

Cálculos similares pueden ser hechos para programas, duración y horas.

A modo de ejemplo de lo importante que es efectuar una adecuada estimación, la Tabla 5.40 ilustra la frecuencia aproximada de entrega de una serie de proyectos.

Resultados del Proyecto	< 100 PF	100-1000 PF	1000-5000 PF	> 5000 PF
Cancelados	3%	7%	13%	24%
Demora >12 meses	1%	10%	12%	18%
Demora >6 meses	9%	24%	35%	37%
App. entregado a tiempo	72%	53%	37%	20%
Entregado antes	15%	6%	3%	1%

Tabla 5.40: Resultados del Proyecto

Como se puede visualizar, mientras más grande sea el proyecto más problemas de entrega en el tiempo acordado tendrá, producto de una mala estimación.

La Tabla 5.41, por su parte, muestra las agendas de desarrollo aproximadas desde la etapa de requerimientos hasta la entrega del producto final.

Planificación promedio	< 100 (meses)	100-1000 (meses)	1000-5000 (meses)	> 5000 (meses)
Planificación esperada	6	12	18	24
Planificación ocurrida	6	16	24	36
Diferencia	0	4	6	12

Tabla 5.41: Estimaciones para tiempos

La Tabla 5.41 ilustra las diferencias entre las agendas de desarrollo estimadas y reales para aplicaciones de software. Parte de la diferencia puede ser debido a una estimación impropia y a un aumento o cambio de los requerimientos después de la estimación inicial o acuerdo contractual.

Continuando con la estimación de proyectos basados en Punto Función, la Tabla 5.42 presenta 14 diferentes proyectos[5]. La ventaja de ella es que presenta tanto la medición en LOC, como en Puntos de Función:

Proyecto	Entradas	Salidas	Archivos	PFNA	Esfuerzo Total	Campos en tablas
1	78	31	130	239	163	271
2	388	332	1789	2509	1011	3567
3	40	0	73	113	22	58
4	63	146	169	378	73	607
5	10	17	24	51	39	82
6	337	170	518	1025	240	1009
7	20	14	121	155	41	227
8	24	64	138	226	53	281
9	398	82	802	1282	595	1139
10	412	105	633	1150	410	1355
11	349	297	650	1296	572	2508
12	72	161	321	554	198	1788
13	81	120	385	586	301	1128
14	74	57	138	269	113	405

Tabla 5.42: Estimación para proyectos basadas en PF

Estimación del esfuerzo total.

Para la estimación de esfuerzo total, la técnica de descomposición es la más usada. Consiste en los siguiente:

A partir de la Definición de Requerimientos se descompone el producto de software en componentes individuales. Se definen las tareas estándares que contribuyen al desarrollo de cada componente, por ejemplo: Diseño Preliminar, Diseño Detallado, Codificación, Prueba e Instalación. Se estima el esfuerzo requerido para cada tarea estándar asociada a cada componente. Se aplican los coeficientes de trabajo (Costo/HM) a cada una de las tareas estándares definidas. Este coeficiente puede variar debido a que el personal involucrado en las tareas de Diseño Preliminar y Prueba puede ser más costoso que el personal encargado del Diseño Detallado, Codificación e Instalación. Se calculan, posteriormente, el costo y esfuerzo total para cada tarea y componente del producto de software. Todos los cálculos pueden resumirse en una Matriz de Costos de Software.

[5] Proyectos desarrollados por la compañía norteamericana Software Productivity Research.

Relación Fases/Recurso Humano.

La duración de las fases del proyecto puede establecerse usando tablas estándares de Distribución de Tiempo a lo largo del proyecto. Estas tablas reflejan el porcentaje de tiempo promedio que suele tomar una fase determinada, en relación a la duración total del proyecto, calculada mediante alguna técnica de estimación, como las vistas anteriormente. La asignación de las personas a cada fase, puede hacerse del mismo modo. En este caso se usa una tabla de Distribución de Esfuerzo, que indique el porcentaje de esfuerzo requerido para las fases, en relación al esfuerzo estimado para todo el proyecto. Tanto la tabla de distribución del tiempo como la de esfuerzo debe hacerse en base a la información histórica de la organización. El modelo COCOMO incluye ambos tipo de tablas. Otro aspecto importante del calendario del proyecto, es que debe incorporar los hitos de éste.

Capítulo 6

Control en Proyectos de Software

Un control apropiado del proyecto es clave para que éste pueda cumplir los objetivos propuestos. El control efectivo requiere esfuerzo (generalmente consume un 15% del esfuerzo total del proyecto). Las revisiones son instancias de control administrativo y técnico. Éstas se clasifican en:

- ✓ Revisiones Administrativas
- ✓ Revisiones Técnicas
- ✓ Inspecciones
- ✓ Recorridos (Walkthroughs)

Las Revisiones Administrativas se orientan a la evaluación del estado del proyecto, en relación a la planificación de éste, mientras que las Revisiones Técnicas, Inspecciones y Recorridos se enfocan a la evaluación de los componentes del producto de software.
Cada uno de los tipos de revisión tienen sus procedimientos y formas, pero ante cualquiera de ellas es necesario tener presente las siguientes directrices:

- ✓ Revisar el producto, no al productor. Limitar el debate.
- ✓ Fijar una agenda y mantenerla. Tomar notas escritas.
- ✓ Enunciar ciertas áreas de problemas, pero no intentar resolver cualquier problema que se ponga de manifiesto.
- ✓ Limitar el número de participantes e insistir en la preparación anticipada.
- ✓ Desarrollar una lista de comprobaciones para cada producto que sea sometido a revisión.
- ✓ Disponer de recursos y una agenda para las revisiones.
- ✓ Llevar a cabo un buen entrenamiento de los revisores. Repasar las revisiones anteriores.

6.1 Revisión Administrativa

La Tabla 6.1 muestra los procedimientos y consideraciones en el proceso de la Revisión Administrativa.

DESCRIPCION	OBJETIVOS
Evaluación formal de estado del proyecto en relación al plan de éste.	✓Asegurar el progreso del proyecto. ✓Recomendar acciones correctivas. ✓Asegurar la adecuada localización de recursos.
ENTRADA	**CRITERIO DE ENTRADA**
✓Objetivos de a revisión. ✓Lista de temas a tratar. ✓Calendario actual del proyecto. ✓Informes de revisiones anteriores. ✓Informe sobre los elementos de software terminados o en proceso de desarrollo.	La revisión puede iniciarse cuando el líder, en conjunto con el jefe de proyecto: ✓Establecen o confirman los objetivos de la revisión. ✓Establecen que los elementos de software y la documentación pertinente, está suficientemente completa.
SALIDA	**CRITERIO DE SALIDA**
Informe de Revisión que incluya: ✓Las entradas para la revisión. ✓El estado actual del proyecto. ✓Lista de recomendaciones.	La revisión se considera completa cuando: ✓Todos los aspectos considerados en los objetivos de la revisión han sido tratados. ✓El informe de la revisión ha sido producido.
EQUIPO DE TRABAJO	**LIDERAZGO**
✓Líder ✓Registrador ✓Revisores ✓Responsables de lo que va a ser examinado Dos o más personas	Usualmente el jefe de proyecto.

RESPONSABILIDADES	
Líder	✓Planificar la revisión ✓Conducir la revisión. ✓Asegurar que se produzca el Informe de Revisión.
Registrador	✓Documentar los problemas, decisiones y recomendaciones hechas por el equipo de revisión.
Revisores	✓Prepararse individualmente para la revisión. ✓Formular recomendaciones para mejorar anomalías detectadas. ✓Personal responsable de lo que va ser examinado. ✓Proveer la información necesaria para entender el objeto de revisión. ✓Preparar el material de entrada que corresponda.

	PLANIFICACION	PREPARACION
PROCEDIMIENTOS	✓Establecer el equipo de revisión ✓Establecer fecha y lugar para llevar a cabo la revisión. ✓Distribuir a tiempo, a los participantes el material de entrada a la revisión.	✓Revisar individualmente el material que se distribuye para la revisión. ✓Confeccionar una lista de preguntas y tópicos a discutir.

	EXAMINACION
PROCEDIMIENTOS	✓Examinar el estado del proyecto y determinar si cumple con el estado esperado, de acuerdo a la planificación predefinida. ✓Examinar el estado del proyecto y determinar si es constantemente restringido por factores externos o internos no considerados originalmente en los planes. ✓Registrar todas las desviaciones con respecto al estado esperado. ✓Generar una lista de tareas y recomendaciones para ser manejada por niveles administrativos superiores. ✓Recomendar el curso de acción a ser tomado en lo sucesivo.

Tabla 6.1: Revisión Administrativa

6.2 Revisión Técnica Formal. (RTF)

Es una actividad de garantía de calidad del software llevada a cabo por los profesionales de la ingeniería del software. Los objetivos de la RTF son:

- ✓ Descubrir errores en la función, la lógica o la ejecución de cualquier representación del software.
- ✓ Verificar que el software bajo revisión cumple con los requerimientos.
- ✓ Garantizar que el software ha sido desarrollado bajo ciertos estándares predefinidos.
- ✓ Obtener un desarrollo del software uniforme.
- ✓ Hacer que los proyectos sean más manejables.

Cada RTF se lleva a cabo mediante una reunión y sólo tendrá éxito si es bien planificada, controlada y atendida.

Existen ciertas restricciones que deben ser tomadas en cuenta al efectuar una RTF:

- ✓ Convocar a la revisión entre 3 a 5 personas.
- ✓ Se debe preparar por adelantado, pero sin requerir más de 2 horas de trabajo por persona.
- ✓ La duración de la reunión de revisión debe ser menor de 2 horas.

Al final de la reunión, todos los participantes en la RTF deben decidir:

- ✓ Si aceptan el producto sin posteriores modificaciones.
- ✓ Si rechazan el producto debido a los serios errores encontrados.
- ✓ Si aceptan el producto provisionalmente.

Una vez tomada la decisión, todos los participantes firman un documento de revisión en el cual expresan que están de acuerdo con las conclusiones tomadas. El centro de atención de la RTF es un producto.
A modo de *ejemplo*, la Tabla 6.2 presenta el número promedio de errores presentes en actividades del desarrollo del software por Punto de Función. Estos datos son resultado de estudios efectuados en los Estados Unidos entre 1980 y 1996.

ORIGEN DE LOS DEFECTOS	DEFECTOS POR PF
Requerimientos	1.00
Diseño	1.25
Codificación	1.75
Documentación	0.60
Malos arreglos	0.40
Total	5.00

Tabla 6.2: Número promedio de errores

La Tabla 6.3 muestra los procedimientos y consideraciones en el proceso de Revisión Técnica.

DESCRIPCION	OBJETIVOS
Evaluación formal de uno o varios elementos de software.	✓Evaluar la conformidad con especificaciones y planes. ✓Asegurar la integridad de los cambios.
ENTRADA	**CRITERIO DE ENTRADA**
✓Objetivos de la revisión. ✓Elemento(s) de software a examinar. ✓Especificaciones de los elementos de software a examinar. ✓Planes, estándares o guías para contrastar el elemento de software a revisar.	La revisión puede iniciarse cuando: ✓Se establecen los objetivos de la revisión. ✓Los responsables de los elementos de software están listos para la revisión. ✓El líder establezca que los elementos de software y la documentación pertinente está suficientemente completa.
SALIDA	**CRITERIO DE SALIDA**
Informe de Revisión que incluya: ✓Las entradas para la revisión. ✓Lista de deficiencias no resueltas, de los elementos de software examinados. ✓Recomendaciones para eliminar las deficiencias no resueltas.	La revisión se considera completa cuando: ✓Todos los aspectos considerados en los objetivos de la revisión han sido tratados. ✓El informe de la revisión ha sido producido.
EQUIPO DE TRABAJO	**LIDERAZGO**
✓Líder ✓Registrador ✓Revisores ✓Autor(es) Tres o más personas	Usualmente el ingeniero líder

RESPONSABILIDADES	
Líder	✓Planificar la revisión. ✓Conducir la revisión. ✓Asegurar que se produzca el Informe de Revisión.
Registrador	✓Documentar los problemas, decisiones y recomendaciones hechas por el equipo de revisión.
Revisores	✓Prepararse individualmente para la revisión. ✓Formular recomendaciones para mejorar anomalías detectadas.
Autor(es) de los elementos de software	✓Proveer la información necesaria para entender el elemento de software a examinar. ✓Rehacer el trabajo necesario para que los elementos de software satisfagan los criterios de salida de la revisión. ✓Preparar el material de entrada que corresponda

	PLANIFICACION	PREPARACION
PROCEDIMIENTOS	✓Establecer el equipo de revisión. ✓Establecer fecha y lugar para llevar a cabo la revisión. ✓Distribuir a tiempo, a los participantes el material de entrada a la revisión.	✓Revisar individualmente el material que se distribuye para la revisión. ✓Confeccionar una lista de preguntas y tópicos a discutir.

	EXAMINACION
PROCEDIMIENTOS	✓Examinar los elementos de software para verificar si cumplen con las especificaciones y estándares predefinidos. ✓Documentar los aspectos técnicos que solucionan las deficiencias encontradas en los elementos de software y determinar los responsables de su resolución. ✓Cuando las deficiencias son críticas o numerosas, el líder de la revisión debe recomendar procesos de revisión adicionales.

Tabla 6.3: Revisión Técnica Formal

6.3 Inspección.

La Tabla 6.4 muestra las consideraciones y procedimientos para realizar una Inspección.

DESCRIPCION	OBJETIVOS
Evaluación formal de uno o varios elementos de software. Las inspeccionar son realizadas por personal altamente calificado. La resolución de defectos es obligatoria y el trabajo rehecho es formalmente verificado.	✓Detectar defectos. ✓Verificar la resolución de defecto.
ENTRADA	**CRITERIO DE ENTRADA**
✓Elemento(s) de software a inspeccionar. ✓Especificaciones de los elementos de software a inspeccionar. ✓Lista de chequeo de la inspección. ✓Planes, estándares o guías para contrastar el elemento de software a inspeccionar	La inspección puede iniciarse cuando: ✓Los elementos de software estén disponibles para su inspección. ✓La documentación pertinente esté disponible. ✓Para reinspecciones, todos los ítems especificados en la lista de defectos deben haber sido resueltos.
SALIDA	**CRITERIO DE SALIDA**
Lista de defectos que incluya: ✓Localización del defecto. ✓Descripción del defecto ✓Categorización del defecto. ✓Resumen de los defectos por categoría	✓La inspección se considera completa cuando todos los defectos de los elementos de software han sido detectados y resueltos.
EQUIPO DE TRABAJO	**LIDERAZGO**
✓Moderador ✓Lector ✓Registrador ✓Inspectores ✓Autor(es) Tres a seis personas	Moderador entrenado

RESPONSABILIDADES	
Moderador	✓Planificar la revisión. ✓Conducir la revisión. ✓Asegurar que se produzcan las salidas de la inspección.
Lector	✓Dirigir al equipo de inspectores a través de los elementos de software a examinar.
Registrador	✓Documentar los defectos que se identi-fiquen en los elementos de software a examinar.
Inspectores	✓Prepararse individualmente para la revisión. ✓Identificar y describir los defectos que posean los elementos de software a examinar.
Autor(es) de los elementos de software	✓Proveer la información necesaria para entender el elemento de software a examinar. ✓Rehacer el trabajo necesario para que los elementos de software satisfagan los criterios de salida de la revisión. ✓Preparar el material de entrada que corresponda.

	PLANIFICACION	PREPARACION
PROCEDIMIENTOS	✓Establecer el equipo de revisión. ✓Establecer fecha y lugar para llevar a cabo la revisión. ✓Distribuir oportunamente a los participantes, el material de entrada a la revisión.	✓Revisar individualmente el material que se distribuye para la revisión. ✓Confeccionar una lista de preguntas y tópicos a discutir.

	EXAMINACION
PROCEDIMIENTOS	✓Examinar el elemento de software. ✓Detectar y re-gistrar defectos. ✓Recolectar información sobre los defectos. ✓Decidir si el elemento de software satisface los criterios de salida de la inspección

Tabla 6.4: Inspección

6.4 Recorrido (Walkthrough)

La Tabla 6.5 muestra las consideraciones y procedimientos para realizar un Recorrido o Walkthrough.

<table>
<tr><th>DESCRIPCION</th><th>OBJETIVOS</th></tr>
<tr><td>Evaluación formal de un elemento de software o de su proceso de desarrollo.
El autor presenta el elemento de software a revisar, el resto de los participantes discute sobre el progreso logrado, errores, cambios posibles y mejoras en general.</td><td>✓Detectar defectos.
✓Examinar alternativas de desarrollo.
✓Ser fuente de aprendizaje para el autor</td></tr>
<tr><th>ENTRADA</th><th>CRITERIO DE ENTRADA</th></tr>
<tr><td>✓Objetivo del recorrido.
✓Elemento de software a examinar.
✓Estándares para el desarrollo del elemento de software a examinar.</td><td>El recorrido puede iniciarse cuando:
✓El autor del elemento de software está listo para su presentación.
✓El moderador establece que el elemento de software está suficientemente completo para su examinación.</td></tr>
<tr><th>SALIDA</th><th>CRITERIO DE SALIDA</th></tr>
<tr><td>Informe de Recorrido que incluya:
✓Elemento de software examinado
✓Lista de deficiencias, omisiones, contradicciones.
✓Recomendaciones para resolver deficiencias.</td><td>El recorrido se considera completo cuando:
✓El elemento de software ha sido presentado en detalle.
✓Todas las deficiencias y mejoras han sido documentadas.
✓El Informe de Recorrido ha sido producido.</td></tr>
<tr><th>EQUIPO DE TRABAJO</th><th>LIDERAZGO</th></tr>
<tr><td>✓Moderador
✓Lector
✓Registrador
✓Revisores
✓Autor
Dos a siete personas.</td><td>Usualmente el autor del elemento de software a examinar.</td></tr>
<tr><th colspan="2">RESPONSABILIDADES</th></tr>
<tr><td>Moderador</td><td>✓Planificar la revisión.
✓Conducir la revisión.
✓Asegurar que se produzcan las salidas de la inspección.</td></tr>
<tr><td>Registrador</td><td>✓Documentar los comentarios hechos durante el recorrido.</td></tr>
<tr><td>Revisores</td><td>✓Prepararse individualmente para la revisión.
✓Participar en la presentación del elemento de software haciendo aportes sobre errores, omisiones, contradicciones, mejoras, enfoques alternativos, entre otros.</td></tr>
<tr><td>Autor(es) de los elementos de software.(Autor y Moderador pueden ser la misma persona).</td><td>✓Proveer la información necesaria para entender el elemento de software a examinar.
✓Rehacer el trabajo necesario para que los elementos de software satisfagan los criterios de salida de la revisión.
✓Preparar el material de entrada que corresponda.
✓Dirigir al equipo de inspectores a través de los elementos de software a examinar.</td></tr>
<tr><td rowspan="2">PROCEDIMIENTOS</td><th>PLANIFICACION</th><th>PREPARACION</th></tr>
<tr><td>✓Establecer el equipo de revisión.
✓Establecer fecha y lugar para llevar a cabo la revisión.
✓Distribuir a tiempo, a los participantes el material de entrada a la revisión.</td><td>✓Revisar individualmente el material que se distribuye para la revisión.
✓Confeccionar una lista de preguntas y tópicos a discutir.</td></tr>
<tr><td rowspan="2">PROCEDIMIENTOS</td><th colspan="2">EXAMINACION</th></tr>
<tr><td colspan="2">✓Examinar el elemento de software.
✓Detectar y registrar defectos, omisiones, contradicciones.
✓Registrar recomendaciones y mejoras sugeridas.</td></tr>
</table>

Tabla 6.5: Recorrido

Capítulo 7

Aseguramiento de la Calidad del Software- SQA

7.1 Introducción a la Calidad

Poco a poco el término CALIDAD se ha introducido en el mundo de la empresa, tanto industrial como de servicios. Pero son muchas las empresas que no conocen o confunden el significado de este concepto.

- Unos lo confunden con un producto de unas cualidades inmejorables. Sin embargo la calidad va más allá de las características de un producto o servicio.
- Otros la asocian con una acumulación de papeles que no sirven sino para ralentizar el trabajo, y el desarrollo de las actividades. Sin embargo, la calidad, o en este caso el SISTEMA DE CALIDAD en una organización, es algo más que una serie de documentos y papeles para rellenar.
- Y muchos otros identifican calidad con CONTROL DE CALIDAD, siendo este último sólo una parte que constituyen un Sistema de Calidad.

Además de aclarar estas dudas y otras posibles, esta página tiene por objeto definir distintas SISTEMÁTICAS DE CALIDAD que pueden implantarse.

CONCEPTOS GENERALES

La Calidad puede definirse como el conjunto de características de una entidad, que le confieren la aptitud para satisfacer las ***necesidades establecidas y las implícitas*** [ISO 8402].
El concepto de entidad se extiende tanto a procesos, como a productos y servicios, una organización, o combinación de ellos.
¿Qué son las necesidades establecidas y las implícitas?
Las necesidades establecidas son las que están especificadas, ya sea por un reglamento (necesidades para un proceso, producto, etc.), por un cliente (características para un producto o servicio), etc.
Las necesidades implícitas son las que no están especificadas, pero que conviene identificar y definir.
Uno de los objetivos (que no el único) que busca la Calidad es la **satisfacción del cliente**.
¿Cómo se puede conseguir la satisfacción del cliente?

- **Con buenas cualidades**, como buen funcionamiento, fiabilidad, durabilidad, facilidad de uso, estética, personalización, servicio a tiempo, seguridad, cortesía.
- **Sin defectos** a la entrega, durante el uso, en los procesos administrativos y en la facturación.

¿Por qué es tan importante la satisfacción del cliente?

- Porque el 96% de los clientes insatisfechos nunca protestan, por tanto no podemos saber su insatisfacción.
- Por cada uno que protesta puede haber 26 con problemas, y probablemente 6 de ellos graves.
- El 90% de los clientes insatisfechos no volverán a confiar en el proveedor.
- Sólo el 80% de los satisfechos vuelve a comprar.
- Los clientes insatisfechos lo comentan con gran cantidad de personas, algunos de ellos posibles clientes.
- Sin embargo el cliente satisfecho raramente lo comenta.

¿Cómo se puede llegar a la Calidad?
Implantando un .....SISTEMA DE CALIDAD
Un Sistema de Calidad identifica, *documenta, coordina y mantiene* las *actividades* necesarias para que los productos/servicios cumplan con los *requisitos de calidad* establecidos, sin tener en cuenta *dónde* estas actividades se producen.
Un Sistema de Calidad pone requisitos a las actividades y procesos que se realizan en la empresa, y documenta cómo se realizan estas actividades.
Cualquier empresa tiene su Sistema de Calidad más o menos eficaz, una forma de llevar a cabo sus actividades.
El objetivo de un Sistema de Calidad es satisfacer las necesidades internas de la gestión de la organización. Por tanto va más allá de satisfacer los requisitos que pone el cliente.
¿Qué alcance tiene un sistema de calidad?
El sistema de calidad debe abarcar todas las actividades que se realizan en la empresa y que puedan afectar (directa o indirectamente) a la calidad del producto/servicio que suministra.
Estas actividades abarcan desde las actividades de compra, control del diseño, control de la documentación, realización de ofertas, identificación de los productos, control de los procesos, inspección de los productos, hasta el tratamiento de productos no conformes, almacenamiento, formación del personal, etc.
Un Sistema de Calidad ayuda a evitar problemas en la ejecución de estas actividades, ya que la filtración de errores a través de las actividades de la empresa puede ocasionar importantes pérdidas. El costo de corregir un error entre proveedor y cliente antes de firmar el contrato, es mucho menor que si el error se detecta en la entrega al cliente del producto/servicio terminado.
El espíritu de los Sistemas de Calidad es prevenir errores para evitar estas filtraciones y pérdidas económicas.
Pero *¿en base a qué se puede implantar una Sistemática de Calidad?*
Para la implantación de estas sistemáticas se pueden seguir distintos...

ENFOQUES DE CALIDAD

Una vez que una empresa se decide a implantar un Sistema de Calidad, puede seguir distintos caminos para conseguirlo. El camino elegido será normalmente función de los siguientes factores:

- Tamaño de la empresa.
- Disponibilidad de recursos (entre ellos el tiempo).
- Motivo por el que necesita un Sistema de Calidad.

Tres posibles enfoques para la implantación de un Sistema de Calidad son los siguientes:

ENFOQUE GLOBAL

Orientado para las medianas/grandes corporaciones, con un plazo de implantación largo (entre 3 y 5 años). Su objetivo es conseguir el liderazgo.
Está orientado a conseguir la satisfacción del cliente mediante la *mejora continua*. Los tres puntos fundamentales para conseguir esto son:

- Liderazgo de la dirección en la implantación del sistema.
- Participación de todo el personal de la organización.
- Formación continua.

Son puntos importantes los siguientes:

1. Entender las necesidades y requerimientos del mercado

¿Cómo se puede conseguir esto?:

- Mediante encuestas
- Identificando los valores clave del éxito, como pueden ser:
  - Valor del producto para el cliente
  - Cumplir con los compromisos
  - Fabricar productos sin defectos
  - Cumplir los plazos
  - Dar un servicio de apoyo al cliente
- Analizando la competencia.

2. Definir y analizar los procesos

- Identificando y determinando sus límites
- Designado sus responsables
- Analizando y mejorando estos procesos.

3. Establecer mediciones y objetivos

- De la satisfacción de clientes
- De la calidad y el rendimiento de los procesos
- De la calidad de los proveedores y servicios
- Del cumplimiento de objetivos y compromisos, etc.

4. Eliminar defectos

5. Reducir los ciclos

- Identificando los puntos críticos en los procesos
- Estableciendo métodos y herramientas

Para la implantación de un Sistema de Calidad bajo este enfoque, existen *modelos* en los que puede basarse la implantación. Los más conocidos son:

- Modelo Deming, o modelo japonés
- Modelo EFQM (*European Foundation for Quality Management*), o modelo europeo
- Modelo Malcom Baldridge, o modelo americano.

ENFOQUE PRÁCTICO
Enfoque que puede ser adoptado por cualquier tipo de empresa o industria (fundamentalmente PYMES), y cuyo objetivo es *mejorar en el corto plazo*.
Una forma de operar en estos casos es:
1. Identificando y cuantificando los defectos
2. Analizando las causas que los producen
3. Estudiando las acciones a tomar para eliminarlos

Existen distintas herramientas para la realización de estas actividades, como son las Metodologías de Evaluación y Mejora de Procesos, la Detección de Problemas,etc.

ENFOQUE TÁCTICO
Orientado a cualquier tipo de empresa. Su objetivo es el **Aseguramiento de la Calidad** como estrategia táctica de la empresa. Este tipo de implantación de Sistemas de Calidad suele tener como fin la certificación del Sistema de Calidad por un organismo externo acreditado para ello. El periodo de implantación suele estar entre 1 y 2 años dependiendo de la madurez de la organización).

Esta implantación se hace en base a un modelo. Los más utilizados son los modelos para el Aseguramiento de la Calidad definidos en las normas ISO 9000.

7.2 Evolución de la calidad

Historia

1- El artesano

El concepto de calidad estaba ligado a la habilidad del artesano que a lo sumo contaba con la colaboración de pocas personas (aprendices).

El entendía cuales eran las exigencias de su cliente y trataba de realizar de la mejor manera e producto solicitado, con el fin de satisfacerlo en su pedido, tanto en la calidad del producto como también el costo.

La capacidad del artesano era fundamental, no solo en lo referido a la construcción del producto solicitado, sino también en la relación comercial con el cliente y su proveedor de materia prima, además de componer el equipamiento necesario (instrumentos, dispositivos, etc.).
Con el correr del tiempo el mercado creció, ya que se dispuso de mayores recursos para adquirir productos, por lo cual el artesano amplió su pequeña organización (incrementó sus ayudantes) y se creó la necesidad de utilizar instrumentos mas sofisticados (máquinas motorizadas, dispositivos

automatizados etc.) que permitían aumentar la productividad y satisfacer la demanda.

Esta evolución originó la necesidad de delegar funciones dentro de su pequeña industria y de implementar conceptos de estandarización (materiales, modelos, procesos, etc.)

2- La pequeña industria

Se llega en ese momento al concepto del pequeño empresario donde aparece la contribución de TAYLOR y FORD quien divide los trabajos complejos que antes eran realizados por el artesano una sucesión de tareas simples realizadas por personas con una capacitación mínima.

En este esquema el operario pierde el concepto global del producto final que antes poseía el artesano, por lo cual se hace necesario el control final que permitía separar lo apto de lo no apto.
Este control final originó muchos descarte, lo cual llevó a los empresarios a considerar que no era una solución del punto de vista económico separar al final de la producción los productos no aptos.
Esto dio lugar a la creación del control intermedio de modo que los inspectores de calidad verificaban el cumplimiento de los standard determinados en las distintas fases del proceso, esto se produjo aproximadamente en 1930.

3- En tiempos de guerra

En esta etapa surge el concepto de control estadístico para la aceptación de mercaderías en diferentes etapas del proceso productivo (recepción de materias primas, control de la producción, control final), dando origen a las técnicas de muestreo.

Durante la 2da. guerra mundial la necesidad de producir rápidamente determinó que las potencias en pugna, especialmente Estados Unidos, instruyan a los ingenieros en la aplicación de técnicas estadísticas.
Cuando la guerra concluye esos profesionales se reintegran a sus empresas y difunden el control estadístico para reemplazar el control 100%.

4- El incipiente aseguramiento de calidad

Este escalón de la historia comienza con el interrogante de cómo evitar los descartes de producción y surgen varios caminos a recorrer para buscar las causas:

- Máquinas mal preparadas por deficiencias en el mantenimiento preventivo
- Falta de capacidad del personal que operaba esos equipos
- Selección del personal
- Compras de equipos, materias primas, etc.

Estos análisis se extienden a 1970 donde el concepto de calidad (tal vez sin tener decidida conciencia de ello) se extiende a otras secciones de la empresa, aunque se mantiene un arrea exclusiva para calidad. Este concepto da lugar a la idea del aseguramiento de calidad, dando paso a la exigencia de documentar el cumplimiento de calidad, en especial en empresas con riesgos potenciales (por ejemplo la energía nuclear) donde se crean normas especificas para esas actividades.

Esta etapa fue designada como de garantía de calidad.

5- El concepto de calidad total

A partir de la década del 80 las relaciones entre contratista y proveedor se hacen más complejas al aumentar las exigencias de este, la cual determina que el cliente se convierte en el eje del sistema.
Los conceptos fundamentales de la llamada calidad total son:

a) La satisfacción del cliente

b) La calidad como factor estratégico de la empresa y orientados a la mejora continua (los japoneses la llamaron Kaizen).

c) En este esquema el precio es importante pero en correlación con la calidad producida, el cliente ocasional, aquel que solo compra por precio, no constituye el futuro de la empresa, si lo es en cambio el cliente consolidado, el que luego de la primero compra lo seguirá haciendo por haber quedado satisfecho con la ecuación producto-precio-atención.

6- La gestión de calidad en la actualidad

A partir del desarrollo que Japón logra en estos temas, aparece un nuevo concepto de gestión de calidad.
Producción (encarnado por el operario) pasa a ser responsable de la calidad de la calidad de lo producido, dando origen a herramientas de calidad (causa y efecto, resolución de problemas en grupo, etc.), se forman grupos interdisciplinarios de trabajo, todo esto para tender a la mejora continua.
Europa introduce las normas ISO 9000, logrando con ello unidad desarrollo rápido de sus empresas hacia la calidad, pero también coloca una barrera en las fronteras del mercado común protegiendo sus productos del mercado externo.
Las consultas del cliente se hacen cada vez más imprescindibles, preguntan colores, formas, sabores, atención, etc.
Estas consultas ponen de manifiesto un aspecto que antes no se había tenido en cuenta; el servicio asociado al producto.
Los clientes con ayuda de la red mundial pueden comparar, adquirir y comprar productos con un solo click. Las empresas deben adaptarse a brindar servicios rápidos, satisfacer una gama extensa de necesidades, culturas diferentes y una calidad constante e innovadora.

Las respuestas de los clientes generan que los cambios se sucedan cada

vez más rápido, el desafío actual es la forma de adaptación de las organizaciones a estos cambios.

Sin lugar a dudas la década del 90 se caracteriza por la palabra CALIDAD, normalmente.... sin importar a qué elementos, entes y áreas se hace referencia; gerencial, objetos de consumo, servicios, administración, gestión, atención al publico, universidades, hospitales, líneas aéreas, gobiernos, bancos (la lista continua). Las ciencias no escapan de este "fenómeno": medicina, economía, ingeniería, biología, sociología, entre otros. En definitiva todos dicen poseerla y dominarla, pero cómo conocer y dominar un concepto tan amplio, subjetivo y muchas veces ambiguo, que ni siquiera la mayoría de las personas podrían dar una definición exacta de ¿qué es calidad?

La Ingeniería de Software no escapa de esta realidad pero sin lugar a dudas se enfrenta con muchos más obstáculos para poder dominar la calidad en comparación con otras ciencias. Para numerar solo algunos:

- ✓ **No existe una definición estándar y universal de qué es calidad.**
En realidad algunos organismos e instituciones como ISO, IEEE, SEI, etc. brindan definiciones aceptables pero no son homogéneas, dando como resultado que cada profesional utilice su propia versión de calidad.
- ✓ **La calidad debe satisfacer a una amplia gama de entes relacionados pero no mutuamente excluyentes.**
Clientes, procesos, organismos, productos (documentos, aplicaciones, mediciones, ...)
- ✓ **Cultura de calidad.**
Esto implica un compromiso constante, tedioso, costoso, y a largo plazo por parte de la organización y las personas que lo componen.

Se verá en más detalle los puntos que encierra este concepto y qué alternativas existen para alcanzarlo.

7.3 ¿ Qué es calidad ?

El término calidad es ambiguamente definido y pocas veces comprendido, esto se debe porque:

- ✓ La calidad no es una sola idea, es un concepto multidimensional;
- ✓ La dimensión de calidad incluye el interés de la entidad, el punto de vista de la entidad, y los atributos de la entidad;
- ✓ Por cada concepto existen diferentes niveles de abstracción;
- ✓ Varía para cada persona en particular.

Se puede clasificar la calidad bajo dos puntos de vista, usual y profesional.

Punto de vista usual

La calidad contiene características intangibles, términos como alta, baja, y buena calidad son utilizados sin intentar definirlos.

Punto de vista profesional

Juran(1970) definió la calidad como *"adaptabilidad de uso"*, esto implica dos parámetros: calidad de diseño y calidad de conformidad. Es decir, adaptable a la necesidad de los usuarios.
Crosby(1979) definió la calidad como la *"conformidad con los "requerimientos."*
Como ocurre con el concepto general de calidad no existe una única definición de calidad del software, según la IEEE:
- ✓ La calidad debe ser mensurable;
- ✓ La calidad debe ser predecible.

Los factores deben ser:
- ✓ Ausencia de defectos;
- ✓ Satisfacción del usuario;
- ✓ Conformidad con los requerimientos.

Existen otras definiciones de Calidad, por ejemplo:
Concordancia con los requerimientos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo y con las características implícitas que se espera de todo software desarrollado profesionalmente.

Esta definición enfatiza tres puntos importantes:
- ✓ Los requerimientos del software son la base de las medidas de la calidad.
- ✓ Los estándares especificados definen un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería de software.
- ✓ Existe un conjunto de requerimientos implícitos que a menudo no se mencionan.

Sin lugar a dudas el factor inherente sobre calidad de software es la ausencia de defectos, este factor usualmente se expresa de dos maneras: tasa de defecto (número de defectos por KLOC, Puntos de Función, otra unidad) y confiabilidad (número de fallas por n horas de operación, tiempo medio entre fallas, otra probabilidad de libre de fallas por unidad de tiempo).

La satisfacción del usuario usualmente es medida por porcentaje de satisfacción o insatisfacción. Para evitar prejuicios se utiliza las técnicas de estudio o encuesta ciega (el entrevistador no sabe quién es el cliente, y el cliente no sabe a qué empresa representa el entrevistador).

La función de SQA interactúa en cada fase del proceso de desarrollo del producto. Una descripción de cada producto resultante de la fase y los atributos deseables en cada producto, deben ser definidos para proveer una base objetiva que identifique el término satisfactorio de la fase.

7.4 La calidad y la infórmatica

Durante las tres primeras décadas de la Informática, el principal desafío era desarrollar el hardware de manera que se redujeran los costos de procesamiento y almacenamiento. Ya a partir de la década del 80, los avances

en la microelectrónica han dado como resultado una mayor potencia de procesamiento, configuraciones de hardware imponentes y una reducción significativa del costo del hardware.

Hoy en día, el problema es diferente. El principal objetivo es reducir el costo, elevar la productividad y la eficiencia en la industria del software y mejorar la calidad para lograr un producto competitivo que se ajuste a los requerimientos de calidad establecidos por el cliente y por el productor respectivamente.

Producir "calidad" es indispensable no sólo para lograr y conservar un segmento de mercado, contra una competencia cada vez más aguerrida, sino porque se esta pasando definitivamente de una concepción de mercado nacional a otra dimensión regional o global. La utilización de métodos y técnicas para incrementar la calidad de los productos de software permite ampliar los propios horizontes comerciales.

Para cualquier organización que se dedique a la investigación, producción o comercialización del software debe tener en cuenta el factor del aseguramiento de la calidad, hoy con más razón, donde existe un mercado en que el cliente es cada vez más exigente, no sólo en lo que se refiere al precio, sino sobre todo, en cuanto a los servicios y a la confiabilidad que brindan los productos de software. El aseguramiento de la calidad asume un rol determinante para la competitividad de la empresa.

La tendencia internacional está fundamentada en la aplicación y certificación en base al grupo de normas ISO 9000. Este aspecto supone un lenguaje común en materia de calidad adoptado por un elevado número de países. La Industria del Software no está exenta de la certificación de su Sistema de Calidad en conformidad con la norma antes mencionada. En la figura 1 se muestran algunas consideraciones importantes sobre la ISO 9000.

Durante mucho tiempo, la confiabilidad ha sido la única manera de medir la calidad del software. Diferentes modelos de calidad han sido propuestos y han sido presentados para su uso. Mientras que por una parte las investigaciones realizadas en el área de aseguramiento de la calidad del software fueron útiles, causaron también confusión debido a la gran cantidad de aspectos de calidad ofrecidos y lo engorroso y costoso de mantener esta actividad. Por lo tanto, era necesario la llegada de un modelo estándar que estableciera las características fundamentales de calidad del software y que ganara el consenso para su aplicación.

Es por esta razón que el comité técnico de la ISO/IEC comenzó un trabajo conjunto para lograr el consenso requerido y alentar la normalización del ACS sobre una base internacional. Las primeras consideraciones se originaron en 1978 y en 1985 se comenzó a desarrollar un estándar internacional. En la fig. 2 se muestra el enfoque actual en materia de calidad para el software.

Como resultado de ese trabajo en diciembre de 1991 se publicó la norma internacional ISO/IEC 9126:1991. Information technology - Software product

evaluation - Quality characteristics and guidelines for their use, la cual establece los lineamientos generales para la evaluación del producto de software a partir de seis características de calidad. Sin embargo, luego de un análisis del documento se pueden mencionar las siguientes limitaciones para su utilización en la práctica:

- ✓ La ISO/IEC 9126 establece que cada característica de calidad está dada por un "conjunto de atributos que demuestran la capacidad" sobre algunos aspectos del software que pueden ser refinados a través de múltiples niveles de subcaracterísticas. Se plantean ejemplos de posibles definiciones de subcaracterísticas en el primer nivel de refinamiento, pero son relegadas a un Anexo que no forma parte de la Norma Internacional.

- ✓ Las subcaracterísticas del primer nivel de refinamiento del Anexo, son definidas como "un conjunto de atributos" y estos atributos para un segundo nivel de refinamiento quedan completamente indefinidos.

- ✓ Por lo tanto, las características y subcaracterísticas no están propiamente definidas y nos son medibles. Luego la ISO/IEC 9126:1991 tiene una base insuficiente sobre la cual evaluar el software en la práctica.

- ✓ Como que las definiciones de cualquier subcaracterística medible al nivel más bajo de refinamiento se dejan al usuario de esta norma, esto no proporciona una estructura conceptual uniforme dentro de la cual poder realizar modificaciones comparables por diferentes partes o puntos de vista de la calidad, por ejemplo, el de los usuarios, el de los productores, el de los administradores, el de los comercializadores, etc.

- ✓ El alcance de la norma es limitado. No contiene lineamientos sobre métodos de medición.

- ✓ Existe el peligro que los usuarios de la ISO/IEC 9126 se engañen con relación a los atributos a medir creyendo que afectan alguna característica de calidad del software sin darse cuenta que esto asume la existencia de un modelo preestablecido que nunca puede ser validado hasta que no sean definidas adecuadamente las características de calidad.

- ✓ No se indica propiamente la problemática de definir los requerimientos de calidad para un producto de software, aspecto básico para la medición de la calidad.

La evaluación de la calidad del software, en la práctica, requiere de características fuera del conjunto que define la ISO/IEC 9126:1991 y requiere de métricas para cada una de ellas. El estado actual en esta esfera no permite la normalización a un nivel de refinamiento más detallado.

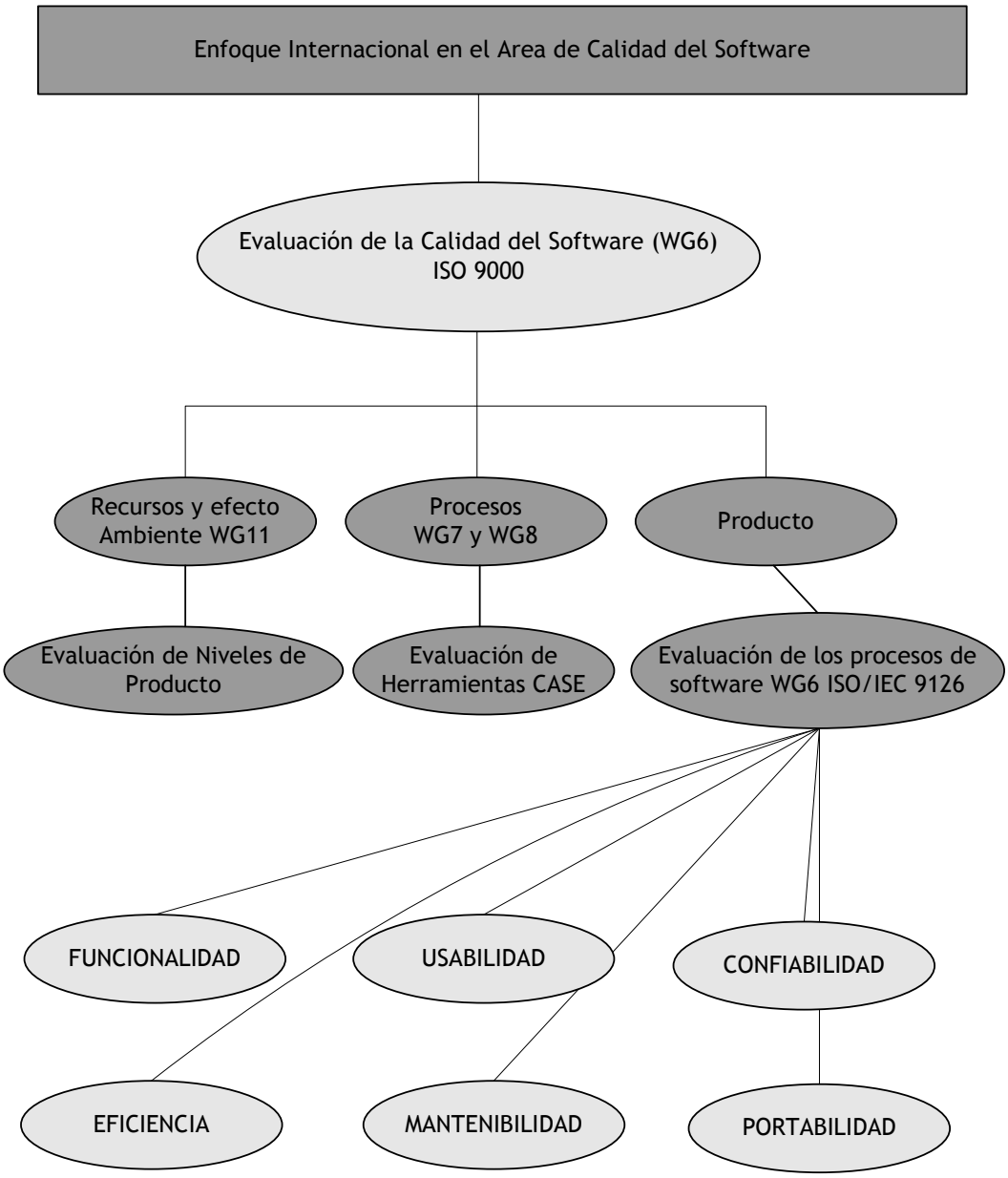


Figura 7.1: Enfoque del Area de Calidad del Software

7.5 Factores de Calidad de Mc CALL

- ✓ Facilidad de auditoría.
- ✓ Exactitud.
- ✓ Normalización de las comunicaciones.
- ✓ Completitud.
- ✓ Concisión.
- ✓ Consistencia.
- ✓ Estandarización de los datos.
- ✓ Tolerancia a errores.
- ✓ Eficiencia en la ejecución.

- ✓ Facilidad de expansión.
- ✓ Generalidad.
- ✓ Independencia del hardware.
- ✓ Instrumentación.
- ✓ Modularidad.
- ✓ Operabilidad.
- ✓ Seguridad.
- ✓ Autodocumentación.
- ✓ Simplicidad.
- ✓ Independencia del sistema de software.
- ✓ Facilidad de traza o Trazabilidad.
- ✓ Formación.

Es difícil desarrollar medidas directas de los anteriores factores de calidad, por lo cual se han definido un conjunto de métricas, estas son:

- ✓ Correctitud.
- ✓ Fiabilidad.
- ✓ Eficiencia.
- ✓ Integridad.
- ✓ Facilidad de uso.
- ✓ Facilidad de mantenimiento.
- ✓ Flexibilidad.
- ✓ Facilidad de prueba.
- ✓ Portabilidad.
- ✓ Reusabilidad.
- ✓ Interoperabilidad

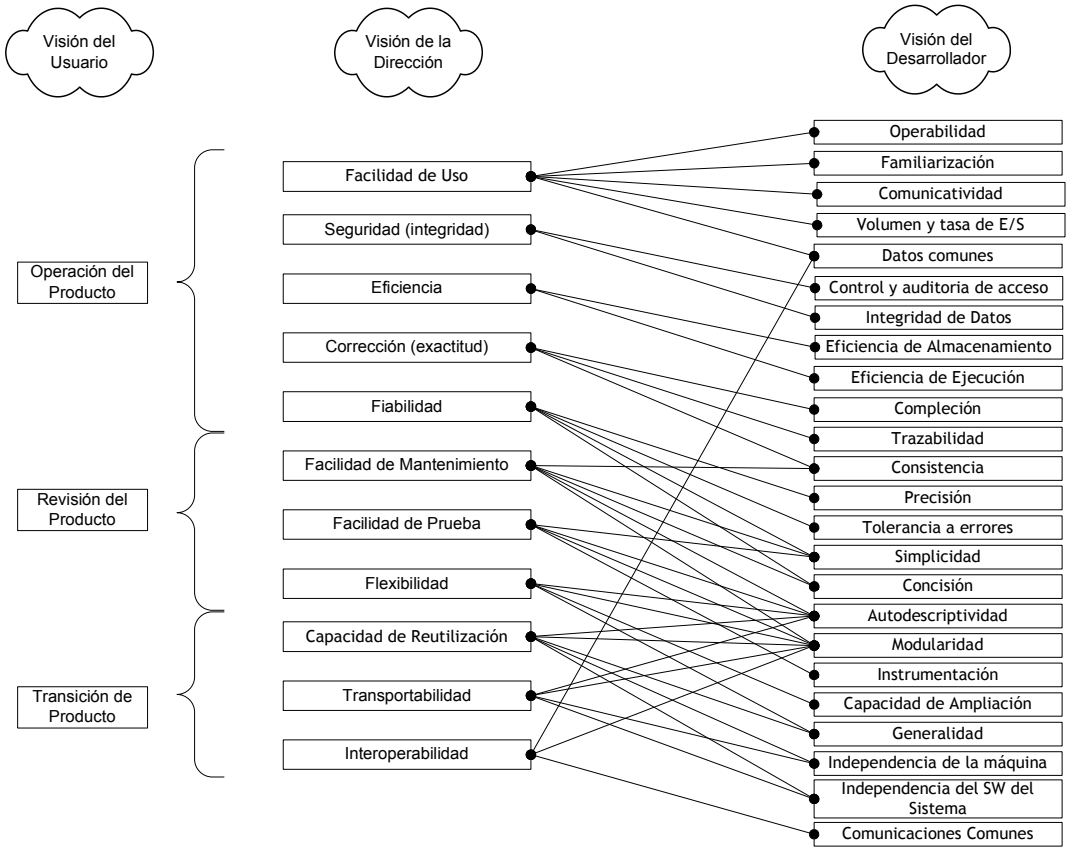


Figura 7.2: Relaciones entre las visiones en base a los factores de calidad de Mc Call

7.6 Factores de Calidad ISO-9126

Funcionalidad

- ✓ *Adecuación:* Capacidad del producto software para proporcionar un conjunto apropiado de funciones para tareas y objetivos de usuario especificados.

- ✓ *Exactitud:* Capacidad del producto software para proporcionar los resultados o efectos correctos o acordados, con el grado necesario de precisión.

- ✓ *Interoperabilidad:* Capacidad del producto software para interactuar con uno o más sistemas especificados.

- ✓ *Seguridad de acceso:* Capacidad del producto software para proteger información y datos de manera que las personas o sistemas no autorizados no puedan leer o modificar, al tiempo que no se niega el acceso a las personas o sistemas autorizados.

- ✓ *Cumplimiento funcional:* Capacidad del producto software para adherirse a normas, convenciones o regulaciones y prescripciones similares relacionadas con funcionalidad.

Fiabilidad

- ✓ *Madurez:* Capacidad del producto de software para evitar fallas del mismo.

- ✓ *Tolerancia a fallos:* Capacidad del software para mantener un nivel especificado de prestaciones en caso de fallos del software o de infringir las interfaces especificadas.

- ✓ *Capacidad de recuperación:* Capacidad del producto software para reestablecer un nivel de prestaciones especificado y de recuperar los datos directamente ante fallas o caídas del sistema.
- ✓ *Cumplimiento de la fiabilidad:* Capacidad del producto de software para adherirse a normas, convenciones o regulaciones relacionadas con al fiabilidad.

Usabilidad

- ✓ *Capacidad para ser entendido:* Capacidad del producto software que permite al usuario entender si el software es adecuado y cómo puede ser usado para una tarea y/o condiciones de uso particulares.

- ✓ *Capacidad para ser aprendido:* Capacidad del producto software que permite al usuario aprender sobre su aplicación.

- ✓ *Capacidad para ser operado:* Capacidad del producto de software que permite al usuario operarlo y controlarlo.

- ✓ *Capacidad de atracción:* Capacidad del producto de software para ser atractivo al usuario.

- ✓ *Cumplimiento de la usabilidad:* Capacidad del producto software para adherirse a normas, convenciones, guías de estilo o regulaciones relacionadas con la usabilidad.

Eficiencia

- ✓ *Comportamiento temporal:* Capacidad del producto software para proporcionar tiempos de respuesta, tiempos de proceso y potencia apropiados, bajo condiciones determinadas.

- ✓ *Utilización de recursos:* Capacidad del producto software para usar las cantidades y tipos de recursos adecuados cuando el software lleva a cabo su función bajo condiciones determinadas.

- ✓ *Cumplimiento de la eficiencia:* Capacidad del producto software para adherirse a normas o convenciones relacionadas con la eficiencia.

Mantenibilidad

- ✓ *Capacidad para ser analizado:* Es la capacidad del producto software para serle diagnosticadas deficiencias o causas de los fallos en el software, o para identificar las partes que han de ser modificadas.

- ✓ *Capacidad para ser cambiado:* Capacidad del producto software que permite que una determinada modificación sea implementada.

- ✓ *Estabilidad:* Capacidad del producto software para evitar efectos inesperados debidos a modificaciones del software.

- ✓ *Capacidad para ser probado:* Capacidad del producto software que permite que el software modificado sea validado.

- ✓ *Cumplimiento de la mantenibilidad:* Capacidad del producto software para adherirse a normas o convenciones relacionadas con la mantenibilidad.

Portabilidad

- ✓ *Adaptabilidad:* Capacidad del producto software para ser adaptado a diferentes entornos especificados, sin aplicar acciones o mecanismos distintos de aquellos proporcionados para este propósito por el propio software considerado.

- ✓ *Instalabilidad:* Capacidad del producto software para ser instalado en un entorno especificado.

- ✓ *Coexistencia:* Capacidad del producto software para coexistir con otro software independiente, en un entorno común, compartiendo recursos comunes.

- ✓ *Capacidad para reemplazar:* Capacidad del producto software para ser usado en lugar de otro producto software, para el mismo propósito, en el mismo entorno.

- ✓ *Cumplimiento de la portabilidad:* Capacidad del producto software para adherirse a normas o convenciones relacionadas con la portabilidad.

- ✓ *Efectividad:* Capacidad del producto software para permitir a los usuarios alcanzar objetivos especificados con exactitud y completitud, en un contexto de uso especificado.

- ✓ *Productividad:* Capacidad del producto software para permitir a los usuarios gastar una cantidad adecuada de recursos con relación a la efectividad alcanzada, en un contexto de uso especificado.

- ✓ *Seguridad física:* Capacidad del producto software para alcanzar niveles aceptables del riesgo de hacer daño a personas, al negocio, al software, a las propiedades o al medio ambiente en un contexto de uso especificado.

- ✓ *Satisfacción:* Capacidad del producto software para satisfacer a los usuarios en un contexto de uso especificado.

7.7 Actividades de SQA

Cada desarrollo de software es en alguna medida único. Aún cuando se usen los mismos métodos y técnicas, existen diferencias por factores que tienen impacto en la calidad, los cuales se mencionan a continuación:

- ✓ Requerimientos de plazo
- ✓ Presupuesto disponible
- ✓ Complejidad técnica
- ✓ Tamaño esperado del producto
- ✓ Experiencia relativa del grupo humano
- ✓ Recursos disponibles
- ✓ Requerimientos contractuales

El aseguramiento de calidad de software comprende una gran variedad de tareas asociadas con siete actividades principales:

- ✓ Aplicación de Métodos Técnicos y Herramientas de Análisis, diseño, codificación y prueba.
- ✓ Realización de Revisiones Técnicas Formales que se aplican durante cada paso de la ingeniería de software.
- ✓ Prueba del software.
- ✓ Control de la documentación y los cambios hechos a éste.
- ✓ Aseguramiento de que se cumple con los estándares de desarrollo del software.
- ✓ Mediciones.
- ✓ Registro y realización de informes.

La actividad principal que permite garantizar la calidad es la revisión técnica formal, donde se reúne el personal técnico con el propósito de encontrar problemas de calidad.

La prueba del software combina una estrategia de múltiples pasos con una serie de métodos de diseño de casos de pruebas que ayudan a asegurar una eficaz detección de errores.

El grado de aplicación de procedimientos y estándares varía de acuerdo a la empresa. En muchos casos viene dado por los clientes o por normas de regulación. Si existen estándares formales, se debe establecer una actividad SQA, para garantizar que se cumple con ellos.

Una de las principales amenazas para la calidad del software viene de una fuente aparentemente benigna: *los cambios*. Debe existir un proceso de control de cambios, el cual contribuye directamente a la calidad del software, al momento que exige formalizar las peticiones de cambios, evaluar la naturaleza del cambio y controlar el impacto de éste sobre el desarrollo del proyecto.

Las métricas del software deben ser establecidas para seguirle la pista a la calidad del software y evaluar el impacto de los cambios de metodología y de procedimientos que intenten mejorar la calidad de éste.

El registro de información y generación de informes, es un conjunto de procedimientos para la recolección y divulgación de información de SQA. Los resultados de las revisiones, auditorias, control de cambios, prueba y otras actividades de SQA, deben convertirse en una parte del registro histórico de un proyecto y deben ser divulgados al personal de desarrollo.

Auditoría de la Configuración

La aceptación final del producto de software se basa frecuentemente en completar un conjunto de auditorías de la configuración. Esta auditoría asegura que el producto ha cumplido satisfactoriamente todos los requerimientos aplicables y puede ser de tres tipos, los cuales se indican a continuación:

- ✓ ***Funcional***

  La auditoría de la configuración funcional asegura que el producto sea probado para demostrar cumplimiento con los requerimientos del contrato.

- ✓ ***Física***

  La auditoría de la configuración física asegura que todos los requerimientos contractuales se han satisfecho en relación a la documentación y los datos.

- ✓ ***De evaluación***

  La auditoría de la configuración de evaluación intenta asegurar el cumplimiento con los requerimientos aplicables a cada producto de software y es realizada, generalmente, por una persona.

La lista a continuación muestra las evaluaciones a realizar en cada fase del proceso de desarrollo:

✓ **Requerimientos de Software**

Las actividades y productos de la fase de requerimientos del software se examinan para evaluar:

- Plan de desarrollo de software
- Estándares del software y manual de procedimientos
- Plan de administración de la configuración
- Plan del programa de calidad del software
- Especificación de requerimientos del software
- Especificación de requerimientos de interfaces
- Documentos con concepto de operación

✓ **Análisis**

Las actividades y productos de la fase de análisis se examinan para evaluar:

- Planes y programas revisados
- Documentos con diseño de alto nivel del testing del producto de software
- Plan de testing del software
- Manual del operador
- Manual del usuario

✓ **Diseño**

Las actividades y productos de la fase de diseño se examinan para evaluar:

- Planes y programas revisados
- Documentos de análisis de software
- Documentos de diseño de interfaces
- Documentos de diseño de bases de datos
- Casos para tests unitarios
- Descripción de test del software
- Manuales revisados

✓ **Codificación y test unitario**

Las actividades y productos de la fase codificación y test unitario se examinan para evaluar:

- Planes y programas revisados
- Código fuente
- Código objeto
- Procedimientos de tests unitarios
- Resultados de tests unitarios
- Procedimientos de tests de integración
- Manuales revisados

✓ **Testing del sistema**

Las actividades y productos de la fase de testing del sistema se examinan para evaluar:

- Planes y programas revisados
- Informe de test de sistema
- Código fuente revisado
- Código objeto revisado
- Manuales

En definitiva, la calidad del software es una compleja mezcla de "ciertos factores" que varían para las diferentes aplicaciones y los clientes que las solicitan.

7.8 Garantía de calidad Estadística

La garantía de calidad estadística refleja una tendencia creciente en toda la industria de establecer la calidad en forma más cuantitativa. Para el software, la garantía de calidad estadística implica los siguientes pasos:

✓ Agrupar y clasificar la información sobre los errores y defectos del software.
✓ Intentar encontrar la causa subyacente de cada defecto.
✓ Usando el principio de Pareto (80% de los problemas de calidad se pueden encontrar en el 20% de todas las posible causas), se aísla el 20% (los pocos, pero vitales).

- ✓ Una vez que se han identificado los defectos vitales, se actúa para corregir los problemas que han producido los defectos.

Las raíces de la mayoría de los defectos (fallas) realmente difíciles, se pueden encontrar en un número relativamente limitado de causas.

7.9 Fiabilidad del Software

La fiabilidad del software, a diferencia de otros factores de calidad, puede ser medida o estimada mediante datos históricos o de desarrollo. La fiabilidad del software se define en términos estadísticos como la probabilidad de operación libre de fallas de un programa de computadora en un entorno determinado y durante un tiempo específico. Falla es cualquier falta de concordancia con los requerimientos del software.

Cuando la aplicación es entregada a los clientes, es posible que continúen existiendo y apareciendo errores durante el primer año. Por ello, sería interesante la posibilidad de llevar una estadística sobre los errores detectados pre-release y post-release :

Ejemplo:

Defectos encontrados durante el desarrollo	→	90
Defectos encontrado por clientes y usuarios	→	10
Total de defectos		100

Es decir, hubo una eficiencia de remoción de errores del 90%.

Un dato referente a lo anterior es que en los Estados Unidos durante 1996 la eficiencia en la remoción de errores fue de un 85%.

Medidas de Fiabilidad y de Disponibilidad del Software

Fiabilidad = Tiempo Medio Entre Fallas (TMEF) donde:

$$TMEF = TMDF + TMDR$$

TMDF corresponde a tiempo medio de falla. **TMDR** es tiempo medio de recuperación.

Además de una medida de fiabilidad debemos obtener una medida que indique la probabilidad de que un programa funcione de acuerdo con los requerimientos en un momento dado. Esto se define como:

$$\text{Disponibilidad} = [\ TMDF\ /\ (TMDF + TMDR)\ ]/100\ \%$$

La medida de fiabilidad TMEF es igualmente sensible al TMDF y TMDR. La medida de Disponibilidad es más sensible al TMDR.

7.10 Necesidad de SQA

Una organización de desarrollo del software debe adoptar procedimientos, métodos y herramientas de ingeniería de software. Esta metodología asociada con un paradigma eficaz para el desarrollo de software, puede hacer mucho por mejorar la calidad de todo software desarrollado por la organización. Para institucionalizar los procedimientos a fin de garantizar la calidad del software, se deben evaluar y examinar los siguientes puntos:

- ✓ ¿Qué principios, estándares y procedimientos existen para cada fase de desarrollo del software?
- ✓ ¿Son ellos rigurosos?
- ✓ ¿Hay una política específica para SQA?
- ✓ ¿Son principios aplicados a actividades de desarrollo y mantenimiento?
- ✓ ¿En qué lugar del organigrama organizacional reside la ingeniería del software?
- ✓ ¿Donde reside garantía de calidad?
- ✓ ¿Cual es la relación entre garantía de calidad y garantía de calidad del software?
- ✓ ¿Cómo interactúa con RTF, con SCM y con actividades de prueba?

7.11 Gestión del Proceso SQA

Hay tres grupos de tareas que se destacan:

✓ *Benchmarking y mejora del proceso*

Para realizar comparaciones de procesos de desarrollo de software, se debe contar con un historial, siendo para esto fundamental el registro de datos de los proyectos exitosos y los fallidos.
Al completar cada proyecto, sin consumir mucho tiempo se puede registrar los datos útiles para que a futuro sirvan como referencia, no se requieren grandes cantidades de datos, sino que basta con registrar el tiempo de desarrollo, el esfuerzo y el tamaño del software para las principales fases del desarrollo. Estos datos, a medida que la experiencia de la empresa crece, conforman una base de datos de un alto valor, que permitirá durante el desarrollo de un nuevo proyecto, comparar si es que las fases avanzan al ritmo y tiempo que debe, y más claramente, facilitarán las estimaciones del esfuerzo y tiempo que se requerirá para un software de un tamaño determinado.

✓ *Estimación y control de riesgos*

Los datos de referencia generados con lo anterior, son utilizados para estimar:

- El tamaño del producto de software (y su margen de error)
- La productividad del proceso de desarrollo
- Parámetros de desarrollo: tiempo, esfuerzo, staff y confiabilidad.
- Niveles de riesgo para cada parámetro.

El tamaño del software es estimado en base a niveles de “lo más pequeño”, “lo más probable” y “lo más grande” que puede llegar a ser, utilizando los métodos de estimación como *Puntos de Función*. Las otras estimaciones tienen mayor dependencia de la empresa desarrolladora, tomando en cuenta la experiencia y la información de productividad disponible, y si esta no existe, se debe recurrir a los promedios de la industria, incorporando aun más errores a las estimaciones.

Algunas preguntas útiles al momento de gestionar un proceso de desarrollo, con respecto a las estimaciones, son las siguientes:

- ¿Los requerimientos se identifican, se les da prioridad y son mapeados a módulos del software?
- ¿Cada módulo se estima en menos de 3000 sentencias con un rango de tamaño para cuantificar incertidumbre?
- ¿La productividad de proceso asumida, es consistente con los proyectos terminados?
- ¿Los apremios del desarrollo se indican claramente y se les da la prioridad con los niveles de riesgo convenidos?
- ¿Se registran y se documentan las estimaciones alternativas?

✓ *Control de proceso y reportes*

La gerencia de una empresa debe mantener el control de todos los proyectos que se llevan a cabo dentro de su campo de acción, así mismo, si un proyecto de desarrollo de software lo realiza una empresa contratista para otra, el proyecto debe ser controlado por ambas partes. Los reportes en este contexto, constituyen una base fundamental, por lo tanto se requiere contar con personal ocupado en esta labor de control, formando la denominada Oficina de Control de Software (SCO por sus siglas en inglés).

Esta oficina, debe seguir un proceso de control periódico, mensualmente por ejemplo, para cada proyecto de software en desarrollo, recibiendo los reportes, revisandolos y manteniendo un registro del estado de cada riesgo, comprobando que estos no estén en rojo, y si lo están, tomar las acciones correctivas (según el plan de contingencia) con el jefe del proyecto. Si la SCO no recibe los reportes, o estos están incompletos, automáticamente se debe clasificar el proyecto como en estado de alerta.

De esta manera, se pretende disminuir la posibilidad de que los riesgos se tornen en problemas, y en caso que suceda, minimizar sus efectos negativos. Como la carga de trabajo de los gerentes y jefes de proyectos normalmente es muy alta, se concluye en el reconocimiento de la necesidad de apoyo por una oficina preocupada de mantener este control de todos los proyectos, aparte del control interno que estos puedan tener, y siempre teniendo en mente los plazos finales e hitos importantes.

7.12 Estándares ANSI/IEEE 730-1984 Y 983-1986

Plan de Garantía de Calidad del Software

I. Propósito Del Plan.
II. Referencias.
III. Administración.
- ✓ Organización.
- ✓ Tareas.
- ✓ Responsabilidades.

IV. Documentación.
- ✓ Propósito.
- ✓ Requerimientos De Ingeniería De Software.
- ✓ Otros Documentos.

V. Estándares, Practicas Y Convenciones.
- ✓ Propósitos.
- ✓ Convenciones.

VI. Revisiones Y Auditorias.
- ✓ Propósitos.
- ✓ Revisión De Requerimientos.
- ✓ Revisión De Requerimientos.
- ✓ Revisión Del Diseño.
- ✓ Revisión De Verificación Y Validación Del Software
- ✓ Auditoria Funcional.
- ✓ Auditoria Física
- ✓ Auditoria De Los Procesos.
- ✓ Revisión De Administración.

VII. Administración De La Configuración Del Software.
VIII. Acción Correctiva Y Reporte De Problemas.
IX. Herramientas, Técnicas Y Metodologías.
X. Control De Código.
XI. Control De Medios.
XII. Control Del Distribución
XIII. Agrupación, Mantenimiento Y Retención De Registros

7.13 Bases conceptuales y Teóricas de la futura Norma ISO/IEC 15504

Debido a la alta competencia en el mercado de desarrollo de software, la difícil tarea de identificar los riesgos, cumplir con el calendario, controlar los costos y mejorar la eficiencia y calidad se creó SPICE (Software Process Improvement and Capability dEtermination). Este engloba un modelo de referencia para los procesos y sus potencialidades sobre la base de la experiencia de compañías grandes, medianas y pequeñas.
El proyecto SPICE provee:

- ✓ Un marco de referencia para determinar las fortalezas y debilidades de los procesos,
- ✓ Un marco de referencia para mejorar los procesos de software, y medir sus mejoras,
- ✓ Un marco de referencia para los clientes que adquieren sistemas para evaluar la capacidad de sus proveedores y,
- ✓ Un marco de referencia para determinar los riesgos de negocio para una empresa que considera desarrollar un nuevo producto de software o servicio.
- ✓ Una ruta para la armonización o migración de los modelos de evaluación de procesos con referencia al modelo de la norma ISO/IEC 15504. La organización o unidad informática puede utilizar SPICE para los siguientes casos:

Modo de nivel de capacidad o potencialidad. Le permite a una organización adquiriente determinar el nivel de capacidad de una empresa que le suministra un sistema de software,
Modo de mejoramiento de procesos. Para ayudar a una organización a mejorar su propio departamento de desarrollo de software y los procesos de mantenimiento y,
Modo auto-evaluación. Para ayudar a una organización determinar su habilidad de implementar un nuevo proyecto de software.
El modelo describe los procesos que una organización puede ejecutar, adquirir, suplir, desarrollar, operar, evolucionar, brindar soporte de software y todas las practicas genéricas que caracterizan las potencialidades de estos procesos.
El Reporte de Pruebas TR- 2 de SPICE sobre el que está guiado esta memoria, lo forman nueve partes:

- ✓ Parte 1: Conceptos y Guía Introductoria,
- ✓ Parte 2: Un modelo de referencia para procesos y madurez de procesos,
- ✓ Parte 3: Realizando una evaluación,
- ✓ Parte 4: Guía para realizar evaluaciones,

- ✓ Parte 5: Un modelo de evaluación y directrices,
- ✓ Parte 6: Guía para la cualificación de los evaluadores,
- ✓ Parte 7: Guía para la mejora de procesos,
- ✓ Parte 8: Guía para la determinación de la madurez de los procesos de proveedores y,
- ✓ Parte 9: Vocabulario.

La Tabla 7.1 muestra la audiencia principal a la que esta dirigido este futuro estándar internacional, por qué cada grupo necesita el modelo, cómo y cuándo debe ser usado.

Quién	Por qué	Cómo	Cuándo
Organización de Software	Comprender qué hay que hacer para mejorar los procesos de software.	✓ Comprender cuáles procesos y prácticas un asesor puede evaluar.Como guía de trabajo en la gestión de los procesos de aplicación de las actividades del proyecto de software.	✓ Durante la aplicación de los procesos de software de la organización.
		✓ Como guía de referencia para destacar los procesos y consideraciones prácticas.	✓ Durante el desarrollo/ revisión de los procesos de software de la organización y como parte de una actividad de mejoramiento continuo.
		✓ Como un documento de entrenamiento.	✓ Idem
		✓ Como una práctica de lista de chequeo (Checklist)	✓ Antes de las evaluaciones
Asesores de los procesos de software	Determinar cómo una organización gestiona y administra los procesos de software y sus resultados	✓ Como una práctica de lista de chequeo (Checklist)	✓ Antes y durante una evaluación de procesos de software

Tabla 7.1: Aplicación del modelo ISO/IEC 15504

Modelo de Referencia

La Estructura del Modelo de Referencia de la norma ISO/IEC 15504 se compone de dos dimensiones: la Dimensión del Proceso y la Dimensión de la Capacidad del Proceso.

Dimensión del Proceso

El modelo de la referencia reune los procesos en tres agrupaciones de ciclo de vida que contiene las cinco categorías de proceso:

- ✓ Los Procesos de Ciclo de Vida Primarios, contiene las categorías del proceso Cliente-Proveedor CUS y de Ingeniería ENG,
- ✓ Los Procesos de Ciclo de Vida de Soporte, contiene la categoría del proceso Soporte SUP.
- ✓ Los procesos de Ciclo de Vida Organizacional, contiene las categorías del proceso Administración MAN y Organización ORG.

Las categorías de procesos están relacionadas y cada categoría es una capa que precede a la otra, como se ilustra en la Figura 7.3.

Figura 7.3: Categoría de los Procesos[12]

La Tabla 7.2. muestra las tres agrupaciones del proceso de ciclo de vida, la categoría y sus procesos respectivos.

Categoría del Proceso		Proceso	
ID	**Título**	**ID**	**Título y Tipo de Proceso**
Procesos del Ciclo de Vida Primario			
CUS	Categoría del Proceso Cliente-Proveedor Consiste en los procesos que impactan directamente al cliente y proveen un correcto funcionamiento y uso del producto de software y/o servicio.		
		CUS1	Adquisición
		CUS.1.1	Preparación para la Adquisición
		CUS.1.2	Selección de Proveedor
		CUS.1.3	Monitoreo del Proveedor
		CUS.1.4	Aceptación del Cliente
		CUS.2	Suministración
		CUS.3	Requerimientos de Elicitación
		CUS.4	Operacional
		CUS.4.1	Uso Operacional
		CUS.4.2	oporte al Cliente
ENG	Categoría del Proceso de Ingeniería Consiste en los procesos que directamente especifican, instrumento o mantiene el producto del software, su relación al sistema y su documentación del cliente.		
		ENG.1	Desarrollo
		ENG.1.1	Análisis y Diseño de los Requerimientos del sistema
		ENG.1.2	Análisis de Requerimientos del software
		ENG.1.3	Diseño del software
		ENG.1.4	Construcción del software
		ENG.1.5	Integración del software
		ENG.1.6	Pruebas del software
		ENG.1.7	Integración y pruebas del sistema
		ENG.2	Mantenimiento del software y del sistema
Proceso del Ciclo de Vida de Soporte			
SUP	Categoría del Proceso de Soporte Consiste en los procesos que pueden emplearse por cualquiera de los otros procesos.		
		SUP.1	Documentación
		SUP.2	Administración de la configuración
		SUP.3	Aseguramiento de la Calidad
		SUP.4	Verificación
		SUP.5	Validación
		SUP.6	Revisión
		SUP.7	Auditoría
		SUP.8	Resolución de problemas
Procesos del Ciclo de Vida de Organizacional			
MAN	Categoría del Proceso Administrativo Consiste en los procesos que contienen prácticas de naturaleza genérica, pueden usarse por cualquiera que maneje cualquier tipo de proyecto.		
		MAN.1	Administración
		MAN.2	Administración del Proyecto
		MAN.3	Administración de la calidad
		MAN.4	Administración de Riesgos
ORG	Categoría del Proceso Organizacional Consiste en los procesos que construyen la infraestructura organizacional sobre la cual cada uno de los procesos descriptos pueden potencialmente ser ejecutados y maximizar los beneficios para la organización.		
		ORG.1	Alineación Organizacional
		ORG.2	Mejora de Procesos
		ORG.2.1	Proceso de Establecimiento
		ORG.2.2	Proceso de Valoración
		ORG.2.3	Proceso de Mejora
		ORG.3	Administración del Recurso Humano
		ORG.4	Infraestructura
		ORG.5	Medida
		ORG.6	Reuso

Tabla 7.2: Agrupaciones, Categoría de Procesos y sus Procesos

Dimensión de la Capacidad del Proceso

Existen seis niveles de capacidad en el modelo:

- ✓ Nivel 0; Incompleto: No se encuentran procesos implementados, o fallan al alcanzar los resultados del proceso. A este nivel hay poco o nada de evidencia de cualquier logro sistemático de cualesquiera de las cualidades definidas
- ✓ Nivel 1; Realizado: Las prácticas bases de los procesos generalmente son ejecutadas. La ejecución de dichas prácticas puede no ser rigurosamente planificadas y seguidas; dependerá del conocimiento y esfuerzo personal. Se identifica algunos procesos. El proceso implementado alcanza sus resultados.
- ✓ Nivel 2; Manejado: La ejecución de las prácticas bases en los procesos es planificada y seguida. El desempeño estará acorde con los procedimientos especificados. La primera distinción entre el nivel 1 y el 2 es que la ejecución de los procesos está planificada y administrada y progresan hacia un proceso bien definido.
- ✓ Nivel 3; Establecido: Las prácticas bases son ejecutadas de acuerdo a una versión adaptada del estándar, procesos aprobados bien definidos y documentados basado en los principios de ingeniería de software y la capacidad de alcanzar estos resultados.
- ✓ Nivel 4; Predecible: Mediciones detalladas del rendimiento o ejecución son recolectadas y evaluadas. Es conocido en forma cuantitativa el rendimiento de los procesos y es posible su predicción. Las prácticas son objetivamente administradas. La calidad de las mismas es cuantitativamente conocida.
- ✓ Nivel 5; Óptimo: Basados en los objetivos de la organización, los procesos y metas son establecidos en forma cuantitativa y eficiente. En forma continua los procesos se van mejorando mediante la retroalimentación obtenido por los resultados de procesos definidos, por ideas pilotos y tecnologías novedosas.

Arquitectura del Modelo

La arquitectura del modelo organiza las prácticas en números de categorías usando diferentes tipos de aproximaciones. La arquitectura distingue entre:

- ✓ Prácticas Base, son las actividades esenciales de un proceso especifico, agrupado por categorías de procedimientos y procesos de acuerdo al tipo de actividad que direccionan.
- ✓ Prácticas Genéricas, aplicables a cualquier proceso, que representa las actividades necesarias para administrar el "proceso" y mejorar su potencialidad.[12]

Cada proceso en el modelo es descrito en términos de prácticas de base, cada una son actividades únicas de ingeniería de software o de administración.

La Categoría de los Procesos, procedimientos y las prácticas de base, están agrupados por tipo de actividad. Los procedimientos y las actividades caracterizan el rendimiento de los procesos, aun cuando el rendimiento no es sistemático. El desempeño de las prácticas bases puede ser informal (*ad hoc*), impredecibles, inconsistentes, pobremente planeado, y/o su resultado es un

producto de baja calidad. La aplicación solamente de las prácticas base en los procesos puede tener un valor mínimo y representar solamente el primer paso en la construcción de la capacidad de los procesos.
La evolución de la capacidad de los procesos esta expresado en términos de niveles de capacidad, características comunes, y prácticas genéricas. Un nivel de capacidad es un conjunto de actividades que trabajan juntas para proveer una mejor ejecución de los procesos. Cada nivel provee una mejor y más compleja ejecución de los procesos que el nivel predecesor.
Los niveles de capacidad proveen dos beneficios: El conocimiento de los procesos esto dependerá del monto de la práctica y la ayuda a la organización de identificar qué "mejora" se debe ejecutar primero, basado en una secuencia racional de aplicación de los procesos.

Frameworks	Tipo de *Frameworks*	Tiempo	Persona Interna/ Externa	Equipos especiales	N° páginas	Costo	Alcance	Focalizado	Niveles y prácticas	N° Empresas
CMM® Vers. 1.1	Modelo de madurez	1-2 nivel, aprox. 2 ½ años.	Interna y Externa	SQA, SEPG, SCM	561	U$ 49,44 Gratis Web	Organización Desarrollo Software	Procesos de software	5 Niveles de madurez 18 KPA	782
LOGOS-CMM Vers. 1.0	Modelo de madurez	No indicado	Interna y Externa	SQA, SEPG, SCM "reducidos"	450	U$ 75 Web solo el articulo	Organización Desarrollo Software	Procesos de software	-	200 Org. (Etapa validación)
IDEAL℠ Vers. 1.6	*Administra-dor* mejora-miento de procesos	Relativo Plan SPI 1 a 5 años	Interna	SEPG, MSG, TWG	236	Gratis Web	Organización Desarrollo Software	Procesos de software	5 Fases 47 Tareas	-
SPICE (ISO TR2 15504) Vers. 1.00 "Borrador"	Modelo de madurez y avaluación de procesos	Relativo	Interna	*Sponsor, owner, assessor* (SQA)	474	U$ 50 a 100 Gratis Web	Software Producción *Organización*	Procesos de software	5 Categorías de procesos 29 Procesos 200 práct. base 6 Niveles de madurez	Fase I 35 Fase II 250 Fase III 314
ISO 9001 Vers. 1994	Estándar Internacional de calidad "tradicional"	Relativo aprox. 2 a 4 años	Interna y Externa	*Managers*	75 (TickIT)	U$ 50 a 100	Producto Producción Organización	Calidad de los procesos	Consta de 20 elementos de calidad	8.100 a 13.500 SW Tickit >1.500 (Mundo)
Trillium Vers. 3.1	Modelo de madurez	18 meses	Interna	*Quality managment*, TQM, (SQA)	130	Gratis Web	Empresas (Telecomuni-caciones)	Procesos	5 Niveles 8 áreas de capacidad 27 guias dtdas. 508 prácticas	< 100

Tabla 7.3: Comparación de los Marcos de Trabajo Pilares

Capítulo 8

Gestión de Configuración del Software - GCS

La Gestión de Configuración del Software (GCS, también conocido como SCM-Software Configuration Management) es un conjunto de actividades desarrolladas para gestionar los cambios a lo largo del proyecto. La Gestión de Configuración del Software es una actividad que complementa a SQA que se aplica en todas las fases del desarrollo del proyecto.
El resultado del proceso de Ingeniería del Software es información que se puede dividir en tres categorías: los programas, la documentación y las estructuras de datos. Los elementos que componen la información que se produce en forma colectiva en las fases del desarrollo se denomina configuración del software a medida que el proyecto avanza el número de Elementos de Configuración del Software (ECS), el problema no es que cada ECS produce otro ECS sino que cuando se introducen los cambios dentro del proyecto.

8.1 Líneas bases

A medida que el tiempo avanza se introducen cambios en el proyecto, una línea base es un elemento de la GCS que ayuda a controlar los cambios sin impedir seriamente los cambios justificados. Se define una línea base como un punto de referencia en el desarrollo de software que queda estipulado por el envío de un ECS y su aprobación mediante un revisión técnica formal, por ejemplo si se revisa los elementos de la especificación de diseño, se encuentran errores y se corrigen. Cuando las partes de la especificación se han revisado, corregido y aprobados, la especificación mencionada pasa a ser una línea base. El objetivo de las líneas bases son.

- ✓ Identificar en forma clara, concisa y precisa, cualquier tipo cambio que se le desee realizar al producto.
- ✓ Controlar la modificación para evitar que se modifique algún otro elemento no previsto.
- ✓ Asegurar que se cumpla adecuadamente con lo especificado anteriormente.
- ✓ Informar a todas las personas involucradas con el proyecto los cambios que se han realizado

Las Líneas Base que más comúnmente son utilizadas, son por ejemplo: Especificación de sistemas, Especificación de requerimientos del software, Especificación de diseño, Código fuente, entre otros.

8.2 Tareas de la Gestión de Configuración de Software

8.2.1 Identificación de objetos en la configuración del Software

Se identificarán los elementos de la GCS en forma única y se organizarán bajo el enfoque de Orientación a Objetos (OO). Se identifican dos tipos de objetos:

- ✓ *Objetos básicos:* es una unidad de texto que pudo ser generada en el análisis, el diseño, la codificación o prueba.
- ✓ *Objetos compuestos:* son varios objetos simples, como por ejemplo la especificación de diseño.

Cada objeto poseerá una estructura que lo define en forma única como: nombre, descripción, número de revisión, fecha actual y hora, Información sobre la versión, información del cambio a realizar, quién solicita la modificación, quién la autoriza la modificación, quién se responsabiliza por la modificación, fecha de la última actualización, otros ECS posiblemente afectados.

8.2.2 Control de versiones

Aquí se combinan procedimiento y herramientas para gestionar las versiones del producto. Las versiones se manejarán en forma simple, pudiendo realizarse con un solo numero, o considerando la fecha, etc. se podrían basar en el siguiente formato:

Nro. Versión = día + mes + año con lo cuál para buscar la última versión basta con buscar la fecha menor o igual a la actual.

8.2.3 Control de cambios

Los cambios incontrolados llevan muy rápido al caos, el control del cambio une procedimientos humanos con herramientas automáticas para proporcionar un mecanismo de control, el proceso de control debería contener lo siguiente:

1. Reconocer la necesidad del cambio
2. Usuario realiza la petición del cambio
3. Evaluación del impacto del cambio
4. Generación del informe del cambio
5. Decisión de la autoridad de cambios, Si la respuesta es NO, se le informa a la persona que el cambio fue rechazada, Si la respuesta es SI, se sigue con el siguiente paso.
6. Se genera la orden de cambio
7. Realizar cambios
8. Revisar los cambios
9. Identificar los ECS que han cambiado
10. Establecer una línea base para la prueba
11. Realizar las actividades de SQA
12. Informar de los cambios, tal de que la próxima versión los contenga
13. Reconstruir la nueva versión

14. Realizar auditoría de parte de la GCS para verificar que se realizar los cambios especificados
15. Cambios incluidos en la nueva información
16. Distribuir la nueva versión.

Finalmente, es necesario generar informes y/o documentos los cuales es conveniente almacenar.
Existen tres tipos de Controles de Cambios:

- ✓ Control de Cambios Informal

Se realiza antes de que el ECS se convierta en una línea base y consiste en que el desarrollador puede hacer cualquier cambio justificado por el proyecto y por los requerimientos.

- ✓ Control de Cambios a Nivel de Proyecto

Se realiza una vez que el ECS ya se ha convertido en una línea base. Al hacer un cambio, el encargado del desarrollo debe recibir la aprobación del gestor del proyecto (si el cambio es local) o de la jefatura de control de cambios (si el cambio impacta a otros ECS).

- ✓ Control de Cambios Formal

Se realiza una vez que el producto de software ya se ha distribuido a los usuarios, siguiendo las normas establecidas en el Control de Cambios a Nivel de Proyecto.

8.2.4 Auditorías de configuración

Las auditorías que realiza la GCS se hacen aparte de las que realizan el grupo de revisiones formales, las auditorias deben plantearse asuntos tales como:

- ✓ ¿Se ha hecho el cambio especificado?
- ✓ ¿Se hizo una revisión técnica formal para comprobar la correctitud técnica?
- ✓ ¿Se han seguido los estándares del proyecto durante el cambio?
- ✓ ¿Se ha incorporado toda la información sobre el cambio en los ECS (fecha, quien, porque, versión)?
- ✓ ¿Se ha divulgado el cambio?
- ✓ ¿Se han actualizados todos los ECS?

Capítulo 9

Testing en Productos de Software

Un producto, para el potencial cliente, es un conjunto complejo de satisfacción de valores. Los clientes le asignan valor a los productos en proporción a la habilidad percibida de los productos en relación con sus problemas.
El testing del software se puede definir como:
"Proceso planificado, basado en estándares previamente establecidos, que se inicia antes del desarrollo del software y continua a lo largo de todo el proceso, incluyendo puesta en marcha y mantenimiento, que incluye actividades preventivas y correctivas en todos los niveles, siendo el objetivo principal de estas actividades el de asegurar la eficacia de la información producida por el sistema en cuestión"

9.1 Plan de prueba

El plan es el medio de comunicación en el proceso de testing. Se inicia después de los requerimientos y su prerequisito es que exista un estándar de testing. Un buen plan debe permitir planificar, monitorear y controlar todas las actividades del proceso de testing, es decir:

- ✓ Definir los objetivos generales y específicos.
- ✓ Proveer una descripción completa de la función y estructura del sistema de software a probar.
- ✓ Establecer el testing a lo largo de todo el proceso de desarrollo y definir puntos de revisión específicos.
- ✓ Definir las tareas a desarrollar en cada fase, con estimación de los recursos necesarios.
- ✓ Identificar los criterios de entrada y salida para cada actividad de testing.
- ✓ Identificar los recursos necesarios para diseñar, almacenar y ejecutar los escenarios y las corridas de prueba y los necesarios para capturar, analizar e informar los resultados del prueba.
- ✓ Identificar y asignar responsabilidades a los individuos participantes en las actividades de prueba.
- ✓ Identificar resultados esperados y procedimientos para resolver problemas.
- ✓ Identificar planes de contingencia.
- ✓ Identificar herramientas de prueba.
- ✓ Definir procedimientos de entrenamiento del personal participante.
- ✓ Identificar fuentes de información.

El contenido de un Plan de Prueba estándar podría ser el siguiente:

1. **Introducción**

Es una breve y concisa descripción del sistema a probar, incluyendo diagramas de análisis y Diseño además del ambiente de operación. Se discute en general la estrategia y los procedimientos específicos de prueba. Se incluye una descripción de los informes y documentación que se generará.

2. Descripción breve del sistema a probar

Incluye componentes, DFD, diagramas de estructura, entre otras y características generales del ambiente en el cual se usará el sistema.

3. Objetivos del Plan

Niveles de prueba: módulo, integración, sistema. Tipos de prueba: funciones, rendimiento, documentación. Aspectos que no se probarán como parte de este plan: instalación, características especiales.

4. Métodos de Testing

Estrategia de integración y regresión. Desarrollo de casos de prueba, inspección y verificación. Flujo de tareas y control de escenarios para aislar problemas. Informe de problemas y su solución. Herramientas de prueba.

5. Plan Global

Contiene los hitos a alcanzar durante el proceso de pruebas. Describe el material de apoyo para los procedimientos planificados y define las condiciones para iniciar y terminar las pruebas.

- ✓ Hitos: malla de pruebas e itinerario
- ✓ Material de prueba
  - Planes de prueba.
  - Resumen de casos de prueba.
  - Especificación de casos de prueba.
  - Escenarios de prueba.
  - Matriz de funciones y casos de pruebas.
  - Matriz de módulos llamados y quién lo llama.
  - Casos de pruebas de programas.
    - → Casos de prueba de pantalla.
    - → Resumen de resultados de prueba.
    - → Bitácora de pruebas.
    - → Informes de discrepancias.
    - → Entrenamientos.
- ✓ Criterios
  - Criterios de entrada.
  - Criterios de salida.

6. Requerimientos de testing

Describe los requerimientos humanos y materiales necesarios para efectuar el proceso de prueba.

- ✓ Hardware

Configuración requerida para los casos de prueba. Configuración requerida para simular ambiente de pruebas. Configuración requerida para pruebas de aceptación.

- ✓ Software
  - Sistema a probar (diversas configuraciones).
  - Sistema operativo.
  - Subsistema de comunicaciones.
  - Simulación de carga de la red.
  - Simulación del ambiente.

- Medidas del software.
- Software de ayuda a las pruebas.
- Itinerario de disponibilidad.

✓ Personal

- Características y número.
- Itinerario de disponibilidad.

Cabe destacar que es necesario completar los siguientes puntos para cada prueba que se realice. Para cada iteración de esta sección se debe definir objetivos detallados y verificables.

✓ Objetivos
✓ Descripción del Software
✓ Método
- Lista de funciones a probar
- Métodos de Diseño de casos de pruebas
- Herramientas de prueba a usar

✓ Hitos, red de pruebas e itinerario
✓ Requerimientos
- Hardware
- Software
- Datos/Archivos
- Personal

✓ Criterios
- De entrada
- De salida

✓ Material con resultados de pruebas
✓ Control de ejecución
- Inicio de pruebas
- Registro de actividades
- Acciones a realizar si hay fallas
- Autorizaciones

9.2 Estrategias de Diseño de caja negra y caja blanca

Los casos de pruebas requieren tiempo y dinero, por lo que los casos de prueba que no encuentran un error son una pérdida de recursos. En consecuencia es primordial diseñar casos de prueba que sean eficaces.

Los métodos de Diseño y construcción de casos de pruebas se pueden dividir en dos grandes categorías basado en la perspectiva usada para generarlos. Si requieren conocer la estructura interna de los componentes del software, reciben el nombre de Caja Blanca o pruebas orientadas a la lógica. Si se basan en los requerimientos funcionales del sistema, se conocen como de Caja Negra o pruebas orientadas a los datos. Las pruebas de caja negra se consideran superiores a las de caja blanca, ya que pueden ser utilizados bien temprano en el desarrollo.

Estrategia de Caja Blanca

El conjunto ideal de datos de prueba de Caja Blanca es aquel que ejecuta exhaustivamente todos los caminos de los flujos de control en un módulo o programa. Esto no es posible en la práctica.

Estrategia de Caja Negra

Para realizar un testing de Caja Negra exhaustivo, se requiere probar todos los posibles casos de entrada al módulo o programa, lo que por lo general es imposible en la práctica.

9.2.1 Testing Básico

La técnica que se describe a continuación es una técnica de Caja Blanca que emplea el diagrama de flujos de control para representar la lógica. Su ventaja es que emplea el número ciclomático de McCabe como medida de complejidad del programa, por lo que existen herramientas que la soportan. Su desventaja es que los diagramas de flujos de control se construyen antes de iniciar la programación o se obtienen al finalizarla.

La técnica básica, descrita por McCabe, usa el código fuente o diagrama de flujo para generar diagramas con flujos de control de módulos de programa. El diagrama de flujo se usa para enumerar caminos independientes y calcular el número ciclomático (C), como una medida de complejidad. Se crea a continuación un conjunto básico de casos de prueba, que cubran todos los caminos independientes.

9.2.2 Diagrama de flujos de control

Es una red en la cual los nodos representan segmentos de programa y ramas son los conectores desde un segmento de programa a otro. Una decisión se representa como un nodo con múltiples ramas emergentes y un lazo es un nodo con una rama que emana y regresa al mismo nodo. Un camino es una ruta a través del grafo, desde un nodo de entrada a uno de salida.

El número ciclomático C, se calcula como:

$$C = E - N + 2$$

donde E es el número de ramas y N es el número de nodos.

Se puede calcular C también como:

$$C = D + 1$$

donde D es el número de decisiones primitivas.

Una decisión primitiva es una que evalúa los estados condicionales asociados con una condición simple. Las condiciones anidadas y condiciones conectadas por operadores lógicos se tratan como si fueran decisiones separadas. La figura a continuación compara los símbolos utilizados en los diagramas de flujos con los diagramas de flujos de control.

9.2.3 Testing Básico, versión Warnier Orr.

Combina las técnicas de prueba de McCabe con una técnica formalizadora como lo es la Metodología de Desarrollo de Sistemas con Datos Estructurados de Warnier Orr (WO).

Un conjunto de procesos WO que contiene conjuntos subordinados jerárquicamente, para los propósitos de pruebas, puede ser representado por un solo nodo en el grafo. De esta forma, los detalles lógicos que no es necesario probar en algún nivel, se pueden omitir. Los conjuntos superordenados pueden representarse por nodos conteniendo paréntesis vacíos. Estos nodos son indefinidos y contienen subdiagramas con procesos lógicos de complejidad desconocida.

9.2.4 Guía para obtener el diagrama de flujos de control desde los diagramas Warnier-Orr.

I. Para un nivel dado en los Diagramas de Warnier Orr, el conjunto de entrada corresponde al nodo de entrada.
II. Para un nivel dado el DWO, el conjunto de salida corresponde al nodo de salida.
III. Las secuencias de operaciones lógicas, secuencias alternativas de operaciones lógicas y repeticiones de operaciones lógicas en el nivel analizado, representan cada una nodos individuales.

IV. Las secuencias de operaciones lógicas, secuencias alternativas de operaciones lógicas y repeticiones de operaciones lógicas que son subordinadas en la jerarquía del DWO en el nivel analizado, son resumidas como un nodo indefinido.
V. El desarrollo de nodos desde el conjunto de operaciones lógicas procede top-down y desde la izquierda hacia la derecha.

9.3 Herramientas Automáticas de Prueba

Debido a que en la actualidad la prueba del software involucra un elevado porcentaje del esfuerzo total de un proyecto, todas las herramientas que tiendan a reducir el tiempo de prueba serán vistas como un valioso aporte.

A continuación se describirán varias categorías de herramientas de prueba :

- ✓ **Analizadores del código**: son filtros de propósito especial usados para comprobar la calidad del software de manera que cumpla con los standard mínimos de codificación.
- ✓ **Procesadores de asertos:** son usados para indicar si las demandas del programador (asertos) se satisfacen realmente durante las ejecuciones reales del programa.
- ✓ **Generadores de archivos de prueba:** estos procesadores generan, y rellenan con valores predeterminados, archivos de entrada típicos para posteriores programas de prueba.
- ✓ **Generadores de datos de prueba:** estos sistemas ayudan al usuario a seleccionar datos de prueba.
- ✓ **Verificadores de prueba:** estas herramientas miden el alcance interno de la prueba (a menudo expresados en términos relativos a la estructura de control del objeto de prueba) informando al experto de calidad.
- ✓ **Controladores de prueba:** estas herramientas ayudan en el procesamiento de las pruebas reduciendo al mínimo la dificultad de instalar un programa candidato en un entorno de prueba, alimentarlo con los datos de prueba y simular el comportamiento de módulos subordinados.
- ✓ **Comparador de salida:** estos sistemas ayudan a la comparación de un conjunto de salidas con otro conjunto (previamente archivados) con el fin de establecer diferencias entre ellos
- ✓ **Sistemas de ejecución simbólica:** estas herramientas llevan a cabo pruebas de programa usando entradas algebraicas, en valor de datos numéricos. Con ello se pretende que el software pruebe clases de datos, en lugar de un caso específico de pruebas.
- ✓ **Simuladores de entorno:** son sistemas especializados que permiten modelar el entorno externo del software en tiempo real y así simular dinámicamente las condiciones reales de operación.
- ✓ **Analizadores de flujos de datos:** estas herramientas siguen la pista al flujo de datos e intentan encontrar referencias a datos indefinidos, indexaciones incorrectas y otros errores relativos a datos.

A menudo los experimentados realizadores de software dicen: “la prueba nunca termina, simplemente se transfiere del encargado del desarrollo al cliente. Cada vez que el cliente usa el programa, lleva a cabo una prueba.”

La Tabla 9.1 muestra la eficiencia en la prevención y remoción de defectos, (% de defectos)

	Requerimientos	Diseño	Codificación	Documentación
JAD	50	25	10	15
Prototipos	40	35	35	15
R. Requerimiento	40	15	0	5
R. Diseño	15	55	0	15
I. Código	20	40	65	25
Sub total	75	85	73	50
Testing unitario	10	5	20	0
Testing funcional	10	15	30	5
Testing del sistema	10	15	35	20
Testing en terreno	20	20	25	25
Sub total	35	45	75	43
Total	**87**	**92**	**94**	**72**

Tabla 9.1 Eficiencia en la prevención y remoción de defectos

De la Tabla 9.1 se pueden derivar una serie de apreciaciones con respecto al testing :

- ✓ El testing se presenta bastante caro para el porcentaje de defectos que evita.
- ✓ El testing es mejor que sea utilizado en la etapa de Diseño que en la de Requerimientos.
- ✓ El testing en la codificación ofrece buenos resultados si se le compara con otras técnicas.

Los errores detectados por el testing son mayoritariamente superficiales, sin mayor impacto.

Capítulo 10

Valores necesarios para el Éxito de un Proyecto

Una de las tareas importantes en el éxito de un proyecto de desarrollo de Software o de implantación de TI es el unir los diversos elementos que constituyen el entorno del proyecto, de optimizar el uso del tiempo, de facilitar los recursos a los miembros del equipo de trabajo, resolver las diferencias de visiones de los usuarios, clientes y desarrolladores, de satisfacer y ajustar las expectativas de todos.
Para ello es necesario que quienes participan en el equipo de proyecto y especialmente quienes asuman roles de liderazgo en él, tengan en consideración que 30% del éxito de éste depende de cuan ajustado, disciplinado, ordenado, armonioso y equilibrado sea el conjunto de personas que componen el equipo de proyecto.
Para ello es necesario practicar y encarnar algunos valores [12] que harán de quienes dirigen un foco de unión, guía y sustento para el grupo de tareas:

Cooperación

La realización humana es como una cordillera con precipicios, riscos, pendientes y valles. Alcanzar la perfección en un esfuerzo colectivo es como desear conquistar la cima más alta. El esfuerzo requiere que cada alpinista esté equipado con habilidades y conocimientos esenciales, mucha determinación y fuerza de voluntad. Sin embargo, no se debe emprender la ascensión sin lo más indispensable: la cuerda de seguridad de la cooperación. La cooperación asegura ecuanimidad, capacitación, facilidad y entusiasmo. La cooperación provee los medios para que cada escalador dé un paso, por pequeño que sea, y que todos esos pasos, unidos, permitan alcanzar la cumbre.
La cooperación no es un mero regateo en el que el éxito de una persona se logra a expensas o gracias a la exclusión del éxito de otras. El objetivo constante de la cooperación es el beneficio mutuo en las interrelaciones humanas; se fundamenta en el principio del respeto mutuo. El valor, la consideración, el cuidado y la participación proporcionan un fundamento a partir del cual puede desarrollarse el proceso de la cooperación.

Si la capacidad de discernir es clara cuando una persona o grupo precisen cooperación y se aplique el método apropiado, habrá éxito en las relaciones e interacciones humanas. El método puede ser tan sencillo como ofrecer una explicación, brindar amor o apoyo, o saber escuchar. Sin embargo, si no se dispone de la capacidad de discernir el tipo de cooperación adecuada ni el método correcto para proporcionarla, no se experimentará éxito en la forma de acuerdo y de satisfacción .
La cooperación es posible cuando hay facilidad, no pesadez. Ser fácil significa ser sincero y de espíritu generoso. Tal generosidad le hace a uno digno de recibir la cooperación de todos. Si uno tiene fe y confianza en los demás, eso,

en retorno, construye la fe y confianza en ellos. Tales sentimientos producen un ambiente de enriquecimiento, respeto, apoyo y solidaridad.
Cooperar es responsabilidad de todos, aunque facilitar el proceso requiere valor y fortaleza interna. A veces, los que asumen la responsabilidad se convierten en el blanco de insultos y críticas. Se requiere una preparación fundamental para crear un mecanismo de apoyo interno mediante el cual las personas sean capaces de protegerse a sí mismas y de mantener la ecuanimidad y el equilibrio. Se necesita una actitud de desapego, en la que nada se tome a nivel personal. Al permanecer desapegado, objetivo e influenciado por los valores más internos y no por las circunstancias externas, surge la cooperación en forma de sabiduría. Mirar a otro con una actitud de amor y de cooperación, aun después de haber sido "difamado" por esa persona, se reconoce como tener una visión misericordiosa. La perspectiva de uno está llena de comprensión, perdón, tolerancia, paciencia y empatía. El que adopta esa actitud, elimina más fácilmente las trabas de la falta de cooperación que pueden haber obstruido el progreso.

La cooperación requiere reconocer el papel único de cada persona, a la vez que mantener una actitud sincera y positiva. Los pensamientos positivos dentro del ser automática y fácilmente crean sentimientos de cooperación en la mente de los demás. El método para ofrecer cooperación es usar la energía de la mente para crear vibraciones de buenos deseos y sentimientos puros hacia los demás y hacia la tarea. Esto afecta al ambiente en una forma positiva y sutil. Las vibraciones colectivas de un esfuerzo tan puro y sutil preparan el terreno para deliberaciones abiertas y profundas, así como para períodos exitosos de cooperación.

La cooperación, con el tiempo y con el orden natural de los acontecimientos, genera paciencia. El tiempo es valioso porque siempre ofrece la oportunidad única de conseguir lo que es mejor y lo que es necesario en el momento adecuado. El tiempo coopera con cada persona si ésta reconoce su importancia.

En el proceso de transformar el mundo, ahora es el momento de que cada persona aporte una pequeña dosis de cooperación; si no es con la mente, entonces con el trabajo físico; si no es con el trabajo físico, entonces con la riqueza; si no es con la riqueza, entonces apoyando o motivando a otros a cooperar. Si cada uno aportara un dedo de cooperación, ¡juntos podríamos levantar una montaña! ¡Y cuando se reconozcan como indestructibles los vínculos espirituales que nos unen en hermandad universal, la cooperación será inevitable y juntos podremos alcanzar nuevas y grandes cimas!

Humildad

La humildad se encuentra en un vasto océano de aguas tranquilas que fluyen en la profundidad. En lo profundo yace la autoestima. Al principio, adentrarse en el océano es como viajar a una zona desconocida de inmensa oscuridad. Pero, así como explorar puede llevar a descubrir tesoros enterrados, en la búsqueda del mundo interior se pueden encontrar joyas enterradas en las profundidades de uno mismo. Y la joya que está enterrada en lo más

profundo, la que más brilla y más luz da es la humildad. Sus rayos penetran en los momentos más oscuros. Elimina el miedo, la inseguridad y abre a la persona a las verdades universales.
Humildad es aceptar los principios naturales que no se pueden controlar. Todo lo que tenemos, desde el cuerpo con el que hemos nacido hasta las posesiones más preciadas, se hereda. Por lo tanto, se vuelve un imperativo moral el utilizar estos recursos de forma valiosa y benevolente. La conciencia de ser un depositario de tales recursos ilimitados y atemporales toca la esencia del alma humana y la despierta para darse cuenta de que, así como en el momento de nacer se heredaron esos recursos, en el momento de morir se tendrán que abandonar. En la muerte, todo lo que acompañará a la persona serán las impresiones de cómo se usaron esos recursos junto con la sabiduría de ser y de vivir como un depositario.

La conciencia de ser un depositario eleva la autoestima y realza las múltiples relaciones diferentes encontradas a lo largo de la vida. Le lleva a uno a un estado de reflexión silenciosa, invitándole a tomarse un tiempo para sí mismo y a mirar la vida desde una perspectiva diferente. El reconocimiento de ser un depositario hace que la persona busque la renovación de las relaciones con el propio ser y con el mundo.
Humildad es dejar hacer y dejar ser. La piedra del conflicto yace en la conciencia del "yo" y del "mío" y la posesividad: de un rol, de una actividad, de un objeto, de una persona, incluso del cuerpo. Paradójicamente, esta conciencia le hace perder a uno aquello a lo que quiere aferrarse y, especialmente, le hace perder lo más significativo, los valores universales que dan valor y sentido a la vida. La humildad elimina la posesividad y la visión limitada que crean límites físicos, intelectuales y emocionales. Estas limitaciones destruyen la autoestima y levantan muros de arrogancia y de orgullo que distancian a las personas. La humildad actúa suavemente en las fisuras, permitiendo el acercamiento.

Todo el mundo se "reverencia" ante una persona que posee la virtud de la humildad, ya que todos se reverencian ante los que se han reverenciado primero. Por tanto, el signo de la grandeza es la humildad. La humildad permite a la persona ser digna de confianza, flexible y adaptable. En la medida en que uno se vuelve humilde, adquiere grandeza en el corazón de los demás. Quien es la personificación de la humildad hará el esfuerzo de escuchar y aceptar a los demás. Cuanto más acepte a los demás, más se le valorará y más se le escuchará. La humildad automáticamente le hace a uno merecedor de alabanzas.

El éxito en el servicio a los demás proviene de la humildad. Cuanto mayor sea la humildad, mayor el logro. No puede haber beneficio para el mundo sin humildad. El servicio se lleva a cabo de la mejor manera cuando 1) nos consideramos un depositario o instrumento y 2) cuando damos el primer paso para aceptar a otro que es diferente.

Una persona humilde puede adaptarse a todos los ambientes, por extraños o negativos que éstos sean. Habrá humildad en la actitud, en la visión, en las palabras y en las relaciones. La persona humilde nunca dirá: "no era mi

intención decirlo, pero simplemente surgieron las palabras". Según sea la actitud, así será la visión; según sea la visión, las palabras reflejarán eso y los tres aspectos combinados asegurarán la calidad de las interacciones. La mera presencia de una persona humilde crea un ambiente atractivo, cordial y confortable. Sus palabras están llenas de esencia, poder y las expresa con buenos modales.
Una persona humilde puede hacer desaparecer la ira de otra con unas pocas palabras. Una palabra dicha con humildad tiene el significado de mil palabras.

Respeto

El respeto comienza en la propia persona. El estado original del respeto está basado en el reconocimiento del propio ser como una entidad única, una fuerza vital interior, un ser espiritual, un alma. La conciencia elevada de saber "quién soy" surge desde un espacio auténtico de valor puro. Con esta perspectiva, hay fe en el propio ser así como entereza e integridad en el interior. Con la comprensión del propio ser se experimenta el verdadero autorrespeto.

El conflicto se inicia cuando falta el reconocimiento de la propia naturaleza original y la del otro. Como resultado, las influencias negativas externas dominan completamente el respeto. Estabilizarse en el estado elevado del propio ser asegura auténtico respeto por y de los demás debido a que se actúa con la conciencia de que todo ser humano tiene un valor innato, que es puro y virtuoso. Esta forma de pensar garantiza la victoria final, porque la interacción sobre esta base asegura que surja la bondad inherente del propio ser y de los demás.

La causa de todas las debilidades se origina en la ausencia de autorrespeto. La persona se llena de diferentes deseos o expectativas, exigiendo consideración o respeto de los demás. La persona, al hacerse dependiente de fuerzas externas en lugar de sus poderes internos, mide el respeto mediante los factores físicos y materiales, tales como la casta, el color, la raza, la religión, el sexo, la nacionalidad, el estatus y la popularidad. Cuanto más se mide el respeto sobre la base de algo externo, mayor es el deseo de que los demás tengan un reconocimiento hacia mí. Cuanto mayor es ese deseo, más se es víctima del mismo y se pierde el respeto hacia uno mismo y hacia los demás. Si las personas renunciaran al deseo de recibir consideración de los demás y se estabilizaran en el estado elevado de autorrespeto, la consideración y el respeto los seguiría como una sombra.

El desafío es desarrollar el valor del respeto en el propio ser y darle una expresión práctica en la vida diaria. Aparecerán obstáculos para probar la solidez del respeto y, con frecuencia, se sentirán en los momentos de más vulnerabilidad. Es necesaria la confianza en uno mismo para tratar con las circunstancias con seguridad, de manera optimista, esperanzadora. En las situaciones en las que parece que todos los apoyos se han desvanecido, lo que permanece fiel es el nivel en que se ha podido confiar internamente en el propio ser.

El poder de discernir crea un ambiente de respeto, en el que se presta atención a la calidad de las intenciones, actitudes, conductas, pensamientos, palabras y acciones. En la medida que exista el poder de la humildad en el respeto hacia el propio ser —y el discernimiento y la sabiduría que permiten ser justo e imparcial con los demás— habrá éxito en la forma de valorar la individualidad, apreciar la diversidad y tomar en consideración la tarea en su totalidad. El equilibrio entre la humildad y el autorrespeto da como resultado el servicio altruista, una actuación honrosa desprovista de actitudes débiles tales como la arrogancia y la estrechez mental. La arrogancia daña o destruye la autenticidad de los demás y viola sus derechos fundamentales. Un temperamento así perjudica también al transgresor. Por ejemplo, la tendencia a impresionar, dominar, o limitar la libertad de los demás se manifiesta con el propósito de imponerse en detrimento del valor interno, de la dignidad y la paz mental. El respeto original se subordina a uno artificial.

Por tanto, pretender ganar respeto sin permanecer consciente del propio valor original se convierte en el método mismo para perderlo. Conocer el valor propio y honrar el de los demás es la auténtica manera de ganar respeto. Puesto que tal principio tiene su origen en ese espacio prístino de valor puro, los demás sienten intuitivamente, la autenticidad y la sinceridad. En la visión y la actitud de igualdad existe una espiritualidad compartida. Compartir crea un sentimiento de pertenecer, un sentimiento de familia.

Capítulo 11

Casos de Estudio

11.1 Caso de Estudio Nº1: "Desarrollo de un Sistema de Control de Gestión para Servicio Nacional de Aduanas"

El Sistema de Control de Gestión[14], debe permitir coordinar, monitorear y evaluar permanentemente los compromisos de gestión de la organización, a nivel nacional y regional. Algunos detalles del sistema son:

- Contar con un sistema computacional, flexible, operando en red, en ambiente Windows, bajo esquema web (modalidad Intranet), vinculado con otros sistemas de la organización, de tal forma que sea capaz de alimentarse con información que le administre dicho sistema.

- Que permita la creación de series estadísticas, gráficos y tablas de datos.

- Que acepte el ingreso de datos de todas las fuentes posibles: red, Internet, datos manuales, disquete, CD's, etc.

- Que permita el acceso diferenciado por claves y distintos niveles de seguridad, acorde a la función de cada usuario. El sistema deberá identificar y ser capaz de discriminar diferentes categorías de usuarios, con privilegios asociados: ej: sólo consulta; consulta e ingreso de información nueva; consulta, ingreso y modificación de la información, esta última categoría será definida sobre la base de especificaciones definidas por el Departamento de Estudios.

- El sistema deberá contar con un módulo de auditoría que permita la revisión de los registros de información y las modificaciones efectuadas.

En este caso existe una restricción, que es el tiempo. El análisis y diseño lógico debe ser en un plazo máximo de 90 días. El diseño y desarrollo del software debe ser terminado en un plazo máximo de 150 días. Esta última es la etapa de construcción propiamente tal.

Estimaciones y Cálculos

Puntos de Función

La finalidad de la técnica de puntos de función es estimar el tamaño de un producto de software y el esfuerzo asociado a su desarrollo, en las etapas previas a su desarrollo.

Para el método de Puntos de Función se identifican dos etapas:
- ✓ Conteo y clasificación de las funciones del usuario
- ✓ Ajuste del modelo en función de la complejidad del proceso

Conteo y clasificación de las Funciones del Usuario

En esta primera etapa se identifican cinco tipos de funciones de usuario:

- Entradas Externas (al sistema): entradas de usuario que proporcionan al sistema diferentes datos orientados a la aplicación.

- Salidas Externas (al sistema): salidas de usuario que le proporcionan a éste información sobre la aplicación.

- Consultas Externas (al sistema): peticiones de usuario que como resultado obtienen algún tipo de respuesta en forma de salida.

- Archivos Lógicos Internos: número de archivos lógicos maestros.

- Archivos de Interface Externa: interfaces legibles utilizados para transmitir información a otros sistemas.

Entradas Externas

i. Modificación de información mostrada por los informes
Consiste en una pantalla que permitirá al usuario configurar y definir aquellos campos que considera relevante para que se muestren en los informes entregados por el sistema.
Clasificación: Complejidad Media

Salidas Externas

i. Listado de Ingresos
Consiste en el despliegue de la información de los ingresos al país a través de aduanas de chile.
Clasificación: Complejidad Simple

ii. Listado de Egresos

Consiste en el despliegue de la información de los egresos desde el país a través de aduanas de chile.
Clasificación: Complejidad Simple

iii. Listado de Mercancías Importadas
Consiste en el despliegue de los datos referentes a las mercancías importadas en detalle según el tipo de mercancía.
Clasificación: Complejidad Simple

iv. Listado de Mercancías Exportadas
Consiste en el despliegue de los datos referentes a las mercancías exportadas desde el país hacia algún todos o algún destino específico según el tipo de mercancía.
Clasificación: Complejidad Simple

v. Listado de Importadores
Considera el despliegue de información relacionada a los importadores de mercancías al país.
Clasificación: Complejidad Simple

vi. Listado de Exportadores
Considera el despliegue de información relacionada a los exportadores de mercancías al país.
Clasificación: Complejidad Simple

vii. Gráfico de Ingresos
Corresponde al despliegue gráfico de indicadores de ingresos al país desde el extranjero bajo criterios definidos por el usuario.
Clasificación: Complejidad Media

viii. Gráfico de Egresos
Corresponde al despliegue gráfico de indicadores de egresos desde el país desde el extranjero bajo criterios definidos por el usuario.
Clasificación: Complejidad Media

ix. Gráfico de Mercancías Importadas
Corresponde al despliegue gráfico de indicadores de las mercancías que han ingresado al país desde el extranjero bajo criterios definidos por el usuario.
Clasificación: Complejidad Media

x. Gráfico de Mercancías Exportadas
Corresponde al despliegue gráfico de indicadores de las mercancías que han ingresado al país desde el extranjero bajo criterios definidos por el usuario.
Clasificación: Complejidad Media

xi. Gráfico de Importadores
Considera gráficos correspondientes a las tendencias de los indicadores relevantes en relación a los importadores de mercancías al país.
Clasificación: Complejidad Media

xii. Gráfico de Exportadores
Considera gráficos correspondientes a las tendencias de los indicadores relevantes en relación a los exportadores de mercancías desde el país.
Clasificación: Complejidad Media

Consultas Externas

i. Pantallas de Ayuda
Corresponde a la ayuda on-line definida por el cliente en las Bases Técnicas.
Clasificación: Complejidad Compleja

ii. Solicitud de Generación de Informes
Consiste en una pantalla por la cual el usuario podrá solicitar los informes que desea se generen.
Clasificación: Complejidad Simple

Archivos Lógicos Internos

i. Registro Documento de transporte
Corresponde a una base de datos de alrededor de 10 campos
Clasificación: Complejidad Simple

ii. Registro Manifiesto de ingreso
Corresponde a una base de datos de alrededor de 20 campos

Clasificación: Complejidad Simple

iii. Registro Manifiesto de salida
Corresponde a una base de datos de alrededor de 20 campos
Clasificación: Complejidad Simple

iv. Registro Documento de pago
Corresponde a una base de datos de alrededor de 10 campos
Clasificación: Complejidad Simple

v. Registro Declaración de ingreso
Corresponde a una base de datos de alrededor de 20 campos
Clasificación: Complejidad Simple

vi. Registro Declaración de salida
Corresponde a una base de datos de alrededor de 20 campos
Clasificación: Complejidad Simple

vii. Registro Mercancía
Corresponde a una base de datos de alrededor de 15 campos
Clasificación: Complejidad Simple

viii. Registro Aviso de retiro
Corresponde a una base de datos de alrededor de 15 campos
Clasificación: Complejidad Simple

ix. Registro Orden de entrega
Corresponde a una base de datos de alrededor de 10 campos
Clasificación: Complejidad Simple

x. Registro Entrega
Corresponde a una base de datos de alrededor de 10 campos
Clasificación: Complejidad Simple

xi. Registro Guía de transporte
Corresponde a una base de datos de alrededor de 10 campos
Clasificación: Complejidad Simple

xii. Registro Importador
Corresponde a una base de datos de alrededor de 10 campos
Clasificación: Complejidad Simple

xiii. Registro Exportador
Corresponde a una base de datos de alrededor de 10 campos
Clasificación: Complejidad Simple

xiv. Registro Transportista Internacional
Corresponde a una base de datos de alrededor de 10 campos
Clasificación: Complejidad Simple

xv. Registro Agente de aduanas
Corresponde a una base de datos de alrededor de 10 campos
Clasificación: Complejidad Simple

xvi. Registro Forwarder
Corresponde a una base de datos de alrededor de 10 campos
Clasificación: Complejidad Simple

xvii. Registro Almacenista
Corresponde a una base de datos de alrededor de 10 campos
Clasificación: Complejidad Simple

Archivos de Interface Externa

i. Contraseñas de ingreso al sistema
Corresponde a la diferenciación de perfiles de usuario a través de contraseñas.
Clasificación: Complejidad Simple

ii. Interfaces de entrada de datos al sistema
Corresponde a la comunicación de datos con el sistema de integración de la información de aduanas.
Clasificación: Complejidad Compleja

iii. Interfaces de salida de datos al sistema
Corresponde a la comunicación de datos con el sistema de integración de la información de aduanas.
Clasificación: Complejidad Compleja

La tabla 11.1 muestra el cálculo de los puntos de función no ajustados, a partir de la información anterior.

Funciones	Complejidad		Cantidad	Total por tipo
Archivos lógicos internos	Simples	7	17	119
	Medios	10	0	0
	Complejos	15	0	0
Archivos de interface externa	Simples	5	1	5
	Medios	7	0	0
	Complejos	10	2	20
Entradas externas	Simples	3	0	0
	Medios	4	1	4
	Complejos	6	0	0
Salidas externas	Simples	4	6	24
	Medios	5	6	30
	Complejos	7	0	0
Consultas externas	Simples	3	1	3
	Medios	4	0	0
	Complejos	6	1	6

Tabla 11.1: Cálculo de PFNA

Ajuste del modelo en función de la complejidad del proceso

En esta segunda etapa es necesario analizar las características propias del proceso respondiendo 14 preguntas, utilizando la tabla 11.2.

Grado	Descripción
0	No está presente o no influye
1	Influencia Mínima
2	Influencia Moderada
3	Influencia promedio
4	Influencia significativa
5	Influencia fuerte

Tabla 11.2: Ponderaciones

Las 14 preguntas a responder se muestran en la Tabla 11.3.

Característica	Factor
(1) ¿Se requiere comunicación de datos?	5
(2) ¿Hay funciones de proceso distribuido?	0
(3) ¿El rendimiento es crítico?	3
(4) ¿El sistema necesita copias de seguridad y recuperación de datos?	5
(5) ¿El sistema ha de ejecutarse en un entorno existente altamente utilizado?	3
(6) ¿El sistema requiere entrada on-line de datos?	0
(7) ¿La aplicación esta diseñada para facilitar la interacción con el usuario?	5
(8) ¿Los archivos maestros son actualizados on-line?	3
(9) ¿El proceso interno es complejo?	4
(10) ¿El código ha de ser diseñado para ser reutilizable?	5
(11) ¿Son complejas la entradas, salidas, consultas o archivos?	5
(12) ¿El diseño incluye la conversión e instalación?	5
(13) ¿El sistema esta diseñado para múltiples instalaciones en diferentes organizaciones?	5
(14) ¿Los datos de entrada on-line requieren su transmisión entre varias pantallas u operaciones?	1
Factor de Ajuste	49

Tabla 11.3: Cálculo del FA

De este modo es posible calcular el factor de ajuste, cuyo valor es 49 y mediante éste, podemos calcular los puntos de función ajustados como se muestra a continuación.

$$PF = PFNA^* (0,65 + 0,01 \times FA)$$
$$PF = 211^*(0,65 + 0,01 \times 49)$$
$$PF = 240,54$$

Luego, los puntos de función del problema son 240.54, los cuales se pueden traducir a Kilo Líneas De Código (KLDC) sabiendo que en Java cada punto de función corresponde a 53 LDC (ver Anexo D). Luego, tenemos que:

$$LDC = 53 \times PF$$
$$LDC = 53 \times 240{,}54$$
$$LDC = 12748{,}62$$
$$KLDC = 12{,}74862$$

Para calcular, el esfuerzo de desarrollo se puede utilizar el factor de productividad para un programador en Java, la cual corresponde a 10 – 20 puntos de función / persona / mes. Con este valor calculamos y considerando 240.54 puntos de función tenemos:

$$ESFUERZO_1 = 240\ [PF]/10\ [PF/PM]$$
$$ESFUERZO_2 = 240\ [PF]/20\ [PF/PM]$$

$$12.07\ [PM] < ESFUERZO < 24.054[PM]$$

Observación: Se considera como costo por hora hombre del personal de desarrollo del sistema un valor de 92.85 [UF/PH], el cual se ha obtenido a partir de la ponderación de los distintos profesionales participantes en el proyecto, entre los cuales se consideran un jefe de proyecto, 3 analistas, 6 programadores, además de un administrador de base de datos, un diseñador y un diseñador de interfaces (Ver Anexo D).

Luego, para 8 [Meses] de desarrollo, considerando un esfuerzo promedio de 18 [PM] necesitamos 2.25 ≈ 3 personas para el desarrollo, lo que traducido en costos resulta en:

$$COSTO = 3 * 116\ [UF/PM] * 8\ [Meses]$$
$$COSTO = 2.784\ [UF]$$
$$COSTO = \$\ 47.347.488$$

CoCoMo

CoCoMo (Constructive Cost Model) es una técnica de estimación de esfuerzo de las actividades de diseño, codificación, pruebas y mantenimiento.

La primera consideración se debe tomar consiste en decidir qué modelo de CoCoMo se aplica en la estimación, en este sentido se consideran CoCoMo básico, intermedio y avanzado. Para el cálculo particular del esfuerzo en este proyecto se utilizará CoCoMo básico e intermedio.

CoCoMo básico

Para aplicar CoCoMo básico es necesario, primero, clasificar el proyecto que se llevará a cabo en una de las siguientes tres categorías.

Orgánico: Proyectos de no más de 50 KLDC (50.000 LDC), sobre áreas muy específicas y bien conocidas por el equipo participante.

Semiconectado: El nivel de experiencia del equipo de desarrollo se sitúa en niveles intermedios y suelen ser sistemas con interfaces con otros sistemas, siendo su tamaño menor a 300 KLDC.

Empotrado (restringido): Proyectos de gran envergadura, con una exigencia de altos niveles de fiabilidad y en los que participan muchas personas.

En este sentido se ha decidido clasificar el proyecto como semiconectado dadas sus características de trabajo con personal experimentado y novato tanto en la tecnología como en la aplicación en sí (según lo descrito en los supuestos), y por su interconexión con otros sistemas existentes, lo que puede volver complejo el manejo de las interfaces del sistema.

Luego, para calcular el esfuerzo del proyecto, se utiliza la siguiente ecuación:

$$ESFUERZO = A*KLDC^{B} [PM]$$

Como se determinó que se trata de un proyecto semiconectado, se toma A = 3 y B=1,12. Luego, reemplazando los valores obtenidos anteriormente, se tiene:

$$ESFUERZO = 3*KLDC^{1,12} [PM]$$
$$ESFUERZO = 51,9 [PM]$$
$$ESFUERZO \approx 52 [PM]$$

Para calcular el tiempo de desarrollo se considera la siguiente ecuación:

$$Tiempo\ de\ Desarrollo = C*KLDC^{D} [Meses]$$

Para un proyecto semiconectado, se toma C = 2,5 y D = 0,35. Al reemplazar los valores en la ecuación, obtenemos:

$$Tiempo\ de\ Desarrollo = 3*KLDC^{1,12} [Meses]$$
$$Tiempo\ de\ Desarrollo = 6,09 [Meses]$$
$$Tiempo\ de\ Desarrollo \approx 6,1 [Meses]$$

Por último el personal requerido sería:

$$Personal = ESFUERZO / Tiempo\ de\ Desarrollo$$
$$Personal = 52 / 6,1 = 8 [personas]$$

Es decir, dado que en realidad se cuenta con 8 [Meses] para el desarrollo del sistema se puede reducir el personal a 6 [personas].

Luego, el costo para 8 [Meses] de desarrollo, y considerando 6 personas en el equipo de trabajo resulta en un costo de:

COSTO = 6 * 116 [UF/PM] * 8 [Meses]
COSTO = 5.568 [UF]
COSTO = $ 94.694.976

CoCoMo intermedio

En este caso se cuenta con la siguiente ecuación para el cálculo del esfuerzo de desarrollo:

$$ESFUERZO = 3*FA*KLDC^{1,12} \text{ [Meses]}$$

Donde FA corresponde al factor de ajuste para el modelo CoCoMo intermedio, el cual se calcula a partir del producto de la ponderación de los siguientes 14 factores, ver Tabla 11.4:

ATRIBUTOS	Factor
Atributos del Producto	
Confiabilidad del software	1.4
Tamaño de la base de datos de la aplicación	1.16
Complejidad del producto	1.15
Atributos del Computador	
Restricciones de tiempo de ejecución	1
Restricciones de memoria principal	1
Volatilidad de la máquina virtual	1.15
Tiempo de respuesta de la computadora	1
Atributos del Personal	
Capacidad del analista	0.86
Experiencia en aplicaciones	0.82
Capacidad del programador	0.86
Experiencia con la máquina virtual	1.1
Experiencia con el lenguaje de programación	0.95
Atributos del Proyecto	
Prácticas modernas de programación	0.91
Uso de herramientas de software	0.91
Calendario de desarrollo requerido	1.1
[Cocomo Intermedio] Factor de Ajuste	**1.2398**

Tabla 11.4: Cálculo del FA

Obtenido el factor de ajuste podemos calcular el esfuerzo que resulta a continuación.

$$ESFUERZO = 3*1,24*KLDC^{1,12} \text{ [PM]}$$
ESFUERZO = 64,36 [PM]
ESFUERZO ≈ 65 [PM]

Luego, considerando un desarrollo de 8 [meses] se requieren 8 personas.

Por lo tanto, el costo para 8 [Meses] de desarrollo y 8 personas en el equipo de trabajo resulta en:

COSTO = 8* 116 [UF/PM]*8 [Meses]
COSTO = 7.424 [UF]

COSTO = $ 126.259.968

CoCoMo II

CoCoMo II es una nueva formulación de CoCoMo y está adaptado a los ciclos de vida de los modelos de desarrollo de software actuales. El modelo CoCoMo original ha resultado muy exitoso, pero resulta imposible de aplicar a aquellas nuevas prácticas no tradicionales de software.

CoCoMo II consiste en realidad en tres diferentes modelos:

- El Modelo de Composición de la Aplicación
- El Modelo de Diseño Temprano (Pre-Arquitectura)
- El Modelo Post-Arquitectura

Dado que se realizará una estimación antes de realizar la arquitectura del sistema utilizaremos el modelo de diseño temprano. Para el cálculo del esfuerzo se considera la siguiente ecuación:

$$ESFUERZO = 2{,}94*EAF*KLDC^{E}\ [PM]$$

En la cual el factor EAF se calcula en base al producto de la ponderación de los siguientes atributos, ver tabla 11.5.

ATRIBUTOS	Factor
Atributos del Producto	
Confiabilidad del software	1.1
Tamaño de la base de datos de la aplicación	1.14
Complejidad del producto	1
Reusabilidad	1.07
Necesidades de documentación	1.11
Atributos del Computador	
Restricciones de tiempo de ejecución	1.11
Restricciones de memoria principal	1
Volatilidad de la plataforma	1.15
Atributos del Personal	
Capacidad del analista	0.85
Experiencia en aplicaciones	0.88
Capacidad del programador	0.88
Experiencia en el lenguaje y las herramientas	1.09
Experiencia en la plataforma	1.09
Continuidad del personal	0.81
Atributos del Proyecto	
Desarrollo Multisitio	1.09
Uso de herramientas de software	1.09
Calendario de desarrollo requerido	1
[Cocomo Intermedio] Factor de Ajuste	**1.4308**

Tabla 11.5: Cálculo EAF

Por otro lado, el exponente E se calcula a partir de los siguientes 5 factores mediante la siguiente ecuación:

$$E = 0{,}91+0{,}01*\sum Fi$$

Estos 5 factores se muestran en la Tabla 11.6.

FACTORES	
Desarrollos previos similares	3.72
Flexibilidad del desarrollo	1.01
Manejo de riesgos y arquitectura	5.65
Cohesión del equipo de desarrollo	5.48
Nivel de madurez estimada	1.56
[CoCoMo II] $\sum$Fi	17.42
[CoCoMo II] E	1.0842

Tabla 11.6: Cálculo E

Reemplazando en la ecuación original, obtenemos:

$$ESFUERZO = 2,94*1,43*KLDC^{1,08}\ [PM]$$
$$ESFUERZO = 66,45\ [PM]$$
$$ESFUERZO \approx 67\ [PM]$$

Para el tiempo de desarrollo se tiene:

$$Tiempo\ de\ Desarrollo = 3,67*\ ESFUERZO^{F}$$

Donde F se calcula mediante:

$$F = (0,28+0,002*\sum Fi)$$

Luego, el tiempo de desarrollo corresponde a:

$$Tiempo\ de\ Desarrollo = 12,04\ [Meses]$$

Esto corresponde a 6 personas trabajando durante alrededor de 12 [Meses]. Luego, si se considera 8 [Meses] para el desarrollo habrá que contratar a 9 personas para el desarrollo del proyecto.

Por lo tanto, el costo para 8 [Meses] de desarrollo y 9 personas en el equipo de trabajo resulta en:

$$COSTO = 9 * 116\ [UF/PM] *8\ [Meses]$$
$$COSTO = 8.352\ [UF]$$
$$COSTO = \$\ 142.042.464$$

MACOS

MACOS (Modelo Algorítmico de Costos de Software) considera como base para su cálculo la estimación de los puntos de función no ajustados, a partir de estos es posible calcular el valor del esfuerzo

$$\text{ESFUERZO} = 1{,}04 * (\text{PFNA})^{1{,}22} * \text{FA}$$

Donde FA corresponde al factor de ajuste que se calcula a partir de la suma de la ponderación de los siguientes 32 factores, ver Tabla 11.7

Factores	Ponderación
FACTORES DIFICILES DE CUANTIFICAR	
Calidad de las interfaces con el usuario	4
Calidad de la especificación de requerlmientos	2
Existencia de un plan de recursos y control del proyecto	2
Existencia de un plan detallado del proyecto	0
Existencia de un plan detallado de pruebas, de verificación y validación	0
Calidad del ambiente de trabajo	-1
Restricciones de seguridad y privacidad existentes	3
Calidad de la documentación	2
Calidad de la administración del proyecto	2
FACTORES CUANTIFICABLES: Perceptibles para el usuario del producto	
Volumen de transacciones a realizar	3
Tamaño de la base de datos	4
Características de captura, validación, actualización y almacenamiento de datos	2
Facilidad de operación	3
Facilidad y flexibilidad de cambio	4
Rendimiento requerido para la operación del producto de software	-2
Confiabilidad del producto de software entregado	4
Restricciones en el tiempo de desarrollo	1
FACTORES CUANTIFICABLES: Del ambiente usuario	
Modo de desarrollo y operación del producto de software.	1
Experiencia del usuario en la aplicación y en el procesamiento de datos	-2
FACTORES CUANTIFICABLES: Perceptibles para el desarrollador	
DEL PERSONAL DE DESARROLLO	
Capacidad de los analistas	1
Experiencia de los analistas	1
Capacidad de los programadores	1
Experiencia del grupo en la plataforma de desarrollo (configuración de hardware y software) necesaria para el desarrollo del producto de software	4
Experiencia en el lenguaje	1
Continuidad del personaL	-1
DEL COMPUTADOR	
Restricciones del almacenamiento principal y de uso de CPU durante el desarrollo	-1
Tiempo de respuesta promedio del computador	0
Volatilidad de la plataforma de desarrollo	3
DEL PROYECTO	
Herramientas de ayuda utilizadas	0
Prácticas modernas de programación utilizadas	0
DEL PRODUCTO	
Complejidad del producto de software	0
El diseño del producto de software contempla su facilidad de instalación, conversión y mantenimiento	3
[MACOS] $\sum$ Fi	44
[MACOS] FACTOR DE AJUSTE	**1.11**

Tabla 11.7: Cálculo del FA para MACOS

Luego, el esfuerzo estimado para el desarrollo del proyecto resulta:

$$ESFUERZO = 1{,}04 * (PFNA)^{1{,}22} * 1{,}11$$
$$ESFUERZO = 790{,}6\ [PH]$$
$$ESFUERZO \approx 791\ [PH]$$
$$ESFUERZO \approx 5\ [PM]$$

La aproximación a 5 [PM], es considerando 160 horas mensuales de trabajo por persona.

Se calcula el tiempo de desarrollo dividiendo el esfuerzo por la cantidad de personas para el desarrollo del proyecto. A partir de las consideraciones realizadas, se deduce que la mejor alternativa resulta al contratar 4 personas (equipo de trabajo más pequeño) y trabajar durante 1 mes y medio.

Para esta estimación el costo del proyecto resulta en:

$$COSTO = 4 * 116\ [UF/PM] * 1{,}5\ [Meses]$$
$$COSTO = 696\ [UF]$$
$$COSTO = \$\ 11.836.872$$

Con toda la información resultante de las estimaciones podemos establecer la Tabla 11.8.

	PF	COCOMO 81 Básico	COCOMO 81 Intermedio	COCOMO II	MACOS
Costo (UF)	2.784	5.568	7.424	8.352	696
Costo ($)	47.347.488	94.649.976	126.259.968	142.042.464	11.836.872
Esfuerzo [PM]	18	52	65	67	5
Personal	3	6	8	9	4
Tiempo	8	8	8	8	1.5

Tabla 11.8: Resumen de Estimaciones

Finalmente, con algún criterio y la experiencia de quien esta estimando, se establece el valor de esfuerzo, plazo y costo a utilizar.

11.2 Caso de Estudio Nº2: "Sistema de Información para Compras y abastecimiento"(SICA).

El Sistema de Información a desarrollar[15] tiene como principal objetivo controlar el proceso de solicitud de compra y abastecimiento de la DT.
Los requerimientos mínimos para el Sistema son:

- ✓ Controlar el proceso de solicitud de compra y abastecimiento, para ello debe indicar las etapas en las cuales se encuentra un determinado documento.
- ✓ Disponer de los elementos que permitan llevar un registro a nivel de artículo de características tales como: stock disponible, número de inventario, unidad de medida, costo, fecha de última compra, ultimo proveedor, stock critico, proveedores posibles, volumen mínimo a comprar, familia, grupo, etc.
- ✓ Disponer de los elementos que permitan llevar un registro de las compras, registro de solicitudes y registro de despachos realizados.
- ✓ Emitir informes tales como: lista de artículos agrupados por familia, grupo, informe de compras, informe de despachos, artículos con stock critico, lista de precios, listado de proveedores, listado de artículos ordenados por inventario, informe de compras realizadas ordenadas por proveedor.
- ✓ Emitir una orden de compra automática para aquellos productos con stock mínimo.
- ✓ Comparar los niveles de compra y consumo de artículos por periodo.
- ✓ Emitir diferencias mensuales y acumuladas entre lo comprado, lo solicitado y lo entregado a despacho.
- ✓ Llevar una hoja de vida por proveedor, indicando hitos relevantes que permitan evaluar su comportamiento, para ello se registrará al menos, artículos comprados, descuentos, monto compra, tiempo despacho, calidad producto, etc.
- ✓ Permitir rebaja electrónica de artículos mediante la utilización de código de barra.

La aplicación deberá contemplar un nivel de privilegios que jerárquico, es decir, se debe estructurar como "árbol" computacional, de esta forma siempre el usuario del nivel superior puede acceder a información del nivel inferior.

El sistema debe contemplar un mantenedor de tablas base, es decir, de todas aquellas tablas que son de la forma código, glosa.

El sistema debe contemplar un mantenedor de cuentas de usuarios, donde se puedan asignar los permisos a las diferentes opciones del sistema. Los privilegios son a nivel de opciones del sistema, pudiendo eventualmente ser agrupadas en categorías de usuarios, el oferente debe considerar que pueden haber tantos perfiles de usuarios como opciones tiene el sistema.

Con respecto al tiempo el proyecto debe durar a lo más 3 meses.
Los requerimientos técnicos se muestran en la tabla 11.9:

Software de contrucción	El sistema deberá estar construido en ASP, con elementos de javascript y utilizando HTML.
Base de Datos	MS Sql Server 6.5 u 7.0
Arquitectura	Tres capas, es decir, debe utilizar procedimientos almacenados
SO	NT 4.x, Window 2000

Tabla 11.9: Requerimientos Técnicos

Estimación y Cálculos
Puntos de Función

Para realizar el cálculo de los puntos de función (PF) para la realización del sistema, se identificarán cada una de las funciones específicas para cada uno de los sub-sistemas con el fin de identificar todos los elementos a considerar para realizar el cálculo de los PF, como son: entradas, salidas, consultas, interfaces y archivos o tablas de base de datos. En las Tablas 11.1 a la Tabla 11.14 detallan el cálculo de los PF de cada uno de los elementos a considerar separados por sub-sistemas.

SALIDAS			
	Cantidad	Complejidad (baja=4,media=5,alta=7)	PF
Sistema de Inventario			
Información artículos	1	5	5
Solicitud de compra	1	4	4
Información de despacho de artículos	1	5	5
Sistema de Compras			
Datos de (estado) compra	3	4	12
Datos de artículos comprados	2	5	10
Sistema de administración de usuarios			
Datos de perfiles de usuarios	1	4	4
Datos de usuarios	1	4	4
Sistema de Gestión de datos			
Informes	12	5	60
Informes específicos	4	7	28
		Total PF	132

Tabla 11.10: Salidas

ENTRADAS			
	Cantidad	Complejidad (baja=3,media=4,alta=6)	PF
Sistema de Inventario			
Ingreso de artículos	1	5	5
Eliminación de artículos	1	3	3
Modificación de datos artículos	1	4	4
Ingreso de despachos	1	5	5
Sistema de Compras			
Ingreso de datos de compra	1	5	5
Eliminación de datos de compra	1	3	3
Modificación de datos de compra	1	4	4
Ingreso de proveedores	1	5	5
Eliminación de proveedores	1	3	3
Modificación de proveedores	1	4	4
Ingreso de artículos-proveedor	1	5	5
Eliminación artículos-proveedor	1	3	3
Modificación artículos-proveedor	1	4	4
Sistema de administración de usuarios			
Ingreso de usuarios	1	5	5
Eliminación de usuarios	1	3	3
Modificación de datos usuarios	1	4	4
Ingreso de perfiles de usuario	1	5	5
Eliminación de perfiles usuarios	1	3	3
Modificación de perfiles de usuario	1	4	4
		Total PF	77

Tabla 11.11: Entradas

TABLAS BD o ARCHIVOS			
	Cantidad	Complejidad (baja=3,media=4,alta=6)	PF
Sistema de Inventario			
Registro de artículos	1	7	7
Registros de despachos	1	7	7
Sistema de Compras			
Registro de compras		7	7
Registro de proveedores	1	7	7
Sistema de administración de usuarios			
Registros de usuarios	1	7	7
Registros de perfiles de usuarios	1	7	7
		Total PF	132

Tabla 11.12: Tablas BD o archivos

CONSULTAS			
	Cantidad	Complejidad (baja=3,media=4,alta=6)	PF
Sistema de Inventario			
Consulta de artículos	1	3	3
Consulta de stock	1	3	3
Consulta de despachos	1	3	3
Sistema de Compras			
Consulta de compras	2	6	12
Consulta de proveedores	1	6	6
Sistema de Administración de usuarios			
Consulta perfiles	1	6	6
Consulta usuarios	1	6	6
Sistema de Gestión de datos			
Consulta de artículos por clasificación	3	4	12
Consulta de estado de compra	2	3	6
Consulta de precios	1	3	3
Consulta de proveedor	1	4	4
Consulta artículos stock crítico	1	3	3
Consultas niveles de compra	2	6	12
Consultas consumo de artículos	3	6	18
Consultas Comparativas	2	6	12
		Total PF	109

Tabla 11.13: Consultas

INTERFACES			
	Cantidad	Complejidad (baja=3,media=4,alta=6)	PF
Sistema de Inventario			
Registro de stock	1	5	5
Histórico de despachos	1	5	5
Sistema de Compras			
Registro de Compras	1	5	7
		Total PF	17

Tabla 11.14: Interfaces

Resumen del cálculo de punto de función	
	PF
Salidas	132
Entradas	77
Consultas	109
Tablas de BD o Archivos	42
Interfaces	17
PFNA	377

Tabla 11.15: Resumen del Cálculo de PF

Para calcular los puntos de función ajustados es necesario calcular un factor de ajuste, que en este caso es 40. Se tiene entonces los siguientes valores:

PFNA=377

FA=40

PF=395.85

Los puntos de función ajustados fueron calculados de la siguiente forma:

PF = PFNA*(0,65+0,01*FA) = 396

El lenguaje que se utilizará para el desarrollo es VBScript (Ver Anexo D) el que considera por cada punto de función 50 líneas de código, por lo tanto se puede calcular los KLDC (Kilo líneas de código) de la siguiente forma:

KLDC = 439 * 50/1000 = 19,8

En resumen:

Puntos de Función = 396
KLDC = 19,8

COCOMO 81

Para la estimación de COCOMO 81 se consideró el proyecto dentro de la clasificación como de carácter *semiacoplado*, ya que es de mediano tamaño y complejidad. Por lo tanto los valores para cada una de las variables que manejan tanto COCOMO Básico como COCOMO Intermedio quedan definidas con los siguientes valores:

Resumen del cálculo de punto de función	
	PF
Salidas	132
Entradas	77
Consultas	109
Tablas de BD o Archivos	42
Interfaces	17
PFNA	377

Tabla 11.15: Resumen Cálculo PF

Cocomo Básico

COCOMO Básico nos permite calcular el esfuerzo en Persona Mes (PM), el tiempo de desarrollo y el personal requerido para el proyecto.

$$\text{Esfuerzo} = A*KLDC^{B} = 3* 19{,}8^{\,1{,}12} = 84{,}99 \text{ [PM]}$$

$$\text{Tiempo de desarrollo} = C*KLDC^{D} = 2{,}5*19{,}8^{0{,}35} = 7{,}1 \text{ [Meses]}$$

$$\text{Personal requerido} = \text{Esfuerzo/Tiempo de desarrollo}$$

$$\text{Personal requerido} = 84{,}99 / 7{,}1 = 11 \text{ [Personas]}$$

En resumen:

Esfuerzo = 84.99 [PM]

Tiempo de Desarrollo = 7.1 [meses]

Personal Requerido = 11 [Personas]

Cocomo Intermedio

COCOMO Intermedio nos permite calcular de otra manera el esfuerzo necesario para realizar el desarrollo para lo cual considera un factor de ajuste (FA) con un valor de 1,107393731 que se calcula a partir de un conjunto de características del sistema.

$$\text{Esfuerzo} = FA * A * KLDC^{B} = 94{,}12 \text{ [PM]}$$

Esfuerzo = 94,12 [PM]

COCOMO II

COCOMO II al igual que COCOMO 81 nos permite estimar el esfuerzo en Hombres Mes, el tiempo de desarrollo y el personal requerido para el proyecto. Pero la diferencia radica en que COCOMO II considera una variedad más amplia de características para determinar el factor de ajuste (FA), además de considerar líneas lógicas de código[6].

Esfuerzo = 2,94 * EAF * $KLDC^{E}$, donde E y EAF son factores que dependen de una variedad de características del sistema, en este caso se estiman en 0,9983 y en 1,043 respectivamente

Por lo tanto:

[6]En este caso las líneas lógicas de código son equivalentes a las líneas de código consideradas en COCOMO 81.

$$\text{Esfuerzo} = 2{,}94 * 1{,}043 * 19{,}8^{\,0{,}9983} = 60{,}42 \text{ [PM]}$$

Tiempo de desarrollo = $3{,}67 * \text{Esfuerzo}^{\,0{,}28+0.002*\Sigma Fi}$, donde los Fi son la suma de los factores que sirven para calcular E.

$$\text{Tiempo de desarrollo} = 3{,}67 * 60{,}42^{\,0{,}28+0.002*8{,}83} = 12{,}44 \text{ [Meses]}$$

Personal requerido = Esfuerzo / Tiempo desarrollo

Personal requerido = 60,42 / 12,44 = 4 [Personas]

En resumen:

Esfuerzo = 60,42 [HM]
Tiempo de Desarrollo = 12,44 [Meses]
Personal Requerido = 4 [Personas]

MACOS

Este modelo nos permite estimar también el esfuerzo requerido para la realización del desarrollo en el se consideran para el ajuste (FA) 32 factores clasificados en cuantificables y difíciles de cuantificar, además considera los puntos de función no ajustados (PFNA) para realizar al cálculo. Para este proyecto el FA es de 1,15.

$$\text{Esfuerzo} = 1{,}04 * (\text{PFNA})^{1{,}22} * \text{FA [HH]} = 1{,}04 * 377^{\,1{,}22} * 1{,}15$$
Esfuerzo = 1662,91 [Horas Hombre]

Tiempo de desarrollo = Esfuerzo / NPTC, donde NPTC es el número de personal de tiempo completo. Para el tiempo de desarrollo se considerará una cantidad variable de personas entre 5 y 15 personas.

NTPC	Tiempo de Desarrollo [Horas]	Tiempo de Desarrollo [Meses]
5	332.58	2.08
6	277.15	1.73
7	237.55	1.48
8	207.86	1.29
9	184.76	1.15
10	166.29	1.04
11	151.17	0.94
12	138.58	0.86
13	127.92	0.79
14	118.77	0.74
15	110.86	0.69

Tabla 11.16: Tiempo de Desarrollo para MACOS

Finalmente, con algún criterio y la experiencia de quien esta estimando, se establece el valor de esfuerzo, plazo y costo a utilizar.

Anexo A

A. Características de Proyectos Cliente/ Servidor

Para introducir el concepto de proyectos basados en la plataforma Cliente/Servidor, se analizan los aspectos teóricos sobre el ambiente Cliente/Servidor.

Se ha modificado substancialmente la forma de operar de las organizaciones, y esto, ha inducido modificaciones en el que hacer de la tecnología computacional dentro de las empresas. Algunos de los aspectos son los siguientes:

- ✓ Las aplicaciones deben ser desarrolladas en un tiempo corto, puesto que los requerimientos del negocio cambian rápidamente y las primeras deben adaptarse a ellos. Se espera, por ejemplo, que una aplicación debe ser desarrollada en un tiempo no mayor a seis meses.
- ✓ Se hace énfasis en la importancia de contar con una buena información, destacan por ello los sistemas de información ejecutivos y los sistemas de soporte a las decisiones.
- ✓ Cada vez es más importante hacer que la información esté disponible donde y en el instante en que se necesite. Para lograr esto, tanto la información como los sistemas para procesarla deben ser distribuidos a una amplia audiencia.
- ✓ A medida que crece la competencia, las organizaciones tienen cada vez menos recursos disponibles para los proyectos internos, incluyendo los sistemas de información. Por esta razón las nuevas aplicaciones deben basarse en tecnologías que disminuyan los costos de desarrollo y mantenimiento, en aspectos relacionados con el hardware, el software, la operación, el entrenamiento, y el personal. Además se requiere que las nuevas aplicaciones se puedan comunicar con las ya existentes.
- ✓ Con el fin de aumentar la productividad y de facilitar el uso de las aplicaciones por parte de los usuarios, se requieren interfaces simples e intuitivas que proporcionen un acceso transparente a la información.
- ✓ Las aplicaciones deben adaptarse al ritmo vertiginoso del desarrollo de la tecnología, para que puedan aprovechar sus potencialidades.

A.1 Modelos Cliente/Servidor.

En la actualidad este modelo de desarrollo ha tenido una importante evolución, sobre todo con la acelerada introducción de la tecnología Internet a las organizaciones. Por esto ahora existen diferentes modelos para cliente/servidor los que se pueden clasificar según su número de capas:

- ✓ Modelo de dos capas (Modelo básico).
- ✓ Modelo de tres capas.
- ✓ Modelo de múltiples capas (n-capas).

El término *Cliente/Servidor* se utiliza frecuentemente como sinónimo de Proceso Cooperativo o Proceso Distribuido, es decir, distribución de aplicaciones y/o datos en una red de ordenadores [a].

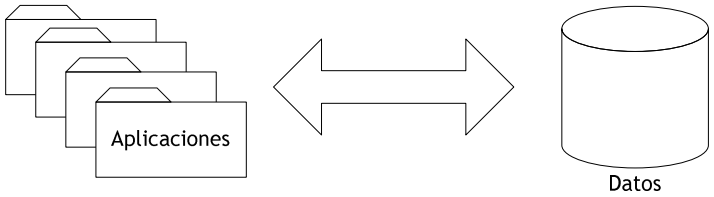


Figura A-1: Esquema Modelo cliente/servidor dos capas

Modelo C/S de dos capas o Modelo C/S Básico

En el esquema Cliente/Servidor básico existen dos unidades básicas: los procesos y los datos, ver figura A-1. Un proceso es una sola tarea que ejecuta alguna parte de una aplicación, como manejo de pantallas, acceso a la Base de Datos, cálculos, etc. Los procesos se comunican entre sí mediante los IPC (Comunicación entre procesos) y los RPC (Llamadas a procedimientos remotos). De esta manera, máquinas de diferentes tipos se encargan de funciones específicas. Lo que se busca con esta inter-operabilidad es proveer al usuario final con la metáfora de la imagen única del sistema, lo que se supone que todo el hardware y el software residente en un sólo procesador –o distribuido en varios físicamente separados- actúan a la vista del usuario final como si fuera una única computadora bajo un único sistema operativo.

En cuanto a los datos, éstos también pueden encontrarse juntos o distribuidos a través de la red. Generalmente son almacenados en bases de datos relacionales Cliente/Servidor, conocidas como motor de datos ya que proveen un alto grado de seguridad y Confiabilidad. Estos datos podrán ser accesados concurrentemente por todos los usuarios del sistema [b].

En este sentido, no es nada nuevo. Los bancos, por ejemplo, comenzaron a distribuir aplicaciones a principios de los años 70; la necesidad de una constante disponibilidad de información en el ámbito de sucursales, para atender a los clientes independientemente de la disponibilidad de la red y del ordenador central, llevó a la incorporación de aplicaciones donde las funciones y los datos se distribuían entre el sistema central y los procesadores inteligentes instalados en las sucursales.

El esquema *Cliente/Servidor* básico "es un modelo de computación en el cual el procesamiento requerido para ejecutar una aplicación o conjunto de aplicaciones relacionadas, se divide entre dos o más procesos que cooperan entre sí".

Usualmente la mayoría del trabajo pesado se hace en el proceso llamado servidor y él (los) proceso(s) cliente(s) sólo se ocupa de la interacción con el usuario (aunque esto puede variar).

Los principales componentes del esquema Cliente/Servidor son entonces los *Clientes*, los *Servidores* y la *infraestructura de comunicaciones*.

Los *Clientes* interactúan con el usuario, usualmente en forma gráfica. Frecuentemente se comunican con procesos auxiliares que se encargan de establecer conexión con el servidor, enviar el pedido, recibir la respuesta, manejar las fallas y realizar actividades de sincronización y de seguridad.

Los *Servidores* proporcionan un servicio al cliente y devuelven los resultados. En algunos casos existen procesos auxiliares que se encargan de recibir las solicitudes del cliente, verificar la protección, activar un proceso servidor para satisfacer el pedido, recibir su respuesta y enviarla al cliente. Además debe manejar los ínter bloqueos, la recuperación ante fallas, y otros aspectos afines. Por las razones anteriores, la plataforma computacional asociada con los servidores es más poderosa que la de los clientes. Por esta razón se utilizan PC poderosos, estaciones de trabajo, mini-computadores o sistemas grandes. También deben manejar servicios como administración de la red, mensajes, control y administración de la entrada al sistema, auditoria, recuperación y contabilidad. Generalmente en los servidores existe algún tipo de servicio de bases de datos.

Para que los clientes y los servidores puedan comunicarse se requiere una *infraestructura de comunicaciones,* la cual proporciona los mecanismos básicos de direccionamiento y transporte.

La mayoría de los sistemas Cliente/Servidor actuales se basan en redes locales y por lo tanto utilizan protocolos no orientados a conexión, lo cual implica que las aplicaciones deben hacer las verificaciones. La red debe tener características adecuadas de desempeño, confiabilidad, transparencia y administración.

Como *ejemplos de clientes* están las interfaces de usuario para enviar comandos a un servidor, APIs para el desarrollo de aplicaciones distribuidas, herramientas para que el cliente pueda tener acceso a servidores remotos (por ejemplo servidores de SQL) o aplicaciones que solicitan acceso a servidores para algunos servicios.

Algunos *ejemplos de servidores* pueden ser servidores de ventanas como X - Windows, servidores de archivos como NFS, servidores para el manejo de bases de datos, los servidores de SQL, servidores de diseño y manufactura asistidos por computador, entre otros.

Modelo C/S de tres Capas

Con el avance de la tecnología Internet han cambiado las formas de implementar las aplicaciones cliente/servidor, y con ello la arquitectura que los sustenta.

Esta arquitectura corresponde a una extensión del modelo básico cliente/servidor, la idea de contar con este modelo es separar de alguna forma los conceptos involucrados en las diferentes capas de forma tal de facilitar el entendimiento y el desarrollo de aplicaciones web. Este modelo considera una capa adicional intermedia entre la capa de aplicaciones de cliente y la capa de datos. A esta capa se le llama lógica de aplicación, la cual independiza de funcionalidad a las aplicaciones clientes, dejándolas simplemente con la interfaz. De esta forma el modelo cliente servidor de tres capas queda formado por la capa de: presentación (interfaz), lógica de aplicación y capa de datos. La figura A-2 muestra de forma esquemática esta nueva arquitectura. [a]

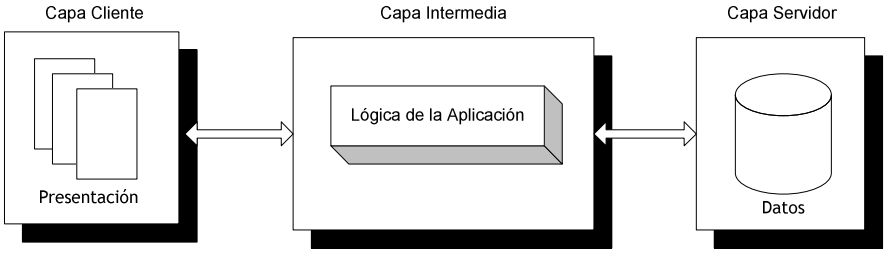


Figura A-2: Modelo Cliente/Servidor 3 capas

En este esquema se considera como capa de *Presentación*, corresponde con todo lo que el cliente interactúa, se utiliza código HTML, Java Script, Flash, entre otros. La capa de *Lógica de aplicación* es la que permite realizar la comunicación entre los datos y la presentación de los mismos, para ello se necesita de lenguajes que permitan realizar esta operación de forma transparente para el usuario como es el caso de PHP y ASP. Finalmente la capa de *Datos* es la que almacena y administra todos los datos del negocio, para ello se debe contar con poderosos servidores de bases de datos como SQL Server, Oracle, Sybase, entre otros.

Modelo cliente/servidor N-capas

Hoy en día las aplicaciones que se desarrollan en la web poseen un fuerte componente de multimedia, por lo cual se requiere de una capa que soporte y permita la implementación de toda esta tecnología. Es por esto que en la actualidad se ha trabajado en el desarrollo de diferentes plataformas que faciliten la creación de aplicaciones con este tipo de características, por ejemplo J2EE y .NET. Ambas, para el desarrollo de aplicaciones, proponen como modelo cliente/servidor de N-capas.

El modelo cliente/servidor de N-capas es una generalización de las anteriores, y la diferencia radica principalmente en la capa de *Lógica de aplicación*. Esta capa es la que concentra todas las nuevas capas que día a día se proponen, por lo tanto esta capa se transforma en un contenedor de nuevas capas, donde cada una de ellas define una nueva función específica para el sistema.

Un ejemplo de este modelo es el que utiliza J2EE que considera 4 capas, la figura A-3 muestra este modelo de forma esquemática. [b]

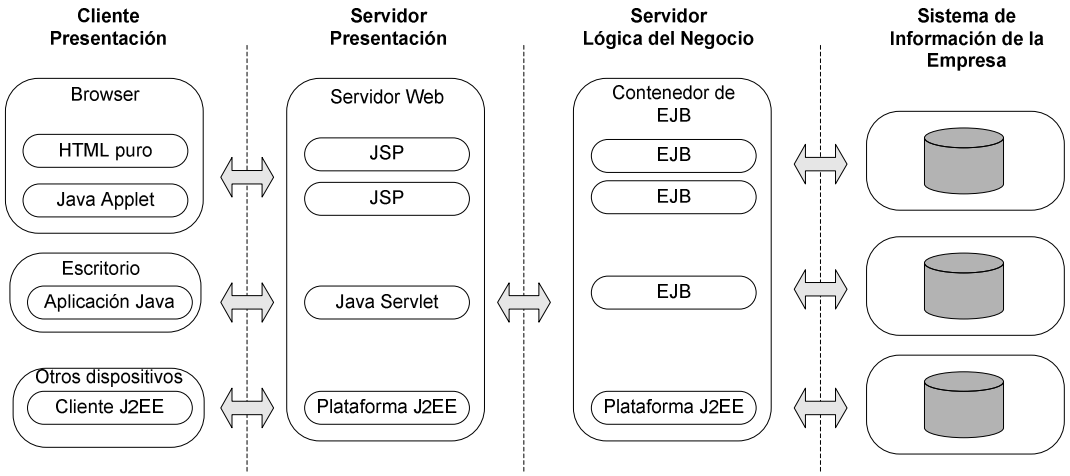


Figura A-3: Arquitectura de 4 capas J2EE.

A.2 Ventajas y desventajas del esquema Cliente/Servidor

Este esquema provee numerosas ventajas, alguna de ellas son:

- ✓ Uno de los aspectos que más ha promovido el uso de sistemas Cliente/Servidor es la existencia de plataformas de hardware cada vez más baratas. Esta es la ventaja más clara de este esquema, utilizar máquinas más baratas que las requeridas por una solución centralizada, basada en sistemas grandes. Se pueden utilizar componentes, tanto de hardware como de software, de varios fabricantes, lo que contribuye a la reducción de costos y favorece la flexibilidad en la implantación y actualización de soluciones.
- ✓ El esquema Cliente/Servidor facilita integrar sistemas diferentes y compartir información, permitiendo, por ejemplo que las máquinas existentes puedan ser manejadas utilizando interfaces más amigables al usuario. Por ejemplo, integrar computadores con sistemas medianos y grandes, sin que todas las máquinas tengan que utilizar el mismo sistema operativo. Favorece el uso de interfaces gráficas interactivas; los sistemas construidos con este esquema tienen una interacción más intuitiva con el usuario. Si se utilizan interfaces gráficas para interactuar con el usuario, el esquema Cliente/Servidor presenta una ventaja con respecto a uno centralizado, que no es siempre necesario transmitir información gráfica por la red pues ésta puede residir en el cliente, lo que permite aprovechar mejor el ancho de la banda de la red.

- ✓ Rápido mantenimiento y desarrollo de aplicaciones, pues se pueden emplear las herramientas existentes (por ejemplo los servidores de SQL o las herramientas de más bajo nivel como los sockets o el RPC). La estructura modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones. El esquema Cliente/Servidor contribuye a proporcionar a los diferentes departamentos de una empresa soluciones locales, y permitiendo la integración de la información relevante en el ámbito global.

Este esquema también posee *desventajas,* alguna de estas son:

- ✓ El mantenimiento de los sistemas es más difícil pues implica la interacción de diferentes partes de hardware y software, distribuidas por distintos proveedores, lo que dificulta el diagnóstico de fallas.
- ✓ Herramientas para la administración y ajuste del desempeño de los sistemas son complejos.

A.3 Ventajas del esquema Cliente/Servidor para la Empresa

Con el uso de este esquema se reducen los costos de producción de software y se disminuyen los tiempos. Esto es así puesto que para la construcción de una nueva aplicación pueden usarse los servidores que haya disponibles, reduciéndose el desarrollo a la elaboración de los procesos del cliente, según los requerimientos deseados. Lo anterior disminuye los costos internos del área de sistemas. Además, se pueden obtener ventajas importantes al reducir el costo del hardware requerido, llevando las aplicaciones a plataformas más baratas, aprovechando el poder de cómputo de los diferentes elementos de la red, y facilitando la interacción entre las distintas aplicaciones de la empresa.

El esquema Cliente/Servidor también contribuye a una disminución de los costos de entrenamiento del personal, pues favorecen la construcción de interfaces gráficas interactivas, las cuales son más intuitivas y fáciles de usar por el usuario final.

Otra de las ventajas es que facilita el suministro de información a los usuarios. Lo que por un lado proporciona una mayor consistencia a la información de la empresa, al contar con un control centralizado de los elementos compartidos, y por otro, facilita la construcción de interfaces gráficas interactivas, las cuales pueden hacer que los "datos" se conviertan en "información".

El esquema *Cliente/Servidor* permite llevar más fácilmente la información a donde se necesita, además que contribuye a aumentar su precisión ya que se puede obtener de su fuente (el servidor) y no de una copia en papel o en medio magnético.

La habilidad de integrar sistemas heterogéneos es inherente al modelo Cliente/Servidor, puesto que los clientes y los servidores pueden existir en múltiples plataformas y tener acceso a datos de cualquier lugar de la red. Además un cliente puede integrar datos de diferentes lugares para presentarlos a su manera, al usuario final. Al favorecer la construcción de interfaces gráficas interactivas y el acceso transparente a diferentes modos de la red, se facilita el uso de las aplicaciones por parte de los usuarios, lo cual aumenta su productividad.

El esquema *Cliente/Servidor* también favorece la adaptación a cambios en la tecnología, pues facilita la migración de las aplicaciones a otras plataformas y, al aislar claramente las diferentes funciones de una aplicación, se hace más fácil incorporar nuevas tecnologías en ésta.

Con el establecimiento de estándares aparecieron los *sistemas abiertos*. Este es un medio en el cual se pueden intercambiar componentes de software y hardware, dando al usuario mayor posibilidad de escoger productos de acuerdo a sus necesidades y fomentando la competencia entre proveedores, los cuales deben mejorar sus servicios para ganar clientes.

Un sistema abierto cuenta con las siguientes propiedades:

- ✓ *Interoperabilidad:* Son Componentes de múltiples proveedores que pueden intercambiar información, por medio de interfaces bien definidas, reduciendo el costo de interconexión e integración.
- ✓ *Portabilidad:* Permite a un sistema instalado en un medio, ser instalado en otro, minimizando el costo de la migración.
- ✓ *Integración:* Permite compartir e intercambiar información, mostrando consistencia de comportamiento y presentación.

Los sistemas abiertos son la plataforma adecuada para desarrollo de aplicaciones distribuidas, ya que se pueden combinar las ventajas de diferentes máquinas y sistemas operacionales. Para implementar el intercambio de información el modelo de comunicación más popular es el modelo Cliente/Servidor, el cual permite que el usuario llame servicios de forma transparente.

Referencias Anexo A

[a] Revista Profesional para Programadores-RPP, Pedro Agulló Soliveres. “Desarrollo cliente/servidor: Ubicación de las reglas del negocio”.
[b] Java 2 Plataform, Enterprise Edition (J2EE).

Anexo B

B Modelos del ciclo de vida.

Los diferentes modelos del ciclo de vida, al reflejar diferentes filosofías para crear el producto de Software, incorporan diferentes tipos de procesos y productos.

Los métodos, herramientas y procedimientos que, aplicados correctamente, conducen a la construcción de un producto de software con una perspectiva de ingeniería.

B.1 Ciclo de Vida Tradicional o Modelo en Cascada [Royce, 1970]

Se basa en la idea de producir secuencialmente tipos de productos en diferentes niveles de abstracción (seguido por la integración del código en orden inverso). Seguir este modelo en un proyecto significa transformar linealmente, el conjunto completo de requerimientos a un producto cada vez más concreto.

Este modelo es el más antiguo y más utilizado. Para ocupar este modelo es necesario tener en cuenta que los proyectos no siempre siguen el flujo secuencial, hay iteraciones, es difícil para el usuario el establecimiento explícito de todos los requerimientos, el usuario debe ser "paciente". Sin embargo, es útil como punto de partida.

En la Figura B.1 se comparan las etapas del Ciclo de Vida Tradicional con el Ciclo de Vida Estándar.

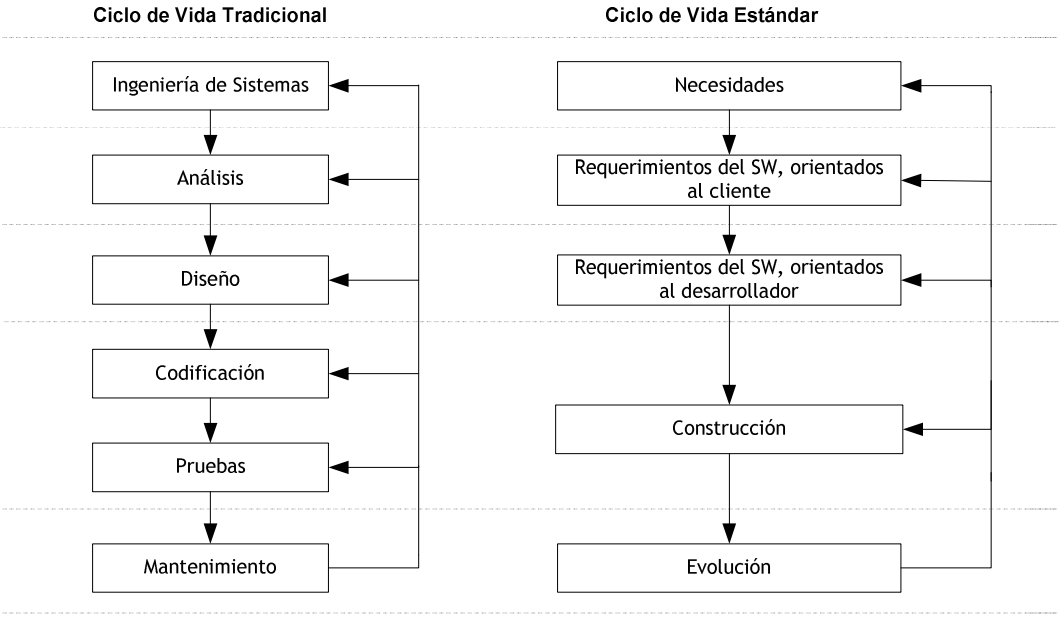


Figura B.1: El ciclo de vida Tradicional vs Ciclo de Vida Estándar

B.2 Modelo de crecimiento iterativo o incremental [Basili1975].

Se basa en la idea de producir los mismos tipos de productos que en el modelo tradicional para algún subconjunto de requerimientos, ver figura B.2. La idea es permitir un mejor aprendizaje y retroalimentación a partir de cada uno de estos miniproyectos de desarrollo.

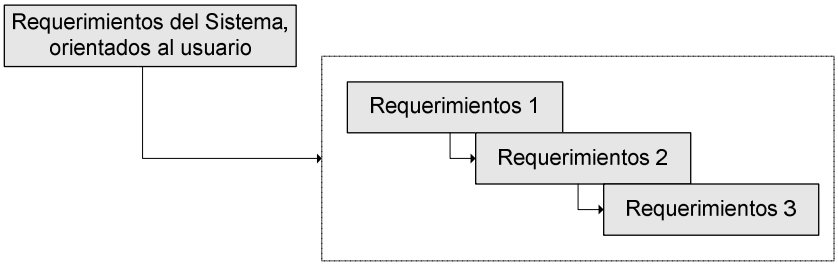


Figura B.2: Requerimientos en Modelo Iterativo

B.3 Construcción de Prototipos [Boehm 1984]

Este paradigma se basa primero en la producción de una versión operacional del software, para un conjunto limitado de requerimientos, excluyéndose parte de la funcionalidad. A menudo las razones para el prototipo son ciertos aspectos cruciales de la interfaz hombre máquina o algunas características de alto rendimiento. Se utiliza para determinar un conjunto aceptable de requerimientos del producto, explorar factibilidad técnica o determinar riesgo asociado. Es una forma rápida de aprender acerca de aspectos claves del proyecto. El objetivo es reutilizar la experiencia ganada durante el proceso de construir el prototipo.

El profesional crea un modelo del software a construir (carcaza). El usuario evalúa un diseño reducido (lo visible para él) y se itera. Idealmente, el prototipo sirve para identificar requerimientos. El prototipo se elimina después de su uso y se construye el sistema desde cero, o el prototipo se refina necesariamente hasta obtener un producto utilizable.

- ✓ *Desechable:* en el cual el prototipo es desechado y no usado en el producto entregado.
- ✓ *Evolucionario:* una parte o el total del prototipo es retenido.

Estudios demostraron que de 39 casos, los desarrolladores que utilizaron el prototipamiento obtuvieron un éxito en 33 de ellos, de los cuales 22 usaron el prototipo Evolucionario, mientras que en 8 casos el desechable (IEEE Software, Enero 1995).

Algunas desventajas son:

- ✓ Debido a que los usuarios se enfrentan fuertemente con la funcionalidad, algunos pueden cometer el error de entusiasmarse

adicionando funcionalidad intranscendente bajo presiones de uso normal.

- ✓ Posibilidad de creer en que el prototipamiento es en sí una especificación y por ende base para el diseño, cuando éste es sólo parte de él.
- ✓ El prototipamiento puede demostrar funcionalidad que no es posible bajo apremios reales y este problema puede no ser descubierto hasta que la fase de prototipo sea finalizada.
- ✓ Puede dar la impresión al usuario de que tiene frente a sí el sistema completo y operable, y no un esqueleto de éste.
- ✓ Existe la posibilidad de subestimar el proyecto debido a que las salidas están rápidamente disponibles.

Algunas ventajas son:

- ✓ Los usuarios tienen la posibilidad de interactuar con el prototipo y realimentar a los desarrolladores. Muchos usuarios no saben que necesitan ciertas funcionalidades hasta que le son expuestas.
- ✓ Se produce una mayor identificación, debido que para los usuarios es más confortable enfrentarse a un prototipo que a una especificación.
- ✓ Decrece el esfuerzo de desarrollo entre un 30% y un 60%. Un diseño rápido es posible cuando los requerimientos son claros.

B.4 Modelo espiral [Boehm 86].

Este enfoque es evolutivo y se basa en un acercamiento orientado a la determinación del riesgo en el desarrollo del Software. Los ciclos de desarrollo iterativo se organizan en forma de espiral, donde los ciclos interiores representan análisis temprano y construcción de prototipo y los ciclos exteriores representan el ciclo de vida clásico. Esta técnica se combina con análisis de riesgo durante cada ciclo.

Este modelo une los dos modelos anteriores, e incorpora *Análisis de Riesgos*

Tiene 4 actividades principales:

- ✓ planificación → objetivos, alternativa, restricciones
- ✓ análisis de riesgo → análisis de alternativas y riesgos
- ✓ ingeniería → desarrollo de producto “próximo”
- ✓ evaluación del cliente

El modelo se acerca paulatinamente a un sistema completo. Es adecuado para proyectos de gran escala.
Algunos problemas son:

- ✓ Criticidad del análisis de riesgo.
- ✓ Difícil de “vender” como algo controlable.

Algunas ventajas son:

- ✓ Demanda una consideración directa de los riesgos técnicos en todas las etapas del proyecto y, si se aplica adecuadamente, debe reducir los riesgos antes que se conviertan en problemas.
- ✓ Permite la utilización de la creación de prototipos como un mecanismo de reducción del riesgo. Pero aún más importante, permite a quién lo desarrolla utilizar el enfoque de creación de prototipos en cualquier etapa de evolución del producto.

Anexo C

C Datos de apoyo para los cálculos

Estimación del número de LDC para diferentes lenguajes de Programación.

La tabla D-1 proporciona estimaciones informales del número medio de líneas de código requerido para construir un punto de función[7] en los lenguajes que se utilizan en los casos.

LENGUAJE	NIVEL	LDC/PF
Java	6	53
VBScript	-	50

Tabla D-1: Lenguaje vs LDF/PF

Remuneraciones del Mercado Laboral Informático[8]

La Tabla D-2 muestra los valores del Mercado Laboral Informático.
(Actualizado MAYO 2002) [9]-[10]-[11]

[7] Para tablas mas completas revisar http://www.davidconsultinggroup.com/indata.htm o http://www.obarros.cl/diredecambio.html
[8] http://www.comunidadinformatica.com/chile/laboral.htm
[9] Valor UF: $ 17005,05 al 10 de Junio de 2003
[6] Considerando 48 horas de trabajo semanales

Profesional Informático	Promedio Mensual SUELDOS BRUTOS ($)	Promedio Mensual SUELDOS BRUTOS UF	Horas Hombre UF	Horas Hombre Empresa UF
Digitador	$127.500	7,50	0,04	0.1
Operador AS/400 Unix Windows NT	$387.500 $355.000 $280.000	22,79 20,88 16,47	0,12 0,11 0,08	0,3 0,28 0,2
Administrador AS/400 Unix Windows NT	$495.500 $470.500 $480.500	29,14 27,67 28,26	0,15 0,14 0,15	0,38 0,35 0,38
Administrador de Base de Datos Sql Server Oracle Sybase Informix	$720.500 $880.500 $815.000 $890.500	42,37 51,78 47,93 52,37	0,22 0,27 0,25 0,27	0,55 0,68 0,63 0,68
Programador Avanzado Visual Basic + SQLServer Oracle Developer Cobol AS/400 Cobol Cics Java HTML + ASP + SQLServer Power Builder Cliper Delphi	$720.500 $790.000 $720.500 $900.000 $700.500 $680.500 $620.000 $450.000 $615.000	42,37 46,57 42,37 52,93 41,19 40,02 36,46 26,47 36,17	0,22 0,24 0,22 0,28 0,21 0,21 0,19 0,14 0,18	0,55 0,6 0,55 0,7 0,53 0,53 0,48 0,35 0,45
Analista de Sistemas	$920.500	54,14	0,28	0,7
Jefes de Proyecto	$1.100.000	64,69	0,34	0,85
PROMEDIO DE LA INDUSTRIA	$600.020	35,28	0,18	0,45
Variación últimos 6 meses	- 9,3%	-	-	-

Tabla D-2: Valores del Mercado Laboral Informático

Referencias

[1] Java 2 Plataform, Enterprise Edition (J2EE).
http://java.sun.com/j2ee/overview2.html
[2] Revista Profesional para Programadores-RPP, Pedro Agulló Soliveres. “Desarrollo cliente/servidor: Ubicación de las reglas del negocio”.
http://www.ctv.es/USERS/pagullo/arti/csbr/csbr.htm
[3] "Estimación de Proyectos de Software para Desarrollo de Aplicaciones Intranet/Internet basada en la Técnica de Puntos de Función". Lautaro Guerra Genskowsky, Pamela Hermosilla Monckton, páginas 91 - 98, Proceedings 1st Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC'01), June 13-15, 2001, Buenos Aires, Argentina [Co-located with 13th International Conference on Software Engineering and Knowledge Engineering]. Editorial Universidad Nacional de Jujuy, San Salvador de Jujuy, Argentina 2001, ISBN 950-721-156-X
[4] OMG Unified Modeling Language Specification. Version 1.4 September 2001
[5] Mass-Updates and Software Project Management, Caper Jones 30/1/99.
http://www.spr.com
[6] Balzer, R., and N. Goodman, "Principles of Good Software Specification", Proc, on Specifications of Reliable Software, IEEE, 1979, pp.58-67
[7] Puntos por función. Una Métrica estándar para establecer el tamaño del software. Boletín de Política Informática. Núm 6, 2003.
[8] "Modelo de estimación para proyectos cliente / servidor basado en el método de punto de función". L. Guerra y C. Vildósola. Páginas 194-203 Actas del XI INFONOR'98 y VI Encuentro Chileno de Computación, Antofagasta, Chile, noviembre 1998.
[9] “COCOMO II”, Mayo 2003.
http://www.softstarsystems.com/cocomo2.htm
[10] "MACOS: Modelo Algorítmico de Costos de Software", Guerra L., Concha S. Actas XIV Conferencia Latinoamericana de Informática, Buenos Aires, Argentina, Septiembre 1988
[11] “La fórmula de Compaq para ganar la carrera” Diario del Navegante- Internet, Informática y Nuevos Medios. Noviembre 1997.
http://www.el-mundo.es/navegante/97/noviembre/16/compaq.html#arriba
[12] BEDINI, Alejandro. SPICE [en línea] [Guayaquil]: Universidad Santa María. Actualizada: Septiembre, 2003. <http://www.usm.edu.ec/~abedini/spice/spicess.htm> [Consulta: mayo 30, 2003].
[13]Controlling Software Development , Laurence H. Putnam,Ware Wyers, IEEE Computer Society Press, 1996.
[14]Licitación Pública, “Diseño y Construcción de un Sistema de Control de Gestión” para el Servicio Nacional de Aduanas de Chile. Junio 2001
[15]Licitación Pública, “Sistema de Información para las Compras y Abastecimientos de la Dirección del Trabajo”, Gobierno de Chile, Julio 2001.
[16] Living Values, J. Kirpalani, M. Panjabi, Ed. Gill-Kozul, 1999

Bibliografía

1. **El ambiente Cliente / Servidor: Una aplicación práctica.** Marcela Rojas Garate, Memoria U.T.F.S.M. 1995
2. **The Essential Client / Server Survival Guide.** Robert Orfali, Dan Harkey, Jeri Edwards. Second Edition, 1996
3. **IT METRICS STRATEGIES.** July 1996: VOLUMEN I, N° 6, 11
4. **INGENIERIA DEL SOFTWARE: Un Enfoque Práctico.** Roger S. Pressman, Tercera Edición, Editorial Mc Graw-Hill Inc., 1993
5. **Econometría.** Damodar N. Gujarati, Segunda Edición, Editorial Mc Graw-Hill Inc.,1992
6. **Diseño y Análisis de Experimentos.** Douglas C. Montgomery, Grupo Editorial Iberoamérica S.A. , 1991
7. **Administración de proyectos informáticos con apoyo de software.** Patricio Aguilera Reyes , Curso TECH Training, Santiago. , 1997
8. **El modelo Cliente/Servidor**, www.hidra.uniandes.edu.co/articulos/cliser.html
9. **Fundamentos Cliente/Servidor**, www.es.hosting.ibm.com/ncsc/cs/fundamcs.html
10. **Function Points Analysis and its Uses**, www.predicate.com/index.html/papers.html
11. **How are Function Points Useful**, www.softwaremetrics.com/download
12. **Estimating and Tracking Software Projects**, www.kallista.com/TechPapers/Track/TrackingProjects.html.
13. **A Manager's Guide to Software Engineering,** Roger S. Pressman, McGraw-Hill, Inc. 1993.
14. **The handbook of MIS application software testing, methods, techniques, and tools for assuring quality through testing,** Daniel J. Mosley. Yourdon Press, 1993
15. **Object Oriented Software Engineering,** Ivar Jacobson,. Addison - Wesley 1992.
16. **Software Management**, Donald J. Reifer. IEEE Computer Society Press, 1993.
17. **Managing The Software Process,** Watts S. Humphrey. Addison - Wesley, 1989
18. **Standards, Guidelines, and Examples on System and Software Requirements Engineering"**, Merlin Dorfman, Ricahard H. Thayer. IEEE Computer Society Press, 1990.
19. **Rapid Development**. McConnell, Steve. Ed. Microsoft Press, U.S.A, 1996.

20. **Total Quality Management for Software**, Gordon Schulmeyer, James Y. McManus. Van Wostrand Reintrold, 1992.
21. **The Handbook of MIS Application Software Testing,** Daniel J. Mosley. Yourdon Press, 1993.
22. **Client/Server Systems Management**, Peter D. Varhol. Computer Technology Research Corp. 1995.
23. **Managing Software Development Projects,** Neal Whitten. 2nd Edition John Wiley, 1995
24. **Software Project Management, Reading and Cases**, Chris F. Kemerer. IRWIN, 1997.
25. **Managing a Programming Project, Processes and People**, Philip Metzger, John Boddie. Prentice Hall, 1996
26. **Software Function, Source Lines of Code and Development Effort Prediction**. IEEE Transactions on Software Engineering. Vol. SE-9. Albrecht, A.1993
27. **Simulation and Comparison of Albreht's Function Points and De Marco's Function Metrics in a Case Enviroment**. IEEE Transactions on Software Engineering. Vol. 19. 1993
28. **Function Points Analysis and its Uses**, www.predicate.com/index.html/papers.html
29. **How are Function Points Useful**, www.softwaremetrics.com
30. **Auditing Function Points Counts**, www.softwaremetrics.com
31. **Forgotten Aspects of Function Points**, www.tbl.co.uk/fafpa.html
32. **Estimating and Tracking Software Projects**, www.kallista.com/TechPapers/Track/ TrackingProjects.html
33. **What are Function Points ?**, www.spr.com /library/funcmente.htm
34. **Software Assesment and Benchmarks**, Casper Jones. June 1995
35. **Conflict and Litigation between Clients and Developers**, Casper Jones, 1996
36. **Fifteen Principles of Software Engineering**, Alan Davis, IEEE Software. Enero 1995, pag 94-101.
37. **Power to the people : end users as developers**, Sorel Reisman. IEEE Software. Marzo 1992, pag.111-112
38. **Metrics Tools: Effort and Scheduling**. David Erickson. Software Engineering Technology. March 1995
39. **Breakthrough Technology Project Management,** B. P. Lientz, K.P. Rea, Academic Press, 1999
40. **Project Management**, P. C. Tinnirello (editor), Auerbach, 2000
41. **Software Project Management: A Unified Approach**, W. Royce, Addison Wesley, 1998
42. **The Rational Unified Process: An Introduction**, P. Krutchen, Addison Wesley, 1999
43. **Desarrollo y gestión de proyectos Informáticos**, Steve McConnell, Microsoft Press, 1997

44. **"Process Improvement: Practical Guidelines for Business Success,** S. Zahran, Addison-Wesley, 1998.
45. **Seven steps for highly effective project management**, N. B. Mingus
46. **Software Requirements**, Kart E. Wiegers, Microsoft Press, 2003

Referencias Web

- ✓ Atributos externos del software
http://www.sc.ehu.es/jiwdocoj/mmis/externas.htm
- ✓ Complejidad de software
http://www.geocities.com/txmetsb/compleji.htm
- ✓ Estimación de Costos
www.itcdguzman.edu.mx/ingsoft/estimac.htm
- ✓ Planificación de Proyectos de Software
http://www.getec.etsit.upm.es/articulos/gproyectos/art4.htm
- ✓ "Estimating Software Projects" ,Bill Meacham
http://ourworld.compuserve.com/homepages/bmeacham/ProfWriting.htm
- ✓ "Rediseño del desarrollo de Software"
http://www.obarros.cl/diredecambio.html