UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DOCUMENTO DE ESPECIFICACION DE REQUISITOS DE UNA BASE DE DATOS HISTORICA PARA PROYECTOS DE SOFTWARE.

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

XOCHITL CRISTINA VILLA AGUIRRE

DIRECTOR DE TESIS: M.C. REYNALDO ALANIS CANTU.

CIUDAD UNIVERSITARIA, MEXICO, D.F.
JUNIO 2008.

Agradecimientos

Gracias a todas aquellas personas que siempre estuvieron allí esperando el momento preciso para ayudarme a subir el escalón siguiente: Papá, Mamá, Ely, Toño, José Luis, mis profesores…

Gracias a todas aquellas personas que creyeron en mí y nunca me permitieron que yo no lo creyerá, a ti Chris.

"La conclusión es que sabemos muy poco y sin embargo es asombroso lo mucho que conocemos. Y más asombroso todavía que un conocimiento tan pequeño pueda dar tanto poder."

Bertrand Russell

Cuando veas una pequeña luz brillar, ¡síguela!
Si te dirige al pantano, pues ya saldrás de él.

Pero si no la sigues,
Toda tu vida vivirás arrepentido
Al no saber si ésa era tu estrella.

Séneca

Si quieres tener éxito, duplica tu porcentaje de fracasos.

Tom Watson, fundador de IBM

"Tras el vivir y el soñar, está lo que más importa: el despertar"

Antonio Machado

ÍNDICE

INDICE DE TABLAS

INDICE DE FIGURAS

ÍNDICE DE ECUACIONES

INTRODUCCIÓN

La transformación de una economía nacional o regional a una economía global, ha permitido una dramática innovación en tecnología y ha creado grandes presiones en mejora de la calidad y contención de precios (Gray, 2003). Por ello, se hace necesario contar con las herramientas que permitan la reducción del tiempo en la entrega de productos o servicios, disminución de costos, mejora en los niveles de servicio, así como el intercambio rápido y eficiente de información, tanto para el interior de la empresa como hacia el exterior, para que las organizaciones puedan responder a las necesidades del mercado.

Se dice que el software ha llegado ser una herramienta crítica para satisfacer estas necesidades, ya que cualquier ámbito del ser humano (empresas, gobierno, vida cotidiana, etc.) está inmerso en los programas de computadora. Esto se debe a que el software ha logrado, al automatizar procesos, acortar tiempo y distancia en la obtención de los bienes de insumo.

Las propuestas académicas y de la industria para mejorar los procesos han favorecido la evolución del software hacia la aparición de una nueva disciplina: *la ingeniería de software* (Borges de Barros, 2002).

Todo proceso de administración de proyectos conlleva un proceso de evaluación. Al decir evaluación surgen vocablos ligados a él, tales como: apreciar, estimar, atribuir valor o juzgar. Al administrador de proyectos de software le interesa evaluar, tanto la información de proyectos previos, como de los proyectos actuales. Es así que se hace necesario diseñar instrumentos adecuados para dicho fin, tal como un producto de software que almacene datos sobre las prácticas de administración de proyectos e ingeniería de software, para su uso posterior en un nuevo proyecto.

Para crear este producto de software es necesario conocer que es tanto un sistema de información y un software de computadora, así como cuales son las herramientas que brinda la ingeniería del software, tales como: el ciclo de vida del software y los procesos de desarrollo. Cabe mencionar que las prácticas de administración de proyectos de desarrollo de software como el ciclo de vida del proyecto de software, sus elementos, su administración y las características de los programas de computación para la administración de proyectos existentes, también son importantes. Asimismo las métricas del software de computadora brindan la oportunidad de conocer cómo se comporta el desarrollo del software y el software mismo, mediante medidas.

Es por ello que la presente investigación está orientada a proponer un documento de especificación de requisitos de una base de datos histórica de proyectos de software con un modelo de ciclo de vida generalizado, en el cual se recopila la información suficiente para establecer este entregable clave en el desarrollo del software.

ABSTRACT

The transformation of a national or regional economy to a global one has allowed a dramatic innovation in technology and has created big pressures in the improvement of quality and containment of prices (Gray, 2003). It becomes necessary to rely on the tools that should allow time reduction in products or services delivery, costs decreases, improvement of service levels, as well as the rapid and efficient exchange of information, for both the interior and the exterior of the company so that the organizations could answer the needs of the market.

The software has become to be a critical tool to satisfy these needs, since any area of the human being (companies, government, daily life, etc.) are immersed in the computer programs. This is due to the fact that the software has achieved, on having automated processes, shortened time and distance in obtaining the goods of input.

The academic and industry offers to improve the processes have favored the evolution of software towards the appearance of a new discipline: *the software engineering* (Borges, 2002).

All project administration process carries an evaluation procedure. When we said evaluation there arise words tied with it such as to estimate, to give value or to judge. The project manager of software is interested in evaluating so much the information of previous projects as the current ones. Thus it becomes necessary to develop suitable tools for this purpose, as a software product that store data on the practices of project management and engineering software, for subsequent use in a new project.

To create this software product is necessary to know who both information system and computer software is; besides what are the tools which provide engineering software, such as the life cycle of software and development processes. It is worth mentioning that the management practices of software development projects as the life cycle of the software project, its elements, its administration and the characteristics of software for managing existing projects, are also important. Also the metric computer software provides an opportunity to learn how it behaves software development and the software itself, through measures.

That is why the current investigation is aimed at proposing a document specifying requirements of a historical database of software projects with a life cycle model widespread, which collects information sufficient to establish this key deliverable software development.

The present research is orientated to proposing a document of requirements of a tool of summary of historical information of software projects, which allows knowing the design of a Database on the practices of project administration and engineering software, for its posterior use in a new project.

1. ANTECEDENTES

1.1 LA INDUSTRIA DEL SOFTWARE

Son tres los tipos de organizaciones que se dedican al desarrollo del software de computadora:

- Fabricantes de software.
- Empresas de servicios.
- Empresas hibridas.

Las características de dichas organizaciones se muestran en la tabla 1.1.

Tabla 1.1. Características de los tres tipos de empresas de software.

Fabricantes de software	Empresas de servicios	Hibridas
◆ Maneja altos costos fijos. Sus costos marginales son pequeños. ◆ 75% de sus ingresos los obtiene por la venta de licencias. ◆ Desarrolla pocos productos y muy estandarizados. ◆ Posee márgenes muy altos: 4/20 personas más ricas del mundo. ◆ Tiene alta concentración de empresas en los Estados Unidos de Norteamérica	◆ Tiene la estructura típica de una empresa de servicios (personal es clave). ◆ Basa su negocio en formación, soporte, y proyectos integrales.	◆ Combinación de los modelos anteriores

Fuente: Masi, 2006.

1.1.1 Estados Unidos de Norteamérica

La industria del software obtiene grandes ingresos, el software empaquetado o de caja produce 190 mil millones de dólares, y emplea a millones de personas. La industria norteamericana de software, tiene una posición dominante en el mercado mundial. Alrededor del 80% de los suministradores de software son originarios de Estados Unidos de Norteamérica (EU), mientras

que un 15% corresponde a Europa. En el año de 1997, en Estados Unidos de Norteamérica se reportó que el software empaquetado contribuyó en 13 mil millones de dólares en su balanza comercial, y para el año de 1998. (Sánchez, 1999; ICC, 2007)

1.1.2 India y China

Las empresas del software se convirtieron en el segundo grupo industrial más grande en manufactura en 2001. No obstante en los últimos años India y China repuntan como centros productores tecnológicos de bajo costo. En India se encuentran 42 de las 52 empresas de software certificadas con el nivel superior en calidad, sus ganancias en la manufactura de software pasaron de rondar los 20 millones de dólares en 1990, a superar los 12,800 millones, en 2004. India planea alcanzar la escalofriante suma de 50,000 millones de dólares en exportaciones de software para el 2008. Mientras el volumen de ventas de software en China creció 23% año tras año para alcanzar los 61 billones de dólares en el 2006. Para el año 2010 se predice una ganancia de aproximadamente los 130 billones de dólares para sus industrias del software y de tecnologías de información. (Qui, 2001; Fernández, 2005; Masi, 2006; Ruffinatti, 2007; Xinhuanet, 2007)

1.1.3 Latinoamérica

En Latinoamérica la situación es muy distinta. En 2001 Brasil era el país líder de la región con casi 1900 millones de dólares y una participación del mercado Latinoamericano del 50%, lo seguía México con casi 600 millones de dólares y una participación del 17% y en tercer lugar se encontraba Argentina con 410 millones de dólares y una participación del 11%. Para el 2005 la industria del software en Latinoamérica tuvo una participación del 2.9% del gasto total en Tecnología de Información del mundo. Dicha cifra refleja que la manufactura de software en esta región todavía se encuentra en desarrollo. (SPME, 2005)

1.1.4 México

En cuanto a México, se estima que alrededor de 300 empresas conforman la industria del software y cerca del 20% se encuentran formalmente estructuradas, ya sea como subsidiarias de grandes empresas internacionales, como organizaciones netamente mexicanas, o una combinación de ambas. En el 2003, la industria de tecnología de información estaba muy fragmentada, con un mercado de software de aproximadamente 700 millones de dólares y un crecimiento anual estimado del 3%. (Peñaloza, 2002; SPME, 2005)

También es importante mencionar cuales son las ventajas de México, es decir, porque es conveniente desarrollar software de calidad en México:

- Se tiene una ubicación cercana a los Estados Unidos de Norteamérica, el cual es el principal cliente potencial.
- Existe un incremento notable en el número de profesionales en disciplinas afines a la informática y con niveles de sueldo competitivos respecto al mercado internacional.
- No se tienen restricciones para la importación de tecnología, lo que facilita su transferencia.

- Se cuenta con convenios con los Estados Unidos de Norteamérica, como es el de arancel cero para la prestación de servicios de tecnología de la información.

Dada la creciente importancia de este sector en la economía mundial y el alto valor agregado de los productos informáticos, un país como el nuestro no puede darse el lujo de renunciar a desarrollar esta industria. Por lo cual se deben virar las políticas de desarrollo del software, desde las gubernamentales (al aumentar la inversión en tecnología de la información que sólo es de aproximadamente 1.5 %, mientras que en otros países es del 7.5%) hasta las empresariales, al buscar nuevos sectores como los de mayor concentración de actividad y mayor atraso relativo a la productividad, como salud, gobierno, sector eléctrico, infraestructura (transporte, comunicaciones, servicios públicos), industria del acero, logística y comercio minorista (Carral, 1999; Favela, 2001; SPME, 2005).

1.2 SISTEMA DE INFORMACIÓN

Se dice que para definir este concepto es posible analizarlo en dos elementos. Para lo cual, primero se debe saber qué es un sistema, y posteriormente qué es la información.

Definición de sistema.
Un sistema es una colección de componentes interrelacionados que trabajan conjuntamente. Para lograr una función específica o sistema de funciones. Esto significa que cualquier colección de elementos organizados orientados hacia una meta puede ser un sistema. Los sistemas pueden tener límites o no, e interactuar recíprocamente con sus ambientes circundantes. Si no presentan intercambios con el ambiente que los rodea, es decir, son herméticos a cualquier influencia ambiental son sistemas cerrados, en cambio los sistemas abiertos presentan relaciones de intercambio con el ambiente a través de entradas y salidas. Sin importar si los productos que resultan de un proyecto de software son sistemas cerrados o abiertos, los programas y los proyectos son sistemas casi siempre abiertos: el trabajo del líder de proyecto implica el trabajar recíprocamente con la demás gente involucrada (Senn, 1987; IEEE, 1990; Futrell, 2002).

Definición de información.
La información es un factor importante a considerar para definir lo que se quiere llevar a cabo en una aplicación de software. La información tiene cuatro características: (Olguín, 1997)

- *Única.* No redundante.
- *Oportuna.* Hay que considerar el principio de *tiempo inútil*, de la información.
- *Específica.* Se debe adecuar a las necesidades reales existentes; tal como al realizar un reporte ejecutivo, éste debe constar de cierta cantidad de hojas de acuerdo al nivel al que va dirigido, tanto menor cuanto más alto sea el nivel al que se dirige.
- *Exacta.* Total realidad de los datos procesados y de los programas utilizados.

1.2.1 Definición de sistema de información

Cuando la información forma parte de un sistema adquiere la cualidad de flujo de la información. El flujo de la información representa la manera en la que los datos cambian conforme pasan a través de un sistema. La entrada se transforma en datos intermedios y más adelante se transforma en la salida; lo anterior se ilustra en la figura 1.1.

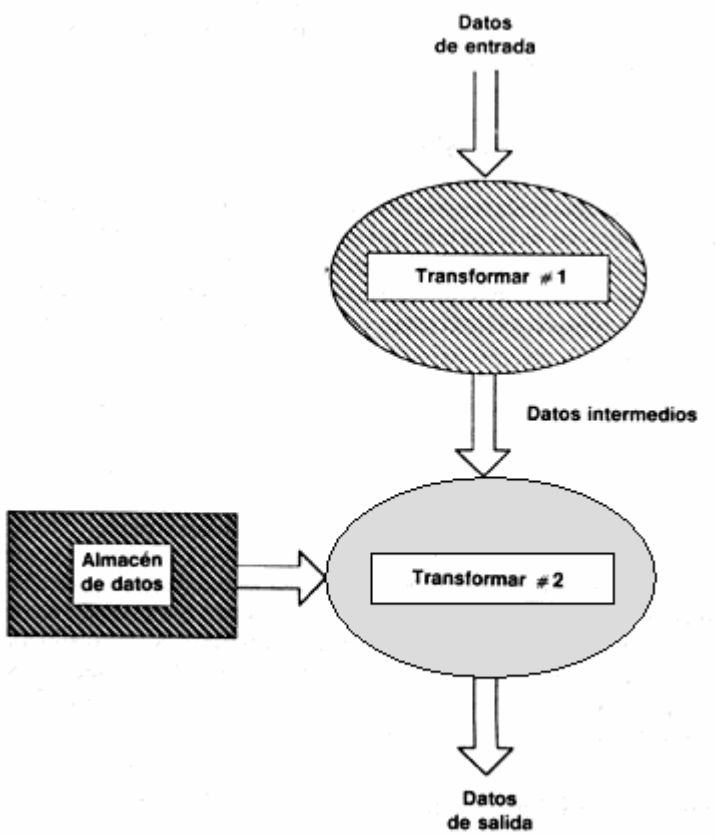


Figura 1.1. Flujo de la información.
Fuente: Bendahan, 2007.

De acuerdo a lo anterior, y a que un sistema basado en computadora procesa información y puede soportar alguna función de negocio, o desarrollar un producto que pueda venderse para generar beneficios, un sistema de información (SI) es un sistema basado en computadoras que acepta datos como entrada, procesa los datos y produce información útil para los usuarios. El realizar un sistema de información, es llevar a cabo un proyecto de software de desarrollo de una nueva aplicación y el producto del proceso puede ser una base de datos, por mencionar algún producto en específico. El desarrollo de un sistema de información requiere de amplia planeación y control para asegurar que el sistema cumpla con los requisitos del usuario y que se termine a tiempo y dentro del presupuesto. Se dice que los elementos de un sistema de información son los elementos de un sistema basado en computadora, los cuales son: (UEAFIT, 1998; Pressman, 2002; Bendahan, 2007)

- *Software.* Programas de computadora, estructuras de datos y su documentación que sirven para hacer efectivo el método lógico, procedimiento o control requerido.

- *Hardware.* Dispositivos electrónicos que proporcionan capacidad de cálculo, dispositivos de interconexión (tales como los conmutadores de red y los dispositivos de telecomunicación) y dispositivos electromecánicos (como los sensores, motores, bombas, etc.) que proporcionan una función extrema del mundo real.

- *Personas.* Usuarios y operadores del hardware y software.

- *Documentación.* Manuales, formularios y otra información descriptiva que plasma el empleo y/o funcionamiento del sistema.

- *Procedimientos.* Los pasos que definen el empleo específico de cada elemento del sistema o el contexto procedimental en que reside el sistema.

1.2.2 Ciclo de vida para el desarrollo de sistemas

El ciclo de vida para desarrollo de sistemas (CVDS), también llamado ciclo de vida de los sistemas de información (CVSI), es una herramienta que asiste en los proyectos de desarrollo de sistemas de información, para planearlos, ejecutarlos y controlarlos. Este ciclo de vida hace énfasis en la identificación de las funciones que realiza la empresa y en el desarrollo de las aplicaciones que lleven a cabo estas funciones. Se dice que el ciclo de vida de desarrollo del software sigue un enfoque orientado a funciones, ya que los sistemas se ven desde el punto de vista de las funciones que llevan a cabo. El CVDS consta de 6 fases, se dice que las dos primeras son abarcadas en parte por la fase de desarrollo del software, las tres siguientes corresponden a la fase de desarrollo y la última corresponde a la etapa de mantenimiento, las cuales son: (Marqués, 2001; Guido, 2003)

- Definición del problema.
- Análisis del sistema.
- Diseño del sistema.
- Desarrollo del sistema.
- Prueba del sistema.
- Puesta en marcha del sistema.

Definición del problema.
Se puede decir que durante esta etapa, se recopilan y analizan los requerimientos del producto, desde las perspectivas del cliente y del desarrollador, y se define con claridad el ámbito del software. El cual describe el control y los datos a procesar, la función, el rendimiento, las restricciones, las interfaces y la fiabilidad del software. Se precisan y estudian los factores técnicos, económicos, de operación y otros de factibilidad para determinar, si el sistema de información es viable (Op. Cit.[1]).

Análisis del sistema.
El equipo encargado del proyecto define el alcance del sistema a desarrollar, entrevista a los posibles usuarios, estudia el sistema ya existente, es decir, la forma en la cual el cliente resolvía su problemática y define las necesidades del usuario. Esta información se puede recoger de varias formas al:

[1] Op. Cit. significa oportunamente citado.

- Entrevistar al personal de la empresa, concretamente, a aquellos que son considerados expertos en las áreas de interés.
- Observar el funcionamiento de la empresa.
- Examinar documentos, sobre todo aquellos que se utilizan para recoger o visualizar información.
- Utilizar cuestionarios para recoger información de grandes grupos de usuarios.
- Utilizar la experiencia adquirida en el diseño de sistemas similares.

Para al fin, obtener como resultado un conjunto de documentos con las especificaciones de requisitos de los usuarios, en donde se describen las operaciones que se realizan en la empresa desde distintos puntos de vista. La información reunida se debe estructurar a través del uso de técnicas de especificación de requisitos, tales como: técnicas de análisis o diseño estructurado y diagramas de flujo de datos, los cuales se detallan en el capítulo cuarto del presente trabajo. Los requisitos deben ser completos y consistentes. (Marqués, 2001; Guido, 2003)

Diseño del sistema.
Se proponen varios diseños conceptuales alternativos que describen las entradas, el procesamiento, las salidas, los equipos, los programas de computadora y la base de datos a un nivel alto. Después se evalúa cada una de estas alternativas y se selecciona la mejor para un diseño y desarrollo adicional. En esta etapa se debe asegurar que toda la funcionalidad especificada en los requisitos de usuario se encuentra en el diseño de la aplicación y además se diseña las interfaces de usuario (Op. Cit.).

Desarrollo del sistema.
Se lleva a la práctica el sistema real. Se compran equipos. El software se compra, se hace a la medida, o se desarrolla. También se desarrollan las bases de datos, las pantallas para captura de datos, los informes del sistema, las redes de telecomunicación, los controles de seguridad y otras características (Op. Cit.).

Prueba del sistema.
Una vez que se desarrollan módulos individuales dentro del sistema se pueden comenzar las pruebas. Las pruebas no sirven para demostrar que no hay fallos, si no para encontrarlos. Si la fase de prueba se lleva a cabo correctamente, descubrirá los errores lógicos, errores de omisión, de seguridad y otros problemas que pudieran impedir el éxito del sistema. Una vez que se prueban los módulos individuales y se corrigen los problemas, se prueba el sistema completo. En las pruebas se puede hacer una medida de la fiabilidad y la calidad del software desarrollado. Cuando los usuarios y los desarrolladores están convencidos de que no tiene errores, se puede poner en marcha (Op. Cit.).

Puesta en marcha del sistema.
El sistema existente se reemplaza por el nuevo, mejorado, y se capacita a los usuarios. Existen varias metodologías para cambiar del sistema existente al nuevo con una interrupción mínima para los usuarios. El ciclo de vida del sistema en si mismo continua con la revisión formal del proceso de desarrollo una vez que el sistema está instalado y en funcionamiento. Después se continúa con el mantenimiento, las modificaciones y las mejorías al sistema. Cuando es

necesario, los nuevos requisitos que aparezcan se incorporan al sistema, de nuevo las etapas del ciclo de vida que se acaban de presentar se revisan (Op. Cit.).

1.2.3 Ciclo de vida para el desarrollo de sistemas con aplicación a bases de datos

La planeación de la base de datos también incluye el desarrollo de estándares que especifiquen cómo realizar la recolección de datos, cómo especificar su formato, qué documentación será necesaria y cómo se va a llevar a cabo el diseño y la implementación. El desarrollo y el mantenimiento de los estándares puede llevar bastante tiempo, pero si están bien diseñados, son una base para el personal informático en formación y para medir la calidad, además, garantizan que el trabajo se ajusta a unos patrones, independientemente de las habilidades y la experiencia del diseñador. Se pueden establecer reglas sobre cómo dar nombres a los datos, lo que evita redundancias e inconsistencias. Se deben documentar todos los aspectos legales sobre los datos y los establecidos por la empresa como qué datos deben tratarse de modo confidencial. (Pressman, 2002; Marqués, 2001)

Es relevante mencionar como se desarrolla el CVDS cuando se realiza un sistema de información con aplicación para una base de datos, ya que mediante el software se crea la conexión entre el usuario y los datos que solicita, es así que las observaciones sobre el CVDS en cada etapa son: (Op. Cit.)

- *Definición del problema.* Al especificar el ámbito del software se especifica el ámbito de la base de datos y los límites de la aplicación de bases de datos, así como con qué otros sistemas interactúa.

- *Análisis del sistema.* En la información recogida se debe incluir las principales áreas de aplicación y los grupos de usuarios, la documentación utilizada o generada por estas áreas de aplicación o grupos de usuarios, las transacciones requeridas por cada área de aplicación o grupo de usuarios y una lista priorizada de los requerimientos de cada área de aplicación o grupo de usuarios.

- *Diseño del sistema.* Para una aplicación de base de datos se debe considerar lo relativo al diseño de bases de datos, expuesto en la sección anterior. Además Si no se dispone de un SGBD, o el que ahí se encuentra obsoleto, se debe escoger un SGBD que sea adecuado para el sistema de información. Esta elección se debe hacer en cualquier momento antes del diseño lógico.

- *Desarrollo del sistema.* Esta etapa y el diseño de la base de datos, son paralelas. Se diseñan los programas de aplicación que usan y procesan la base de datos. En la mayor parte de los casos no se puede finalizar el diseño de las aplicaciones hasta que se termine con el diseño de la base de datos. Por otro lado, la base de datos existe para dar soporte a las aplicaciones, por lo que se realimenta desde el diseño de las aplicaciones al diseño de la base de datos. Se debe asegurar de que toda la funcionalidad especificada en los requisitos de usuario se encuentra en el diseño de la aplicación y se diseñan las interfaces de usuario. Además se requiere implementar la base de datos y si la base de datos es vieja es necesario convertir y cargar datos:

    - Implementación. Se crean las definiciones de la base de datos a nivel conceptual, externo e interno, así como los programas de aplicación. La implementación de la base

de datos se realiza mediante las sentencias del lenguaje de definición de datos del SGBD escogido. Estas sentencias se encargan de crear el esquema de la base de datos, los archivos en donde se almacenan los datos y las vistas de los usuarios. Los programas de aplicación se implementan mediante lenguajes de tercera o cuarta generación. Partes de estas aplicaciones son transacciones sobre la base de datos, que se implementan mediante el lenguaje de manejo de datos del SGBD. Las sentencias de este lenguaje se pueden embeber en un lenguaje de programación anfitrión como Visual Basic, Delphi, C, C++, Java, COBOL o Pascal. En esta etapa, también se implementan los menús, los formularios para la introducción de datos y los informes de visualización de datos. Para ello, el SGBD puede disponer de lenguajes de cuarta generación que permiten el desarrollo rápido de aplicaciones mediante lenguajes de consultas no procedimentales, generadores de informes, generadores de formularios, generadores de gráficos y generadores de aplicaciones. También se implementan en esta etapa todos los controles de seguridad e integridad. Algunos de estos controles se pueden implementar mediante el LDD y otros pueden implementarse mediante utilidades del SGBD o mediante programas de aplicación.

- Conversión y carga de datos. Si se reemplaza un sistema viejo por uno nuevo los datos se cargan desde el sistema viejo al nuevo directamente o, si es necesario, se convierten al formato que requiera el nuevo SGBD y luego se cargan. Si es posible, los programas de aplicación del sistema antiguo también se convierten para que se puedan utilizar en el sistema nuevo.

- *Prueba del sistema*. Se verifica que no existan fallos en la definición, ni defectos durante las solicitudes de información o las operaciones sobre la base de datos.

- *Puesta en marcha del sistema*. Al poner en marcha el sistema se está en la fase de mantenimiento, por lo cual si las prestaciones caen por debajo de un determinado nivel, puede ser necesario reorganizar la base de datos.

1.3 SISTEMAS DE GESTIÓN DE BASES DE DATOS

Los sistemas de gestión de bases de datos se diseñan para manejar grandes cantidades de información. El manejo de los datos incluye tanto la definición de las estructuras para el almacenamiento de la información como los mecanismos para el manejo de la información. El sistema de bases de datos debe cuidar la seguridad de la información almacenada en la base datos, tanto contra las caídas del sistema como contra los intentos de acceso no autorizado. Si los datos son compartidos por varios usuarios, el sistema debe evitar la posibilidad de obtener fallos en los resultados (Korth y Silberschatz, 2006).

1.3.1 Definición de sistema de gestión de bases de datos

Un sistema de gestión de bases de datos (SGBD o DBMS, por las siglas en ingles de Database Management System), consiste en un conjunto de datos relacionados entre sí y un grupo de programas para tener acceso a esos datos. En la figura 1.2 se puede ver una representación simplificada del sistema. El conjunto de datos se conoce como base de datos. El objetivo de un SGBD es crear un ambiente en el cual sea posible guardar y recupera la información de la base de datos de forma conveniente y eficiente. Comprende un conjunto de componentes, los cuales son: (Date, 1990; Korth y Silberschatz, 2006)

- Un lenguaje de definición de esquema conceptual.
- Un sistema de diccionario de datos.
- Un lenguaje de especificación de paquetes de entrada/salida.
- Un lenguaje de definición de esquemas de base de datos.
- Una estructura simétrica de almacenamiento de datos.
- Un módulo de transformación lógica a física.
- Un subsistema de privacidad de propósito general.
- Un subsistema de integridad de propósito general.
- Un subsistema de reserva y recuperación de propósito general.
- Un generador de programas de aplicación.
- Un generador de programas de informes.
- Un lenguaje de consulta de propósito general.

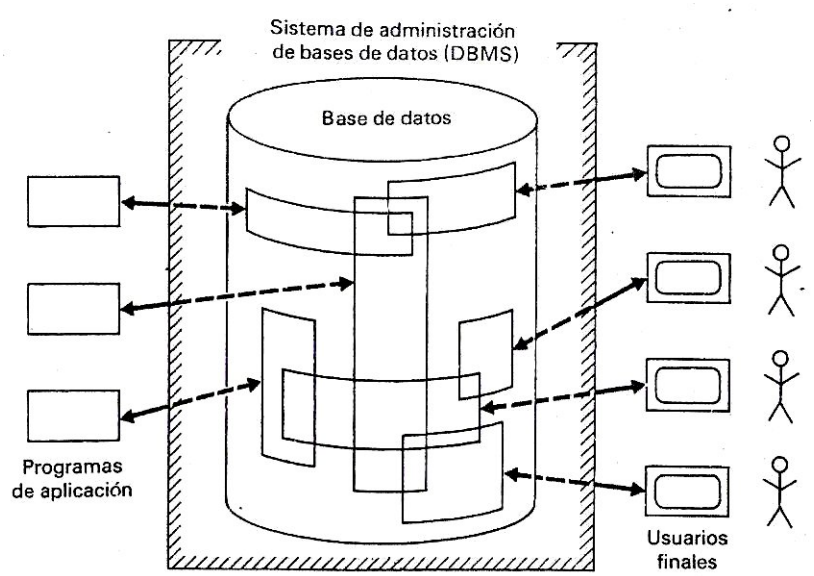


Figura 1.2. Representación simplificada de un Sistema de Gestión de base de datos.
Fuente: Date, 1990.

El SGBD incorpora como herramienta fundamental dos lenguajes, para la definición y la manipulación de los datos. El lenguaje de definición de datos (LDD o DDL, por las siglas en inglés de Data Definition Language) provee de los medios necesarios para definir los datos con precisión, al especificar las distintas estructuras. Acorde con el modelo de arquitectura de tres niveles, habrá un lenguaje de definición de la estructura lógica global, otro para la definición de la estructura interna, y un tercero para la definición de las estructuras externas. El lenguaje de manipulación de datos (LMD o DML, por las siglas en inglés de Data Manipulation/ Management Language), que es el encargado de facilitar a los usuarios el acceso y manipulación de los datos. Pueden diferenciarse en procedimentales (aquellos que requieren qué datos se necesitan y cómo obtenerlos) y no procedimentales (que datos se necesitan, sin especificar cómo obtenerlos), y se encargan de la recuperación de los datos almacenados, de la inserción y supresión de datos en la base de datos, y de la modificación de los existentes. (Op. Cit.)

1.3.2 Definición de bases de datos

Es una colección de datos u objetos correspondientes a las diferentes perspectivas de un sistema de información (de una empresa o institución), existentes en algún soporte de tipo físico (normalmente de acceso directo), agrupados en una organización integrada y centralizada en la que figuran no sólo los datos en sí, sino también las relaciones existentes entre ellos, y de forma que se minimiza la redundancia y se maximiza la independencia de los datos de las aplicaciones que los requieren. Su definición y descripción son únicas y se almacenan junto a los mismos. (Marqués, 2001, Korth y Silberschatz, 2006)

1.3.3 Diseño de la base de datos

El sistema oculta ciertos detalles relativos a la forma en la cual se almacenan y mantienen los datos. Los objetivos del diseño de la base de datos son: (Op. Cit.)

- Representar los datos que requieren las principales áreas de aplicación y los grupos de usuarios, y representar las relaciones entre dichos datos.
- Proporcionar un modelo de datos que soporte las transacciones que se vayan a realizar sobre los datos.
- Especificar un esquema que alcance las prestaciones requeridas para el sistema.

Se definen tres niveles de abstracción de la información en los que puede considerarse a la base de datos, estos son: (Korth y Silberschatz, 2006)

- *Diseño de visión.* Consiste en la producción de un esquema, describe sólo una parte de la base de datos y simplifica la interacción entre usuarios y sistema.
- *Diseño conceptual.* Se describe cuales son los datos reales que están almacenados en la base de datos y que relaciones existen entre los datos. Es el utilizado por los administradores de bases de datos.
- *Diseño físico.* El esquema lógico se traduce en un esquema físico para el SGBD escogido. La fase de diseño físico considera las estructuras de almacenamiento y los métodos de acceso necesarios para proporcionar un acceso eficiente a la base de datos en memoria secundaria.

1.3.4 Modelos de bases de datos

Las necesidades de almacenamiento de datos crecen y con ellas las necesidades de transformar los mismos datos en información de muy diversa naturaleza. Además el diseño o la interpretación de datos pueden dar lugar a información incorrecta y conducir al usuario a la toma de decisiones equivocadas. Por lo cual se creó la metodología de diseño de bases de datos para brindar estructuras correctas y fiables, al minimizar los tiempos de diseño y explotar todos los datos. Los modelos más comunes en el desarrollo de bases de datos se revisan en la Tabla 1.2. También existen las bases de datos distribuidas, las cuales no están almacenadas en un solo lugar físico si no que se distribuyen a lo largo de una red de computadoras

separadas geográficamente, conectadas entre sí. Tienen como objetivo percibirse como un sistema centralizado. (Korth y Silberschatz, 2006; Casares, 2007)

Tabla 1.2. Modelos de bases de datos.

Modelos	Descripción	Tipos
Físicos de los datos	◆ Sirven para describir los datos en el nivel más bajo. ◆ Capturan aspectos de la implantación de los SGBD. ◆ Son poco utilizados.	◆ Modelo unificador. ◆ Memoria de cuadros.
Lógicos basados en registros	◆ Se utilizan para describir los datos en los niveles conceptual y de visión. ◆ Sirven para especificar la estructura lógica general de la base de dados como una descripción en un nivel más alto de la implantación.	◆ Modelo relacional. ◆ Modelo de red. ◆ Modelo jerárquico.
Modelos lógicos basados en objetos	◆ Se utilizan para describir los datos en los niveles conceptual y de visión. Permiten una estructuración flexible. ◆ Permiten especificar las limitantes de los datos.	◆ Modelo entidad-relación (E-R). ◆ Modelo binario. ◆ Modelo semántico de datos. ◆ Modelo infológico.

Fuente: Korth y Silberschatz, 2006.

1.3.5 Modelo relacional

El método convencional de más amplia aceptación en estos momentos y con más productos comerciales que lo soportan, es el modelo relacional. El modelo relacional se basa en el concepto matemático denominado relación, que gráficamente se puede representar como una tabla a nivel lógico (en los niveles externo y conceptual de la arquitectura de tres niveles), y a nivel físico puede implementarse mediante distintas estructuras de almacenamiento. Algunos Sistemas de Gestión de Base de Datos Relacionales son: (Codd, 1990)

- Oracle.
- Informix.
- PostgreSQL.
- Sybase.
- MySQL.

Características del Modelo relacional.
El modelo relacional, se compone de tres aspectos, los cuales son: (Marqués, 2001; Quiroz, 2003)

- *Relaciones.* Existen entre los datos, cada una con un nombre que es único y con un conjunto de columnas. Se representan gráficamente como una tabla bidimensional en la que las filas corresponden a registros individuales y las columnas corresponden a los campos o atributos de esos registros.

- *Elementos de una relación.* Son una colección no ordenada de elementos diferentes, corresponden a las filas o tuplas en una relación. No siguen ningún orden.

- *Atributos.* Un atributo es el nombre de una columna de una relación. Se utilizan para almacenar información sobre los elementos u objetos que se representan en la base de datos. Los atributos pueden aparecer en la relación en cualquier orden.

Otras características importantes del modelo relacional son: (Op. Cit.)

- *Especifica un domino para los atributos*. Un dominio es el conjunto de valores legales de uno o varios atributos. Cada atributo de una base de datos relacional se define sobre un dominio, pudiendo haber varios atributos definidos sobre el mismo dominio. Permite que el usuario defina, en un lugar común, el significado y la fuente de los valores que los atributos pueden tomar. Esto hace que exista más información disponible para el sistema cuando se va a ejecutar una operación relacional, de modo que las operaciones que son semánticamente incorrectas, se pueden evitar. Los SGBD relacionales no ofrecen un soporte completo de los dominios ya que su implementación es extremadamente compleja. (Ver Figura 1.3 b)

- *Define el grado de una relación.* Es el número de atributos que contiene la BD y no cambia con frecuencia.

- *Establece la cardinalidad de una relación.* Es el número de tuplas que tiene la BD. Ya que en las relaciones se insertan y borran tuplas a menudo, la cardinalidad de las mismas varía constantemente.

- *Es un conjunto de relaciones normalizadas.* Una relación **R** definida sobre un conjunto de dominios $\mathbf{D_1, D_2,\ldots D_n}$ consta de: (Ver Figura 1.3 c)

  - Cabecera. Es el conjunto fijo de pares atributo:dominio, se representa así: {(A1:D1), A2:D2…, An:Dn}, donde cada atributo Aj corresponde a un único dominio Dj y todos los Aj son distintos, es decir, no hay dos atributos que se llamen igual. El grado de la relación R es n.
  - Cuerpo. Es el conjunto variable de tuplas. Cada tupla es un conjunto de pares *atributo:valor,* se representa así: $\mathbf{\{(A_1:v_{i1}), A_2:D_{i2}\ldots, A_n:D_{in}\}}$, con *i=1,2,…,n*, donde **m** es la cardinalidad de la relación **R**. En cada par $\mathbf{(A_j:v_{ij})}$ se tiene que $\mathbf{v_{ij}}$ pertenece a $\mathbf{D_j}$.

Representación de las relaciones.
Como se ven la figura 1.3a la relación oficina se representa mediante una tabla, los nombres de las columnas corresponden a los nombres de los atributos, la columna identificada como numofi es la clave primaria de la relación y las filas son cada una de las tuplas de la relación. Los valores que aparecen en cada una de las columnas pertenecen al conjunto de valores del dominio sobre el que está definido el atributo correspondiente. En la figura 1.3b se exponen los dominios de los atributos de la relación OFICINA. Hay dos atributos que están definidos sobre el mismo dominio, los cuales son Teléfono y Fax. La figura 1.3c se revisa la representación formal de la relación OFICINA.

OFICINA

Numofi	Calle	Ciudad	Estado	Teléfono	Fax
O5	Av. Universidad, 3000	D.F.	D.F.	964 201 240	964 201 340
O7	Circuito interior, s/n	Morelia	Michoacán	964 215 760	964 215 670
O3	13 de Septiembre, 1	Chihuahua	Chihuahua	964 520 250	964 520 255
O4	Niños héroes, 23	Sn. Cristóbal	Chiapas	964 284 440	964 284 420
O2	20 de Septiembre, 26	Mérida	Yucatán	964 525 810	964 252 811

a)

Atributo	Nombre del Dominio	Descripción	Definición
Onum	NUM_OFICINA	Posibles valores de número de oficina	3 caracteres; rango O1-O99
Calle	NOM_CALLE	Nombres de calles	25 caracteres
Ciudad	NOM_CIUDAD	Nombres de ciudades	20 caracteres
Estado	NOM_ESTADO	Nombres de estados	15 caracteres
Teléfono	NUM_TEL_FAX	Números de teléfono	9 caracteres
Fax	NUM_TEL_FAX	Números de teléfono	9 caracteres

b)

La relación OFICINA tiene la siguiente cabecera:
{ (Onum:NUM_OFICINA), (Calle:NOM_CALLE), CIUDAD:NOM_CIUDAD),
(ESTADO:NOM_ESTADO), (Teléfono:NUM_TEL_FAX), (Fax:NUM_TEL_FAX)}.
Ésta es una de sus tuplas o filas:
{ (Onum:O5), (Calle: Av. Universidad, 3000), Ciudad:D.F.),
(Estado:), (Teléfono:964 201 240), (Fax:964 201 340)}.

c)

Figura 1.3. Representación de una relación de una base de datos relacional de oficinas de una empresa. a) Muestra la relación OFICINA. b) Expone los dominios de los atributos de la relación OFICINA. c) Revisa la representación formal de la relación OFICINA.
Fuente: Marqués, 2001.

En un SGBD relacional pueden existir varios tipos de relaciones, aunque no todos manejan todos los tipos, estos tipos son: (Marqués, 2001; Quiroz, 2003)

- *Relaciones base.* Son relaciones reales que tienen nombre y forman parte directa de la base de datos almacenada (son autónomas).
- *Vistas.* También denominadas relaciones virtuales, son relaciones con nombre y derivadas, se representan mediante su definición en términos de otras relaciones con nombre, no poseen datos almacenados propios.

- *Instantáneas.* Son relaciones con nombre y derivadas. Pero a diferencia de las vistas, son reales, no virtuales: están representadas no sólo por su definición en términos de otras relaciones con nombre, sino también por sus propios datos almacenados. Son relaciones de sólo de lectura y se refrescan periódicamente.

- *Resultados de consultas*. Son las relaciones resultantes de alguna consulta especificada. Pueden o no tener nombre y no persisten en la base de datos.

- *Resultados intermedios*. Son las relaciones que contienen los resultados de las subconsultas. Normalmente no tienen nombre y tampoco persisten en la base de datos.

- *Resultados temporales.* Son relaciones con nombre, similares a las relaciones base o a las instantáneas, pero la diferencia es que se destruyen automáticamente en algún momento apropiado.

Claves.
Ya que en una relación no hay tuplas repetidas, éstas se pueden distinguir unas de otras, es decir, se pueden identificar de modo único. La forma de identificarlas es mediante los valores de sus atributos. Una superclave es un atributo o un conjunto de atributos que identifican de modo único las tuplas de una relación. Una clave candidata es una superclave en la que ninguno de sus subconjuntos es una superclave de la relación. El atributo o conjunto de atributos **K** de la relación **R** es una clave candidata para **R** si y sólo si satisface las siguientes propiedades: (Marqués, 2001; Quiroz, 2003)

- *Unicidad: nunca* hay dos tuplas en la relación **R** con el mismo valor de **K**.

- *Irreductibilidad (minimalidad):* ningún subconjunto de **K** tiene la propiedad de unicidad, es decir, no se pueden eliminar componentes de **K** sin destruir la unicidad.

Cuando una clave candidata esta formada por más de un atributo, se dice que es una clave compuesta. Una relación puede tener varias claves candidatas. Las claves candidatas se identifican al establecer el significado real de los atributos, ya que esto permite saber si es posible que aparezcan duplicados. La clave primaria de un relación es aquella clave candidata que se escoge para identificar sus tuplas de modo único (Ver Figura 1.3 a). Ya que una relación no tiene tuplas duplicadas, siempre hay una clave candidata y, por lo tanto, la relación siempre tiene clave primaria. En el peor caso, la clave primaria se forma por todos los atributos de la relación, pero normalmente habrá un pequeño subconjunto de los atributos que haga esta función. Las claves candidatas que no son escogidas como clave primaria son denominadas claves secundarias. Una clave ajena es un atributo o un conjunto de atributos de una relación cuyos valores coinciden con los valores de la clave primaria de alguna otra relación, en ocasiones es la misma (Op. Cit.).

2. INGENIERIA DE SOFTWARE

2.1 DEFINICIÓN DE SOFTWARE

El término fue creado para diferenciar las instrucciones del hardware, que son los componentes físicos de un sistema computacional. Un programa de computadora es un conjunto de instrucciones que dirigen el hardware de la computadora para ejecutar una tarea. (Enciclopedia Británica, 2006)

Este término es todavía más amplio, se puede decir que el software es la suma total de los programas de computadora, de cualquier tamaño o arquitectura, los procedimientos, las reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo, tales como: números texto y representaciones de audio, video e imágenes. De esta forma un producto de software es un artículo diseñado por la ingeniería de software para un usuario específico. (Pressman, 2002)

Las características generales del software se pueden revisar en la tabla 2.1.

Tabla 2.1. Características generales del software.

El software se desarrolla, no se fabrica en un sentido clásico.
En el software sólo existen problemas durante el desarrollo y en la afinación de los detalles para la entrega.
El software no se estropea, los agentes externos (temperatura, humedad, etc.) no le afectan.
El software, en general, se construye a la medida.

Fuente: Pressman, 2002.

2.2 TIPOS DE SOFTWARE

Se dice que los dos tipos genéricos del software de computadora son: de sistema y de aplicación. El software de sistema controla el funcionamiento interno de la computadora, principalmente a través de un sistema operativo, y también controla los periféricos tales como monitores, impresoras, y dispositivos de almacenaje. En contraste, el software de aplicación, dirige a la computadora e incluye cualquier programa que procesa datos para ejecutar instrucciones dadas por el usuario. El software de aplicación por lo tanto, incluye procesadores de texto, hojas de cálculos, el inventario, programas de nómina y muchas otras aplicaciones. (Pressman, 2002)

De acuerdo con Pressman (2002), las categorías del software en cuanto a su aplicación especifica, es decir, al tener en cuenta su complejidad son:

- De sistemas.
- De tiempo real.
- De gestión.
- De ingeniería y científico.
- Empotrado.
- Basado en Web.
- Software de computadoras personales.
- De inteligencia artificial.

2.2.1 Software de sistemas

El software de sistemas es un conjunto de programas que han sido escritos para servir a otros programas. Se caracteriza por: (Op. Cit.)

- La fuerte interacción con el hardware de la computadora.
- Su gran utilización por múltiples usuarios
- Una operación concurrente que requiere una planificación.
- El reparto de recursos, y
- Una sofisticada gestión de procesos.

2.2.2 Software de tiempo real

Es el software que coordina, analiza y controla sucesos del mundo real en el momento en que estos ocurren. Entre los elementos del software de tiempo real se incluyen: (Op. Cit.)

- Un componente de adquisición de datos que recolecta y da formato a la información recibida del engomo externo.
- Un componente de análisis que transforma la información según lo requiera la aplicación.
- Un componente de control entrada/salida que responda al entorno externo.
- Un componente de autorización que coordina a todos los demás componentes de forma que pueda mantenerse la repuesta en un tiempo real.

2.2.3 Software de gestión

Las aplicaciones en esta área reestructuran los datos existentes para facilitar operaciones comerciales o gestionar la toma de decisiones. Además de las tareas convencionales de procesamiento de datos y las aplicaciones del software de gestión también realizan cálculo interactivo. Al analizar estas tareas se logro evolucionar al software de sistemas de información de gestión (SIG) que accede a una o más bases de datos que contienen información comercial. (Op. Cit.)

2.2.4 Software de ingeniería y científico

El software de ingeniería y científico se caracteriza por los algoritmos de gestión numérica, el diseño asistido por computadora (CAD por las siglas en ingles de Computer Assistant Design), la simulación de sistemas y otras aplicaciones interactivas. Su aplicación es dada en las ciencias, tales como: astronomía, vulcanología o biología molecular. Su evolución lleva a este tipo de software a tomar características del software de tiempo real e incluso del software de sistemas. (Pressman, 2002)

2.2.5 Software empotrado

El software empotrado reside en la memoria de sólo lectura, y se utiliza para controlar productos y sistemas de los mercados industriales y de consumo. El software empotrado puede ejecutar funciones muy limitadas, pero importantes, o suministrar una función significativa y con capacidad de control. (Op. Cit.)

2.2.6 Software de computadoras personales

Existen muchas aplicaciones para este tipo de software, tales como: el procesamiento de textos, las hojas de cálculo, los gráficos por computadora, multimedia, entretenimientos, gestión de bases de datos, aplicaciones financieras, de negocios y personales y redes o acceso a bases de datos externas, etc. Este nicho del software es participe de un gran desarrollo desde hace unas décadas. (Op. Cit.)

2.2.7 Software basado en Web

En esencia, la Web viene a ser una gran computadora que proporciona un recurso software casi ilimitado que puede ser accedido por cualquiera con posibilidad de conectarse a Internet. Las páginas Web buscadas por un explorador son software que incorpora instrucciones ejecutables y datos. Para su elaboración utiliza lenguajes de programación como Perl o Java y lenguajes de etiquetado como HTML (por las siglas en inglés de Hyper Text Markup Language). Presentan de manera ágil desde contenidos de texto hasta de audio y visual. (Op. Cit.)

2.2.8 Software de inteligencia artificial

El software de inteligencia artificial (AI, por las siglas en inglés de Artificial Intelligence), hace uso de algoritmos no numéricos para resolver problemas complejos, para los que no son adecuados el ciclo o el análisis directo. Los sistemas expertos, también llamados sistemas basados en el conocimiento, reconocimiento de patrones (imágenes y voz), redes neuronales artificiales, prueba de teoremas, y los juegos son representativos de las aplicaciones de esta categoría. (Op. Cit.)

2.3 DEFINICIÓN DE INGENIERÍA DE SOFTWARE

El IEEE (1993) indica que la Ingeniería de Software (SE de las siglas en inglés de Software Engineering) es:

> *"Ingeniería de software: (1) La aplicación de in enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software; es decir, la aplicación de ingeniería de software. (2) El estudio de enfoques como en (1)".*

Así que la ingeniería de software es una entidad de múltiples capas que integra métodos, herramientas, procedimientos y gestión de calidad para el desarrollo de software de computadora. La definición anterior se ilustra en la figura 2.1. Así mismo debe indicarse que éstos permiten elaborar consistentemente productos confiables, utilizables, eficientes y mantenibles. (Pressman, 2002; UVP, 2007)

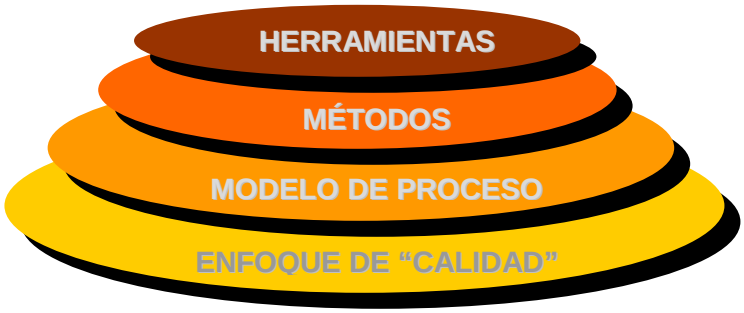


Figura 2.1 Definición de Ingeniería de Software.
Fuente: Pressman, 2002.

La capa de proceso es la que une las capas de tecnología y permite el desarrollo razonado y acertado, por tanto es el fundamento de la ingeniería de software. En esta capa se define lo que se llama Áreas Claves del Proceso (ACP), las cuales son necesarias para la entrega efectiva de la tecnología, más adelante se explica en qué consiste un proceso desde la perspectiva de la ingeniería de software y como se definen la ACP. La capa de enfoque de calidad implica la gestión total de la calidad y las filosofías similares para fomentar una cultura continua de mejoras de procesos y conducir al desarrollo de enfoques cada vez más robustos para la ingeniería del software. Los métodos indican cómo construir el software técnicamente e incluyen un amplio espectro de procedimientos, mismos que definen su secuencia de aplicación, para la planeación, la estimación, análisis, diseño, codificación, prueba y mantenimiento. Las herramientas brindan un enfoque automático y semiautomático al apoyar a la aplicación de los métodos cuando se integran, de forma que la información creada por una herramienta puede ser usada por otra. (Pressman, 2002)

2.4 MODELOS DE CALIDAD DEL SOFTWARE

Si la calidad contiene características intangibles, términos como alta, baja, y buena calidad, es adaptable a la necesidad de los usuarios y es conforme con los requerimientos, entonces la calidad del software se mide y predice, respecto a ausencia de defectos, satisfacción del usuario y conformidad con los requerimientos. La calidad de software requiere de métricas para su análisis, en el capítulo 4 del presente estudio se revisan las métricas de calidad del software.

En este apartado se revisan las características de calidad y el modelo de uso general para evaluar el rango de resultados esperados que se pueden tener presentes en un proceso de software. (Pressman, 2002; ISO/IEC 15504, 2004; Bedini, 2007)

2.4.1 Características de calidad del software

En 1977 McCall, Richards, y Walters, propusieron la idea de dividir el concepto de calidad del software en un cierto número de características. Posteriormente se propuso analizar la calidad del producto del software mediante un modelo jerárquico de calidad, en una colección de características y subtipos, los cuales están unidos a métricas e indicadores apropiados. Para 2001, la Organización Internacional de Estándares (ISO, por las siglas en inglés de International Organization for Standardization) y la Comisión Internacional de Electrónica (IEC, por las siglas en inglés de International Electrotechnical Comision) determinaron un conjunto estándar de características del software de calidad, denominado ISO-9126. Este estándar define seis características de calidad y sus subtipos mostrados en la tabla 2.2. (Veenendaal et al., 2002)

Tabla 2.2. Características del software de calidad.

Funcionalidad	Confiabilidad	Facilidad de uso	Eficiencia	Capacidad de mantenimiento	Portabilidad
Idoneidad	Madurez	Facilidad de comprensión	Tiempo de funcionamiento	Capacidad de análisis	Adaptabilidad
Exactitud	Tolerancia a fallos	Facilidad de aprender	Utilización de recursos	Variabilidad	Capacidad de instalación
Interoperabilidad	Capacidad de recuperación	Operabilidad		Estabilidad	Capacidad de reemplazo
Seguridad				Facilidad de prueba	

Fuente: Veenendaal et al., 2002.

La International Standards for Language Engineering (ISLE) (2000) menciona la descripción de cada una de las características del software de calidad dada por la ISO-9126. Estas características se pueden revisar en la tabla 2.3. No todas las características de calidad tienen igual importancia para un producto de software. El transporte puede tener poca importancia cuando el objetivo de desarrollo es para una plataforma dedicada y la capacidad de mantenimiento no es relevante cuando el software desarrollado es un producto con un ciclo de vida corto. Por tanto, es necesario identificar y seleccionar la característica de calidad más importante mediante una evaluación de riesgo. Esto se puede lograr a través una buena administración del proyecto del software mediante la consulta de los involucrados internamente en el proyecto (tales como el administrador del producto, el administrador del proyecto, y el diseñador del software) y fuera del proyecto (usuarios).

Tabla 2.3. Descripción de las características del software de calidad.

Funcionalidad
◆Idoneidad. Atributos de software relacionados con la presencia y el adecuado uso de un conjunto de funciones para tareas específicas. ◆Exactitud. Atributos de software relacionados con el aseguramiento de los efectos o de los resultados acordados. ◆Interoperabilidad. Atributos de software relacionados con su capacidad de actuar recíprocamente con otros sistemas. ◆Seguridad. Atributos de software relacionados con su capacidad de prevenir el acceso no autorizado, accidental o deliberado, a programas o datos.
Confiabilidad
◆Madurez. Atributos de software relacionados con la frecuencia de fallas por desperfectos en el software. ◆Tolerancia a fallos. Atributos de software relacionados con su capacidad de mantener un nivel de funcionamiento especificado en casos de fallas de software o de infracción de su interfaz. ◆Capacidad de recuperación. Atributos de software relacionados con la capacidad de restablecer su nivel de funcionamiento, recuperar los datos directamente afectados en caso de una falla y sobre el tiempo y el esfuerzo necesario para ello.
Facilidad de Uso
◆Facilidad de comprensión. Atributos de software relacionados con el esfuerzo de los usuarios para reconocer el concepto lógico y su aplicabilidad. ◆Facilidad de Aprender. Atributos de software relacionados con el esfuerzo de los usuarios para aprender su uso (por ejemplo, el control de operación, entrada, salida). ◆Operatividad. Atributos de software relacionados con el esfuerzo de los usuarios para la operación y el control de operación.
Eficiencia
◆Tiempo de funcionamiento. Atributos de software que impactan la respuesta, el tiempo de procesamiento y las tasas de rendimiento en la realización de su función. ◆Utilización de recursos. Atributos de software relacionados con la cantidad de recursos usados y la duración de tal uso en la realización de su función.
Capacidad de mantenimiento
◆Capacidad de análisis. Atributos de software relacionados con el esfuerzo necesario para el diagnóstico de las deficiencias, las causas de fallas, así como para la identificación de las partes a ser modificadas. ◆Variabilidad Atributos de software relacionados con el esfuerzo necesario para la modificación, el retiro de falla o para el cambio en el entorno. ◆Estabilidad Atributos de software relacionados con el riesgo de efectos inesperados de modificaciones hechas. ◆Facilidad de prueba Atributos de software relacionados con el esfuerzo necesario para validar el software.
Portabilidad
◆Adaptabilidad. Atributos de software relacionados con la oportunidad para la adaptación del software a diferentes ambientes sin aplicar otras acciones o medios que aquellos provistos para este propósito. ◆Capacidad de instalación. Atributos de software relacionados con el esfuerzo necesario para instalar el software en un ambiente especificado ◆Capacidad de reemplazo. Atributos de software relacionados con la oportunidad y el esfuerzo de usarlo en el lugar y en el entorno de otro software.

Fuente: ISLE, 2000.

2.4.2 Modelo de capacidad de madurez

La calidad en las empresas desarrolladoras de software en Estados Unidos, Europa y Asia se puede medir mediante el modelo de capacidad de madurez (CMM *por las siglas en inglés de Capability Maturity Model*). (García, 2004)

El modelo CMM es un modelo que establece los niveles por los cuales las organizaciones de software hacen evolucionar sus definiciones, implementaciones, mediciones, controles y mejorías de sus procesos de software. (Paulk et al., 1993)

Al permitir la definición del grado de madurez de las prácticas de gestión y de ingeniería de software de las organizaciones, se puede determinar cómo madura una determinada organización y cuáles son las acciones de mejora prioritarias para sus procesos de software. El enfoque inicial del CMM fue el proceso de software; sin embargo, se puede encontrar aplicaciones del modelo en otros campos, tal es el caso del CMM para personas (*People CMM*), CMM para ingeniería de sistemas (*Systems Engineering CMM*), CMM para la gestión de productos integrados (*Integrated Product Management CMM*) y otros. (Zahran, 1998)

El esquema del modelo CMM está compuesto por cinco niveles de madurez de acuerdo con la capacidad del proceso de software, y están definidos por los objetivos de los procesos que, cuando se satisfacen, permiten evolucionar al próximo nivel, ya que uno o más componentes importantes del proceso de software se han estabilizado. Lo anterior se ilustra en la figura 2.2. (Op. Cit.)

De esta manera, los niveles de madurez ayudan a las organizaciones a definir las prioridades para sus esfuerzos de mejora. Debe indicarse que cada nivel superior es la suma de los niveles anteriores. Dichos niveles son los siguientes: (Op. Cit.)

- *Nivel 1: Inicial.* Se caracteriza como ad hoc o caótico. Se definen pocos procesos. Representa una situación sin ningún esfuerzo en la garantía de calidad y gestión del proyecto, donde cada equipo del proyecto puede desarrollar software de cualquier forma al elegir los métodos, estándares y procedimientos a utilizar, los cuales pueden variar desde lo mejor hasta lo peor.

- *Nivel 2: Repetible.* Se caracteriza como disciplinado y se establecen los procesos básicos de la gestión. Se definen ciertas actividades tales como el informe del esfuerzo y tiempo empleado, y el informe de las tareas realizadas.

- *Nivel 3: Definido*. Se caracteriza como estándar y consistente. Se definen tanto procesos técnicos como de gestión. Se pretende conseguir utilizando estándares como el ISO 9001, y en pocas ocasiones se supera este nivel, ya que si las notaciones de las actividades no están bien definidas no se puede lograr el nivel siguiente.

- *Nivel 4: Gestionado.* Se caracteriza como predicable, es decir, hay una preocupación en la medición detallada de la calidad del proceso de software y del producto. Comprende el concepto de medición y el uso de métricas. Dichas métricas se utilizan para supervisar y controlar un proyecto de software.

- *Nivel 5: Optimización.* Se caracteriza como mejora continua a partir de la realimentación (feedback) cuantitativa. Es el nivel más alto y difícil de alcanzar. Representa la analogía del software con los mecanismos de control de calidad que existen en otras industrias de

mayor madurez. Se puede decidir la calidad del producto al basarse en los diferentes procesos.

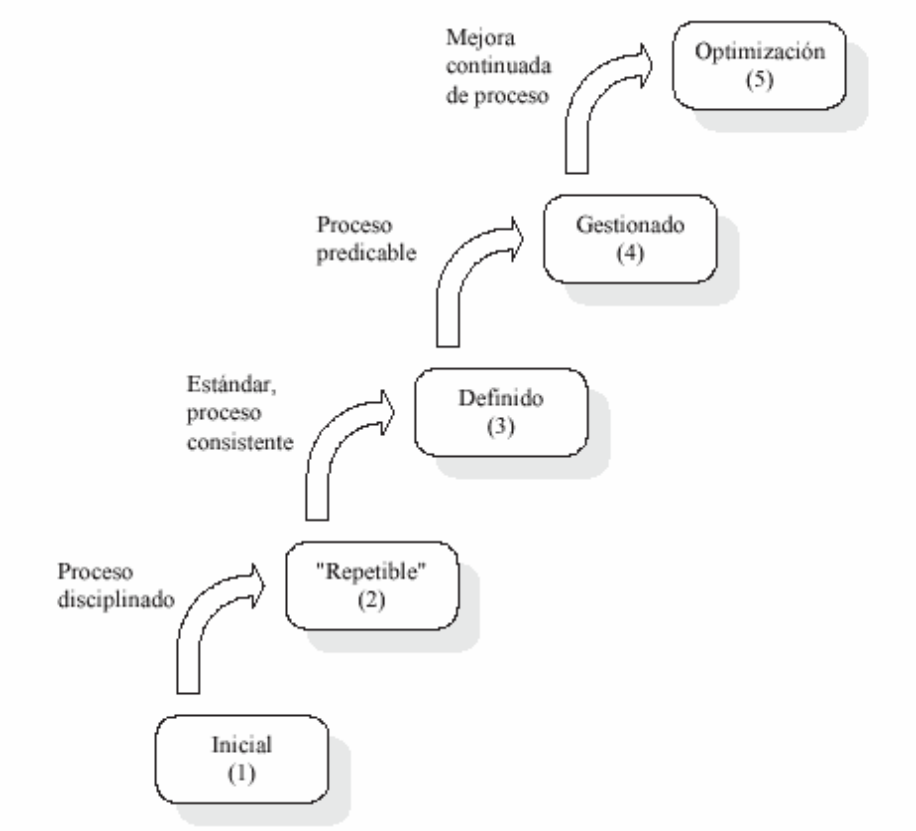


Figura 2.2. Los niveles de madurez del proceso de software del CMM.
Fuente: Paulk et al., 1993.

Como se ve en la figura 2.3 cada nivel de madurez tiene una estructura interna compuesta por los siguientes componentes: (Zahran, 1998)

- *Nivel de madurez.* Representa un indicador evolutivo que permite alcanzar la madurez del proceso del software.

- *Áreas de proceso clave* (ACP o en inglés KPA, por las siglas de Key Process Areas). Son las subestructuras de cada nivel, que indican las áreas a las que una organización debería dirigir su atención con el propósito de mejorar su proceso de software. Se asigna cada conjunto de ACP a un nivel de madurez como se aprecia en la figura 2.4 (excepto al nivel uno). El CMM no detalla todas las áreas del proceso involucradas en el desarrollo y mantenimiento del software, sino que se enfoca en las áreas clave que contribuyen en la mayoría de las capacidades de proceso.

- *Objetivos*. Este componente delimita las ACP a través de la definición de su alcance, límites e intenciones. Los objetivos, por lo tanto, determinan las restricciones que se deben superar por la organización, para que ésta pueda alcanzar mejores niveles de madurez.

- *Características comunes*. Son atributos que indican si la implementación e institucionalización de una ACP son eficaces, repetidas y duraderas. Tiene cinco características comunes:
  - El compromiso en desempeñar las acciones que garantizan el establecimiento del proceso.
  - La habilidad en desempeñar dichas acciones.
  - Las actividades desempeñadas para implementar las ACP.
  - La medición y el análisis del estado y eficacia de las actividades desempeñadas.
  - La implementación para verificar las actividades desempeñadas respecto a los procesos establecidos.
- *Practicas clave.* Estas describen la infraestructura y las actividades que contribuyen para la implementación e institucionalización efectiva de la ACP.

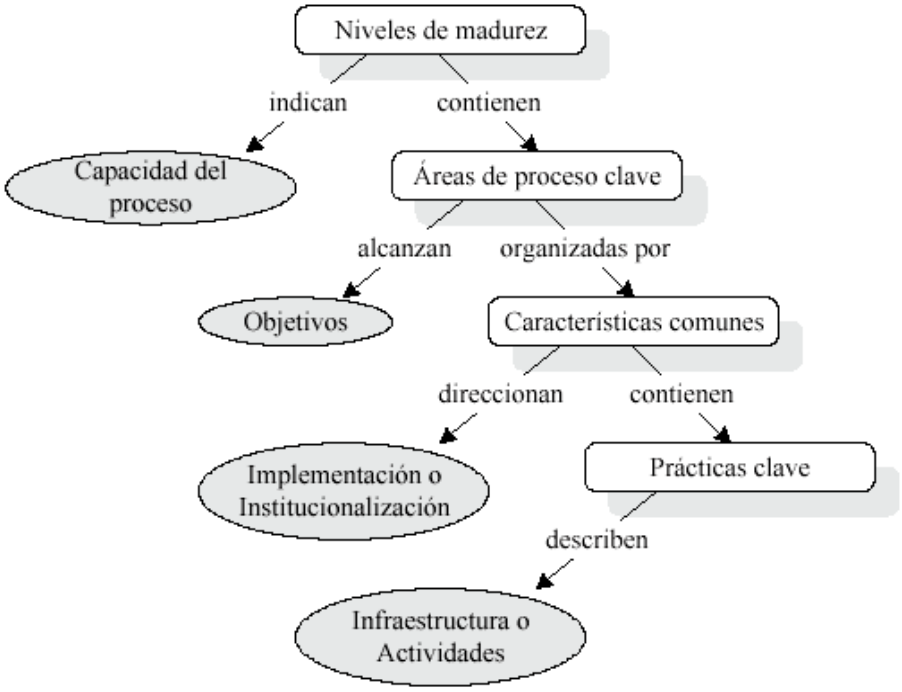


Figura 2.3. Estructura interna del CMM.
Fuente: Paulk et al., 1993.

El CMM es un mapa para la definición de los pasos que una organización necesita realizar para moverse desde procesos caóticos hacia procesos de mejora continua. Las evaluaciones basadas en CMM utilizan el cuestionario del Instituto de Ingeniería de Software (SEI por las siglas en inglés de *Software Engineering Institute*) como un instrumento de evaluación. Su enfoque se dirige a temas relacionados con procesos, y sus cuestiones se basan en los objetivos de las ACP del CMM. (Paulk et al., 1993)

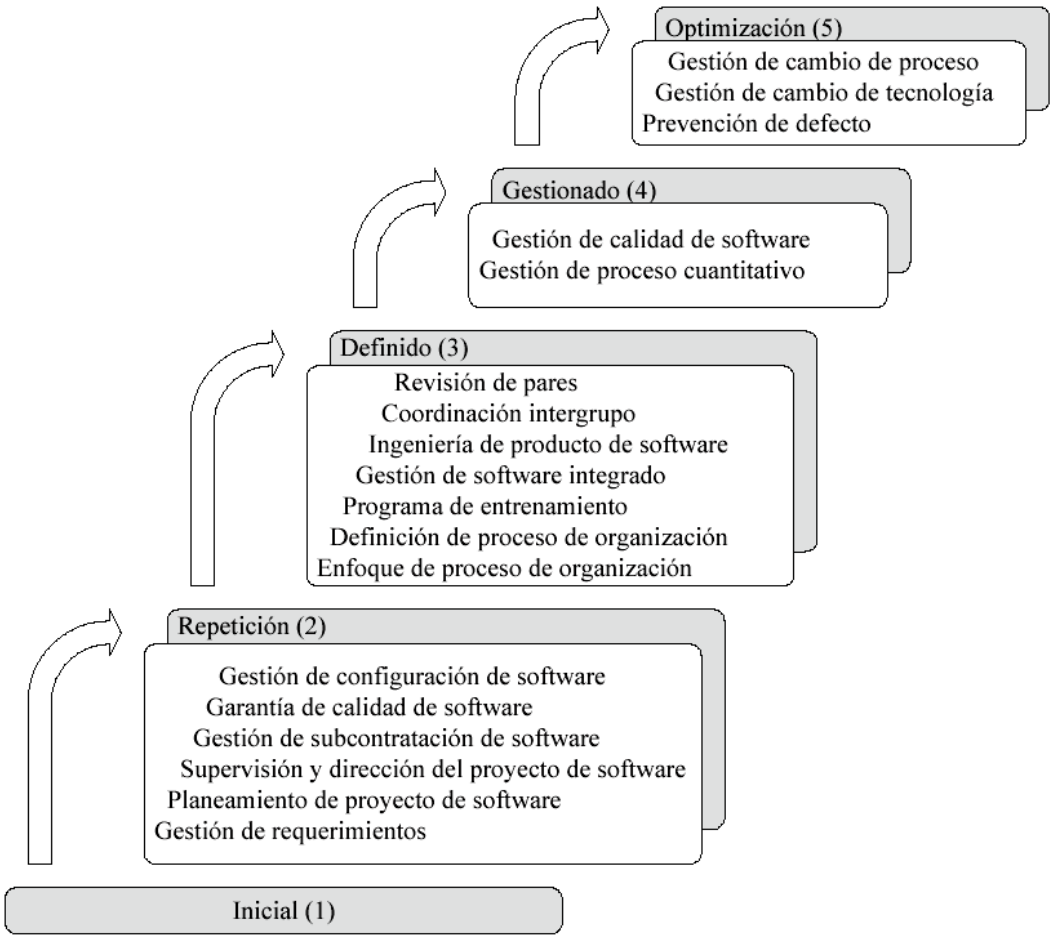


Figura 2.4. Las ACP representadas en cada uno de los cinco niveles de madurez del proceso de software del CMM.
Fuente: Paulk et al., 1993.

Modelo de capacidad de madurez integrado.
El modelo de capacidad de madurez integrado (CMMI *por las siglas en inglés de Capability Maturity Model Integrated*), es usado también para evaluar la calidad de las empresas desarrolladoras de software y se basa en el modelo CMM. Son tres las principales diferencias entre CMM y CMMI: (Royce, 2002)

- El CMM se enfoca sólo en software, mientras el CMMI incorpora otras disciplinas, tales como sistemas, proveedores y desarrollo integrado de proyectos.
- El CMMI tiene dos formas de revisar el desarrollo de un proyecto de software, las cuales son:
  - Por etapas. En el cual hay un camino predefinido que deben seguir las empresas que busquen escalar de nivel CMMI.
  - Continuo. En el que las empresas eligen que áreas potenciar sin necesariamente buscar un nivel general CMMI, sino un nivel por área (aunque el modelo permite obtener

un nivel CMMI general aun si se sigue el modelo continuo). Esto se hace sin eliminar los niveles en el modelo continuo, es decir se sube de nivel por área.

- El CMMI también incorpora varias áreas de proceso, las cuales se conocen como KPA en el CMM, tales como: gestión de riesgo y todo lo relacionado con el desarrollo integrado de proyectos.

2.4.3 Estándar ISO/IEC 15504

El desarrollo del estándar ISO/IEC 15504 se establece a partir del proyecto SPICE (por las siglas en inglés de Software Process Improvement and Capability determination) que tiene como objetivo garantizar una ruta de desarrollo rápida y solicitar las opiniones de los expertos más importantes del mundo. (Zahran, 1998; ISO/IEC15504, 2004).

- Dimensión de proceso.
- Dimensión de capacidad.

Dimensión de proceso.
Se caracteriza por los propósitos de proceso (tales como los objetivos de medición esenciales de un proceso) y el resultado esperado del proceso (como la indicación de su finalización exitosa). Para esta dimensión son atribuidas cinco categorías:

- Categoría de proceso cliente-proveedor.
- Categoría de proceso de ingeniería.
- Categoría de proceso de soporte.
- Categoría de proceso de gestión.
- Categoría de proceso de organización.

Dimensión de capacidad.
Esta dimensión consiste en un conjunto de atributos interrelacionados que ofrecen los indicadores de medición necesaria para gestionar un proceso y mejorar la capacidad de desempeñar un proceso. Como se muestra en la figura 2.5 la dimensión de capacidad está compuesta de seis niveles y tiene características evolutivas similares a las del CMM. Las cuales son los siguientes: (Zahran, 1998)

- *Nivel 0*. Incompleto: se caracteriza por un incumplimiento general para lograr el propósito del proceso.

- *Nivel 1*. Proceso realizado: se caracteriza por el logro de manera general del propósito del proceso.

- *Nivel 2*. Proceso gestionado: se caracteriza por la identificación de la calidad aceptable con definición de tiempos y recursos. Los productos del trabajo están de acuerdo con los estándares especificados y los requerimientos.

- *Nivel 3*. Proceso establecido: se caracteriza por la gestión y el desempeño del proceso usando el proceso estándar basado en principios estables de ingeniería de software.

- *Nivel 4*. Proceso predecible: se caracteriza por la consistencia de su desempeño en la práctica con límites de control definidos para alcanzar sus objetivos de proceso definido.

- *Nivel 5*. Proceso optimizado: se caracteriza por la optimización del desempeño del proceso para encontrar las necesidades de negocio actuales y futuras, y por el grado de repetición que el proceso alcanza encontrando sus objetivos de negocio definidos.

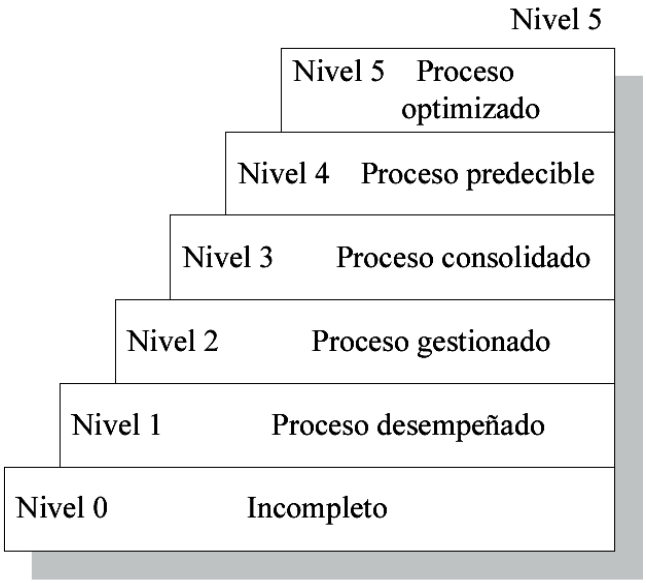


Figura 2.5. Los niveles de capacidad de proceso según ISO/IEC 15504.
Fuente: Zahran, 1998.

2.5 PROCESO Y CICLO DE VIDA

El proceso se compone de estados simples que se encausan con una secuencia de pasos definidos para llegar a un propósito. También se dice que es una serie de acciones que transforman una colección de entradas en un resultado. Lo cual se ilustra en la figura 2.6. (IEEE, 1990; PMI, 2004)

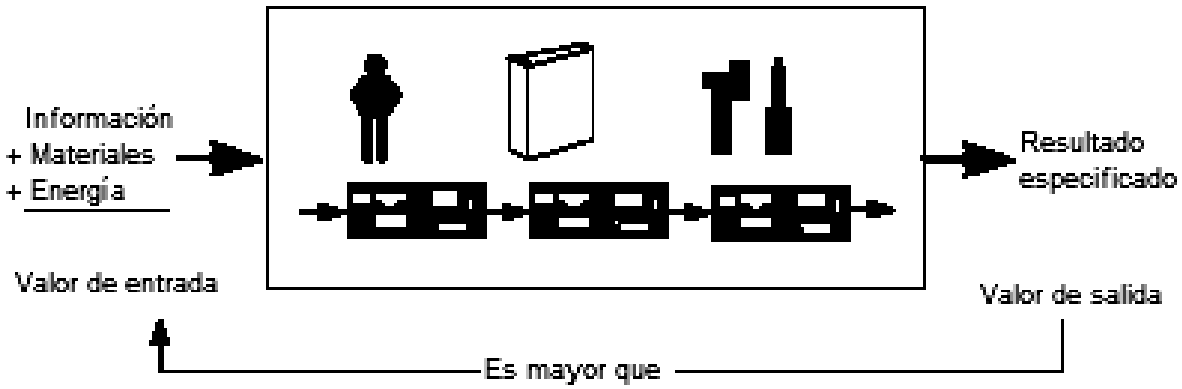


Figura 2.6. El proceso del software.
Fuente: PMI, 2004.

El producto del proceso es cualquier artefacto entregable o documento de resultados de una actividad, y los recursos son entidades requeridas para que se efectúe alguna actividad del

proceso. Existen dos tipos de proceso a considerar, los procesos de gestión de proyectos y los procesos orientados al producto; estos se explican en la tabla 2.4. (PMI, 2004)

Tabla 2.4. Procesos del proyecto y del producto.

Procesos del proyecto	Procesos del producto
➢ Describen, organizan y completan el trabajo del proyecto. ➢ La mayoría son aplicables, en la mayoría de los proyectos.	➢ Especifican y crean el producto del proyecto. ➢ Están definidos por el ciclo de vida del proyecto y varían según el área de aplicación.

Fuente: PMI, 2004.

El enfoque orientado a procesos permite: (Zahran, 1998)

- La sincronización y armonía entre las actividades desempeñadas por cada individuo y los objetivos comunes del equipo.
- La garantía de la consistencia de las actividades y sus resultados.
- El uso de técnicas objetivas de medición del desempeño de cada individuo respecto a las actividades asignadas.
- Debido a la reducción de la dependencia en cada individuo, se consigue que el equipo pueda repetir un determinado proceso, aunque individuos ajenos sean asignados a la actividad.

2.5.1 Proceso del software

Un proceso de software está compuesto por tres aspectos: (Zahran, 1998; Borges de Barros, 2002)

- El primer aspecto es la definición del proceso que generalmente está representado por un documento que especifica las actividades y procedimientos para el proceso, mediante el establecimiento de este marco común del proceso se define un número de actividades del marco de trabajo que son aplicables a todos los proyectos del software, con independencia de su tamaño o complejidad. Tiene como acción la entrada de datos.
- El segundo aspecto se refiere a la transferencia del conocimiento del proceso para aquellos que lo van a ejecutar, mediante el procesamiento, para ello se declara un numero de conjuntos de tareas y actividades de protección. El conjunto de tareas incluye una colección de trabajos de labor de ingeniería del software, hitos de proyectos, productos de trabajo, y puntos de garantía de calidad, para permitir que las actividades del marco de trabajo se adapten a las características del proyecto del software y a los requisitos del equipo del proyecto. Las actividades de protección son independientes de cualquier actividad del marco de trabajo y aparecen durante todo el proceso.
- El tercer aspecto son los resultados obtenidos del proceso después de su ejecución y se reflejan mediante la salida de la información procesada.

2.5.2 Ciclo de vida del software

Las funciones primarias de un modelo de proceso de software consisten en determinar el orden de las fases involucradas en el desarrollo y evolución del software, y establecer los criterios de transición para avanzar de una fase a la próxima. El proceso de desarrollo de software requiere una metodología y un lenguaje propios. Así, los proyectos de software están divididos en fases. Al conjunto de fases y su secuencia se le denomina *ciclo de vida del software*, y comprende cuatro grandes fases: (Pressman, 2002; Sommerville, 2002; Luna, 2004)

- *Concepción*. Define el alcance del proyecto y desarrolla un caso de negocio.
- *Elaboración.* Define un plan del proyecto, especifica las características y fundamenta la arquitectura.
- *Construcción.* Crea el producto.
- *Transición.* Transfiere el producto a los usuarios.

El ciclo de vida del software describe todo el proceso del software de un sistema dado, desde su creación hasta su retiro (concepción, introducción, aceptación o crecimiento, madurez, saturación, obsolescencia y retiro). Así, se puede decir que el ciclo de vida del software es un tiempo determinado en el cual un producto de software se desarrolla y se usa, hasta que se retira. (SEI, 1987; Pressman, 2002)

2.6 MODELOS DEL PROCESO DEL SOFTWARE

El reconocimiento de la “crisis del software” a fines de 1960 y la noción de que el desarrollo de software como una disciplina de ingeniería, llevaron a visualizar el proceso de desarrollo de software con modelación. Un modelo del proceso del software es una representación simplificada de un proceso del software, representada desde una perspectiva específica. Por su naturaleza los modelos son simplificados, por lo tanto un modelo del proceso del software es una abstracción de un proceso real (Sommerville, 2002).

El proceso de software es un proceso complejo y variable que no puede describirse fácilmente mediante un modelo simple, por lo que los modelos más detallados del proceso de software aún son objeto de investigación. No obstante, es posible identificar diferentes modelos generales. Estos son: (Op. Cit.)

- Modelo clásico del software.
- Modelo de construcción de prototipos.
- Modelos de evolutivos de proceso del software.
- Modelo de desarrollo incremental.
- Modelo de desarrollo en espiral.
- Modelo de desarrollo rápido de aplicaciones (RAD).
- Modelo de métodos formales.
- Desarrollo basado en componentes.
- Modelado basado en Técnicas de cuarta generación.

2.6.1 Modelo clásico del software

Winston Royce (1970) desarrolló el modelo del ciclo de vida clásico, denominado modelo lineal secuencial que, debido al escalonamiento entre fases, se le conoce también como modelo en cascada. Este modelo representó un refinamiento en el estudio de proyectos de software al introducir la iteración entre las fases, junto con la restricción de proporcionar iteraciones sólo entre las fases sucesivas, de tal forma que se redujera el exceso de revisión que resulta de las iteraciones en fases múltiples y al proporcionar salidas para la fase de validación en el ciclo de vida del software. En este modelo resaltan dos aspectos importantes: la documentación y el aseguramiento de la calidad. Durante la fase de desarrollo, la documentación debe permitir la comunicación entre las personas involucradas, y en la fase de terminación del desarrollo, ésta debe ayudar al uso y mantenimiento del producto de software. El aseguramiento de la calidad involucra mediciones organizacionales, de diseño y analíticas para la planeación de la calidad, y completar los criterios de calidad tales como la exactitud, confiabilidad, amigabilidad con el usuario, mantenimiento, eficiencia y portabilidad. El modelo del ciclo de vida clásico se compone de cinco fases, las cuales se pueden observar en la figura 2.7.

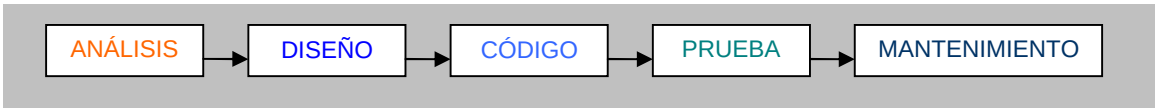


Figura 2.7. Modelo del ciclo de vida clásico.
Fuente: Pressman, 2002.

Análisis de los requisitos del software.
Esta actividad involucra establecer las fronteras del software a través de las cuales debe interactuar (interfaces) con el entorno operacional (requerimientos de nivel superior del software). El objetivo es lograr una clara comprensión de qué se trata el sistema y cuáles son los conceptos fundamentales. Además involucra una fuerte interacción con los usuarios para obtener los requerimientos del software, los cuales son: (Barra, 2007)

- Objetivos.
- Servicios.
- Funcionalidades.
- Interfaces.
- Restricciones.
- Rendimiento, etc.

Estos requerimientos o requisitos se fundamentan en un documento llamado *Especificación de Requerimientos del Software*, de un modo comprensible tanto para el usuario como para el desarrollador, con base en el cual, y mediante un análisis, se construye un modelo conceptual del sistema que especifica su comportamiento. (Op. Cit.)

Diseño.
Este proceso traduce los requerimientos en un modelo físico que incluye: estructura de datos, arquitectura del software, funciones y/o procedimientos, e interfaces. Esto significa incluir, aparte de las características del entorno operacional, las características de la plataforma de implementación, con lo cual se define cómo se logrará el comportamiento especificado a partir de los componentes de la implementación. (Op. Cit.)

Generación de código.
El diseño se traduce en un modelo comprensible para la máquina, esto es, en un conjunto de programas o unidades de programa, mediante un lenguaje de programación el cual tiene como resultado al código. (Barra, 2007)

Pruebas.
Se centra en los procesos lógicos internos del software asegurando que todas las sentencias se han comprobado. En esta etapa se realizan las pruebas para la detección de errores y asegura que la entrada definida produce resultados reales de acuerdo con los resultados requeridos. (Op. Cit.)

Mantenimiento.
Ésta es la fase de mayor duración del ciclo. El sistema se instala y se pone en marcha para su operación, durante la cual se corrigen errores más específicos y de menor envergadura, en comparación con la etapa anterior, con la finalidad de detectar errores no encontrados anteriormente. Es aquí donde se busca el mejoramiento de la implementación de unidades para lograr un mejor rendimiento. A su vez, es relevante revisar si es necesario aumentar las capacidades del sistema de acuerdo a los nuevos requerimientos, etc. El modelo secuencial lineal es el paradigma más antiguo y más extensamente usado en la administración de proyectos de software, porque refleja la forma básica en la cual muchas compañías afrontan la administración del software. Especifica que la progresión de un proyecto es lineal, comenzando con los requerimientos, continua a través del diseño, codificación, depuración, pruebas y mantenimiento. En la tabla 2.5 se indican las ventajas y desventajas que se presenta este modelo. Este modelo es la base para los otros modelos del ciclo de vida. (McConnell, 1997, Barra, 2007)

Tabla 2.5. Ventajas y desventajas del modelo lineal secuencial.

Ventajas	Desventajas
◆Es simple. ◆Describe el ciclo de vida del software mediante un enfoque sistemático y secuencial. ◆Es práctico para ciclos de productos en los que se tiene una definición estable del producto y metodologías y técnicas conocidas. ◆Facilita la localización de errores en las primeras etapas del proyecto a un bajo costo. ◆El modelo es útil para la planeación y los reportes. ◆Funciona si se dispone de personal poco cualificado o inexperto, porque presenta el proyecto con una estructura que ayuda a minimizar el esfuerzo inútil.	◆Requiere explícitamente del cliente, todos los requisitos necesarios para desarrollar el software, esto es difícil de lograr y provoca incertidumbre al comienzo de los proyectos. ◆Para un proyecto de desarrollo rápido, el modelo en cascada puede suponer una cantidad excesiva de documentación. ◆La disponibilidad del producto es mínima, es preciso esperar hasta el final del proyecto para tener un producto de software rentable y por tanto un error no detectado puede ser fatal. ◆Es inflexible cuando los requerimientos tienden a cambiar.

Fuente: McConnell, 1997.

2.6.2 Modelo de construcción de prototipos

Este modelo involucra el desarrollo de un programa funcional, sin embargo, el objetivo del desarrollo es establecer los requerimientos del sistema; a esto le sigue una reimplementación del software para lograr un sistema con calidad de producción. La construcción de un prototipo es una manera útil de extraer los requerimientos del cliente e identificar y eliminar las partes riesgosas de un proyecto. Comienza con la recolección de requisitos, con el trabajo del desarrollador y del cliente, se identifican los requisitos conocidos o las áreas del esquema en donde se requiere más información y se crea un diseño rápido, el cual se centra en una representación de los aspectos del software que serán visibles para el usuario/cliente y es evaluado por el cliente. En la tabla 2.6 se muestran las ventajas y desventajas que se tiene este modelo. (Pressman, 2002)

Tabla 2.6. Ventajas y desventajas del modelo de construcción de prototipos.

Ventajas	Desventajas
Es útil cuando en un proyecto los requerimientos cambian con rapidez, cuando el cliente es reacio a especificar el conjunto de los requerimientos, o cuando no se puede identificar de forma apropiada el área de aplicación. También es útil cuando no se está seguro de la arquitectura o los algoritmos adecuados a utilizar. Se pueden utilizar especialmente cuando existe una gran demanda en la velocidad del desarrollo. Genera signos visibles de progreso.	El cliente no ve un producto terminado, ve sólo un prototipo, pero no necesariamente tiene conocimiento de ello. Tiene la imposibilidad de conocer al comienzo del proyecto el tiempo que se tardará en crear un producto aceptable. Puede convertirse fácilmente en una excusa para realizar el desarrollo sin una administración adecuada Peligro de familiarización con malas elecciones iníciales.

Fuente: Pressman, 2002.

2.6.3 Modelos de evolutivos de proceso del software

Los modelos evolutivos soportan iteraciones. Son adaptables cuando los requisitos de gestión y de productos cambian conforme el avance del desarrollo del proyecto. Sus enfoques principales son dos, los cuales son:

- El modelo de desarrollo incremental.
- El modelo desarrollo en espiral.

Modelo de desarrollo incremental.
El enfoque de desarrollo incremental es una forma de reducir la repetición del trabajo en el proceso y da la oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema. Aplica secuencias lineales de forma escalonada mientras progresa el tiempo estipulado para el desarrollo del proyecto. Entrega el software en partes pequeñas, pero utilizables, llamadas incrementos. Cada incremento se construye sobre aquel que ya entregado. En la figura 2.8 se muestra gráficamente este modelo, mientras que en la tabla 2.7 se presentan las ventajas y desventajas que se tiene. (Mills, 1980)

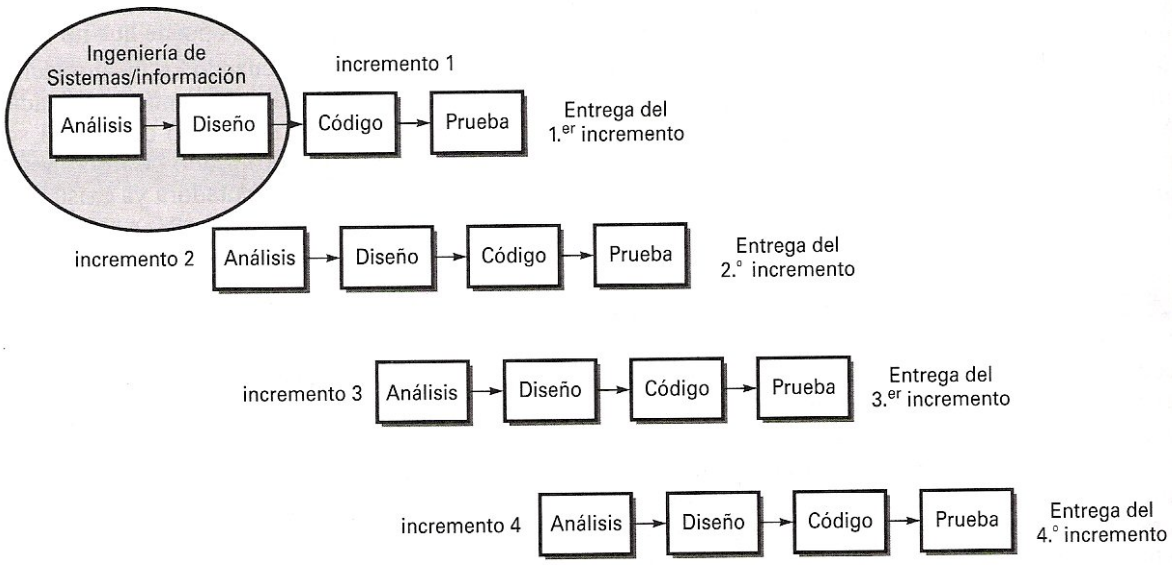


Figura 2.8. Modelo de desarrollo incremental.
Fuente: Pressman, 2002.

Tabla 2.7. Ventajas y desventajas del modelo de desarrollo incremental.

Ventajas	Desventajas
◆Los clientes no esperan hasta el fin del desarrollo para utilizar el sistema. Pueden empezar a usarlo desde el primer incremento. ◆Los clientes pueden aclarar los requisitos que no tengan claros conforme ven las entregas del sistema. ◆Se disminuye el riesgo de fracaso de todo el proyecto, ya que se puede distribuir en cada incremento. ◆Las partes más importantes del sistema son entregadas primero, por lo cual se realizan más pruebas en estos módulos y se disminuye el riesgo de fallos.	◆Cada incremento debe ser pequeño para limitar el riesgo (menos de 20 000 líneas). ◆Cada incremento debe aumentar la funcionalidad. ◆Es difícil establecer las correspondencias de los requisitos contra los incrementos. ◆Es difícil detectar las unidades o servicios genéricos para todo el sistema.

Fuente: Pressman, 2002.

Modelo de desarrollo en espiral.

Este modelo fue desarrollado por Boehm (1991). El diagrama de este modelo puede apreciarse en la figura 2.9. El modelo de espiral es un modelo de desarrollo de software que combina los modelos de cascada y de prototipos o los incluye como casos especiales. Cada ciclo incluye la misma secuencia de pasos para cada parte del producto y para cada etapa necesaria para obtenerlo. (Xuan, 1999)

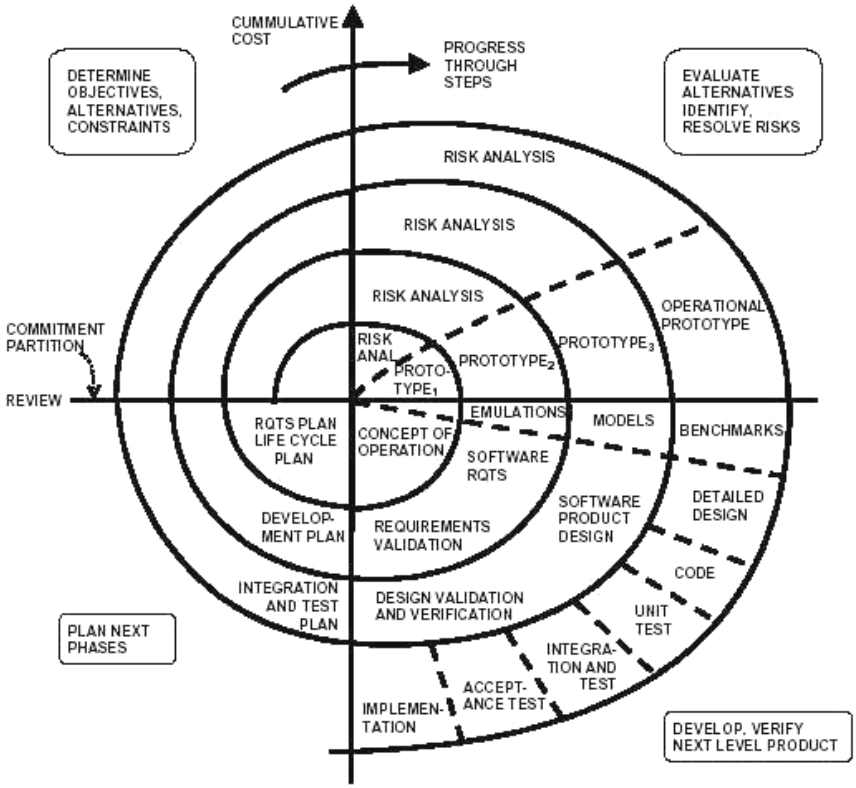


Figura 2.9. Diagrama del modelo en espiral.
Fuente: Boehm, 1998.

Cada ciclo de desarrollo se divide en cuatro regiones de tareas o actividades de marco de trabajo: (Boehm, 1998)

- *Definición de objetivos.* Se definen los objetivos. Se definen las restricciones del proceso y del producto. Se realiza un diseño detallado del plan administrativo. Se identifican los riesgos y se elaboran estrategias alternativas dependiendo de estos.

- *Evaluación y reducción de riesgos.* Se realiza un análisis detallado de cada riesgo identificado. Pueden desarrollarse prototipos para disminuir el riesgo de requisitos dudosos. Se llevan a cabo los pasos para reducir los riesgos.

- *Desarrollo y validación.* Se escoge el modelo de desarrollo después de la evaluación del riesgo. El modelo que se utilizará (cascada, sistemas formales, evolutivo, etc.) depende del riesgo identificado para esa fase.

- *Planeación*. Se determina si se continúa con otro ciclo. Se planea la siguiente fase del proyecto.

A su vez cada una de las regiones está compuesta por un conjunto de tareas que se adaptan a las características del proyecto que va a realizarse. Este modelo a diferencia de los otros toma en consideración explícitamente el riesgo, ésta es una actividad importante en la administración de proyectos de software. Cada ciclo tiene su propia etapa de validación que incluye a todas las personas que participan en el proyecto y a los usuarios futuros que vayan a ser afectados o a la

unidad organizacional. La validación incluye a todos los productos que se obtuvieron del ciclo, incluyendo la planeación para el siguiente ciclo y los recursos necesarios. (Boehm, 1998)

En la tabla 2.8 se presentan las ventajas y desventajas que tiene el modelo en espiral.

Tabla 2.8. Ventajas y desventajas del modelo en espiral.

Ventajas	Desventajas
◆Puede adaptarse y aplicarse a lo largo de la vida del software. ◆Es un modelo realista del desarrollo de sistemas y de software a gran escala. ◆Reduce los riesgos mediante la construcción de prototipos. ◆Permite a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto. ◆Es apropiado para el desarrollo de sistemas orientados a objetos.	◆Es difícil convencer a grandes clientes de que este enfoque es controlable. ◆Requiere una considerable habilidad para la evaluación de riesgo. ◆Todavía no se determina con absoluta certeza la eficacia de este paradigma.

Fuente: Boehm, 1998.

2.6.4 Modelo de desarrollo rápido de aplicaciones (RAD)

Este modelo fue desarrollado en el Pentágono, como un método para controlar los costos de proyectos de defensa. Consiste en dividir el proyecto en fases más cortas de análisis, diseño, construcción y evaluación. Luego de cada fase se evalúa la viabilidad del trabajo terminado, con una estimación de lo que falta. Resuelve el problema de la entrega rápida de código. (UEAFIT, 1998)

El modelo de desarrollo rápido de aplicaciones divide el proyecto en cuadros de tiempo, que son un conjunto de características definido que se promete entregar a los usuarios dentro de un marco de tiempo fijo, puede ser de 60 a 90 días. Primero se realiza un rápido análisis mediante los modelados de gestión y de datos al definir la fuente, el objetivo, el resultado y los procedimientos de la información y al refinar esta información gestionada para obtener un conjunto de objetos de datos, sus características y relaciones. Le sigue modelado de proceso, en el cual los objetos de datos definidos con anterioridad son transformados para lograr el flujo de información necesario para implementar una función de gestión y se crean las descripciones del proceso para añadir, modificar, suprimir o recuperar objetos de datos. Así, se usan herramientas de desarrollo de cuarta generación, para construir un prototipo funcional. El cliente revisa el prototipo y se solicitan modificaciones. El ciclo de codificación y refinación del prototipo se realiza varias veces, yendo en espiral, volviendo a analizar, diseñar y evaluar. Al final del cuadro de tiempo, se instala la aplicación resultante. En la tabla 2.9 se presentan las ventajas y desventajas que tiene el modelo de desarrollo rápido de aplicaciones. Este modelo es de uso común en el desarrollo de software de inteligencia artificial (IA) en que el cliente no puede formular una especificación de requerimientos detallada y donde la adecuación es el objetivo de los diseñadores. (Pressman, 2002; UEAFIT, 1998)

Tabla 2.9. Ventajas y desventajas del modelo de desarrollo rápido de aplicaciones.

Ventajas	Desventajas
◆ Produce código muy temprano. ◆ Involucra al usuario en la creación de prototipos. ◆ Es útil cuando el cliente no puede formular una especificación de requerimientos detallada. ◆ Es apropiado para proyectos pequeños, en los cuales cada una de las funciones principales puedan completarse en menos de tres meses.	◆ Al producir código muy temprano, se tiende a abandonar todas las actividades de análisis y diseño previas. ◆ Los cuadros de tiempo hacen difícil el enfoque de realizar componentes de código reutilizables. ◆ En proyectos grandes es necesario disponer de un gran número de personas. ◆ Requiere de alto compromiso en tiempo. ◆ En general, no es modularizable. ◆ Es de baja reutilización de componentes. ◆ No es apropiado cuando existen riesgos tecnológicos altos o alta interoperatividad con programas ya existentes.

Fuente: Pressman, 2002; EAFIT, 1998.

2.6.5 Modelo de métodos formales

Se dice que un método es formal si tiene una base matemática estable, dada por un lenguaje de especificación formal. Este lenguaje contiene esquemas, estructuras en forma de cuadro que muestran las variables y que especifican las relaciones entre estas variables. Al ser la matemática discreta[2] el fundamento de los métodos formales, para manejar la información, induce a los modelos a que utilicen uno de los dos dominios siguientes: (Marcianiak, 1994; Pressman, 2002)

- *Dominio sintáctico*. Utiliza una simbología que sigue estrechamente a la notación de conjuntos y al cálculo de predicados.

- *Dominio semántico*. Capacita al lenguaje para expresar requisitos de forma concisa.

Un lenguaje de especificación formal describe un conjunto de sucesos que provocan la producción de un estado dentro del sistema , otra relación formal representa todas aquellas funciones que se producen dentro de un cierto estado, al intersectar estas dos relaciones se proporciona una indicación de los sucesos que darán lugar a que se produzcan funciones especificas. Por tanto, los métodos formales tienen tres conceptos fundamentales: (Pressman, 2002)

- *Los invariantes de datos*. Son condiciones que son ciertas a lo largo de la ejecución del sistema que contiene una colección de datos.

- *El estado*. Son los datos almacenados a los que accede el sistema y que son alterados por él.

[2] La matemática discreta es la notación y práctica asociada a los conjuntos y a la especificación constructiva, a los operadores de conjuntos, a los operadores lógicos y a las sucesiones.

- *La operación*. Son una acción que tiene lugar en un sistema y que lee o escribe datos en un estado. Una operación está asociada con dos condiciones: una precondición y una poscondición.

Un método formal proporciona marcos de referencia y la posibilidad de describir las propiedades del software, tales como consistencia, completitud, especificación, implementación y corrección, de manera precisa. Los desarrolladores pueden especificar, desarrollar y verificar los programas de computadora de forma sistemática, en lugar de hacerlo ad hoc. (Marcianiak, 1994)

En la tabla 2.10 se presentan las ventajas y desventajas que tiene el modelo de desarrollo rápido de aplicaciones.

Tabla 2.10. Ventajas y desventajas de los modelos formales.

Ventajas	Desventajas
Son útiles para sistemas de seguridad y sistemas críticos. Conjunta especificaciones sin ambigüedades porque usa notación rigurosa. Tiene buen nivel de manejo de Lógica y Algebra.	Dificultan validación con el cliente. No es adaptable, tiene problemas con la combinación con otras técnicas semiformales. El costo de arranque puede ser muy alto.

Fuente: Pressman, 2002.

2.6.6 Desarrollo basado en componentes

El desarrollo basado en componentes está fundado en el modelo en espiral, el cual es evolutivo, iterativo e incremental y en las tecnologías de objetos, al configurar aplicaciones de componentes de software. Además enfatiza la reutilización. Un esquema de este modelo se muestra en la figura 2.10. Los componentes son unidades de software, llamados clases, se rigen por los principios del paradigma de orientación a objetos: unificación y comportamiento de datos e identidad y encapsulamiento. Las clases creadas se almacenan para su posterior reuso. Al iniciar otro proyecto se revisa la biblioteca de clases o diccionario de datos o repositorio, si ya existe la clase necesaria, se usa. En caso de que la clase no se encuentre, se utilizan técnicas orientadas a objetos para desarrollarla. La especificación de un componente está formado por un conjunto de interfaces que describen el comportamiento del componente. (Pressman, 2002; Vignaga y Perovich, 2005)

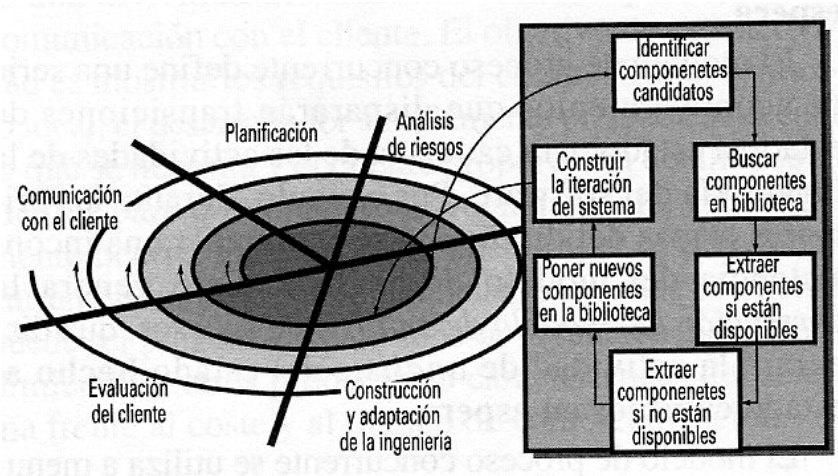


Figura 2.10. Diagrama del modelo de desarrollo basado en componentes.
Fuente: Pressman, 2002.

Las interfaces describen este comportamiento en función de la especificación. Las dependencias entre componentes son dependencias de uso de interfaces, no son dependencias directas sobre el componente. Si la especificación del componente es rigurosa permite que éste pueda ser reemplazable. Cada cliente de un componente depende exclusivamente de la especificación del componente y no de su implementación. Esta importante separación es la clave para reducir el error de acoplamiento y mejorar el manejo de las dependencias. (Pressman, 2002)

Proceso Unificado para Desarrollo de Software.

El Proceso Unificado para Desarrollo de Software (Rational Unified Process, siglas en inglés RUP) es un proceso de ingeniería de software que representa un conjunto de modelos de desarrollo basados en componentes, es decir, es una metodología de desarrollo orientada a objetos y es habilitada para Web. Proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo de software. Tiene como meta asegurar la producción de software de alta calidad que cumpla con las necesidades de los usuarios finales, dentro de un calendario predecible y dentro del presupuesto. Provee un curso de acción, plantillas y ejemplos para los aspectos y las etapas de desarrollo de software(como ciclos, fases, flujos de trabajo, mitigación de riesgos, control de calidad, administración del proyecto y control de configuración) mediante herramientas de ingeniería de software que combinan los aspectos procedimentales de desarrollo (tales como etapas definidas, técnicas y prácticas) con otros componentes de desarrollo (tales como documentos, modelos, manuales y código) dentro de un entorno unificado. El RUP se maneja a través de casos de uso, para reemplazan la antigua especificación funcional tradicional y se centra en la arquitectura. Los casos de uso guían el desarrollo de la arquitectura y la arquitectura se realimenta en los casos de uso, los dos juntos permiten conceptualizar, gestionar y desarrollar adecuadamente el software. (Rumbaugh et al., 1999; Pressman, 2002)

Son cuatro las fases de desarrollo del RUP, cada una de las cuales está organizada en un cierto número de iteraciones separadas que deben satisfacer criterios definidos antes de que se tome la siguiente fase: (Prince, 2003)

- *Fase de inicio o concepción.* Los desarrolladores definen el enfoque del proyecto y el caso de uso.

- *Fase de elaboración.* Los desarrolladores analizan las necesidades del proyecto a profundidad, definen sus características y establecen la arquitectura.

- *Fase de construcción*. Los desarrolladores crean el diseño de la aplicación y el código fuente.

- *Fase de transición*. Los desarrolladores liberan el sistema para los usuarios. RUP proporciona un prototipo cada vez de que termina cada iteración.

Además esta estructura es capaz de cubrir a vendedores y desarrolladores de herramientas para soportar la automatización del proceso, soportar flujos individuales de trabajo, para construir los diferentes modelos e integrar el trabajo a través del ciclo de vida y a través de todos los modelos. El Proceso Unificado usa el Lenguaje de Modelado Unificado (UML, por sus siglas en ingles Unified Model Language) en la preparación de todos los planos del sistema. De hecho, UML es una parte integral del Proceso Unificado, fueron desarrollados al mismo tiempo. (Rumbaugh et al., 1999)

Lenguaje de modelado unificado.
El UML permite a un ingeniero del software expresar un modelo de análisis al utilizar una notación de modelado con reglas sintácticas, semánticas y prácticas. Para analizar un sistema mediante UML se representa mediante cinco vistas diferentes que lo describen desde diferentes perspectivas, a su vez, cada vista se representa mediante un conjunto de diagramas. Las vistas son: (Alhir, 1998; Pressman, 2002)

- *Vista de usuario*. Representa el producto desde el aspecto de los actores, que son los usuarios. Para modelar esta vista se emplea el caso de uso. Es una representación del análisis, que describe un escenario de uso desde la perspectiva del usuario.

- *Vista estructural*. Los datos y la funcionalidad se muestran desde dentro del sistema, es decir, modela la estructura estática (clases, objetos y relaciones).

- *Vista del comportamiento*. Esta parte del modelo del análisis representa los aspectos de comportamiento del sistema. También muestra las interacciones o colaboraciones entre los diversos elementos estructurales descritos en las vistas anteriores.

- *Vista de implementación*. Los aspectos estructurales y de comportamiento se representan aquí tal y como van a ser implementados.

- *Vista del entorno*. Los aspectos estructurales y de comportamiento en el que el sistema a implementar se representa.

En general, el modelo de análisis de UML se centra en las vistas del usuario y estructural. El modelo de diseño de UML se dirige más a las vistas del comportamiento y del entorno. (Alhir, 1998)

Después de revisar lo que conlleva el desarrollo basado en componentes en la tabla 2.11 se presentan las ventajas y desventajas del este modelado.

Tabla 2.11. Ventajas y desventajas del modelo de desarrollo basado en componentes.

Ventajas	Desventajas
◆Prioriza el comportamiento esperado del sistema. ◆Evoluciona fuera del proceso. ◆Notación que permite desarrollar de manera consistente. ◆Enfatiza la reusabilidad.	◆Alto costo ◆Puede contener ambigüedad

Fuente: Pressman, 2002.

2.6.7 Modelado basado en técnicas de cuarta generación

El paradigma técnicas de cuarta generación (T4G) hace posible especificar el software usando formas de lenguaje especializado o notaciones graficas que describa el problema a resolver en de forma que el cliente lo entienda. Un entorno para el desarrollo de software que soporte el paradigma T4G puede incluir todas o algunas de las siguientes herramientas: lenguajes no procedimentales de consulta a bases de datos, generación de informes, manejo de datos, interacción y definición de pantallas, generación de códigos, capacidades gráficas de alto nivel y capacidades de hoja de cálculo, y generación automatizada de HTML y lenguajes similares utilizados para la creación de sitios Web usando herramientas de software avanzado. (Pressman, 2002) Se describe el paradigma T4G para la ingeniería de software en la figura 2.11.

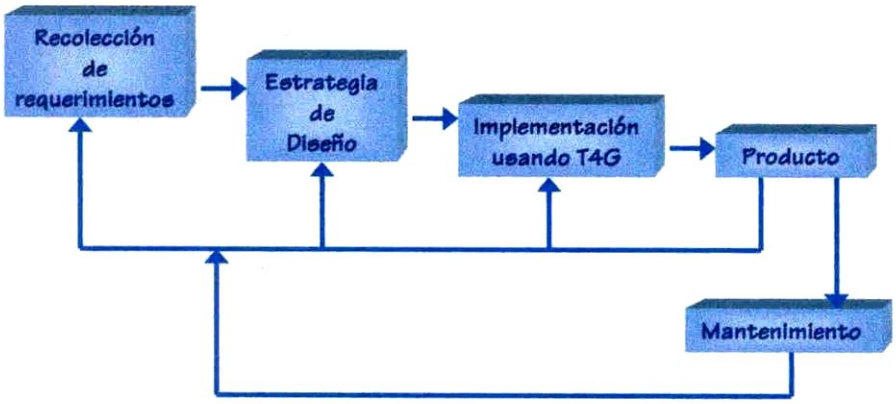


Figura 2.11. Un Paradigma T4G.
Fuente: http://148.202.148.5/cursos/cc321/fundamentos/unidad1/T4G.htm

Este modelado comienza con la recolección de los requisitos del producto de software, parte muy importante en todos los modelados para el desarrollo de proyectos de software. Es relevante que al utilizar las herramientas de T4G se realice una prueba completa, se desarrolle con sentido una documentación, se ejecute el resto de las actividades de integración que también son requeridas por otros paradigmas de ingeniería del software y se diseñe en vista de un mantenimiento ágil. (Op. Cit.)

El modelado mediante técnicas de cuarta generación tiene ventajas y desventajas las cuales se muestran en la tabla 2.12.

Tabla 2.12. Ventajas y desventajas del modelo basado en técnicas de cuarta generación.

Ventajas	Desventajas
◆Facilita la realización de especificaciones a alto nivel. ◆Reduce el tiempo de desarrollo. ◆Ofrece una solución fiable a muchos problemas del software mediante su uso junto con las herramientas de ingeniería de software asistida por computadora (CASE) y los generadores de código.	◆El Código resultante puede ser ineficiente. ◆Las herramientas de uso corriente pueden ser más fáciles de usar que las propuestas por T4G. ◆El mantenimiento es difícil de realizar si no se plantea correctamente desde el diseño. ◆En sistemas grandes, aunque se usen T4G se debe hacer análisis, diseño y pruebas.

Fuente: Pressman, 2002.

Mediante la aplicación de T4G en un proyecto pequeño se puede pasar de la adquisición de los requisitos a la implementación, pero si el proyecto es de grandes dimensiones se requiere de un diseño para no acarrear dificultades como: poca calidad, mantenimiento pobre y el rechazo del producto por parte del cliente. (Pressman, 2002)

3. ADMINISTRACIÓN DE PROYECTOS

3.1 PROYECTO DE DESARROLLO DEL SOFTWARE

El punto de partida de todo proyecto surge de la existencia de una necesidad y el objetivo es la respuesta a la formulación de dicha necesidad (Gaddis, 1991).

3.1.1 Definición de proyecto de software

Un proyecto de software es un conjunto planificado de acciones tendientes a resolver exitosamente una situación problemática o una demanda planteada, mediante una cantidad de recursos humanos y materiales, los cuales restringen al mismo proyecto, un presupuesto, un tiempo para su realización, ya que cada proyecto tiene un inicio y un final definido, y el cumplimiento de las especificaciones predeterminadas. Tiene la misión de crear un producto o servicio de software diferente de los productos o servicios similares, y se considera con una evolución progresiva al derivar en pasos continuos e incrementales, es decir, que el proceso del producto o servicio se basa en un modelo de desarrollo. A grosso modo consta de 4 fases, mismas que se ilustran en la figura 3.1, las cuales son: (Guido, 2003; PMI, 2004)

- La primera fase en el ciclo de vida del proyecto consiste en descubrir una necesidad, un problema o una oportunidad, pudiendo suceder que el cliente pida propuestas a individuos, a un equipo de proyecto o a empresas (contratistas) para atender la necesidad o resolver el problema. Corresponde a la reunión de los requerimientos y a la estimación del proyecto.

- La segunda fase del ciclo de vida del proyecto reside en encontrar una solución a la necesidad o al problema. Se plantea el objetivo, el modelo de desarrollo y el plan del proyecto de acuerdo al análisis de los requerimientos.

- La tercera fase en el ciclo de vida del proyecto radica en implementar el modelo de desarrollo del software elegido, lo anterior también se conoce como ejecución del proyecto.

- La cuarta fase, que es la final, consiste en terminar el proyecto y evaluar la ejecución del proyecto.

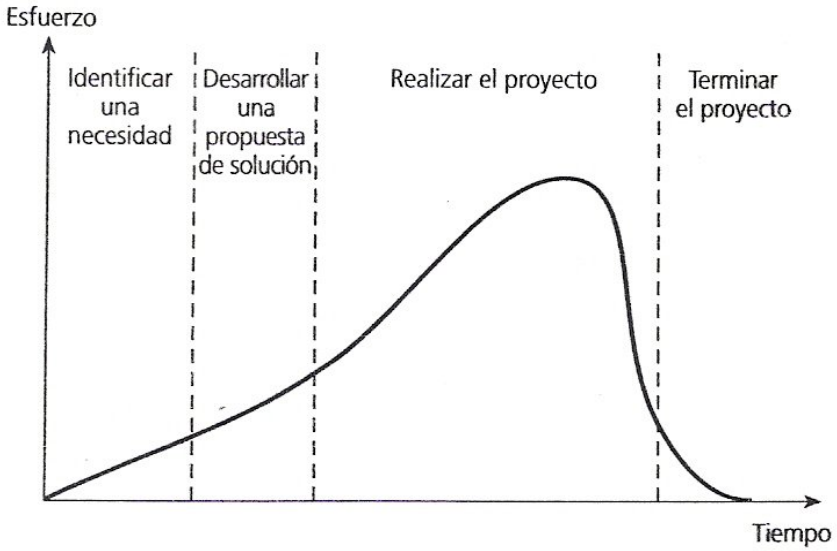


Figura 3.1 Ciclo de vida del proyecto.
Fuente: Guido, 2003.

3.1.2 Objetivo del proyecto de software

El objetivo del proyecto es el primer paso en el desarrollo de un proyecto de software, este deber detallarse con base en el resultado esperado o en el producto final, en la etapa de la definición del problema. El objetivo debe ser alcanzable, especifico, medible y establecido de común acuerdo entre ambas partes (el cliente y quien lo desarrolle), para que cada uno pueda percibir el logro del objetivo del proyecto cuando éste se alcance. Idealmente el objetivo del proyecto debe ser claro y conciso al inicio del proyecto. Sin embargo, en ocasiones es necesario modificarlo de acuerdo al avance. El gerente del proyecto y el cliente tienen que estar de acuerdo con todos los cambios al objetivo inicial. Debe tenerse en cuenta que cualquier modificación puede afectar el alcance del trabajo, la fecha de terminación y el costo final (Pressman, 2002; Guido, 2003).

3.1.3 Tipos de proyectos de software

Los proyectos de software comprenden desde el desarrollo del concepto, desarrollo de una nueva aplicación, mejoras de aplicaciones, mantenimiento de aplicaciones y de reingeniería. Los tipos de proyectos son: (Op. Cit.)

- *Proyectos de desarrollo de concepto.* Se inician para explorar algún nuevo concepto de negocios o aplicación de alguna nueva tecnología.
- *Proyectos de desarrollo de una nueva aplicación.* Se aceptan como consecuencia del encargo de un cliente especifico.
- *Proyectos de mejoras de aplicaciones.* Ocurren cuando un software existente sufre grandes modificaciones de su funcionamiento, rendimiento o interfaces que son observables por el usuario final.

- *Proyectos de mantenimiento de aplicaciones.* Corrigen, adaptan o amplían un software existente en métodos que no son obvios de forma inmediata para el usuario final.

- *Proyectos de reingeniería.* Se lleva a cabo con la intención de reconstruir un sistema existente en su totalidad o en parte.

3.2 ADMINISTRACIÓN DE PROYECTOS DE DESARROLLO DE SOFTWARE

3.2.1 Definición de la administración de proyectos de software

El proceso de desarrollo de programas de computadora se centra en la ingeniería de software, no es claro cuando se habla de como realizar las siguientes acciones: (Jalote, 1997)

- La administración de los recursos en las distintas actividades del proceso.
- La división del trabajo dentro de cada fase.
- El aseguramiento de que cada fase sea hecha apropiadamente.
- El reconocimiento del riesgo del proyecto y su mitigación.
- La especificación de los tiempos de desarrollo.

La administración de proyectos busca resolver estas situaciones. Se define como el proceso de planear, organizar, proveer de personal, monitorear, controlar y dirigir un proyecto de software. Abarca todo el ciclo de vida del proyecto desde el inicio hasta su terminación, es un esfuerzo de equipo y está altamente relacionado con la gente y el proceso. Para ejecutar las acciones establecidas en el programa del proyecto, se tienen 4 actividades, las cuales son: (IEEE, 1987; Kimmons, 1990; Jalote, 1997; Johnson, 1999; PMI, 2004)

- Colaboración con el cliente en la definición y determinación de los objetivos del proyecto y establecerlos en términos de tiempo, costo y parámetros de rendimiento.

- Planeación del proyecto a través de la identificación de las tareas a realizar, los recursos y las personas idóneas para la realización de dichas actividades.

- Dirección y coordinación de todos los recursos empleados para la realización de dicho proyecto.

- Combinar de manera correcta las actividades para no realizar duplicidad de tareas.

Se dice que un proyecto puede formar parte de un conjunto o portafolio de proyectos, donde cada proyecto se considera como una tarea y todos estos proyectos son diferentes en tiempo o tamaño.

3.3 ESTRUCTURA DE LA ORGANIZACIÓN

La administración de los recursos humanos del proyecto, es el proceso requerido para hacer de manera más efectiva el uso de las personas involucradas en el proyecto, lo que incluye identificar, documentar y asignar roles, responsabilidades y relaciones, ya sea a una persona o a un equipo.

3.3.1 Participantes del proyecto

Los individuos participantes de un proyecto de software se encuentran en alguna de estas cinco categorías: (Pressman, 2002; Luna, 2004; PMI, 2004)

- *Gestores superiores.* Son los personajes que definen los aspectos de negocios que a menudo tienen una significativa influencia en el proyecto. Dentro de estos aspectos destacan el estudio de viabilidad y el estudio económico, que se desarrollan junto con el líder de proyecto, definición de requisitos del proyecto, identificación de necesidades de información e intercambio, identificación de procesos, elaboración de la documentación funcional, diseño del modelo de entidades, diseño del flujo de datos y diseño lógico de archivos, las tareas de diseño se realizan a la par de los gestores del proyecto. Se pueden dividir en dos tipos, los cuales son:

  - Organización ejecutante. Son las personas que trabajan en una empresa que están directamente involucrados en el trabajo del proyecto.

  - Cliente. Es el individuo o grupo interno o externo a la organización ejecutante que otorga los recursos financieros, en efectivo o al cambio para el proyecto.

- *Gestores del proyecto.* Son los que deben planificar, motivar, organizar y controlar a los profesionales que realizan el trabajo de software. Un personaje muy importante dentro de este grupo es el administrador, gerente o líder del proyecto, quien conforma al equipo de trabajo y se encarga de coordinar, dirigir y controlar la totalidad del proyecto.

- *Profesionales.* Tienen a su cargo proporcionar las capacidades técnicas necesarias para la ingeniería de un producto o aplicación.

- *Clientes.* Los clientes especifican los requisitos para la ingeniería del software y otros elementos que tienen menor influencia en el resultado. Existen dos tipos de clientes para un proyecto: los internos y los externos. Los primeros se consideran cuando el proyecto es una subfase y el producto resultante es una entrada para la subfase siguiente. Los proyectos externos a la organización pueden ser dictaminados por una empresa, gobierno, compañía, etc.

- *Usuarios finales.* Son quienes interaccionan con el software una vez que se entrega para la producción.

Estructura del equipo.
Los participantes del proyecto deben tener un equipo estructurado. La mejor estructura de equipo depende del estilo de gestión de una organización, el número de personas que componen el equipo, sus niveles de preparación y la dificultad general del problema.

El balancear el equipo de trabajo, así como la cobertura, son dos de los aspectos más importantes en la excelencia de los equipos, la ausencia de balance hace al equipo vulnerable. Los atributos para el éxito en la administración de proyectos de software, respecto a la selección de personal, que requieren más atención son: (Royce, 1998)

- *Adquisición de talento.* Pocas decisiones son tan importantes como las de contratación.
- *Habilidades de comunicación con el cliente.* Para tener éxito es preciso evitar relaciones adversas con los clientes.
- *Habilidades de tomador de decisiones.* Es relevante elegir con el mejor criterio la opción adecuada.
- *Habilidades de construcción de equipos.* El equipo de trabajo requiere que el administrador establezca confianza, motivación, elimine a los inadaptados, y consolide diversas opiniones en una sola dirección. También es relevante adecuar las tareas a las habilidades y motivaciones del personal disponible y capacitar y actualizar a éste mismo personal para hacer una mejor organización en el largo plazo.

3.4 PLANEACIÓN DEL PROYECTO DE SOFTWARE

Cuando un cliente solicita el desarrollo de un producto de software además de describir problema, pregunta el costo de la solución del problema y en cuanto tiempo estará listo. Para resolver esta situación se requiere un perfecto entendimiento del problema, definir los objetivos de la solución y de hacer una buena planeación.

Definición de planeación de un proyecto de software.
La planeación auxilia a conocer el marco del sistema, incluye la selección del ciclo de vida del software apropiado, el establecimiento de las políticas, procedimientos y procesos necesarios para lograr los objetivos. Implica una estimación para determinar cuánto dinero, esfuerzo, recursos y tiempo supondrá construir un sistema o producto específico de software. El plan de proyecto es el documento donde se concreta el objetivo del proyecto y que elementos o actividades de trabajo se necesitan realizar para que se logre, es decir, se asignan las partidas de trabajo[3] (Futrell, 2002; Guido, 2003).

3.4.1 Estructura común del proceso

Para una mejor planeación, se puede usar la estructura común del proceso (ECP), la cual es una herramienta que considera todas las funciones que se deben tratar dentro de un proyecto de desarrollo de software. Se dice que tales funciones se examinan dentro de un conjunto de actividades estructurales[4] definidas para una organización de software, las cuales se adaptan tanto al tipo de proyecto como al paradigma de ingeniería de software que resulte mejor para el proyecto, éstas son: (Pressman, 2002)

[3] La partida de trabajo es cada una de las piezas en la cuales su puede subdividir a un proyecto para su desarrollo.
[4] Las actividades estructurales son aquellas que mediante diferentes conjuntos de tareas (hitos, productos de trabajo y puntos de garantía de la calidad), se adaptan a las características de un proyecto de software dado.

- *Comunicación con el cliente.* Contempla las tareas requeridas para establecer la obtención de requisitos eficiente entre el desarrollador y el cliente.

- *Planeación.* Revisa tareas requeridas para definir los recursos, la planificación temporal del proyecto y cualquier información relativa a él.

- *Gestión del riesgo.* El riesgo como tal puede o no ocurrir, si el riesgo se convierte en una realidad habrán consecuencias no deseadas o pérdidas.

- *Gestión de la calidad.* El proceso de mejora continua puede aplicarse en todas las etapas del proyecto de software. Al revisar estas acciones se asegura el cumplimiento del plan del proyecto en los mejores términos.

- *Ingeniería.* Se examina a los trabajos necesarios para construir una o más representaciones de la aplicación.

- *Construcción y entrega.* Reconoce las tareas obligadas para construir, probar, instalar y proporcionar asistencia al usuario.

- *Evaluación del cliente*. Conjunta las labores requeridas para obtener información de la opinión del cliente basadas en la evaluación de las representaciones de software creadas durante la fase de ingeniería e implementadas durante la fase de instalación.

- *Análisis postmortem.* Se analizan las tareas de recopilación de información de cada una de las actividades anteriores para archivo y su uso posterior para otros proyectos.

Se dice que aunque todas las actividades estructurales forman un todo. Las actividades de planeación, gestión de riesgo y de calidad son propias de la administración de proyectos de software. Las actividades de comunicación con el cliente, ingeniería, construcción y entrega, y evaluación del cliente corresponden a la ingeniería de software. Por último, el análisis postmortem es un conjunto de actividades, el cual no es considerado en muchos casos por los gerentes del proyecto, pero tiene una gran relevancia, debido a que es el fundamento para establecer la base de datos histórica de los proyectos desarrollados en una empresa. Para realizarlo se requiere el registro detallado de todas las demás actividades, mediante el uso de métricas y documentación. La manera de usar una ECP consiste en realizar una matriz con las actividades estructurales y las tareas de trabajo de ingeniería de software. Con la información agrupada mediante esta herramienta se prosigue a completar la planeación mediante la estimación de costos y de recursos, así como de la planeación temporal. (Royce, 2002)

3.4.2 Estimación de Costos

Se dice que un aspecto relevante en la planeación y gestión del proyecto de software es comprender el probable costo del mismo. Un costo estimado de forma correcta al comienzo de la vida de un proyecto brinda el conocimiento necesario para conocer la cantidad de desarrolladores adecuada y preparar el grupo correspondiente para que esté disponible en el momento que se soliciten. Un problema crítico en la estimación de costos del software es la falta de casos de proyectos bien documentados que usen el enfoque de desarrollo iterativo, además, es difícil establecer un conjunto homogéneo de datos del proyecto dentro de una organización, y aún más difícil entre diferentes organizaciones, esto se debe a que en la industria del software, las métricas y unidades de medida pueden estar definidas de forma

inconsistente, lo que provoca que los datos obtenidos sean altamente dudosos en cuanto a consistencia y comparación. La mayoría de los modelos de costos pueden sintetizarse en una función de cuatro básicos, estos son: (Op. Cit.)

- La factibilidad.
- El tamaño.
- El problema.
- El proceso.

Los modelos de tamaño, problema y proceso son modelos que descomponen el problema y lo vuelve a definir como un conjunto de pequeños problemas para que sean más manejables, a continuación se describen estos tres modelos y el de factibilidad.

Factibilidad.
Es el aspecto que revisa si es posible y recomendable realizar un cierto producto de software. Tiene cuatro dimensiones, las cuales son: (Putman, 1992; Pressman, 2002)

- *Factibilidad técnica.* Plantea la revisión para conocer si se tiene el suficiente potencial científico para realizar el proyecto. Toma en cuenta la complejidad del proyecto mediante medidas subjetivas (tales como los factores de ajuste de la complejidad del software) y el grado de incertidumbre estructural, es decir, el grado en el cual los requisitos se definen, la facilidad con la que pueden subdividirse en funciones y la naturaleza jerárquica de la información que debe procesarse.

- *Factibilidad financiera.* Revisa si la empresa de software o el cliente tienen la posibilidad de poder asumir los costos de desarrollar cierto producto de software.

- *Factibilidad temporal.* Contempla los aspectos de cuestionarse si se podrá terminar el proyecto en el tiempo establecido.

- *Recursos.* Establece si existen los suficientes recursos o la posibilidad de adquirirlos para desarrollar el producto de software. Tales recursos se ilustran en la figura 3.2 y se describen a continuación:

  - *Recursos humanos*. Se dice que la gente adecuada que trabaja en la actividad correcta.

  - *Entorno de ingeniería del software (EIS).* Incorpora herramientas de hardware y software proporcionan la infraestructura de soporte al esfuerzo de desarrollo. Se debe determinar para el hardware y el software la fecha cronológica en la cual se requiere el recurso y el tiempo durante el cual es aplicado, a su vez se debe verificar la disponibilidad de estos recursos y especificar cada elemento.

  - *Componentes de software reutilizables*. Son los bloques de software que pueden reducir drásticamente los costos de desarrollo y acelerar la entrega.

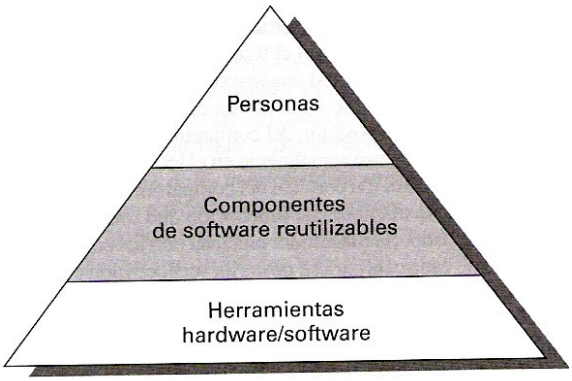


Figura 3.2. Recursos del proyecto.
Fuente: Pressman, 2002.

Tamaño.
Se refiere a una producción cuantificable del proyecto de software. Se puede estimar el tamaño de cuatro formas diferentes (Putman, 1992), las cuales son:

- *Tamaño en lógica difusa.* Utiliza técnicas aproximadas de razonamiento. Se identifica el tipo de aplicación, se establece su magnitud en una escala cuantitativa y se refina la magnitud dentro del rango original.
- *Tamaño en punto de función.* Se desarrollan estimaciones características del dominio de la información (tales como: puntos de función).

- *Tamaño de componentes estándar.* Se supone al software como un todo conformado de cierto número de componentes genéricos para un área en particular de aplicación, se pueden utilizar las líneas de código para medir el tamaño del software.

- *Tamaño de cambio.* Se utiliza cuando un proyecto comprende la utilización de software existente que se debe modificar de alguna manera como parte del proyecto.

Problema.
Se utilizan las estimaciones por proceso y por tamaño, es decir, la estimación se realiza mediante líneas de código (LDC) o puntos de función (PF). Para una orientación de líneas de código se comienza con un enfoque limitado, con líneas de código y en este estado se descompone en funciones que se puedan estimar individualmente. Se asume que las funciones pueden descomponerse en subfunciones que se asemejan a entradas de una base de datos histórica. En cambio desde la perspectiva de puntos de función se estiman cada una de las características del dominio de la función (tales como: entradas, archivo de datos o interfaces externas) y los valores de ajuste de la complejidad que se revisan en el siguiente capítulo. Independientemente del la forma de la variable de estimación se comienza por estimar un rango de valores para cada función o valor del dominio. Para ello se requiere calcular el valor esperado (VE) del tamaño del software, el cual se calcula mediante la ecuación 3.1. (Pressman, 2002)

$$VE = \frac{S_{opt} + 4S_m + S_{pess}}{6} \qquad (3.1)$$

Donde $\mathbf{S_{opt}}$ es la media ponderada de las estimaciones optimistas, $\mathbf{S_m}$ es la media ponderada de las estimaciones probables y $\mathbf{S_{pass}}$ es la media ponderada de las estimaciones pesimistas. Al obtener el valor esperado se procede a comparar el valor con los datos históricos de las métricas de productividad de línea base (tales como: LDC/persona-mes o PF/persona-mes). De acuerdo a esto se realizan cálculos, es decir se extrae el costo o el esfuerzo y se toman decisiones. (Op. Cit.)

Proceso.
La estimación basada en el proceso comienza con un esbozo de las funciones del software obtenidas a partir del ámbito del proyecto. El proyecto se descompone en un conjunto de actividades o tareas y el esfuerzo requerido para llevar a cabo la estimación de cada tarea, se utiliza la herramienta de marco del trabajo común del proceso, la cual se revisa más adelante. Se calculan los costos y el esfuerzo de cada función (tales como: persona-mes) y la actividad del proceso del software. (Op. Cit.)

Modelos empíricos.
Todos los datos que se obtienen mediante la aplicación de los modelos anteriores se registran en tablas. En la actualidad existen varios modelos empíricos de estimación de costos que estiman los valores LDC o PF por medio de los modelos empíricos de estimación, los cuales se clasifican en los tres diferentes tipos que son descritos en breve: (Pfleeger, 2002; Pressman, 2002; Boehm, 1981)

- *Juicio experto.* En su mayoría son técnicas informales, basadas en el conocimiento de los gerentes en proyectos similares en cuanto a: competencia, experiencia, objetividad y percepción. En general, dicha estimación se hace mediante una conjetura informada acerca del esfuerzo necesario para construir un sistema completo o sus subsistemas. La estimación completa se calcula mediante un análisis descendente (en inglés top-down) o ascendente (en inglés bottom-up) de lo preciso.

- *Métodos algorítmicos.* Estos modelos expresan la relación entre el esfuerzo y los factores que influyen. Los modelos se describen por medio de ecuaciones, donde el esfuerzo es la variable dependiente, y varios factores (tales como experiencia, dimensión y tipo de sistema) son las variables independientes. La mayoría de estos modelos reconocen que la dimensión del proyecto es el factor de mayor influencia en la ecuación 3.2 se expresa al esfuerzo en términos de una dimensión S estimada del sistema , constantes a, b y c, un vector de factores de costo,X[x1…xn], y un multiplicador m de ajuste basado en estos factores.

$$E = \left(a + bS^{C}\right)m\left(X\right) \qquad (3.2)$$

  En otras palabras, el esfuerzo está determinado principalmente por la dimensión del sistema propuesto, ajustado por los efectos de otras varias características del proyecto, proceso, producto o recurso. Algunos modelos son: modelo de Watson, Modelo de Bailey y modelo COCOMO (por las siglas en inglés de Constructive Cost Model o Modelo Constructivo de Costo) y COCOMO II. Éstos últimos de los modelos más difundidos y mejor documentados y estiman la complejidad del software.

- *Métodos de aprendizaje.* Se considera las máquinas que aprenden pueden ayudar a obtener estimaciones de producción. Estos métodos pueden utilizar una red neuronal de

tal forma que representen las variadas actividades involucradas en la generación de un producto de software.

Por último es importante comentar la necesidad de sustentar la técnica elegida de estimación con los valores arrojados por otra y además la estimación del proyecto no es exacta pero si se combina con datos históricos de calidad y modelos sistemáticos mejora la precisión de resultados (Pressman, 2002).

3.4.3 Planeación temporal

En la planeación temporal de un proyecto de software se pueden aplicar herramientas y técnicas generales de proyectos (Op. Cit.).

Actividad e hito.
En el análisis del proyecto se debe distinguir entre hitos y actividades. Una actividad es una pieza de trabajo establecida que exige tiempo, tiene un principio y un fin, no siempre requiere el esfuerzo de los participantes, y quien o quienes se encarguen de cada tarea pueden definir las actividades individuales, es decir, distribuir y reasignar el trabajo. Por el contrario un hito es la totalidad de una actividad en un determinado momento, es decir es el final de una actividad especialmente diseñada. Cuando se definen todas las actividades para cada uno de los paquetes de trabajo y los hitos correspondientes, el paso siguiente es mostrar la información en forma gráfica ya sea en un diagrama de red o un cronograma que muestren el orden apropiado y las interrelaciones para lograr el alcance global del trabajo o del proyecto, lo cual es planeación temporal del proyecto (Pfleeger, 2002; Guido, 2003).

Estructura de división del trabajo (EDT).
Cuando se decide el objetivo del proyecto, se debe determinar que elementos o actividades del trabajo son necesarios realizar para lograrlo, para lo cual se requiere elaborar una relación de todas las actividades. Hay dos enfoques para preparar esta relación: (Guido, 2003)

- *Tormenta de ideas.* Este enfoque es apropiado para proyectos pequeños. Sin embargo, para proyectos más complejos, es difícil desarrollar una lista amplia de actividades sin olvidar algunas partidas.

- *Estructura de división del trabajo (EDT).* Divide un proyecto en piezas o partidas manejables para ayudar a asegurar que se identifiquen todos los elementos que se necesiten para completar el alcance del proyecto. Es un árbol jerárquico de partidas de trabajo que logrará o producirá el equipo durante el proyecto. EI logro de la producción de todas estas partidas constituye la terminación del alcance del trabajo del proyecto.

Mediante una estructura gráfica se subdivide el proyecto en piezas más pequeñas denominadas partidas de trabajo. Por lo general la EDT señala la organización o la persona que tiene la responsabilidad de cada partida de trabajo. Los criterios para decidir el detalle o los niveles que se deben colocar en la EDT son: (Op.Cit.)

- El nivel en el cual a una persona individual o una organización se le puede asignar la responsabilidad de realizar el paquete de trabajo.

- El nivel al que se desea controlar el presupuesto, supervisar y recopilar información de costos durante el proyecto.

- No existe una EDT única. Dos equipos diferentes pueden desarrollar una EDT diferente para el mismo proyecto.

Matriz de responsabilidades.
Muestra en un formato tabular el personal que tiene a cargo la responsabilidad de realizar las partidas de trabajo en una EDT, y da a conocer el papel de cada persona en respaldar el proyecto global. Además se puede especificar quién tiene la responsabilidad principal y quién la de respaldo para una partida de trabajo específica (Guido, 2003).

Red de tareas.
La red de tareas o actividades es una representación gráfica del flujo consecutivo de las tareas de un proyecto, muestra las interrelaciones de las actividades y es aplicable para la planeación, la programación y el control de proyectos. Dos de las técnicas más importantes de planeación en redes son: (Op. Cit.).

- Técnica de evaluación y revisión de programas (PERT, por las siglas en inglés de Program Evaluation and Review Technique).

- Método de la ruta critica CPM (CPM, por las siglas en inglés de Critical Path Method).

Las interdependencias entre las tareas deben definirse empleando a su vez una red de tareas. Tanto PERT como CPM proporcionan herramientas cuantitativas que permiten al encargado de planear el software: (Guido, 2003)

- Estimaciones de esfuerzo.

- Una descomposición de la función del producto.

- La selección del modelo de proceso adecuado y del conjunto de tareas.

- La descomposición de tareas.

- Determinar el camino crítico.

- La cadena de tareas que determina la duración del proyecto.

- Establecer las dimensiones de tiempo más probables para las tareas individuales aplicando modelos estadísticos.

- Calcular las limitaciones de tiempo que definen una ventana de tiempo de una tarea determinada.

Los cálculos de las limitaciones de tiempo pueden ser muy útiles en la planeación temporal de proyectos de software algunas limitaciones de tiempo que pueden descifrarse de una red PERT o de una red CPM, son: (Riggs, 1981)

- Lo rapidez con que se puede empezar una tarea cuando las tareas precedentes se completen lo antes posible.

- Lo más tarde que se puede empezar una tarea antes de que se retrase el tiempo mínimo para finalizar el proyecto.

- La fecha más temprana de finalización, es la suma de la fecha más temprana de inicio y la duración de la tarea.

- La fecha límite de finalización, es la fecha más tardía de inicio sumada a la duración de la tarea.

- El margen total, es la cantidad de tiempo extra o atrasos permitidos en la planeación temporal de las tareas de manera que el camino crítico de la red se mantenga conforme a la planeación temporal.

Los cálculos de los tiempos límite llevan a la determinación del camino crítico y proporcionan al gestor un método cuantitativo para evaluar el progreso a medida que se completan las tareas. Tanto PERT como CPM están implementadas en varias herramientas automáticas disponibles virtualmente para todas las computadoras. Tales herramientas son fáciles de usar y hacen asequibles los métodos de la planeación temporal descritos anteriormente a todos los gestores de proyectos de software. (The, 1993)

Gráficos de tiempo.
Un gráfico de tiempo, también denominado cronograma, permite conocer qué tareas serán conducidas en un punto determinado en el tiempo. Al emplear herramientas automáticas se introducen la red o esquema de tareas. La entrada de cada tarea es el esfuerzo, la duración y la fecha de inicio. Del procesamiento de la herramienta elegida resulta un gráfico de tiempo, como el que se muestra en la figura 3.3, al cual también se le conoce como *gráfica de Gantt* (Guido, 2003).

En las gráficas de Gantt la programación de las actividades ocurre en forma simultánea a su planeación. Pero el separar estas dos funciones hace que sea más fácil revisar un plan y calcular un programa actualizado. Las técnicas de redes separan las funciones de planeación y programación. Así el resultado o la producción de la función de planeación es un diagrama de red y no se dibuja a escala de tiempo y con base en este esquema se desarrolla el programa de desarrollo del producto de software (Op. Cit.).

Con frecuencia la planeación temporal no se realiza formalmente, razón por la cual un gran porcentaje de los proyectos de software se terminan mucho mas tarde de lo que se tenía previsto, o nunca llegan a su fin. Uno de los factores más importantes en la programación efectiva, es llegar a duraciones estimadas de las actividades que sean tan realistas como resulte posible al basarse en la experiencia.

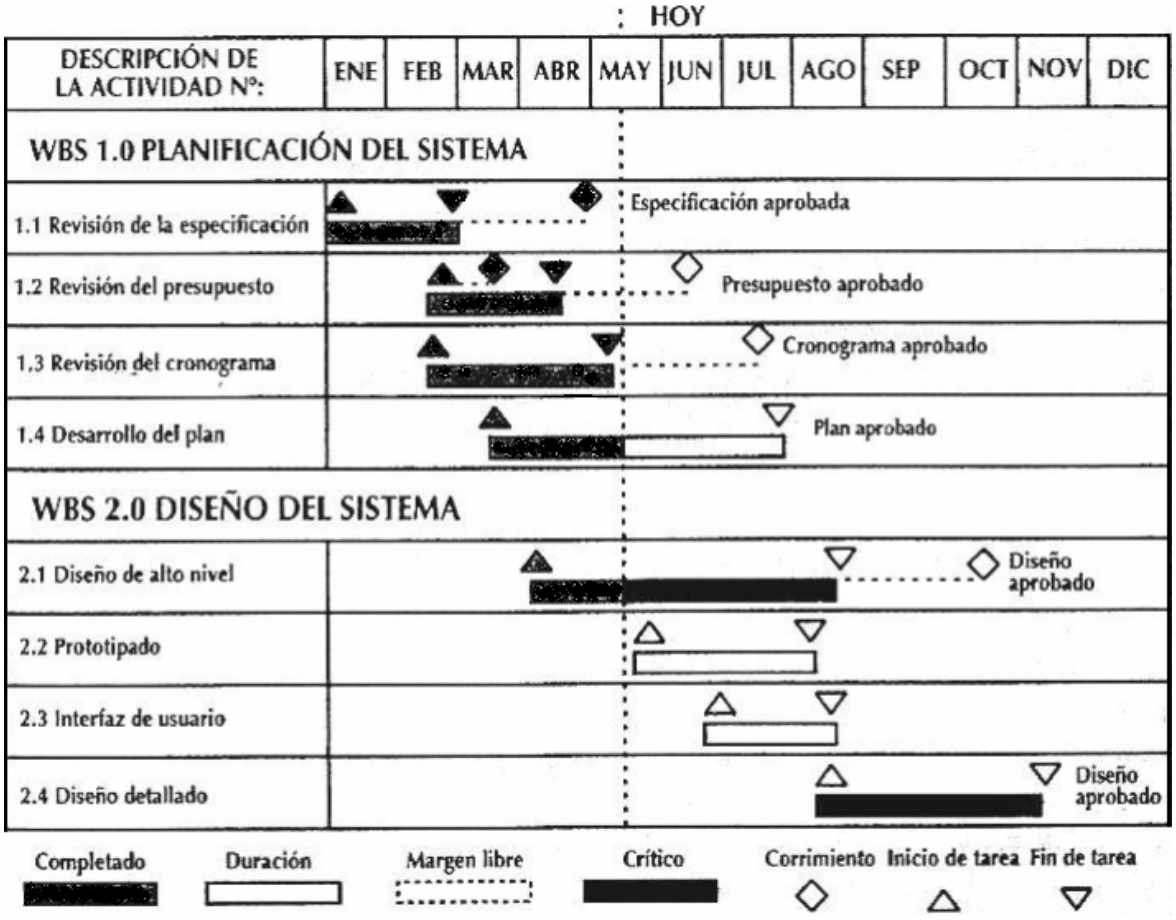


Figura 3.3. Ejemplo de una gráfica de Gantt.
Fuente: Pfleeger, 2002.

3.5 ADMINISTRACIÓN DEL RIESGO

El riesgo como la exposición a la posibilidad de pérdidas o daño, esto es, dar la oportunidad de que algo negativo pueda ocurrir. El riesgo del software implica dos características, estás son: (Higuera, 1995; Jalote, 1997)

- *Incertidumbre.* El acontecimiento que caracteriza al riesgo puede o no ocurrir.
- *Pérdida.* Si el riesgo se convierte en una realidad pueden ocurrir no deseadas o pérdidas.

En el contexto de los proyectos de software, algo negativo implica que tiene un efecto adverso sobre el costo, el tiempo y la calidad. Cualquier proyecto grande involucra cierto riesgo. La administración del riesgo es un área cuyo objetivo es identificar y dirigir los problemas asociados con el proyecto de software. La administración del riesgo puede considerarse como el proceso que trata con las ocurrencias o posibilidades de los eventos que no son “regulares” o esperados comúnmente, esto es, que trata con eventos que son poco frecuentes, fuera de control y son lo suficientemente largos para justificar una atención especial. La administración del riesgo se divide principalmente en la evaluación del riesgo y en el control del riesgo. (Pressman, 2002; Jalote, 1997)

3.5.1 Evaluación del riesgo

La evaluación del riesgo es una actividad que se debe llevar acabo durante la planeación del proyecto. Ésta involucra identificar el riesgo, analizarlo y priorizarlo. Tan pronto como el proyecto se acerca a su fin, el riesgo puede evaluarse de forma más precisa. Aunque la evaluación del riesgo debe llevarse a cabo a través de todo el proyecto, es más importante durante las fases iniciales. Una lista de comprobación de elementos de riesgo, es un método útil para identificar riesgos. Se puede componer de las respuestas a esos riesgos, la probabilidad de ocurrencia y de los componentes y controladores del riesgo. En la figura 3.4 se muestra la caracterización de las consecuencias potenciales de errores (filas etiquetadas con 1) o la imposibilidad de conseguir el producto deseado (filas etiquetadas con 2). La categoría del controlador del impacto sobre los componentes se elige mediante la caracterización que mejor encaja con la descripción de la tabla.

COMPONENTES / CATEGORÍA		RENDIMIENTO	SOPORTE	COSTE	PLANIFICACIÓN TEMPORAL
CATASTRÓFICA	1	Dejar de cumplir los requisitos provocaría el fallo de la misión		Malos resultados en un aumento de costes y retrasos de la planificación temporal con gastos que superan un límite.	
	2	Degradación significativa para no alcanzar el rendimiento técnico	El software no responde o no admite soporte	Recortes financieros significativos, presupuestos excedidos	Fecha de entrega inalcanzable
CRÍTICA	1	Dejar de cumplir los requisitos degradaría el rendimiento del sistema hasta donde el éxito de la misión es cuestionable		Malos resultados en retrasos operativos y/o aumento de coste con un valor esperado de cierto intervalo.	
	2	Alguna reducción en el rendimiento técnico	Pequeños retrasos en modificaciones de software	Algunos recortes de los recursos financieros, posibles excesos del presupuesto	Posibles retrasos en la fecha de entrega
MARGINAL	1	Dejar de cumplir los requisitos provocaría la degradación de la misión secundaria		Los costes, impactos y/o retrasos recuperables de la planificación temporal con un valor estimado de cierto intervalo.	
	2	De mínima a pequeña reducción en el rendimiento técnico	El soporte del software responde	Recursos financieros suficientes	Planificación temporal realista, alcanzable
DESPRECIABLE	1	Dejar de cumplir los requisitos provocaría inconvenientes o impactos no operativos		Los errores provocan impactos mínimos en el coste y/o planificación temporal con un valor esperado de menos de un límite.	
	2	No hay reducción en el rendimiento técnico	Software fácil de dar soporte	Posible superávit de presupuesto	Fecha de entrega fácilmente alcanzable

Figura 3.4. Evaluación del impacto del riesgo.
Fuente: Pressman, 2002.

A su vez, la lista de comprobación de elementos de riesgo permite al gestor del proyecto estimar el impacto del riesgo. Se centra en subconjunto de riesgos conocidos y predecibles en categorías genéricas, las cuales son: (Jalote, 1997; Pressman, 2002)

- *Tamaño del producto.* Riesgos asociados con el tamaño general del software a construir o modificar.

- *Impacto en el negocio*. Riesgos asociados con las limitaciones impuestas por la administración o el mercado.

- *Características del cliente*. Riesgos asociados con la sofisticación del cliente en los momentos oportunos.

- *Definición del proceso* .Riesgos asociados con el grado de definición del proceso del software y su seguimiento para la organización de desarrollo.

- *Entorno de desarrollo.* Riesgos asociados con la disponibilidad y la calidad de la herramientas que se van a emplear en la construcción del producto.

- *Tecnología a construir*: Riesgos asociados con la complejidad del sistema a construir y la tecnología de punta contenida en el sistema.

- *Tamaño y experiencia del equipo de trabajo*. Riesgos asociados con la experiencia técnica y de proyectos de quienes van a realizar el proyecto.

3.5.2 Control del riesgo

Mientras que la evaluación del riesgo es una actividad pasiva que identifica el riesgo y sus impactos, el control del riesgo comprende medidas correctivas que son llevadas a cabo por el administrador del proyecto para minimizar el impacto del riesgo. Al igual que cualquier otra actividad, el control del riesgo comienza con la planeación de la administración del riesgo. Debe notarse que los planes se desarrollan para cada riesgo identificado que requiere ser controlado (Jalote, 1997).

El desarrollo de la respuesta al riesgo involucra definir pasos para responder a las amenazas, los cuales generalmente caen en una de las tres categorías:(Op. Cit.)

- *Evitar.* Eliminar una amenaza, mediante la eliminación de la causa.

- *Mitigar.* Reducir el valor monetario del evento de riesgo, reduciendo la probabilidad de que suceda.

- *Aceptar*. Aceptar las consecuencias. Ya sea activa, al desarrollar un plan de contingencia, o pasiva, al aceptar menores ganancias si algunas actividades salen fuera de lo planeado.

3.6 ADMINISTRACIÓN DE LA CALIDAD

La calidad es la totalidad de las características de una entidad para satisfacer las necesidades implicadas o establecidas. La calidad del software se percibe mediante características mesurables, tales como: (Pressman, 2002; PMI, 2004)

- Calidad del diseño. Comprende los requisitos, especificaciones y el diseño de los sistemas, los cuales son características detalladas para un producto de software.

- Calidad de concordancia. Está centrada en la implementación del producto de software. Es el grado de cumplimento de las especificaciones de diseño durante su realización.

La administración de la calidad incluye el proceso requerido para asegurar que el proyecto cumpla con las necesidades del cliente. Involucra todas las actividades administrativas que determinan las políticas de calidad, los objetivos y las responsabilidades, así como el de implementarlas por medios dentro del sistema de calidad tales como, planeación de la calidad, aseguramiento de la calidad y el control de la calidad. Las etapas de la administración de la calidad son las siguientes: (PMI, 2004)

- Planeación de la calidad
- Aseguramiento de la calidad
- Control de la calidad

3.6.1 Planeación de la calidad

La planeación de la calidad involucra identificar los estándares de calidad que son relevantes al proyecto y determinar como se cumplirán. La planeación de la calidad reconoce la importancia de los siguientes puntos:

- *Satisfacción del cliente.* Entender, manejar e influenciar las necesidades para lograr o exceder las expectativas del cliente. Esto requiere de una combinación entre ajustarse a las especificaciones y ser adecuado para su uso.

- *Prevención.* El costo de evitar errores es siempre menor que el costo de corregirlos.

- *Responsabilidad administrativa.* El éxito requiere de la participación de todos los miembros del equipo, y es responsabilidad de la administración proporcionar los recursos necesarios para lograrlo.

3.6.2 Aseguramiento de la calidad

El aseguramiento de la calidad o garantía de la calidad es el conjunto de todas las actividades y planes implementados dentro del sistema de calidad para proporcionar la confianza de que el proyecto cumpla con los estándares relevantes de calidad. Tiene como objetivo la gestión para informar de los datos necesarios de la calidad del producto. Hay un conjunto de actividades que se enfrentan con la planeación de la garantía de calidad, supervisión, mantenimiento de registros, análisis e informes. Estas actividades son: (Pressman, 2002; PMI, 2004)

- *Establecimiento de un plan de calidad para un proyecto.* El plan se desarrolla durante la planeación del proyecto y se revisa por todas las partes interesadas.

- *Participación en el desarrollo de la descripción del proceso del software del proyecto.* El equipo de ingeniería del software selecciona un proceso para el trabajo que se va a realizar. Los encargados de evaluar la calidad revisa la descripción del proceso para ajustarse a la política de la empresa, los estándares internos del software, los estándares impuestos externamente (como lo es el ISO 9001), y a otras partes del plan del proyecto del software.

- *Revisión de las actividades de ingeniería de software para verificar su ajuste al proceso de software definido.* Se identifica, documenta y sigue la pista de las desviaciones desde el proceso y verifica la realización de las correcciones.

- *Auditoria de los productos de software designados para verificar el ajuste con los definidos como parte del proceso del software.* Se revisa los productos seleccionados; se identifica, se documenta y se sigue la pista de las desviaciones; se verifica la realización de las correcciones, y se informa periódicamente de los resultados de su trabajo al gestor del proyecto.

- *Asegurar que las desviaciones del trabajo y los productos del software se documentan y se manejan de acuerdo con un procedimiento establecido.* Las desviaciones se pueden encontrar en el plan del proyecto, en la descripción del proceso, en los estándares aplicables o en los productos técnicos.

- *Registrar lo que no se ajuste a los requisitos e informar a sus superiores.* Los elementos que no se ajustan a los requisitos están bajo seguimiento hasta que se resuelven.

Además de estas actividades, se coordina el control y la gestión de cambios, y se ayuda a recopilar y analizar las métricas del software (Paulk, 1993).

3.6.3 Control de la calidad

El control de la calidad involucra monitorear resultados específicos del proyecto para determinar si cumplen con los estándares de calidad e identificar formas para eliminar las causas de resultados no satisfactorios (PMI, 2004).

Royce (1998), señala que es primordial hacer tangible la calidad y el progreso del software a través de la instrumentación de la administración del cambio, pero es inherentemente difícil manejar lo que no puede ser medido objetivamente. Cualquier proceso de software cuya métrica está dominada por procedimientos y actividades manuales tendrá un éxito limitado. Las prácticas clave para mejorar la calidad del software incluyen las siguientes: (Royce, 1998, Pressman, 2002)

- Definir todos los productos y las especificaciones mesurables en las cuales se pueda comparar los resultados de cada proceso.

- Enfocarse en los requerimientos y en los casos de uso críticos en las fases tempranas del ciclo de vida.

- Emplear métricas e indicadores para medir el progreso y la calidad de la arquitectura implementada.

- Proporcionar ambientes de ciclo de vida integrados que proporcionen un control de la configuración de forma temprana, administración del cambio, métodos de diseño rigurosos y automatización de documentos.

- Usar modelado visual y lenguajes de alto nivel que apoyen el control de la arquitectura, abstracción, reutilización, programación fiable y autodocumentación.
- Incursión temprana y continua en los problemas de desempeño a través de evaluaciones basadas en demostración.

3.6.4 Costos de la calidad

Incluye los costos debidos a la búsqueda de la calidad o en las actividades relacionadas en la obtención de la calidad. Se realizan estudios sobre los costos de calidad para proporcionar una línea base del costo actual de calidad, para identificar oportunidades de reducir dicho costo, y para proporcionar una base normalizada de comparación. La base de normalización siempre tiene un precio. Una vez que se normaliza, los costos de calidad sobre un precio base, se obtienen los datos necesarios para evaluar el lugar en donde hay oportunidades de mejorar nuestros procesos. Por lo tanto, se puede evaluar como afectan los cambios en términos de dinero. Los costos de calidad se pueden dividir en tres categorias, los cuales son: (Pressman, 2002)

- *Costos de prevención*. Son los referentes a las actividades de planear, revisar y probar el producto.
- *Costos de evaluación.* Son los ocasionados por las actividades para tener una visión mas profunda de la condición del producto de cada proceso.
- *Costos de fallos.* Son los costos que desaparecerían si no surgieran defectos antes del envió de un producto a los clientes. Estos costos se pueden subdividir en:
  - Costos de fallos internos. Se producen cuando se detecta un error en el producto antes de su envío.
  - Costos de fallos externos. Son los que se asocian a los defectos encontrados una vez enviado el producto al cliente.

Los costos relativos para encontrar y reparar un defecto aumentan a medida que se cambia de prevención a detección, y desde el fallo interno al externo, tal como se muestra en la figura 3.5. (Boehm, 1981)

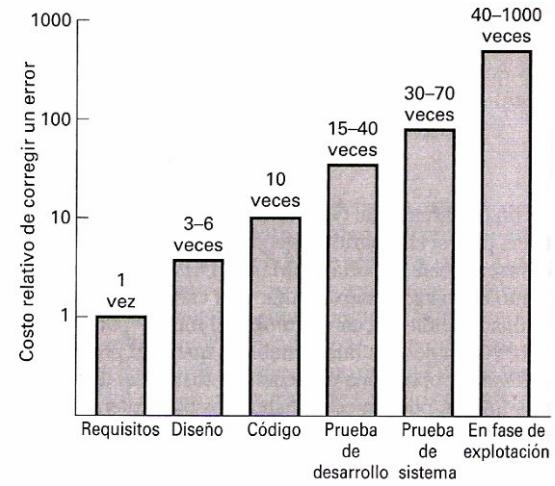


Figura 3.5. Costo relativo de corregir un error.
Fuente: Boehm, 1981.

3.7 PLAN DEL PROYECTO

Para comunicar a los clientes el análisis del problema, la solución propuesta, el diagrama de tiempo, la organización y la administración del riesgo y de la calidad, se escribe un documento denominado plan de proyecto. El plan deja por escrito las necesidades del cliente, así como lo que se espera hacer para satisfacerlas. El cliente puede obtener mediante el plan, la información sobre las actividades del proceso de desarrollo y le es sencillo seguir el avance del proyecto durante el desarrollo. También se puede usar el plan para corroborar cualquiera de las suposiciones hechas, especialmente sobre costos y diagrama de tiempo. Debe aclararse que se debe actualizar el plan durante el ciclo de vida del proyecto. Un buen plan de proyecto incluye las siguientes características: (PMI, 2004; Pfleeger, 2002)

- *Alcance del proyecto.* Establece los límites del sistema, en cuanto a costos, recursos, etc., de tal manera que el cliente se asegure de que los desarrolladores comprendan sus necesidades.

- *Cronograma del proyecto.* Se utilizan las herramientas descritas con anterioridad, tales como los EDT's, hitos y diagramas de Gantt, para mostrar lo que ocurre en cada punto durante el ciclo de vida del proyecto e ilustrar la naturaleza paralela de alguna de las tareas del desarrollo.

- *Organización del equipo del proyecto.* Incluye a las personas del equipo de desarrollo, la forma en la cual se organiza y las actividades o tareas a su cargo. Como se ve, no es forzosa la disponibilidad de todo el equipo de trabajo durante todo el tiempo del proyecto, por lo que generalmente el plan contiene un diagrama de asignación de recursos para mostrar los niveles de personal en diferentes momentos.

- *Descripción técnica del sistema propuesto.* Esta descripción enlista el hardware y el software, incluyendo compiladores, interfaces y equipos de software, para un propósito especial. Cualquier restricción especial en el cableado, tiempo de ejecución, tiempo de respuesta, seguridad u otros aspectos de la funcionalidad o del desempeño son documentados en el plan.

- *Estándares del proyecto, procedimientos y técnicas, y herramientas propuestas*. Detalla cualquiera de los estándares o métodos que deben emplearse, tales como:

  - Algoritmos.
  - Herramientas.
  - Técnicas de revisión o inspección.
  - Lenguajes o representaciones de diseño.
  - Lenguajes de codificación.
  - Técnicas de prueba.

- *Plan de aseguramiento de calidad*. Busca satisfacer de la mejor manera las necesidades del cliente.

- *Plan de gestión de la configuración.* Le dice al cliente cómo se rastrean los cambios de los requerimientos, diseño, código, planes de prueba y documentos.

- *Plan de la documentación.* Enumera y describe los documentos producidos, describe quién se encarga de realizarlos y cuándo, y además indica como se cambian los documentos.

- *Plan de gestión de datos.* Declara como se reúnen, se almacenan, se procesan y se archivan los datos.

- *Plan de gestión de recursos.* Explica como se utilizan los recursos.

- *Plan de prueba.* Describe el enfoque global del proyecto para las pruebas. Se debe establecer ciertas características para realizar las pruebas del producto de software, éstas son:

  - Como se generan los datos de prueba.

  - Como se prueba a cada módulo del programa.

  - Como se integran los módulos del programa.

  - Se establece la forma en la cual se prueban unos módulos con otros.

  - Quién es el responsable de ejecutar las pruebas.

  - Si los sistemas se producen en estadios o fases, es preciso explicar como se realiza el estadio.

  - Si se agrega una nueva funcionalidad a una fase, se debe asegurar que la funcionalidad existente todavía sea correcta.

- *Plan de entrenamiento.* Describe cada clase del software, documentos de soporte y el nivel de especialización requerido. Si existen requerimientos de seguridad, el plan de seguridad consigna la forma en la cual se protegen los datos, usuarios y software.

- *Plan de seguridad.* Se debe explicar como afecta cada faceta de seguridad al desarrollo del sistema, tales facetas son:

  - Confidencialidad.
  - Disponibilidad.
  - Integridad.

- *Plan de gestión del riesgo.* Establece la forma mediante la que se anticipan y resuelven las eventualidades.

- *Plan de mantenimiento.* Discute las responsabilidades para cambiar de códigos, reparar el hardware, actualizar la documentación de soporte y los materiales de entrenamiento.

3.8 CONTROL DEL PROYECTO DE SOFTWARE

Cuando se realiza un proyecto se requiere supervisar el avance para asegurar el plan del proyecto. El control lo usa el gestor para administrar los recursos del proyecto, enfrentarse a los problemas y dirigir al personal del proyecto (Pressman, 2002).

El control del proyecto incluye la recopilación de información periódicamente sobre el desempeño del proyecto y compara el avance real con el planeado para llevar a cabo acciones correctivas si el desempeño real es inferior al planeado. Se pude establecer un diagrama de flujo con los pasos en el proceso de control del proyecto, tal como se ilustra en la figura 3.6. (Guido, 2003)

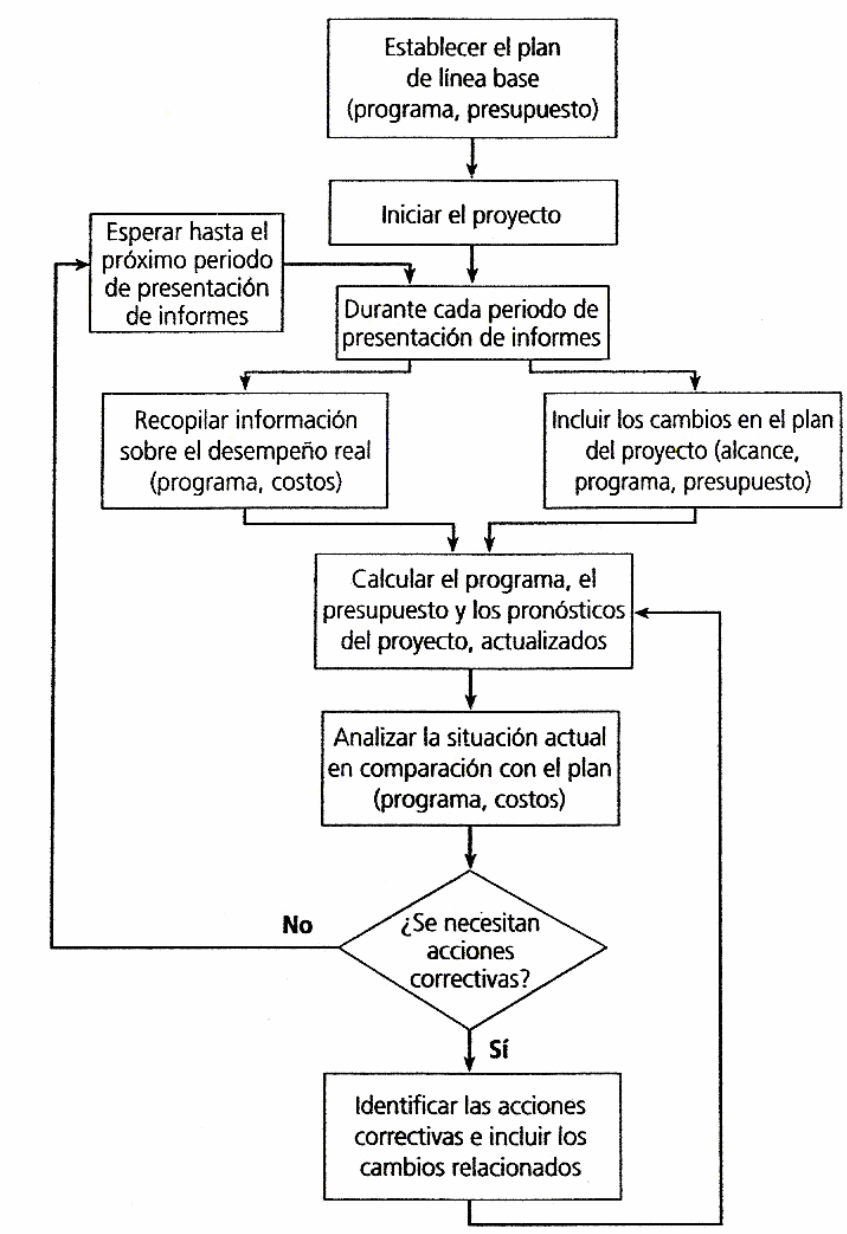


Figura 3.6. Diagrama de flujo del control de un proyecto.
Fuente: Guido, 2002.

3.8.1 Seguimiento de la planeación temporal

El seguimiento de la planeación temporal se puede hacer de diferentes maneras, tales como: (Pressman, 2002)

- Al realizar reuniones periódicas del estado del proyecto en las que todos los miembros del equipo informan del progreso y de los problemas.
- Al evaluar los resultados de todas las revisiones realizadas a lo largo del proceso de ingeniería del software.
- Al determinar si se consiguen los hitos formales del proyecto en la fecha programada, los cuales se pueden representar en la gráfica tiempo. Al comparar la fecha real de inicio con las previstas para cada tarea del proyecto listada en la tabla del proyecto.
- Al reunirse informalmente con los profesionales del software para obtener sus valoraciones subjetivas del progreso hasta la fecha y los problemas que se avecinan.
- Al evaluar el progreso cuantitativamente.

Para medir el avance real es importante mantener un seguimiento de cuales actividades ya se iniciaron y/o terminaron, cuando lo hicieron y cuanto dinero se gasta o está comprometido. Si en cualquier momento del proyecto la comparación del avance real con lo programado muestra que se está retrasado con respecto al plan, excediendo en el presupuesto o que no se cumple con las especificaciones, es necesario llevar a cabo acciones correctivas para hacer que el proyecto esté de nuevo de acuerdo a lo planeado. (Guido, 2003)

4. ANÁLISIS DE REQUISITOS

4.1 MÉTRICAS DEL SOFTWARE

La medición de un proyecto de software tiene dos objetivos principales. Éstos son predecir y analizar. Predecir el progreso futuro mediante las métricas puede servir para planear y estimar el efecto de los cambios en el proyecto. Si un administrador de proyecto puede predecir con claridad los costos del proyecto, entonces existe mayor probabilidad de que el cliente reciba un sistema apropiado y a tiempo. La información de las métricas colectadas durante el análisis puede ser usada para monitorear el progreso, y si un proyecto falla, entonces las mediciones tomadas a través del proyecto, pueden ser usadas en retrospectiva para identificar que es lo que se hizo mal y ayudar por lo tanto a evitar tales fallas en el futuro. (Ince et al., 1993)

Medida, medición, métrica e indicador.
Algunos conceptos importantes en lo que respecta a realizar mediciones durante el desarrollo del software son: (IEEE, 1993; Pressman, 2002)

- *Entidad*. Es un objeto que se caracteriza mediante una medición de sus atributos (tales como: recurso, proceso, producto, proyecto o servicio).
- *Característica*. Es una relación abstracta entre atributos de una o más entidades, y una necesidad de información.
- *Atributo*. Es una característica medible de una entidad. Pueden ser de los siguientes dos tipos:
  - Interno. Son aquellos que pueden ser medidos al examinar el proceso, producto o recurso mismo.
  - Externo. Se miden con respecto a como el proceso, producto o recurso se relaciona con su entorno.

Es importante distinguir entre medida, medición, métrica e indicador, debido a que estos conceptos se confunden con mucha frecuencia, por lo tanto se definen a continuación: (IEEESGS, 1993; Ragland, 1995; Pressman, 2002; Perampalam, 2007)

- *Medida.* Es una indicación cuantitativa de la extensión, cantidad, dimensiones, capacidad o tamaño de algunos atributos de un proceso o un producto.

- *Medición.* Es el proceso mediante el cual números o símbolos son asignados a atributos de entidades en el mundo real, de tal forma que los describen de acuerdo a claras y no ambiguas reglas (Véase figura 4.1).

- *Métrica.* Es una medida cualitativa del grado en el cual un sistema, componente o proceso posee un cierto atributo. Como lo muestra la figura 4.2 tales atributos son tan relevantes que utilizan en la toma de decisiones.

- *Indicador.* Es una métrica o combinación de métricas que proporcionan una visión profunda del proceso de desarrollo del software.

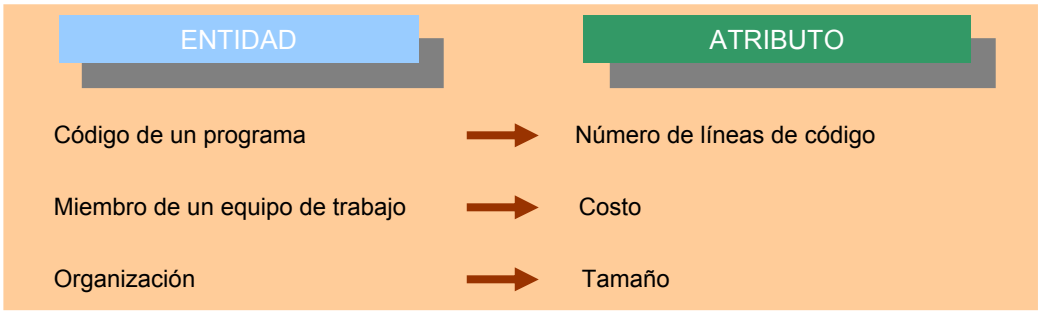


Figura 4.1. Ejemplo de atributos de diferentes entidades.
Fuente: Perampalam, 2007.

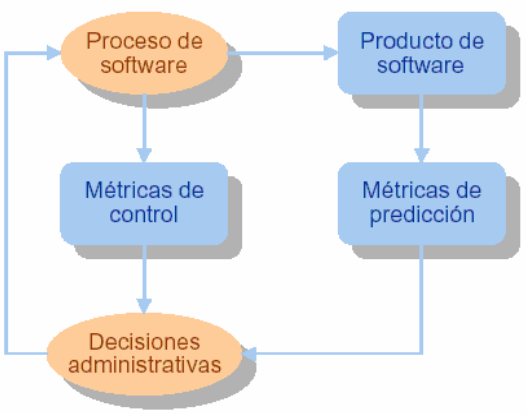


Figura 4.2.Relación de las métricas con la toma de decisiones.
Fuente: Barreiro, 2007.

4.1.1 Clasificación de las métricas del software

Las métricas del software se analizan de diversas formas, se dividen en grupos por el ámbito que abarcan y algunas no pertenecen a una sola categoría. Para plantear un punto de inicio en la clasificación de las métricas del software de computadora se considera dos componentes básicos, que definen dominios de aplicación y permiten diferentes estrategias de medición del software. Además cada una tiene su aplicación en cierta o ciertas etapas del ciclo de vida de los sistemas de información. De acuerdo a lo establecido por Pressman (2002) y Guido (2003) en cuanto al desarrollo del software, en la tabla 4.1 se muestra una propuesta de las métricas más

representativas del software y la etapa del ciclo de vida de los sistemas de información donde se pueden aplicar. Los grupos de métricas son: (Mills, 1980; Kan, 2002; Klasky, 2003)

- Métricas del proceso.
- Métricas del producto.

Tabla 4.1. Métricas del Software.

MÉTRICAS DEL SOFTWARE	MÉTRICAS DEL PROCESO			LISTADO DE MÉTRICAS DEL SOFTWARE		ETAPA DEL CVDS
				MPS01	Factibilidad técnica.	Definición del problema.
				MPS02	Factibilidad temporal.	
				MPS03	Factibilidad financiera.	
				MPS04	Recursos.	
		Métricas de madurez.	Métricas de la organización, recursos, personal y entrenamiento, y estructura de equipo.	MPS05	Número de desarrolladores y sus niveles de habilidades.	Análisis.
				MPS06	Estructura de la organización.	
				MPS07	Distribución del trabajo.	
				MPS08	Distribución del esfuerzo.	
				MPS09	Costo de desarrollo C_{das}.	
				MPS10	Esfuerzo de desarrollo.	
			Métrica de gestión	MPS11	Facilidad de auditoria.	
				MPS12	Índice de Fog.	
				MPS13	Autodocumentación.	
			Métricas para estándares de documentación.	MPS14	Páginas de documentación por líneas de código o por punto de función.	Diseño.
			Métrica de gestión y análisis de datos.	MPS15	Declaración dependencia de datos.	
			Métricas de control de proceso.	MPS16	Rendimiento.	
				MPS17	Índice de errores IE.	
		Métricas de gestión.	Métricas de logros.	MPS18	Número de tareas planeadas y completadas.	
				MPS19	Puntos de función completados.	
			Métricas del riesgo.	MPS20	Clasificación de los riesgos.	
				MPS21	Cantidad de riesgos.	
				MPS22	Categoría del riesgo.	
			Métricas de revisión.	MPS23	Análisis de valor ganado AVG.	Diseño.
				MPS24	Índice de desarrollo de planeación IDP.	
				MPS25	Índice de desarrollo del costo IDC.	
				MPS26	Varianza del costo VC.	
				MPS27	Defectos encontrados luego de realizar cambios en el diseño.	
				MPS28	Número de errores detectados mediante inspecciones de software.	Desarrollo y Pruebas.
				MPS29	Número de errores detectados durante la integración de las pruebas.	

Fuente: Pressman, 2002; Guido, 2003.

Tabla 4.1. Métricas del Software… *Continuación*

LISTADO DE MÉTRICAS DEL SOFTWARE					ETAPA DEL CVDS
MÉTRICAS DEL SOFTWARE	MÉTRICAS DEL PROCESO	Métricas de gestión.	Métricas de revisión.	MPS30 Número de cambios de código requeridos.	Desarrollo y Pruebas.
				MPS31 Número de mejoras sugeridas al sistema.	
				MPS32 Número de defectos encontrados mediante integración y	
				MPS33 Número de defectos encontrados mediante inspecciones.	
				MPS34 Código cubierto por unidad de prueba.	
				MPS35 Número de defectos encontrados por unidad de prueba.	
				MPS36 Porcentaje de casos de prueba aprobados.	
				MPS37 Niveles libre de defectos.	
				MPS38 Número potencial de defectos.	
				MPS39 Eficiencia de eliminación de defectos **EED**.	
		Métricas de gestión.	Métricas de productividad.	MPS40 Errores por líneas de código.	
				MPS41 Defectos por líneas de código.	
				MPS42 Costo por línea de código.	
				MPS43 Líneas de código/ persona-mes.	
				MPS44 Errores por persona-mes.	
				MPS45 Costo por página de documentación.	
				MPS46 Errores por punto de función.	
				MPS47 Defectos por PF.	
				MPS48 Costo por PF.	
				MPS49 Páginas de documentación por PF.	
				MPS50 PF por persona-mes.	
				MPS51 Índice de madurez del software **IMS**.	
			Métricas del análisis y especificación de requerimientos.	MPS52 Especificidad de los requerimientos (Ausencia de	Análisis.
				MPS53 Compleción de los requerimientos Q_2	
				MPS54 Grado de validación los requerimientos Q_3	
				MPS55 Estatus de los requerimientos **ER**.	
				MPS56 Tiempo medio de fallos.	Pruebas.
				MPS57 Disponibilidad.	Desarrollo y Pruebas.
				MPS58 Calidad de la función desplegada.	
			Métricas de hardware.	MPS59 Métricas de performance.	
				MPS60 Métricas de fiabilidad.	
				MPS61 Métricas de disponibilidad.	

Fuente: Pressman, 2002; Guido, 2003.

Tabla 4.1. Métricas del Software… *Continuación*

LISTADO DE MÉTRICAS DEL SOFTWARE				ETAPA DEL CVDS
MÉTRICAS DEL SOFTWARE	MÉTRICAS DEL PRODUCTO	**Métricas del tamaño.**	MPD01 Longitud del código (LDC).	Desarrollo y Pruebas.
			MPD02 Puntos de función (PF).	
			MPD03 Complejidad ciclomática.	
		Métricas de arquitectura.	MPD04 Número de componentes.	
			MPD05 Número de paradigmas de lenguaje en el producto.	
			MPD06 Niveles de métricas.	
		Métricas de estructura.	MPD07 Estandarización de datos.	
			MPD08 Longitud de los identificadores.	
		Métricas de profundidad.	MPD09 Trazabilidad.	
			MPD10 Capacidad de expansión.	
			MPD11 Generalidad.	
			MPD12 Instrumentación.	
		Métricas de acoplamiento.	MPD13 Acoplamiento de flujo de datos y control.	
			MPD14 Acoplamiento global.	
			MPD15 Acoplamiento de entorno.	
		Métricas del diseño arquitectónico.	MPD16 Complejidad estructural **S (i).**	
			MPD17 Complejidad del sistema **C (i).**	
			MPD18 *Complejidad de datos* **D (i).**	
			MPD19 Métrica Henry-Kafura **MHK.**	
			MPD20 Tamaño.	
			MPD21 Profundidad.	
			MPD22 Anchura.	
			MPD23 Relación arco-a-nodo.	
		Métricas de diseño a nivel de componentes.	MPD24 Porción de datos.	Desarrollo y Pruebas.
			MPD25 Muestras (tokens) de datos.	
			MPD26 Señales de unión.	
			MPD27 Señales de superunión.	
			MPD28 Adhesividad.	

Fuente: Pressman, 2002; Guido, 2003.

Tabla 4.1. Métricas del Software … *Continuación*

LISTADO DE MÉTRICAS DEL SOFTWARE					ETAPA DEL CVDS
MÉTRICAS DEL SOFTWARE	MÉTRICAS DEL PRODUCTO	Métricas cualitativas de la calidad.	Métricas de funcionalidad.	MPD29 Idoneidad.	Pruebas y puesta en marcha.
				MPD30 Exactitud.	
				MPD31 Interoperabilidad.	
				MPD32 Seguridad.	
			Métricas de confiabilidad.	MPD33 Madurez.	
				MPD34 Tolerancia a fallos.	
				MPD35 Capacidad de recuperación.	
			Métricas de facilidad de uso.	MPD36 Facilidad de comprensión.	
				MPD37 Facilidad de aprender.	
				MPD38 Operatibilidad.	
			Métricas de eficiencia.	MPD39 Tiempo de funcionamiento.	
				MPD40 Utilización de recursos.	
			Métricas de capacidad de mantenimiento.	MPD41 Capacidad de análisis.	
				MPD42 Variabilidad.	
				MPD43 Estabilidad.	
				MPD44 Facilidad de prueba.	
			Métricas de portabilidad.	MPD45 Adaptabilidad.	
				MPD46 Capacidad de instalación.	
				MPD47 Capacidad de reemplazo.	
		Métricas de bases de datos relacionales.		MPD48 Número de atributos NA.	Desarrollo y Pruebas.
				MPD49 Grado de referenciabilidad GR.	
				MPD50 Profundidad del árbol referencial PAR.	
				MPD51 Radio de normalidad RN.	

Fuente: Pressman, 2002; Guido, 2003.

4.1.1.1 Métricas del proceso

Las métricas del proceso del software que se exponen en este trabajo permiten medir el desarrollo del producto de software, así como todo el tiempo de desarrollo y el tipo de metodología que se emplea, entre otros atributos. Éstas son: (Wiegers, 2007; Klasky, 2003; McCall, 1977; Davis et al., 1993; SML@b, 2007)

- Métricas de madurez.
- Métricas de gestión.
- Métricas del análisis y especificación de requerimientos.
- Métricas de hardware.

Métricas de madurez.
Se dice que estas medidas indican el estado de la empresa al inicio del desarrollo de un proyecto de software. Éstas son:

- Métricas de la organización, recursos, personal y entrenamiento, y estructura de equipo.
- Métrica de gestión tecnológica.
- Métricas para estándares de documentación.
- Métrica de gestión y análisis de datos.
- Métricas de control de proceso.

Métricas de la organización.

- *Estimación de la factibilidad.* Hasta dónde se puede esperar que un programa lleve a cabo su función con la exactitud requerida. El modelo COCOMO es una herramienta de gran uso en este aspecto. Es necesario reportar los valores de cada una de las estimaciones que se realizan, éstas son: (Putman, 1992)

  - Factibilidad técnica (MPS01). Se confirma o no si se tiene el suficiente potencial científico para realizar el proyecto.

  - Factibilidad temporal (MPS02). Estimación de la duración del proyecto (diagrama de tiempo o cronograma) y de cada etapa, seguimiento individual, tareas e hitos.

  - Factibilidad financiera (MPS03). Se corrobora o no si se tiene la capacidad de cubrir todos los costos referentes al desarrollo del producto de software.

  - Recursos (MPS04). Se ratifica si se dispone de los suficientes recursos para desarrollar el producto de software.

- *Número de desarrolladores y sus niveles de habilidades* (MPS05). Se puede presentar mediante una matriz que en su lado izquierdo muestre los nombres de los integrantes del equipo contra la información de las habilidades en la parte superior.

- *Estructura de la organización* (MPS06). Es la estructura con la cual se distribuyen las tareas a los diferentes integrantes del equipo de trabajo.

- *Distribución del trabajo* (MPS07). Registro del tiempo invertido en las actividades de desarrollo y las actividades de mantenimiento.

- *Distribución del esfuerzo* (MPS08). Es una medida subjetiva, pero se puede usar como directriz. Se sigue la regla 40-20-40, que asigna 40% a las tareas de análisis y diseño, un 20% a la creación del código y otro 40% a las pruebas. Esta regla se adapta a las condiciones específicas de cada proyecto y se puede compara con la distribución real al finalizar el proyecto.

- *Costo de desarrollo* $\mathbf{C_{des}}$ (MPS09). Es el esfuerzo de desarrollo con la inclusión de un factor del costo laboral del trabajo, $\mathbf{F_{clt}}$, ($/personas-año) (Ec. 4.1).

$$C_{des} = EF_{clt} \tag{4.1}$$

- *Esfuerzo de desarrollo* (MPS10). Es una medida que indica el esfuerzo invertido (personas-año) durante el ciclo entero de la vida del desarrollo de software y del mantenimiento (Véase la Ec. 4.2).

$$E = \frac{L^3}{\left(P^3 t^4\right)} \tag{4.2}$$

  Donde **L** es el número de líneas de código entregadas**, P** es un parámetro de productividad que refleja una variedad de factores que llevan a un trabajo de ingeniería del software de alta calidad (los valores típicos de **P** se encuentran entre 2,000 y 12, 000); y **t** es la duración del proyecto en meses.

Métrica de gestión tecnológica.

Facilidad de auditoria (MPS11). La facilidad con la que se puede comprobar el cumplimiento de los estándares.

Métricas para estándares de documentación.

- *Índice de Fog* (MPS12). Medida de la longitud promedio de las palabras y declaraciones en los documentos. Cuanto mayor sea el índice de Fog, más difícil será comprender el documento.

- *Autodocumentación* (MPS13). El grado en que el código fuente proporciona documentación significativa.

- Páginas de documentación por líneas de código o por punto de función (MPS14).

Métrica de gestión y análisis de datos.
La declaración dependencia de datos (MPS15) es una métrica de análisis de datos. Se realiza de acuerdo a la estructura del modelo de análisis que se ilustra en la figura 4.3. En la fase del análisis se obtienen los requisitos y se establece el fundamento para el diseño mediante los diagramas mostrados en la figura 4.3 y el diccionario de datos.

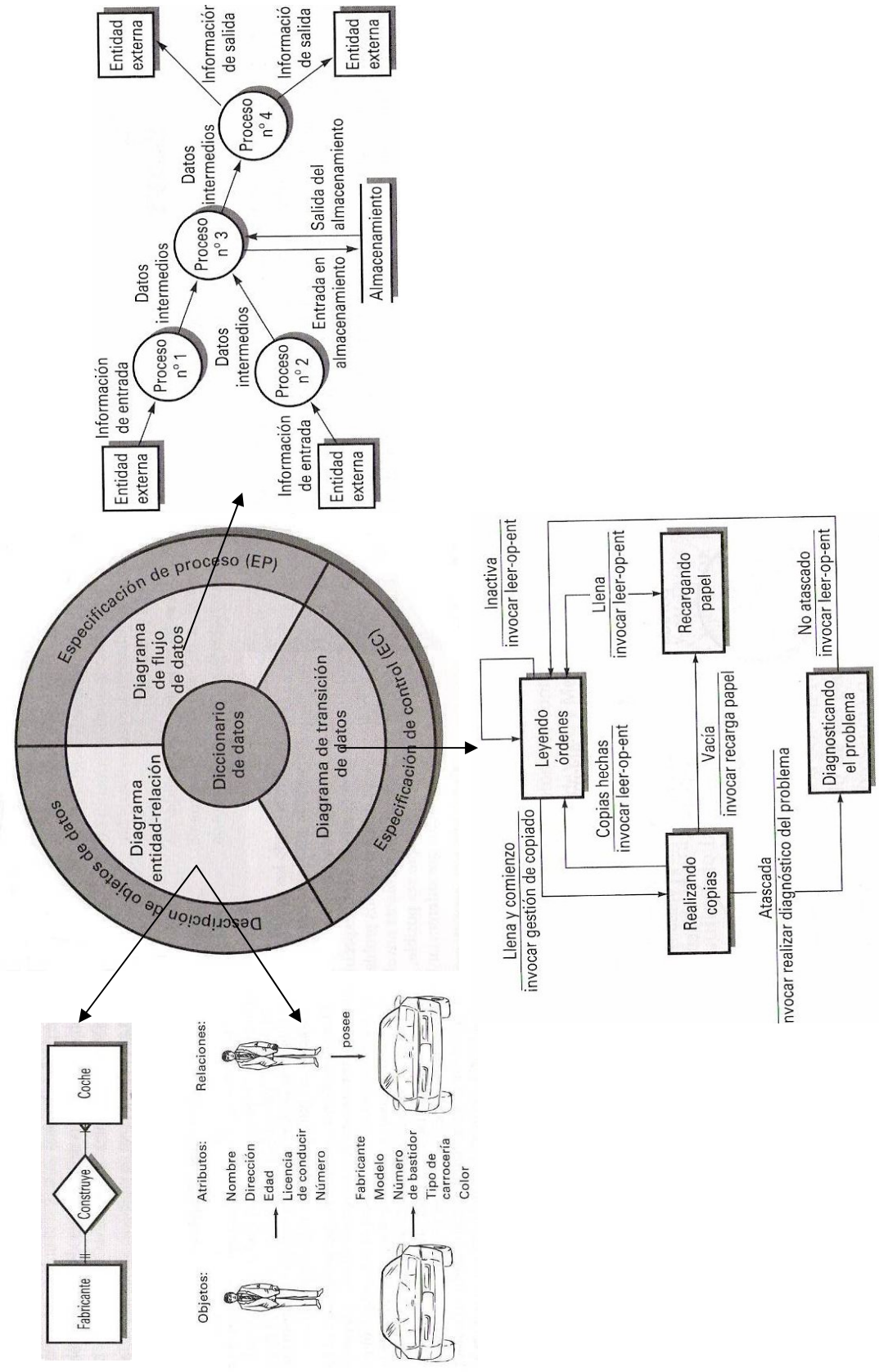


Figura 4.3. Modelo de análisis y sus componentes.

Métricas de control de proceso.

- *Rendimiento* (MPS16). Se mide por la velocidad de procesamiento, el tiempo de respuesta, consumo de recursos, rendimiento efectivo total y eficacia.

- *Índice de errores* **IE** (MPS17). Se calcula un índice de errores IF para cada etapa principal del proceso de ingeniería del software: análisis, el diseño, la codificación, la prueba y la entrega (Ver ecuación 4.3), de toma los datos de la tabla 4.2. El IE se obtiene mediante el cálculo del defecto acumulativo de cada fase, asignando más peso a los errores encontrados más tarde en el proceso de ingeniería del software, que a los que se encuentran en las primeras etapas. (Ver ecuación 4.4)

Tabla 4.2 Medidas para el índice de errores.

Medida	Descripción
E_i	Número total de defectos descubiertos durante la etapa i-ésima del proceso de ingeniería del software.
S_i	Número de defectos graves.
M_i	Número de defectos moderados.
T_i	Número de defectos leves.
PS	Tamaño del producto (LDC, sentencias de diseño, paginas de documentación) en la etapa i-ésima.
W_s, W_m, W_t	Factores de peso de errores graves, moderados, y leves, en donde los valores recomendados son $W_s = 10$, $W_m = 3$, $W_t = 1$. Los factores de peso de cada fase deben agrandarse a medida que el desarrollo evoluciona. Esto favorece a la organización que encuentra los errores al principio.

Fuente: Pressman, 2002.

$$IF_i = W_s \frac{S_i}{E_i} + W_m \left(M_i E_i\right) + W_t \frac{T_i}{E_i} \tag{4.3}$$

$$IE = \frac{\sum_{i=1}^{n} \left(i * IF_i\right)}{PS} \tag{4.4}$$

Métricas de gestión.
Se dice que estas medidas de gestión se enfocan a revisar el avance del proceso del software respecto a los objetivos del proyecto. Éstas son:(Klasky, 2003; Barreiro, 2007)

- Métricas de logros.
- Métricas del riesgo.
- Métricas de revisión.
- Métricas de productividad.
- Métricas del análisis y especificación de requerimientos.

Métricas de logros.

- Número de tareas planeadas y completadas (MPS18).
- Puntos de función completados (MPS19).

Métricas del riesgo.

Las métricas de riesgo se centran en la identificación y evaluación de los riesgos potenciales que pueden producir un impacto negativo en el software y hacer que falle el sistema completo. Si se identifican de manera temprana, los riesgos en el proceso de ingeniería del software se pueden especificar las características del diseño del software que permitan eliminar o controlar los riesgos potenciales. Esta información está contenida en la hoja de información de riesgo. Las métricas son:

- Clasificación o impacto de los riesgos, tales como: catastrófica, critica, marginal y despreciable. (MPS20).
- Cantidad de riesgos. (MPS21).
- Categoría del riesgo, tales como: rendimiento, soporte, coste y planificación temporal. (MPS22).

Métricas de revisión.

- *Análisis de valor ganado* **AVG** (MPS23). Es una técnica cuantitativa para evaluar el progreso a medida que el equipo de software avanza a través de las tareas de trabajo asignadas en la planeación del proyecto. Proporciona las lecturas exactas y fiables del desarrollo desde estados iniciales como cuando tan sólo se haya realizado un 15% del proyecto. Utiliza un presupuesto desglosado a través de todas las actividades estructurales del proyecto, y distribuido en el tiempo. Esta proyección temporal se obtiene en base a una programación de todas las actividades del proyecto (tal como un diagrama de Gantt) y un criterio para distribuir temporalmente el costo de cada una de las tareas.
- *Índice de desarrollo de planeación* **IDP** (MPS24). Es un indicador de la eficiencia con la cual se utilizan los recursos de la planeación. Un valor IDP cercano a 1.0 indica una ejecución eficiente de la planeación del proyecto.
- *Índice de desarrollo del costo* **IDC** (MPS25). Un valor de IDC cercano a 1.0 proporciona una indicación evidente de que el proyecto está dentro del presupuesto definido con anterioridad.
- *Varianza del costo* **VC** (MPS26). Es un índice absoluto de los ahorros en costo (en relación con los costos planificados) o de las carencias en una etapa particular del proyecto.
- *Métricas de defectos.* Un defecto se define como una falta verificada de conformidad con los requisitos, y que se detecta luego del diseño del producto de software, si se encuentra antes se dice que es un error y si se encuentra en la puesta en marcha del sistema se denomina falla. Los errores o defectos localizados se documentan y comparan contra la información histórica, para que el líder del proyecto tome las previsiones pertinentes,

debido a que si se encuentra que el porcentaje de errores y/o defectos encontrados en las tareas de recopilación de requisitos, diseño y código, es una proporción considerablemente mayor con respecto a la información de proyectos anteriores, se debe investigar todo el proceso de desarrollo del producto. Es importante registrar también su tipo y severidad. Las métricas relacionadas con defectos o errores son:

- Defectos encontrados luego de realizar cambios en el diseño (MPS27).
- Número de errores detectados mediante inspecciones de software (MPS28).
- Número de errores detectados durante la integración de las pruebas (MPS29).
- Número de cambios de código requeridos (MPS30).
- Número de mejoras sugeridas al sistema (MPS31).
- Número de defectos encontrados mediante integración y pruebas (MPS32).
- Número de defectos encontrados mediante inspecciones (MPS33).
- Código cubierto por unidad de prueba (MPS34).
- Número de defectos encontrados por unidad de prueba (MPS35).
- Porcentaje de casos de prueba aprobados (MPS36).
- Niveles libre de defectos (MPS37).

- *Número potencial de defectos* (MPS38). Es una medida de la cantidad de defectos del producto de software basada en puntos de función, (Ec. 4.5), la cual está basada en reglas especificadas en el Manual de practicas del conteo por puntos de función.

$$num\_potencial\_defectos = PF^{1.25} \tag{4.5}$$

- *Eficiencia de eliminación de defectos* **EED** (MPS39). Proporciona un índice de la efectividad de las actividades de garantía de calidad, también se utiliza para determinar qué se realiza durante la duración del proyecto. Los errores o defectos localizados se documentan y comparan contra la información histórica, para que el líder del proyecto tome las previsiones pertinentes, debido a que si se encuentra que el porcentaje de errores y/o defectos encontrados en las tareas de recopilación de requisitos, diseño y código, es una proporción considerablemente mayor con respecto a la información de proyectos anteriores, se debe investigar todo el proceso de desarrollo del producto (Ec. 4.6).

$$EED = \frac{E}{E + D} \tag{4.6}$$

Métricas de productividad.

Se dice que las métricas de productividad sirven para mostrar que tanto avance en el proyecto se obtiene de cierto proceso, pueden relacionar las métricas del producto (tales como: líneas de código y puntos de función) con costos y/o tiempo. Algunas de éstas métricas son:

- Errores por líneas de código (MPS40).
- Defectos por líneas de código (MPS41).
- Costo por línea de código (MPS42).
- Líneas de código/ persona-mes (MPS43).

- Errores por persona-mes (MPS44).
- Costo por página de documentación (MPS45).
- Errores por punto de función (MPS46).
- Defectos por PF (MPS47).
- Costo por PF (MPS48).
- Páginas de documentación por PF (MPS49).
- PF por persona-mes (MPS50).
- *Índice de madurez del software* **IMS** (MPS51). Proporciona una indicación de la estabilidad de un producto software (basada en los cambios que ocurren con cada versión del producto). Se determina la información de la tabla 4.3 mediante la ecuación 4.7. A medida que el IMS se aproxima a 1.0 el producto se empieza a estabilizar. El IMS se emplea también como métrica de planeación de las actividades de mantenimiento del software. El tiempo medio para producir una versión de un producto software puede correlacionarse con el IMS desarrollar modelos empíricos para el mantenimiento.

Tabla 4.3. Medidas para el índice de madurez del software.

Medida	Descripción
M_T	Número de módulos en la versión actual.
F_c	Número de módulos en la versión actual que se cambian.
F_a	Número de módulos en la versión actual que se añaden.
F_d	Número de módulos de la versión anterior que se borran en la versión actual.

Fuente: Pressman, 2002.

$$IMS = \frac{M_T - (F_a + F_c + F_d)}{M_T} \qquad (4.7)$$

Métricas del análisis y especificación de requerimientos.
Las métricas de la calidad de la especificación son una lista de características que se emplean para valorar la calidad del modelo de análisis y la correspondiente especificación de requisitos, éstas son: (Davis et al., 1993)

- *Especificidad de los requerimientos* (Ausencia de ambigüedad) **Q1** (MPS52) (Ver Ec. 4.8) y su clasificación, debido a que un requerimiento puede ser funcional o no funcional. Entre más se aleje de la unidad el nivel de especificación de requerimientos, mayor es la ambigüedad de los requerimientos.

$$Q_1 = \frac{nui}{nf + nnf} \tag{4.8}$$

Donde **nui** es el número de requerimientos que tienen interpretación unánime de quienes los revisan; **nf** es el número de requerimientos funcionales, y **nnf** es el número de requerimientos no funcionales.

- *Compleción de los requerimientos* **Q_2** (MPS53) (ver Ec. 4.9) Es el valor de requerimientos funcionales y mide el porcentaje de necesidad de las funciones que se especifican por el sistema.

$$Q_2 = \frac{nu}{ni * ns} \tag{4.9}$$

Donde **nu** es el número de requerimientos funcionales; **ni** es el número de entradas (todos los datos de entradas) definidas o derivadas mediante una especificación de documento, y **ns** es el número de escenarios y de estados en la especificación.

- *Grado de validación los requerimientos* **Q_3** (MPS54) (Ec. 4.10). Es el grado en el cual se validan los requerimientos.

$$Q_3 = \frac{nc}{nc * nnv} \tag{4.10}$$

Donde **nc** el es número de requerimientos validados y **nnv** es el número de requerimiento que todavía no se validan.

- *Estatus de los requerimientos* **ER** (MPS55). Número de especificaciones aprobadas, implementadas y verificadas. Las especificaciones de alta calidad deben estar almacenadas electrónicamente, ser ejecutables o al menos interpretables, anotadas por importancia y estabilidad relativas, con su versión correspondiente, organizadas, con referencias cruzadas y especificadas al nivel correcto de detalle.

Tiempo medio de fallos (MPS56).
Es una métrica de fiabilidad, se calcula con la suma del tiempo medio de fallo TMDF y del tiempo medio de reparación TMDR (Ec. 4.11).

$$TMEF = TMDF + TMDR \tag{4.11}$$

Disponibilidad (MPS57).
La disponibilidad del software es la probabilidad de que un programa funcione de acuerdo con los requisitos en un momento dado. Es una métrica de fiabilidad se calcula mediante el tiempo medio de fallo (TMDF) y el tiempo medio de reparación (TMDR) (Ec. 4.12).

$$Disponibilidad = \left[TMDF * \left(TMDF + TMDR\right)\right] * 100\% \tag{4.12}$$

Calidad de la función desplegada (MPS58).
Se usa para desarrollar un amplio valor de prioridades para cada caso de uso. Los casos de uso se evalúan desde el punto de vista de todos los actores definidos para el sistema. Se asigna un valor de prioridad a cada caso (del 1 al 10) para cada actor, se calcula una prioridad para indicar la importancia percibida de cada caso de uso.

Métricas de hardware.
Se dice que son medidas que examinan si durante el proceso del software se tiene los dispositivos físicos necesarios para el desarrollo del producto. Éstas métricas son:

- *Métricas de performance* (MPS59). Revisa si se tiene el perfil correcto para el desarrollo del producto de software.
- *Métricas de fiabilidad* (MPS60). Examina si los equipos de cómputo o los demás dispositivos utilizados son eficaces.
- *Métricas de disponibilidad* (MPS61). Inspecciona si son utilizables los equipos de cómputo o los demás dispositivos físicos.

4.1.1.2 Métricas del producto

Son métricas de predicción y describen las características de un producto de software. Las relaciones entre estas características del software pueden variar de acuerdo a diversos factores, tales como: proceso, tecnología de desarrollo o tipo de sistema. Las métricas del producto son: (Demarco, 2007; SML@b, 2007; Barreiro, 2007; IFPUG, 1998; Kan, 2002; Klasky, 2003)

- Métricas del tamaño.
- Métricas de arquitectura.
- Métricas cualitativas de la calidad.
- Métricas de bases de datos relacionales.

Métricas del tamaño.
Tamaño de los atributos del proyecto. Se dice que existen cuatro atributos clave en la medición del software, lo son porque forman el cimiento para conocer diferentes indicadores de los programas de computadoras. Éstos son:

- Longitud del código (LDC) (MPD01).
- Puntos de función (PF) (MPD02).
- Complejidad ciclomática (MPD03).

Longitud del código.
La longitud del código (Tamaño del producto por líneas de código), es una medida del tamaño del programa en líneas de código (LDC). Cuanto mayor sea el tamaño de un componente, más complejo y susceptible de incorporar errores. Se debe definir la forma del conteo de LDC desde el análisis, es decir, si se toma en cuenta las líneas en blanco o los comentarios (Pressman, 2002).

Puntos de función.

Las métricas basadas en la función pueden estimar el tamaño del sistema, al examinan el modelo de análisis, entre otras medidas. Para calcular un punto de función se determinan cinco parámetros de medición del dominio o ámbito de la información, esto puede variar respecto al estilo de desarrollo de cada organización, con su correspondiente cantidad de elementos, los parámetros son: (IFP, 1994; Pressman, 2002)

- *Número de entradas del usuario.* Se cuenta cada entrada del usuario que proporciona datos orientados a la aplicación. Las entradas se deben diferenciar de las peticiones, las cuales se cuentan de forma separada.

- *Número de salidas de usuario.* Se cuenta cada salida que proporciona al usuario información orientada a la aplicación. En éste contexto la salida se refiere a informes, pantallas, mensajes de error, etc. Los elementos de datos particulares dentro de un informe no se cuentan de forma separada.

- *Número de peticiones de usuario.* Una petición se define como una entrada interactiva que produce la generación de alguna respuesta del software inmediata en forma de salida interactiva. Se cuenta cada petición por separado.

- *Número de archivos.* Se cuenta cada grupo lógico de datos que puede ser una parte de una gran base de datos o un archivo independiente.

- *Número de interfaces externas.* Se cuentan todas las interfaces legibles por la maquina (tales como: archivos de datos de cinta o disco) que se utilizan para la transmisión de información a otro sistema.

También se pondera la complejidad, ya sea simple, media o alta respecto a un valor numérico para cada dominio, después se utiliza la ecuación 4.13.

$$PF = CT \times \left[0.65 + 0.01 \times 6 \sum_{i=1}^{n} \left(F_i \right) \right] \qquad (4.13)$$

Donde **CT** es la suma de todas las entradas de los puntos de función que resultan del producto del número de elementos de cada parámetro de medición por el factor de ponderación que se le asigne, los valores constantes son los factores de peso que se aplican al número de elementos de los dominios y $\mathbf{F_i}$, con **i = 1, 2, …, 13, 14**, son valores de ajuste de la complejidad, los cuales se establecen de acuerdo a 14 preguntas, las cuales son: (Arthur, 1997):

a. ¿Requiere el sistema copias de seguridad y de recuperación fiables?
b. ¿Se requiere comunicación de datos?
c. ¿Existen funciones de procesamiento distribuido?
d. ¿Es crítico el rendimiento?
e. ¿Se ejecuta el sistema en un entorno operativo existente y fuertemente utilizado?
f. ¿Requiere el sistema entrada de datos interactiva?
g. ¿Requiere la entrada de datos interactiva que las transacciones de entrada se lleven a cabo sobre múltiples pantallas u operaciones?
h. ¿Se actualizan los archivos maestros de forma interactiva?
i. ¿Son complejas las entradas, las salidas, los archivos o las peticiones?
j. ¿Es complejo el procesamiento interno?

k. ¿Se diseño el código para ser reutilizable?
l. ¿Están incluidas en el diseño la conversión y la instalación?
m. ¿Se diseño el sistema para soportar múltiples instalaciones en diferentes organizaciones?
n. ¿Se diseño la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?

Cada una de las preguntas anteriores se responden de acuerdo a una escala de rangos (Véase Tabla 4.4) (Pressman, 2002).

Tabla 4.4. Factores de ajuste de complejidad.

Factores	Valor cualitativo
0	Sin influencia
1	Incidental
2	Moderado
3	Medio
4	Significativo
5	Esencial

Fuente: Barreiro, 2007.

Complejidad ciclomática.
Proporciona una medición cuantitativa del flujo de control y se usa en el contexto de camino básico de un grafo de flujo dirigido. Al definir el número de caminos independientes[5] del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez. En la figura 4.4 se puede apreciar que un camino independiente es el formado por 1-11 ó 1-2,3-4,5-10. La complejidad ciclomática de este grafo de flujo es V(G)=11 aristas – 9 nodos+2= 4. La complejidad se puede calcular de tres formas (Véase Figura 4.4), estas son:

- De acuerdo al número de regiones del grafo de flujo, debido a que éste coincide con el valor de la complejidad ciclomática.
- Mediante a la ecuación 4.14. La complejidad ciclomática de un grafo de flujo se representa con **V (G)**.

$$V(G) = A - N + 2 \qquad (4.14)$$

  Donde **A** es el número de aristas del grafo de flujo y **N** es el número de nodos del mismo.

- La complejidad ciclomática, **V (G)**, de un grafo de flujo G también se puede calcular mediante la ecuación 4.15.

[5] Un camino independiente es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición, está constituido por lo menos por una arista no recorrida, con anterioridad, a la definición del camino.

$$V(G) = P + 1 \qquad (4.15)$$

Donde **P** es el número de nodos predicado contenidos en le grafo de flujo **G**.

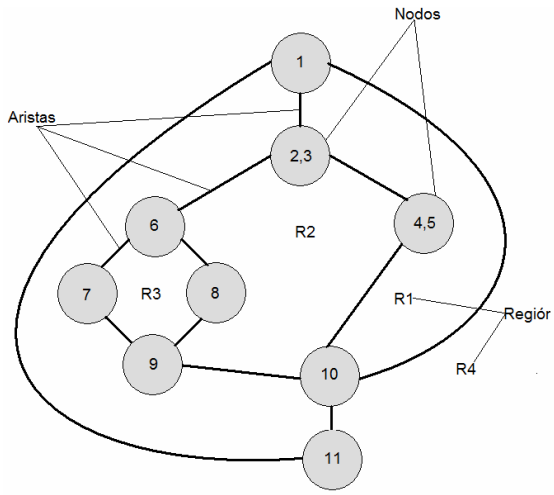


Figura 4.4. Grafo de flujo.
Fuente: Pressman, 2002.

Métricas de arquitectura.
Se dice que estas métricas inspeccionan lo referente a la estructura del producto de software, éstas son:

- Número de componentes (MPD04).
- Número de paradigmas de lenguaje en el producto (MPD05).
- Niveles de métricas (MPD06).
- Métricas de estructura.
- Métricas de profundidad.
- Métricas de acoplamiento.
- Métricas del diseño arquitectónico.
- Métricas de diseño a nivel de componentes.

Número de componentes.
Se debe registrar la cantidad de elementos que conforman al producto de software, tales como: interfaz de usuario o la base de datos.

Número de paradigmas de lenguaje en el producto.
Es necesario definir el número de los tipos de lenguaje a utilizar, así como el establecer cuando se usan y cual es su función.

Niveles de métricas.
Cuando se usan muchas métricas para medir el software, se debe establecer la prioridad entre ellas.

Métrica de estructura.
Estandarización de datos (MPD07). El empleo de estructuras y tipos de datos a lo largo del programa.

Métricas de profundidad.

- *Longitud de los identificadores* (MPD08). Medida de la longitud promedio de los diferentes identificadores de un programa. Cuanto mayor sea esta longitud, más probable es que tengan significado, y por tanto el programa será más comprensible.
- *Trazabilidad* (MPD09). La capacidad de seguir una representación del diseño o un componente real del programa hasta los requisitos.
- *Capacidad de expansión* (MPD10). El grado con que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
- *Generalidad* (MPD11). La amplitud de aplicación potencial de los componentes del programa.
- *Instrumentación* (MPD12). El grado con que el programa vigila su propio funcionamiento e identifica los errores que ocurren.

Métricas de acoplamiento.

El acoplamiento de módulo proporciona una indicación de la conectividad de un módulo con otros módulos, datos globales y el entorno exterior. La métrica de mayor uso para el acoplamiento de un módulo combina tres tipos de acoplamientos. Las medidas necesarias para definir el acoplamiento a través de las tipos anteriores se revisan en la tabla 4.5.

Tabla 4.5. Medidas para determinar el acoplamiento de un módulo.

Tipo	Medidas
Acoplamiento de flujo de datos y control (MPD13).	Número de parámetros de datos de entrada, d_i. Número de parámetros de control de entrada, c_i. Número de parámetros de datos de salida, d_o. Número de parámetros de control de salida, c_o.
Acoplamiento global (MPD14).	Número de variables globales usadas como datos, $g_{d.}$ Número de variables globales usadas como control, $g_{c.}$
Acoplamiento de entorno (MPD15).	Número de módulos llamados (expansión), w. Número de módulos que llaman al módulo en cuestión (concentración), r.

Fuente: Dhama, 1995.

Mediante la utilización de estas medidas se define el *grado de acoplamiento del módulo*, **C** (Véase Ec. 4.16).

$$C = 1 - \frac{k}{\left(d_i + a \times c_i + d_o + d \times c_o + g_d c \times g_c + w + r\right)} \qquad (4.16)$$

Donde **k=1**, **a=b=c=2**, con la consideración de que estos valores se ajustan a medida que se verifican experimentalmente. El grado de acoplamiento aumenta conforme el valor de las medidas de acoplamiento crece. Pero no aumenta linealmente entre un valor mínimo en el rango de 0.66 hasta un máximo que se aproxima a 1.0.

Métricas del diseño arquitectónico.
Estas métricas no requieren ningún conocimiento del trabajo interno de un módulo en particular del sistema. Las métricas de diseño arquitectónico son:(Henry y Kafura, 1981; Card y Glass, 1990; Fenton, 1991)

- *Complejidad estructural* **S (i)** (MPD16). Se define con la ecuación 4.17:

$$S(i) = f^{2}_{out}(i) \tag{4.17}$$

  Donde $\mathbf{f_{out}}$ **(i)** indica el número de módulos invocados directamente invocados por el módulo **i**, es decir el número de llamadas desde el módulo **i**.

- *Complejidad de datos* **D (i)** (MPD17). Proporciona una indicación de la complejidad en la interfaz interna de un módulo i (Ec. 4.18).

$$D(i) = \frac{v(i)}{[f_{out}(i)+1]} \tag{4.18}$$

- *Complejidad del sistema* **C (i)** (MPD18). Se define como la suma de las complejidades estructural y de datos (Ec. 4.19).

$$C(i) = S(i) + D(i) \tag{4.19}$$

- *Métrica Henry-Kafura* **MHK** (MPD19). Es una métrica de diseño arquitectónico de alto nivel, emplea la expansión y la concentración, también se denomina métrica del flujo de la información (Ec. 4.20).

$$MHK = longitud(i) \times [f_{in}(i) + f_{out}(i)]^2 \tag{4.20}$$

  Donde **longitud (i)** es el número de sentencias en lenguaje de programación en el módulo **i** y $\mathbf{f_{in}}$ es la concentración del módulo **i**, es decir el número de llamadas hacia el módulo **i** que tiene. Para su cálculo se emplea el diseño procedimental para estimar el número de sentencias del lenguaje de programación del módulo **i**.

- *Métricas de morfología simples*. Permiten comparar diferentes arquitecturas de programa mediante un conjunto de dimensiones directas. A partir de la figura 4.5 se pueden definir las siguientes métricas: (Fenton, 1991)

  - Tamaño (MPD20) =nodos+arcos.
  - Profundidad (MPD21) = el camino más largo desde el nodo raíz (parte más alta) a un nodo hoja.
  - Anchura (MPD22) = máximo número de nodos de cualquier nivel de la arquitectura.
  - Relación arco-a-nodo (MPD23), r=arcos/nodos. Mide la densidad de conectividad de la arquitectura y puede proporcionar una sencilla indicación del acoplamiento de la arquitectura.

A partir de la figura 4.5 se ve que el tamaño es de 36, profundidad tiene un valor de 4, la anchura es 6 y el valor de la relación arco-nodo es 1, mismo que indica que la arquitectura propuesta se acopla en una realción 1 a 1.

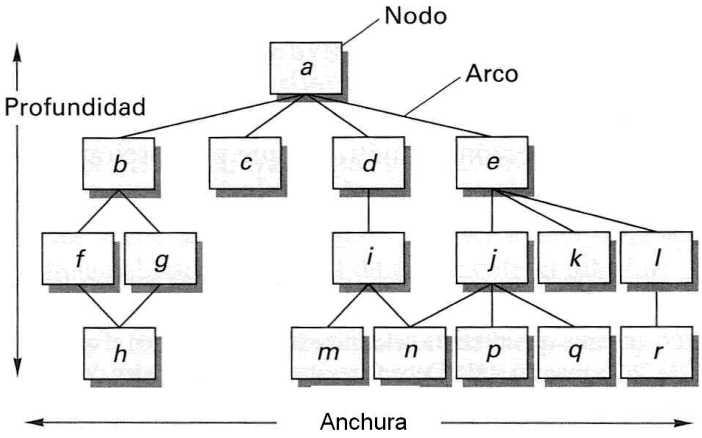


Figura 4.5. Métricas de morfología. Los valores de las medidas propuestas son: profundidad=4, anchura=6, r=1.
Fuente: Pressman, 2002.

Métricas de diseño a nivel de componentes.

Se concentran en las características internas de los componentes del software y ayudan al desarrollador de software a juzgar la calidad de un diseño a nivel de los componentes. Requieren conocimiento del trabajo interno del módulo en cuestión. Se pueden aplicar una vez que se desarrolla un diseño procedimental y también se pueden retrasar hasta tener disponible el código fuente. La cohesión es una indicación cualitativa del grado que tiene un módulo para centrarse en una cosa. Las métricas de cohesión son métricas de diseño a nivel de componentes y se basen en cinco medidas, las cuales se pueden revisar en la tabla 4.6. (Bieman y Ott, 1994; Sears, 1993)

Tabla 4.6. Medidas de la cohesión.

Medida	Descripción
Porción de datos (MPD24).	Es una marcha atrás a través de un módulo que busca valores de datos que afectan a la loca1izacion de módulo en el que empezó la marcha atrás. Se pueden definir tantas porciones de programas (que se centran en enunciados y condiciones) como porciones de datos.
Muestras (tokens) de datos (MPD25).	Variables definidas para un módulo pueden definirse como muestras de datos para el módulo.
Señales de unión (MPD26).	Con junto de muestras de datos que se encuentran en una o más porciones de datos.
Señales de superunión (MPD27).	Muestras de datos comunes a todas las porciones de datos de un módulo.
Adhesividad (MPD28).	Es el grado relativo con el que las señales de unión ligan juntas porciones de datos. La pegajosidad relativa de una muestra de unión es directamente proporcional al número de porciones de datos que liga.

Fuente: Bieman y Ott, 1994.

Las métricas de cohesión presentadas en la tabla 4.6 tienen valores que van desde 0 hasta 1. Tienen un valor de 0 cuando un procedimiento tiene más de una salida y no muestra ningún atributo de cohesión indicado por una métrica particular. Un procedimiento sin señales de superunión, sin señales comunes a todas las porciones de datos, no tiene una cohesión funcional fuerte (no hay señales de datos que contribuyan a todas las salidas) Un procedimiento sin señales de unión, es decir, sin señales comunes a más de una porción de datos (en procedimientos con más de una porción de datos), no muestra una cohesión funcional débil y ninguna adhesividad (no hay señales de datos que contribuyan a más de una salida). La cohesión funcional fuerte y la adherencia se obtienen cuando las métricas toman un valor máximo de 1.

Métricas de la calidad.
Se dice que las métricas de calidad son la parte más importante del desarrollo de un producto de software. Las métricas de calidad de la norma son: (ISO/IEC 9126-1, 2001)

- *Métricas de funcionalidad:* idoneidad (MPD29), e*xactitud* (MPD30), *interoperabilidad* (MPD31) y seguridad (MPD32).
- *Métricas de confiabilidad:* madurez (MPD33), tolerancia a fallos (MPD34) y capacidad de recuperación (MPD35).
- *Métricas de facilidad de uso:* facilidad de comprensión (MPD36), facilidad de aprender (MPD37) y operatibilidad (MPD38).
- *Métricas de eficiencia:* tiempo de funcionamiento (MPD39) y utilización de recursos (MPD40).
- *Métricas de capacidad de mantenimiento:* capacidad de análisis (MPD41), variabilidad (MPD42), estabilidad (MPD43) y facilidad de prueba (MPD44).
- *Métricas de portabilidad:* adaptabilidad (MPD45), capacidad de instalación (MPD46) y capacidad de reemplazo (MPD47).

Métricas de bases de datos relacionales.
Mario Piattini (2000) propone cuatro métricas para medir la complejidad de las bases de datos relacionales, las cuales se dividen en métricas orientadas a las tablas y métricas orientadas al diagrama entidad-relación. Éstas son:

- *Número de atributos* NA (MPD48). Cantidad de atributos en todas las tablas del diagrama y NA(A) es el número de atributos de la tabla A.
- *Grado de referenciabilidad* GR (MPD49). Número de claves ajenas del diagrama y GR(A) es el número de claves foráneas de la tabla A.
- *Profundidad del árbol referencial* PAR (MPD50). Longitud del máximo camino referencial del diagrama entidad-relación. Los ciclos sólo se consideran una vez.
- *Radio de normalidad* RN (MPD51). Número de tablas en tercera forma normal o superior dividido entre el número de tablas en el diagrama (Véase la Ec.4.21).

$$RN = \frac{Número\ de\ tablas\ en\ 3FN}{Número\ de\ tablas\ en\ el\ diagrama} \quad (4.21)$$

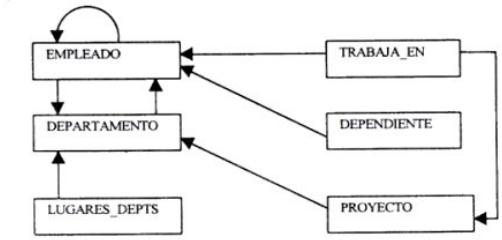
En la figura 4.6b se muestra el modelo entidad-relación de una empresa, donde las flechas indican las relaciones de integridad referencial. En la figura 4.6a se puede ver el pseudocódigo del modelo relacional de la empresa. Los valores de las métricas para la complejidad de bases de datos relacionales se muestran en la figura 4.6c, y por último, a partir del camino referencial que se muestra en la figura 4.6d se ve que el valor para la métrica DRT es cinco.

a)

```
Create table empleado
    (
   nombrep varchar(15), not null,
   inic          char,
   apellido     varchar(15) not null,
   nss           char(9)        not null,
   fechaen     date,
   direccion   varchar(30),
   sexo          char,
   salario       decimal (10,2),
   nsssuper   char(9),
   nd             int               not null,
constraint clpemp
   primary key (nss),
constraint clesuperemp
   foreign key (nsssuper)references empleado(nss)
                ondelete set null on update cascade),
constraint cledepoemp
   foreign key (nd) references departamento (numerod)
               ondelete set default on update cascade;
  Create table departamento
    (
   nombred varchar(15) not null,
   numerod int               not null,
   nssgte    char(9)         not null,
   fechainicgte     date,
constraint clpdepto
   primary key (numerod),
constraint clsdepto
   unique(nombred),
constraint clegtedepto
   foreign key (nssgte) references empleado(nss)
               ondelete set default on update cascade);
```

```
Create table lugar_depts
    (
   numerod  int              not null,
   lugard       varchar(15) not null,
 primary key (numerod, lugard),
 foreign key (numerod) references departamento
(numerod)
          on delete set null on update cascade);
Create table proyecto
    (
   nombrepr varchar(15)       not null,
   numeropr       int               not null,
   lugarpr    varchar(15)        not null,
   numd            int               not null,
 primary key (numerod),
 unique (nombrep),
 foreign key (numd) references departamento (numerod));
Create table trabaja_en
    (
   nsse                  char(9)    not null,
   nump                  int          not null,
   horas       decimal(3,1)       not null,
 primary key (nsse, num),
 foreign key (nsse) references empleado (nss));
 foreign key (nump) references proyecto (numerop));
Create table dependiente
    (
   nss                   char(9)             not null,
   nombredepend    varchar(15)        not null,
   sexo       char,
   fechaan  date,
   relacion  varchar(8),
 primary key (nsse, nombre_depend),
 foreign key (nsse) references empleado (nss));
```

b)



c)

	NA	RD	DRT	NR
EMPLEADO	10	2		
DEPARTAMENTO	4	1		
LUGARES_DEPTS	2	1		
PROYECTO	4	1		
TRABAJA_EN	3	2		
DEPENDIENTE	5	1		
ESQUEMA	28	8	5	1

d)

TRABAJA EN → PROYECTO → DEPARTAMENTO → EMPLEADO → EMPLEADO → DEPARTAMENTO

Figura 4.6. Métricas de bases de datos relacionales para una empresa. a) Modelo entidad-relación de la empresa. b) Pseudocódigo del modelo relacional de la empresa. c) Valores de las métricas para la complejidad de bases de datos relacionales. d) Camino referencial para medir el DRT. Fuente Piattini, 2000.

4.2 CARACTERÍSTICAS DEL SOFTWARE PARA LA ADMINISTRACIÓN DE PROYECTOS

Se presenta una relación de las características deben ofrecer la mayor parte de los programas de software para la administración de proyectos, éstas son: (Guido, 2003)

- *Control del costo*. Un software de computadora para a administración de proyectos brinda la posibilidad de relacionar los costos con cada actividad y con cada recurso en un proyecto. Además, admite desarrollar fórmulas preestablecidas para manejar funciones de costos, con la intención de ayudar a calcular los costos estimados del proyecto y darles seguimiento durante el mismo. En cualquier momento se debe poder comparar los costos reales con los presupuestos para recursos individuales, para grupos de recursos, o para todo el proyecto.

- *Calendarios*. Los calendarios se usan para establecer los días y horas laborables para cada recurso individual o grupo de recursos y también para calcular el programa del proyecto. Por lo que un programa de administración de proyectos debe permitir registrar, modificar y comunicar esta información.

- *Capacidades de Internet*. De ser requerido los programas de administración de proyectos permiten colocar directamente en un sitio Web la información concerniente al proyecto, protegida con contraseñas, para facilitar la adquisición de la información a los miembros del equipo y con el cliente.

- *Gráficas*. La capacidad de producir con facilidad y rapidez diversas graficas, tales como: gráficas de Gantt y diagramas de red, es relevante el un software para la administración.

- *Presentación de informes*. La capacidad de crear reportes es un atributo generalizado en los programas de administración, sobre el proyecto como un conjunto y sobre los pasos principales (puntos de referencia) de un proyecto.

- *Planeación*. Los programas de software para la administración de proyectos crear una estructura de división del trabajo (EDT) para ayudar en el proceso de planeación.

- *Seguridad*. Esta característica consiste en proporcionan el acceso mediante contraseñas al propio programa de administración de proyectos o bien a sus archivos individuales o la información especifica dentro de un archivo.

4.3 INGENIERÍA DE REQUISITOS

Al revisar cada uno de los modelos del proceso de desarrollo del software, es clara la necesidad de capturar los requerimientos para conocer las expectativas respecto al sistema, tanto los clientes y los usuarios. Un requisito es una capacidad o funcionalidad con la cual debe cumplir el sistema, describe el comportamiento de un sistema: a medida que el sistema actúa sobre los datos o las instrucciones, los objetos y las entidades se mueven desde un estado de ser a otro. Describen a su vez las actividades del sistema como una reacción a la entrada y el estado de cada entidad el sistema antes y después de que ocurra dicha actividad. Lo primero es obtener la información acerca de producto por parte del cliente y/o usuario. (Pfleeger, 2002; Torres, 2007)

4.3.1 Requerimientos funcionales y no funcionales

Los requerimientos se pueden definir de maneras dos diferentes maneras, éstos son: (Op. cit.)

- *Requerimientos funcionales*. Describen interacciones en el sistema y su ambiente y como deben comportarse el sistema ante determinado estímulo.

- *Requerimientos no funcionales*. Describen una restricción sobre el sistema que limita las elecciones en la construcción de una solución del problema, es decir, se refieren a atributos del sistema o a atributos del ambiente del sistema.

4.3.2 Características de los requerimientos

Las características de un requerimiento son sus propiedades principales. Un conjunto de requerimientos en estado de madurez, deben presentar una serie de características tanto individualmente como en grupo. Tales características son: (Torres, 2007)

- *Necesario*. Un requerimiento es necesario si su omisión provoca una deficiencia en el sistema a construir, y además su capacidad, características físicas o factor de calidad no pueden ser reemplazados por otras capacidades del producto o del proceso.

- *Conciso.* Un requerimiento es conciso si es fácil de leer y entender. Su redacción debe ser simple y clara para aquellos que vayan a consultarlo en un futuro.

- *Completo.* Un requerimiento está completo si no necesita ampliar detalles en su redacción, es decir, si se proporciona la información suficiente para su comprensión.

- *Consistente.* Un requerimiento es consistente si no es contradictorio con otro requerimiento.

- *No ambiguo.* Un requerimiento no es ambiguo cuando tiene una sola interpretación. El lenguaje usado en su definición, no debe causar confusiones al lector.

- *Verificable.* Un requerimiento es verificable cuando se puede cuantificar de manera que permita hacer uso de los siguientes métodos de verificación: inspección, análisis, demostración o pruebas.

4.3.3 Ingeniería de requisitos

Se define como un enfoque sistemático para obtener, organizar y documentar los requisitos del sistema, así como mantener un acuerdo entre el cliente y el equipo del proyecto en los cambios a los requisitos. Se debe considerar que la clave para una efectiva administración de requisitos es mantener un enunciado claro de los requisitos, junto con atributos para cada tipo de requisitos y su seguimiento con otros requisitos o elementos del proyecto. Se puede mencionar que se deben considerar las siguientes actividades: (Torres, 2007)

- *Análisis del problema*. Se entiende el problema, así como las necesidades del cliente y proponer una solución. Se debe de identificar los límites de la solución y las restricciones de ésta.

- *Entendimiento de las necesidades de los clientes*. Se determina cuales son las mejores fuentes de información, tener acceso a ellas y determinar cual es la mejor forma de obtener información de ellas. Las actividades para la obtención de información utilizan técnicas tales como: entrevistas, tormentas de ideas, prototipos conceptuales o cuestionarios.

- *Definición del sistema*. Se traducen las necesidades de los clientes a una descripción significativa del sistema que va a ser construido.

- *Administrar el alcance del proyecto*. Para llevar eficientemente un proyecto es necesario asignar prioridades a los requisitos, dado que el objetivo se enfoca en tareas que mitiguen el riesgo del proyecto o estabilizar la arquitectura de la aplicación. Requiere administrar las salidas del proyecto en sus diferentes fases.

- *Refinar la definición del sistema*. Se detalla la definición del sistema para que ésta pueda ser entendida por el cliente. También se define como se prueba el sistema.

- *Administrar el cambio de los requisitos*. Incluye las actividades de establecer una línea base, establecer que dependencias son importantes de dar seguimiento, establecer un seguimiento entre elementos relacionados y controlar el cambio.

4.3.4 Clasificación de los requisitos

Aunque existen muchas clasificaciones de requisitos. Una de éstas es el modelo FURPS+, usando el acrónimo FURPS (por las siglas en inglés), para describir las principales categorías de requisitos: (Torres, 2007)

- Funcionalidad.
- Facilidad de Uso.
- Confiabilidad.
- Rendimiento.
- Soporte.

El signo “+” dentro del nombre del modelo indica que se deben de incluir requisitos tales como: (Torres, 2007)

- Restricciones de diseño.
- Requisitos de Implementación.
- Requisitos de Interfase.
- Requisitos físicos.

En la tabla 4.7 se revisa la clasificación de requisitos de acuerdo al modelo FURPS+ y los requisitos básicos que contiene.

Tabla 4.7. Medidas para el índice de madurez del software.

Categoría	Requisitos
Funcionalidad	Conjunto de Características. Capacidades. Seguridad.
Facilidad de uso.	Factores humanos. Estéticos. Consistencia en la interfaz de usuario. Ayuda en línea. Asistentes. Documentación del usuario. Material de capacitación.
Confiabilidad.	Frecuencia y severidad de fallas. Recuperación a fallos. Tiempo entre fallos.
Rendimiento.	Impone condiciones a los requisitos funcionales. Como a una acción dada. Se pueden especificar los siguientes parámetros de rendimiento: Velocidad. Eficiencia. Disponibilidad. Tiempo de Respuesta. Tiempo de Recuperación. Utilización de Recursos.
Soporte.	Requisitos de instalación. Requisitos de Configuración. Requisitos de Adaptabilidad. Requisitos de Compatibilidad.
Requisitos de diseño.	También llamados restricciones de diseño, especifican o restringen el diseño de un sistema.
Requisitos de Implementación.	Un requisito de implementación especifica o restringe la codificación o construcción de un sistema. Algunos son: Estándares requeridos. Lenguajes de implementación. Políticas para la integridad de la Base de Datos. Limites de los recursos. Ambientes de operación.
Requisitos de Interfase.	Un elemento externo con el cual el sistema debe de interactuar. Restricciones en formato, tiempos u otros factores.
Requisitos físicos.	Este tipo de requisitos puede ser usado para representar requisitos de hardware, así como la configuración de red requerida: Material. Forma. Peso. Tamaño.

Fuente: Pfleeger, 2002; Torres, 2007.

4.3.5 Técnicas de ingeniería de requerimientos

Las principales ventajas y desventajas de cada una de las técnicas que se utilizan en las etapas de la ingeniería de requerimientos se muestran en la tabla 4.8.

Tabla 4.8. Comparación de las técnicas utilizadas en la ingeniería de requisitos.

Técnica	Ventajas	Desventajas
Entrevistas y Cuestionarios	Mediante ellas se obtiene una gran cantidad de información correcta a través del usuario. Pueden ser usadas para obtener una idea general del dominio del problema. Son flexibles. Permiten combinarse con otras técnicas.	La información obtenida al principio puede ser redundante o incompleta. Si el volumen de información manejado es alto, requiere mucha organización de parte del analista, así como la habilidad para tratar y comprender el comportamiento de todos los involucrados.
Lluvia de Ideas	Los diferentes puntos de vista y las confusiones en cuento a terminología, son aclaradas por expertos. Ayuda a desarrollar ideas unificadas basadas en la experiencia de un experto.	Es necesaria una buena compenetración del grupo participante.
Prototipos	Ayudan a validar y desarrollar nuevos requerimientos. Permite comprender aquellos requerimientos que no están muy claros y que son de alta volatilidad.	El cliente puede llegar a pensar que el prototipo es una versión del software que será desarrollado. A menudo, el desarrollador hace compromisos de implementación con el objetivo de acelerar la puesta en funcionamiento del prototipo.
Análisis Jerárquico	Permite determinar el grado de importancia de cada requerimiento. Ayuda a identificar conflictos en los requerimientos. Muestra el orden en que deben ser implementados los requerimientos.	Debe construirse un estándar claro de evaluación, que incluya la participación del cliente.
Casos de Uso	Representan los requerimientos desde el punto de vista del usuario. Permiten representar más de un rol para cada afectado. Identifica requerimientos estancados, dentro de un conjunto de requerimientos.	En sistemas grandes, toma mucho tiempo definir todos los casos de uso. El análisis de calidad depende de la calidad con que se haya hecho la descripción inicial.

Fuente: Herrera y Lizka, 2007.

4.3.6 Casos de uso

Una vez recopilados los requisitos, se puede crear un conjunto de escenarios que identifiquen una línea de utilización para el sistema que va a ser construido. Los escenarios, algunas veces llamados casos de uso, facilitan una descripción de cómo puede ser le uso del sistema. Para crear un caso de uso, se debe identificar los diferentes tipos de personas (o dispositivos) que utiliza el sistema o producto. Un actor es algo que comunica con el sistema o producto, es externo al sistema en sí mismo y representa una clase de entidades externas (a veces, pero no siempre personas) que lleva a cabo un papel. Los actores interactúan para conseguir funciones del sistema y derivar el beneficio deseado del sistema, trabajan directa y frecuentemente con el software y dan soporte al sistema. Una vez que se identifican los actores, se pueden desarrollar los casos de uso. EI caso de uso describe la manera en que los actores interactúan con el sistema. Los casos de uso describen entornos que se perciben de distinta forma por distintos actores. (Jacobson, 1992)

5. DOCUMENTO DE ESPECIFICACIÓN DE REQUISITOS DEL SISTEMA

5.1 CONSIDERACIONES PRELIMIRARES

Este documento de especificación de requisitos está basado en la investigación que se realizó en los capítulos anteriores. Los escenarios se concretan en casos de uso, diagramas de casos de uso y en los modelos de flujo de datos. Las métricas y demás información a almacenar se explican en el diccionario de datos. Para elaborar esta propuesta de un documento de especificación de requisitos se toma como requerimiento establecido por el cliente, que la herramienta se desarrolle en ambiente Web mediante el uso de PHP, MySQL, Samba y Linux.

5.1.1 Propósito del sistema

El sistema soluciona la necesidad de la implementación de un sistema de gestión de una base de datos histórica para el desarrollo de proyectos de software basados en el modelo de ciclo de vida de los sistemas de información, o CVSI, con base de datos relacional, donde se almacene la información de los proyectos de software finalizados que desarrolla una empresa.

5.1.2 Alcance del sistema

Hacer disponible una herramienta desarrollada con PHP y MySQL, basada en Linux y cuyo administrador de archivos sea SAMBA que proporcione las interacciones mínimas necesarias para el establecimiento de un sistema de gestión de una base de datos histórica de proyectos de software, en una intranet. El programa definido sólo será aplicable para proyectos basados en el ciclo de vida de los sistemas de información con bases de datos relacionales.

5.1.3 Objetivos del proyecto

Mediante el diccionario de datos se establecerán los parámetros mínimos necesarios sobre la ingeniería de software, el desarrollo de proyectos de software y métricas de software que se almacenarán en una base de datos relacional.

Se concentrará toda la información mínima necesaria para establecer la base de datos histórica de proyectos de software a partir de los datos resultantes de los proyectos finalizados.

Se realizarán plantillas que brinden la capacidad de ajustar a la medida las cuantificaciones de las métricas del software que se reunirán en la base de datos histórica de proyectos de software.

5.1.4 Panorama

La industria del software es una empresa altamente rentable, por lo tanto, es relevante que en México se desarrolle software de calidad. Un grave problema de las organizaciones de desarrollo de productos de software es que no registran de forma adecuada el historial de proyectos que llevan a cabo. Al emprender un nuevo proyecto sin las bases adecuadas: conocimiento técnico disponible, conocimiento de las características de modelos de ciclo de vida ya empleados, información acerca de los problemas existentes en la administración de proyectos anteriores o si el uso de ciertas medidas logra ayudar al equipo de desarrollo del software a obtener un producto de calidad, entre otras características, provoca la pérdida de tiempo y dinero. El establecer un documento de especificación de los requisitos para el desarrollo de un sistema de gestión de una base de datos histórica de proyectos de software con un modelo de ciclo de vida de los sistemas de información, es una propuesta para resolver esta problemática.

5.2 SISTEMA ACTUAL

Se considera como sistema actual, el archivar sin ninguna técnica automatizada, todo lo que se cree importante acerca de los proyectos de desarrollo del software. Esta forma de archivo no necesariamente se da en una base de datos, de manera electrónica y menos de forma centralizada.

5.3 SISTEMA PROPUESTO

5.3.1 Panorama

El medio para ponderar las tareas realizadas en el proyecto de desarrollo de software es establecer que se va a medir, por lo tanto es relevante tomar en cuenta las métricas del software, tanto del proceso como del producto. Así también es importante la información pertinente de la administración de proyecto, tales como: el tipo de proyecto, la estructura de la organización, estructura del equipo de desarrollo, planeación, estimación, es decir, los estudios de factibilidad, la administración del riesgo, el plan de proyecto y el control. Por último son significativos de los procesos que conllevan la ingeniería del software, tales como: tipo del software a desarrollar, calidad, nivel de madurez de la empresa, estándares y modelos de proceso. El nombre de la herramienta propuesta es TOOLBDH.

5.3.2 Requerimientos no funcionales

Interfaz de usuario y factores humanos.
La interfaz de usuario se debe establecer en ambiente Web, es decir, los diferentes escenarios se deben mostrar en formato HTML. El sistema debe tener una interfaz de uso intuitiva y sencilla, complementada con un buen sistema de ayuda. Cada vez que seleccione una opción

debe aparecer una descripción de la opción elegida. Debe proveer de guías para realizar tareas que no sean habituales, permitir que el usuario pueda salir ágilmente de la aplicación, y pueda dejar respaldado el estado de avance de su trabajo, para continuar desde ese punto en otra oportunidad. Se debe admitir realizar la inversa de cualquier acción que pueda llegar a ser de riesgo, es decir contar con un comando deshacer. Se deben establecer selectores de opciones tales como combobox, grupos de radiobutton y de checkbutton. Se debe dar un esquema base de Métricas del SW y el Gestor del proyecto puede editarla. Además debe permitir las siguientes capacidades:

- Cuestiones de ingeniería de software: tipo de la aplicación, modelo de desarrollo y plan de proyecto.
- Cuestiones de Administración de proyectos: Nivel CMMI de la empresa al momento de desarrollar el proyecto y tipo del proyecto.
- *Planeación*. Para ayudar en el proceso de planeación se debe permitir la creación de EDT y diagramas de Gantt.
- *Elección de tipo de gráficas*. Se debe producir diversas graficas, tales como: pastel y barras.
- *Análisis.* Permite al gestor superior y al gestor del proyecto controlar mejor los riesgos relacionados con el proyecto. El análisis de debe presentar como un reporte. Se debe dar la capacidad de elegir calcular, guardar e imprimir los valores. Se puede optar por lo siguiente: (Véase Tabla 4.1)
  - Métricas de organización.
  - Métricas de revisión.
  - Métricas de productividad: mediante las cantidades de errores encontrados y/o defectos respecto a LDC, PF, tiempo o costo. Cuando se llena la EDT deben aparecer los miembros del equipo automáticamente para que se les asigne la tarea que le corresponde.
  - Métricas de logros.
  - Para el control de costos se utilizan algunas métricas de revisión, otro tipo de métricas y la comparación de los costos reales con el presupuesto para todo el proyecto.
- *Presentación de informes*. Los reportes deben mostrar los tipos de gráficas ya especificados, así como presupuestos, evaluaciones de métricas, EDT y diagrama de Gantt.
- Cuando los usuarios profesionales ingresen información, al validar su ID y contraseña debe aparecer sólo que datos puede ingresar o modificar de acuerdo a las actividades asignadas.

Características de los usuarios.
El sistema va a acceder mediante Web, por lo cual puede ser utilizado por varios usuarios con distintos niveles culturales, experiencia y especialización técnica, por lo tanto éste debe desarrollarse de forma que todos los posibles usuarios puedan acceder a él sin problemas.

Tiempo de Respuesta.
Las consultas a la BD no deben contener o solicitar información redundante e innecesaria para que el tiempo de respuesta del TOOLBDH sea lo menor posible.

Tolerancia a fallas.
El Sistema no debe fallar, es decir, debe ser tolerante en caso de que se realice una acción inválida relacionada con la conexión a la BD o las consultas, tal como la inserción de información redundante o la petición de información inexistente.

Documentación.
Como documentación se requiere de un Manual de Usuario, con el propósito de este manual es proveer de información sobre la forma de operar el sistema, que servirá de apoyo a los usuarios. El manual deberá tener una guía de las funcionalidades del sistema y de cómo aplicarlas, expresadas en un lenguaje fácil de comprender por usuarios inexpertos. Dicho manual debe ser presentado en formato Web, documento de Word o documento PDF.

Guías de instalación, configuración y archivo Léame.
La instalación deberá ser automática por lo cual no es necesaria una guía de instalación. Se debe contar con un archivo Léame con la información de la instalación de la aplicación y requerimientos para la misma. Se debe generar un archivo en formato PHP con información necesaria para la instalación automática.

Consideraciones de hardware.
Como performance mínimo la herramienta debe poder ser ejecutada en una computadora con 96 MB en RAM, procesador Pentium II o su equivalente y tener la disponibilidad de estar conectada a una intranet con una velocidad de 96Kbps. Para el caso del servidor es requerida una configuración con memoria RAM de 1GB o superior, disco duro de 120 GB y con una velocidad mínima de 128 Kbps.

Requerimientos de implementación.
TOOLBDH se debe desarrollar como un servicio web basado en herramientas bajo licencia GNU GPL, accesible desde cualquier navegador Explorer 5.0, Netscape 5.0, Mozilla Firefox 2.0.0.6 o superior, y estar instalado en un servidor Linux CENTOS 5.0, actuando como servidor de páginas web el servidor HTTP Apache 2.2.4 .

En aplicaciones Web hay baja concurrencia en la modificación de datos y el entorno es intensivo en lectura de datos, por lo cual el SGBD MySQL 5.0.45 es ideal como el gestor de BD de TOOLBDH. En cuanto la configuración de directorios Linux como recursos para compartir a través de la red es idóneo usar SAMBA 3.0.23d.

Cuestiones de calidad.
La calidad de la presente aplicación radica en cumplir con las siguientes características:

- La aplicación debe ser correcta, al cumplir con los requerimientos funcionales y no funcionales.

- TOOLBDH debe cumplir los requerimientos siempre.
- El sistema debe tener un comportamiento eficiente en todas las computadoras en las cuales se instale la aplicación.
- Se debe verificar que TOOLBDH cumpla con las características de calidad mencionadas en el apartado sobre calidad del presente documento.
- La herramienta debe tener una interfaz amigable y útil para los usuarios.
- Si existe algún problema con la aplicación durante la ejecución el mismo sistema debe ofrecer la capacidad de corregirlo.
- TOOLBDH debe poseer la cualidad de ser flexible ante algunos cambios especificados en el apartado de modificaciones al sistema.
- Se debe poder realizar pruebas especificadas.
- Al ser portable, la aplicación debe poder emplearse en todas las computadoras que tengan las características indicadas en el apartado de consideraciones del hardware del presente documento.
- La herramienta debe ser reutilizable al ser mejorada para adaptarse a una empresa de mayor envergadura.
- TOOLBDH debe interactuar con las demás aplicaciones existentes en la empresa ya sean de oficina, de gestión de BD o de desarrollo.

Ambiente físico.
Esta aplicación debe tener la portabilidad de implementarse en una empresa de desarrollo de productos de software pequeña o mediana.

Cuestiones de seguridad.
Además, en el desarrollo de TOOLBDH se debe tener en cuenta las directrices de política de seguridad siguientes:

- Los usuarios deben contar con navegadores y plataformas seguras, libres de virus y vulnerabilidades. También debe garantizarse la privacidad de los datos del usuario, la seguridad del servidor Web y de los datos almacenados.
- Se debe garantizar la operación continua del servidor, que los datos no sean modificados sin autorización (integridad) y que la información sólo sea distribuida a las personas autorizadas (control de acceso).
- Se debe garantizar que la información en tránsito, es decir la información que viaja entre el servidor Web y el usuario, no sea leída (confidencialidad), modificada o destruida por terceros.
- También es importante asegurar que el enlace entre cliente y servidor no pueda interrumpirse fácilmente (disponibilidad).

- Se debe realizar un control de acceso a la aplicación, solicitando login y password a los usuarios: gestor superior, gestor de proyecto y profesional.

- El gestor superior tiene permiso para revisar toda la información del proyecto y para ingresar, modificar o eliminar datos de configuración del sistema y del proyecto.
- El gestor del proyecto tiene la capacidad de revisar y modificar, ingresar o eliminar la información referente a la configuración del proyecto.

- Los profesionales tienen la capacidad de revisar, modificar o eliminar los datos del esquema de métricas.

- Debe existir una bitácora de acciones que registre quien las hace y cuando.

Control de Acceso.
Los usuarios de la aplicación tienen tres roles asignados los cuales indican a que tipo de funcionalidades y datos pueden acceder. Al ingresar el usuario, la aplicación debe tener en cuenta esta información para determinar que funcionalidades dicho usuario puede realizar y que datos puede manipular.

Modelo del proceso.
Se debe seguir un proceso de desarrollo claro, correcto y ordenado, basado en UML.

Acceso a la Base de Datos.
El acceso a la BD debe hacerse mediante el uso de un formulario desarrollado en PHP. Este acceso se debe realizar mediante consultas. El manejo de los archivos deber ser hecho mediante SAMBA, solo se debe guardar en la BD las rutas de acceso a esos archivos.

Retorno de Información desde la Base de Datos.
Los resultados de cada inserción, modificación o petición de datos deben ser mostrados en una página Web para que el usuario pueda examinarlos.

5.3.3 Requerimientos funcionales.

Los cuatro los grupos de requerimientos funcionales que se establecen para TOOLBDH además de sus elementos se pueden ver en la figura 5.1, éstos son:

- Configuración del sistema.
- Configuración del proyecto.
- Ingreso de información.
- Revisión de la información.

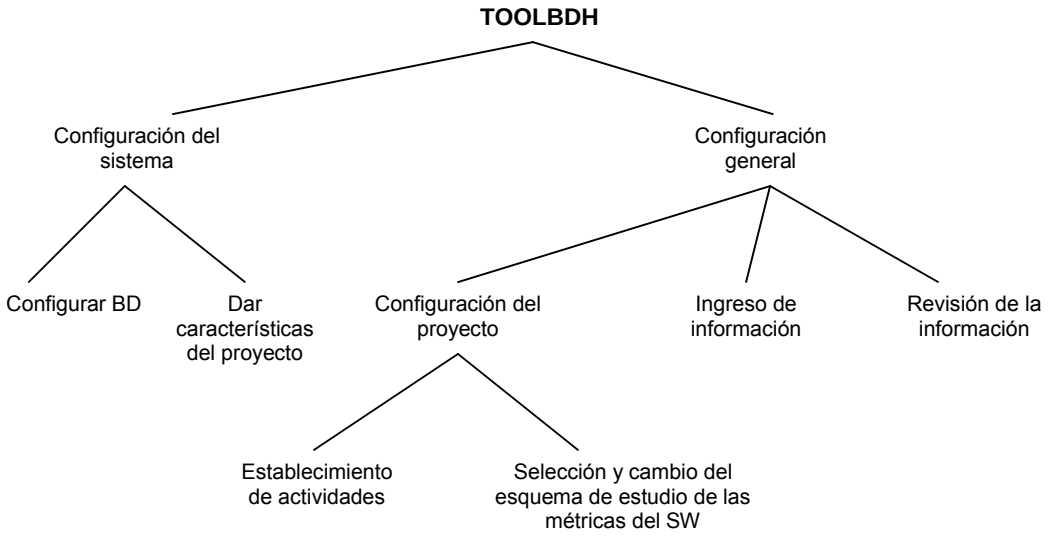


Figura 5.1. Partición vertical de TOOLBHD.

Configuración del sistema.
Para establecer lineamientos generales en cuanto a la forma de trabajar de TOOLBDH, se considera que la aplicación debe admitir lo siguiente:

- Establecer el ID y el PWD del superusuario.
- Decidir la dirección donde se guarda la BD.
- Validación del personal que puede acceder a ella de acuerdo a diferentes permisos.
- El nombre del proyecto y sus características principales (tales como: tipo de la aplicación, modelo de desarrollo, plan de proyecto, nivel CMMI de la empresa al momento de desarrollar el proyecto y tipo del proyecto).
- El registro de la ubicación de plan de proyecto.
- El registro las fechas importantes del proyecto.
- El registro de la duración del proyecto.
- Introducir la información referente a los costos.
- El registro de las horas laboradas, habilidades experiencia, antigüedad, salario para cada integrante del equipo del proyecto.
- Sólo el gestor superior puede configurar el sistema.

Configuración del proyecto.
La configuración de proyecto se refiere a establecer un conjunto de tablas que deben contener la información de este proyecto. Estos esquemas deben definirse antes de llenarse con los datos. Por lo cual la aplicación debe permitir establecer:

- El registro de las actividades del proyecto y quien se encarga de ella.
- El establecimiento de la duración de las actividades.
- La capacidad de organizar la información referente a las métricas que se va a ingresar es respecto a las perspectivas de proceso y producto y al CVSI.
- La posibilidad de evaluar las métricas de productividad, seguimiento, estadísticas y costos, respecto a l tipo de métrica básica, ya sea LDC o PF.

Con la información anterior se crean tanto la BD como las tablas. El gestor superior y el gestor del proyecto pueden modificar este esquema.

Ingreso de información.
El ingreso de la información se refiere a las acciones necesarias para que se puedan guardar los datos en la BD. Los profesionales son los encargados de alimentar esta información a la BD mediante la aplicación, aunque el gestor superior y el de proyecto también tienen la misma capacidad. Por lo tanto, TOOLBDH debe permitir lo siguiente:

- Llenado del esquema de APSW (Tabla 3.1).
- Llenado del esquema de IPSW (Tabla 3.2).
- Llenado de métricas de acuerdo a las tres siguientes perspectivas(Tabla 3.3 y 3.4):
  - Proceso.
  - Producto.
  - CVSI.

Revisión de la información.
La revisión de la información se refiere al medio para conocer la información que se introdujo a la BD. Todos los usuarios tienen la posibilidad de revisar la información. Son dos las acciones que dan esta posibilidad estas son:

- Análisis de la información.
- Creación de reportes.
- Realizar predicciones.

5.3.4 Representación de los requisitos funcionales

Mediante casos de uso se realizan los esquemas de los requerimientos funcionales. Con tal motivo se necesita definir lo siguiente:

- Actores.
- Casos de uso.
- Diagramas de caso de uso de alto nivel.
- Diagramas de caso de uso de detallado.
- Modelo de flujo de datos.
  - DFD de nivel contextual.
  - DFD de nivel 1.
  - DFD de nivel 2 que refina el proceso Iniciar TOOLBDH.
  - DFD de nivel 2 que refina el proceso de Configurar proyecto.
- Diccionario de datos.

Actores.
Son tres los actores que participan en los casos de uso definidos, los cuales se pueden ver en la tabla 5.1.

Tabla 5.1. Actores.

ID	NOMBRE	DESCRIPCIÓN
AC-1	Gestor superior	Es el responsable de definir los aspectos de negocios que a menudo tienen una significativa influencia en el proyecto. Dentro de estos aspectos destacan el estudio de viabilidad y el estudio económico, que se desarrollan junto con el líder de proyecto, definición de requisitos del proyecto, identificación de necesidades de información e intercambio, identificación de procesos, elaboración de la documentación funcional, diseño del modelo de entidades, diseño del flujo de datos y diseño lógico de archivos, las tareas de diseño se realizan a la par de los gestores del proyecto.
AC-2	Gestor de proyecto	Es el comisionado que conforma motiva, organiza y controla al equipo de trabajo y se encarga de coordinar, dirigir y controlar la totalidad del proyecto.
AC-3	Profesional	Es el encargado de proporcionar las capacidades técnicas necesarias para la ingeniería de un producto o aplicación. Puede de especificar los requisitos para la ingeniería del software y otros elementos que tienen menor influencia en el resultado.

Casos de uso.

Los casos de uso desarrollados para los requerimientos funcionales se revisan en las tablas 5.2, 5.3, 5.4, 5.5, 5.6 y 5.7.

Tabla 5.2 Caso de uso CU-01.

ID:	**CU-01**
Nombre:	Configurar la BD
Descripción:	Este caso de uso permite establecer el ID y el PWD del superusuario, decidir la dirección donde se guarda la BD y el establecimiento de los datos del superusuario.
Evento que inicia el caso de uso:	Ejecución del programa.
Actores:	Gestor superior.
Precondiciones:	1. Conocer el ID, PWD y ubicación de la BD. 2. Conocer la configuración a establecer. 3. Conocer quienes pueden acceder a la BD.
Poscondiciones:	1. Creación del ID y el PWD del superusuario. 2. Conocimiento de la ubicación de la BD.
Flujo normal:	1. Se establece el ID de la BD. 2. Se establece el PWD de la BD. 3. Se establece el nombre y la ubicación de acceso de la BD.
Flujo alternativo 1:	1. Se elige eliminar la BD. 2. Se elimina la BD.
Excepciones:	(No aplica)
Incluye:	(No aplica)
Suposiciones:	(No aplica)
Notas:	(No aplica)

Tabla 5.3. Caso de uso CU-02.

ID:	**CU-02**
Nombre:	Dar generalidades del proyecto
Descripción:	Este caso de uso permite ingresar el nombre del proyecto y sus características esenciales, la declaración de los miembros del equipo y sus características, así como la asignación del gestor del proyecto.
Evento que inicia el caso de uso:	Alta de miembros del equipo
Actores:	Gestor superior.
Precondiciones:	1. Conocer el nombre del proyecto. 2. Conocer la localización del plan del proyecto. 3. Conocer las características principales del proyecto. 4. Conocer el nombre de los miembros de equipo y sus características. 5. Conocer el nombre del gestor del proyecto.
Poscondiciones:	1. Establecimiento del nombre del proyecto y sus características. 2. Establecimiento de los miembros del equipo y del gestor del proyecto, así como la definición de sus características.
Flujo normal:	1. Se establece el nombre del proyecto. 2. Se establecen las características principales del proyecto: inicio y fin del proyecto, tiempo de duración estimado, tiempo de duración real, objetivos, objetivos no logrados, costo estimado del proyecto, costo real del proyecto. 3. Se establece la ubicación de las partes del plan del proyecto o se señala que todo está contenido en un solo documento y se da la ubicación. 4. Se define ID y PWD de cada miembro del staff de la empresa. 5. Se describe cada miembro staff: nombre, horas que trabaja, habilidades, experiencia, temporalidad y monto del pago. 6. Se asigna el proyecto para cada miembro del staff de la empresa y se elige que el gestor del proyecto de entre los miembros del equipo del proyecto.
Flujo alternativo 1:	1. Se establece el nombre del proyecto. 2. Se establecen las características principales del proyecto: inicio y fin del proyecto, tiempo de duración estimado, tiempo de duración real, objetivos, objetivos no logrados, costo estimado del proyecto, costo real del proyecto. 3. Se establece la ubicación de las partes del plan del proyecto o se señala que todo está contenido en un solo documento y se da la ubicación. 4. Se describe cada miembro del staff de la empresa: nombre, horas que trabaja, habilidades, experiencia, temporalidad y monto del pago. 5. Se asigna el proyecto para cada miembro del staff de la empresa y se elige al gestor del proyecto como el gestor superior.
Flujo alternativo 2:	1. Se escribe el nombre del proyecto. 2. Se elimina el proyecto o se modifican los datos del proyecto o del gestor del proyecto.
Excepciones:	(No aplica)
Incluye:	(No aplica)
Suposiciones:	(No aplica)
Notas:	El gestor superior y el gestor del proyecto pueden ser el mismo sólo se tiene que indicar.

Tabla 5.4. Caso de uso CU-03.

ID:	**CU-03**
Nombre:	Establecer Actividades
Descripción:	Este caso de uso permite establecer las partidas en las cuales se divide el trabajo, sus responsables y su status.
Evento que inicia el caso de uso:	(No aplica)
Actores:	Gestor del proyecto.
Precondiciones:	1. Haber llevado a cabo los CU-01 y CU-02. 2. Conocer las actividades a desarrollar de cada miembro del equipo del proyecto. Conocer la duración de la jornada laborar de cada integrante del equipo. 3. Conocer el inicio y término de cada actividad.
Poscondiciones:	1. Establecer las actividades
Flujo normal:	1. Se establecen las actividades y se declara el status de la actividad de acuerdo al nivel en la cual esté. 2. Se establecen las fechas de inicio y término de cada actividad. 3. Se selecciona al responsable principal y al de apoyo.
Flujo alternativo:	1. Se elige modificar datos. 2. Se selecciona que información se va a eliminar o modificar. 3. Se cambian los datos referentes a las actividades y sus encargados o bien se elimina alguna tarea.
Excepciones:	(No aplica)
Incluye:	(No aplica)
Suposiciones:	(No aplica)
Notas:	(No aplica)

Tabla 5.5. Caso de uso CU-04.

ID:	**CU-04**
Nombre:	Seleccionar el esquema de estudio de las métricas del SW y observaciones de algunas generalidades del proyecto
Descripción:	Este caso de uso permite establecer la forma en la cual se revisan las métricas: proceso, producto o al CVSI.
Evento que inicia el caso de uso:	Elección de métricas.
Actores:	Gestor del proyecto.
Precondiciones:	1. Conocer la el tipo de métrica adecuada al proyecto.
Poscondiciones:	1. Establecimiento de las métricas.
Flujo normal:	1. Se elige el esquema del proceso, producto o CVSI como forma de estudio de las métricas. 2. Se eligen las métricas adecuadas. 3. Se elige la métrica básica. 4. Se establecen observaciones del tiempo de duración, observaciones de los objetivos, observaciones del costo del proyecto.
Flujo alternativo 1:	1. Se elige el esquema del proceso, producto o CVSI como forma de estudio de las métricas. 2. Se agregan las métricas adecuadas. 3. Se definen dichas métricas. 4. Se elige la métrica básica. 5. Se establecen observaciones del tiempo de duración, observaciones de los objetivos, observaciones del costo del proyecto.
Flujo alternativo 2:	1. Se elige modificar métricas. 2. Se eliminan o modifican métricas, o modificar observaciones.
Excepciones:	(No aplica)
Incluye:	(No aplica)
Suposiciones:	(No aplica)
Notas:	(No aplica)

Tabla 5.6. Caso de uso CU-05.

ID:	**CU-05**
Nombre:	Ingresar métricas
Descripción:	Este caso de uso permite instaurar un conjunto de tablas que deben contener la información de éste proyecto. Contempla el llenado de los esquemas de APSW, IPSW y métricas.
Evento que inicia el caso de uso:	Ejecución del programa.
Actores:	Profesionales.
Precondiciones:	1. Haber realizado el CU-04. 2. Conocer el ID y PWD correctos. 3. Conocer las características de los esquemas de APSW, ISW y métricas. 4. Conocer quienes pueden acceder a la BD para realizar esta acción.
Poscondiciones:	Se puede tener como poscondiciones el llenado de los valores en el esquema de métricas elegido en el CU-04.
Flujo normal:	1. Se elige ingresar datos. 2. Se llena la información adecuada. 3. Se visualiza la parte del esquema llenado.
Flujo alternativo:	1. Se elige modificar datos. 2. Se modifica la información a la cual puede acceder. 3. Se visualiza la parte del esquema llenado.
Excepciones:	(No aplica)
Incluye:	(No aplica)
Suposiciones:	(No aplica)
Notas:	(No aplica)

Tabla 5.7. Caso de uso CU-06.

ID:	**CU-06**
Nombre:	Crear reportes
Descripción:	Este caso de uso permite conocer la información que se introdujo a la BD, mediante el análisis dentro del cual se pude hacer predicciones y la creación reportes.
Evento que inicia el caso de uso:	Ejecución del programa.
Actores:	Gestor superior, gestor del proyecto y profesionales.
Precondiciones:	1. Conocer que se va a analizar. 2. Conocer de que información se va a crear uno o algunos reportes.
Poscondiciones:	Se puede tener como poscondiciones una, algunas o todas de las siguientes situaciones: ➢ Análisis de la información. ➢ Creación de reportes sobre la información.
Flujo normal:	1. Si se accede mediante el ID y PWD del gestor superior o gestor del proyecto, se puede elegir realizar una o todas las siguientes acciones: • Revisar los proyectos respecto al número de proyectos que cumplen con ciertas características (tales como: tiempo de duración, proyectos terminados antes o después del tiempo estimado, objetivos cumplidos o no, costo estimado contra costo real y observaciones de los mismos, y Gestor del proyecto). • Revisar los proyectos respecto a las Métricas de costo del SW. • Revisar los proyectos respecto a las Métricas de revisión del SW. • Revisar los proyectos respecto a las Métricas de calidad del SW. • Revisar los proyectos respecto a las Métricas de análisis del SW. • Revisar los proyectos respecto a las Métricas de productividad del SW. • Revisar información del Staff de la empresa para desarrollar productos de SW respecto a sus características (tales como: habilidades, experiencia, salario, antigüedad y proyectos en los cuales se fue Gestor del proyecto o Responsable principal o de apoyo de alguna actividad). • Revisar EDT de un proyecto. • Revisar matriz de responsabilidades de un proyecto. • Revisar diagrama de Gantt de un proyecto. • Revisar documentos del plan de un proyecto. 2. Se llena o selecciona la información necesaria. 3. Se visualiza la parte llenada y/o se realiza una gráfica y/o se imprime.

Flujo alternativo:	1. Elegir realizar una o todas las siguientes acciones: ➢ Revisar los proyectos respecto al número de proyectos que cumplen con ciertas características (tales como: tiempo de duración, proyectos terminados antes o después del tiempo estimado, objetivos cumplidos o no, costo estimado contra costo real y observaciones de los mismos, y Gestor del proyecto). ➢ Revisar los proyectos respecto a las Métricas de costo del SW. ➢ Revisar los proyectos respecto a las Métricas de revisión del SW. ➢ Revisar los proyectos respecto a las Métricas de calidad del SW. ➢ Revisar los proyectos respecto a las Métricas de análisis del SW. ➢ Revisar los proyectos respecto a las Métricas de productividad del SW. ➢ Revisar información del miembro del Staff de la empresa que ingreso ID y PWD, respecto a sus características (tales como: habilidades, experiencia, salario, antigüedad y proyectos en los cuales se fue Gestor del proyecto o Responsable principal o de apoyo de alguna actividad). ➢ Revisar EDT de un proyecto. ➢ Revisar matriz de responsabilidades de un proyecto. ➢ Revisar diagrama de Gantt de un proyecto. ➢ Revisar documentos del plan de un proyecto. 2. Se llena o selecciona la información necesaria. 3. Se visualiza la parte llenada y/o se realiza una gráfica y/o se imprime.
Excepciones:	(No aplica)
Incluye:	(No aplica)
Suposiciones:	(No aplica)
Notas:	Es uno de los casos de uso principales. Todos los usuarios tienen la posibilidad de acceder a éste caso de uso.

Diagramas de caso de uso de alto nivel.

Dichos diagramas para los tres actores participantes en la herramienta a desarrollar se muestran en las figuras 5.2, 5.3 y 5.4.

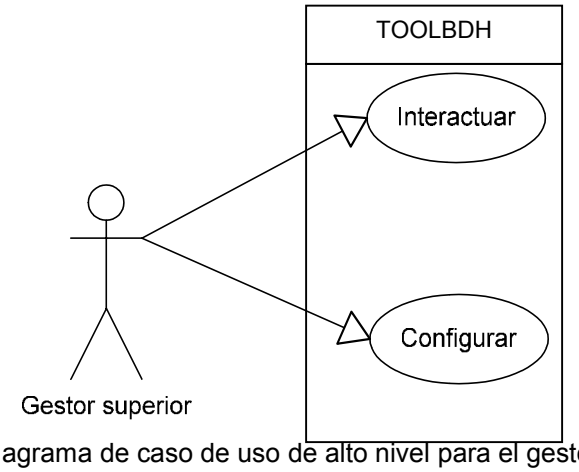


Figura 5.2. Diagrama de caso de uso de alto nivel para el gestor superior.

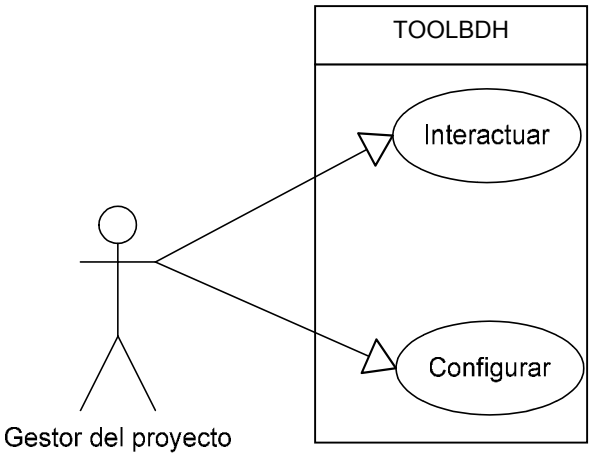


Figura 5.3. Diagrama de caso de uso de alto nivel para el gestor del proyecto.

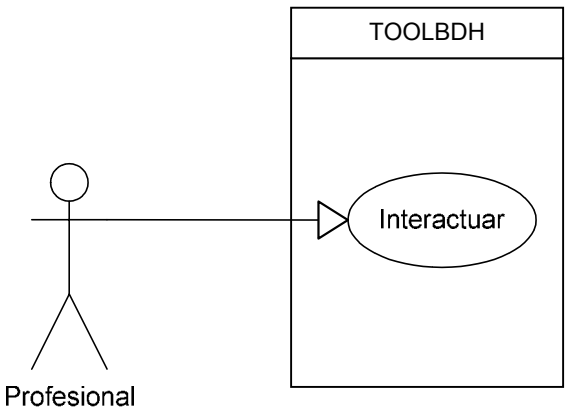


Figura 5.4. Diagrama de caso de uso de alto nivel para el profesional.

Diagramas de caso de uso detallados.

Los diagramas de caso de uso a detalles se revisan en las figuras 5.5, 5.6 y 5.7.

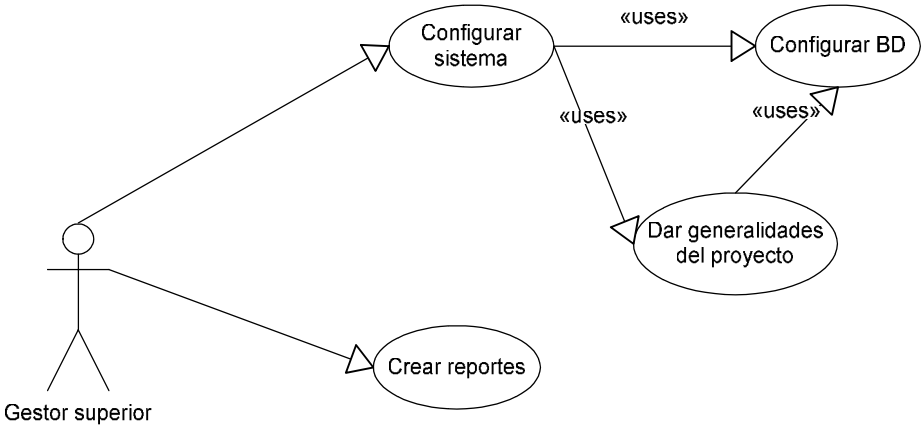


Figura 5.5. Diagrama de caso de uso detallado para el gestor superior.

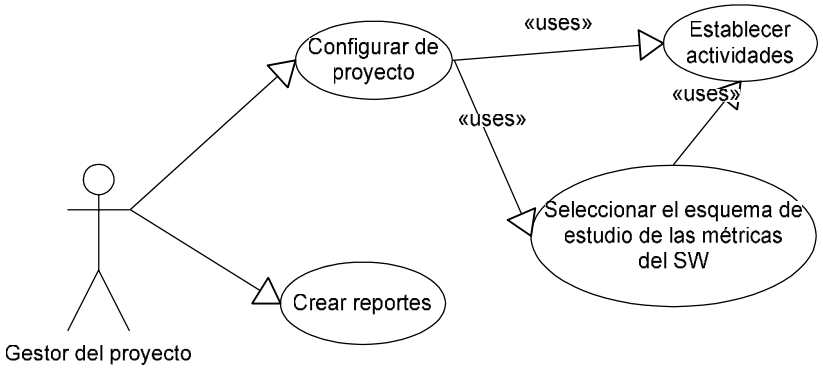


Figura 5.6. Diagrama de caso de uso detallado para el gestor del proyecto.

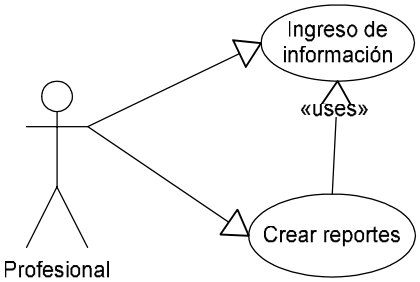


Figura 5.7. Diagrama de caso de uso detallado para el profesional.

Modelo de flujo de datos.

Los modelos de flujo de datos (DFD) de nivel contextual y de nivel 1 y 2 se esquematizan en las figuras 5.8, 5.9, 5.10 y 5.11.

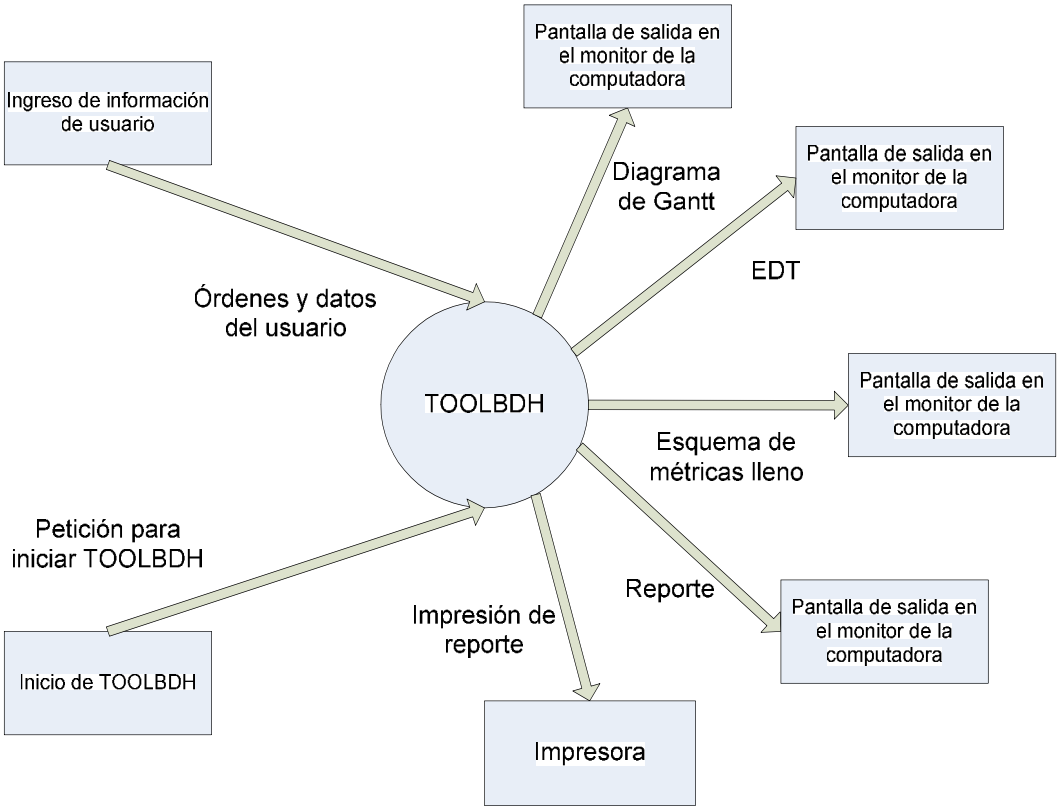


Figura 5.8. Diagrama DFD de nivel contextual

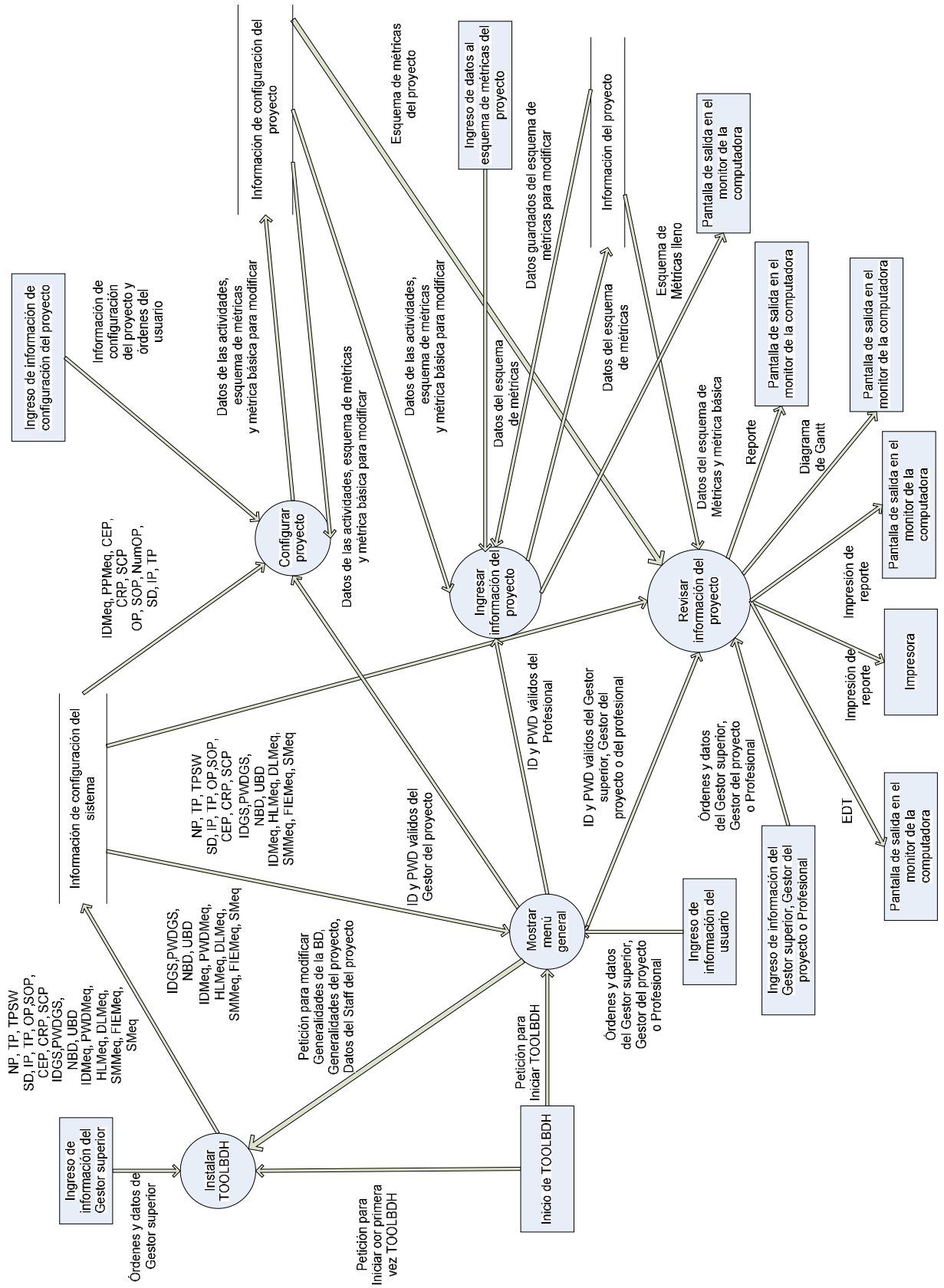


Figura 5.9. Diagrama DFD de nivel 1.

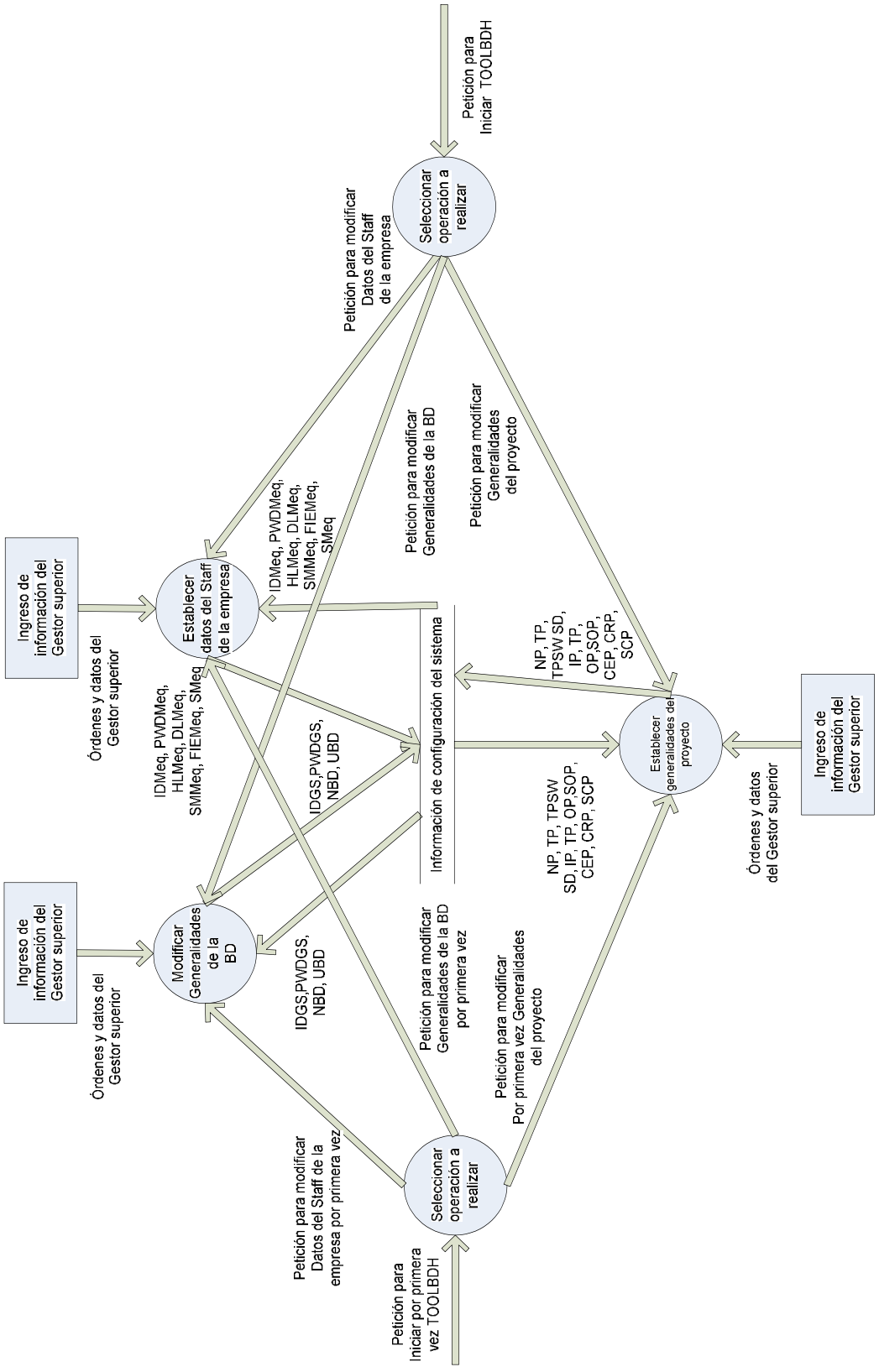


Figura 5.10. Diagrama DFD de nivel 2 que refina el proceso Iniciar TOOLBDH.

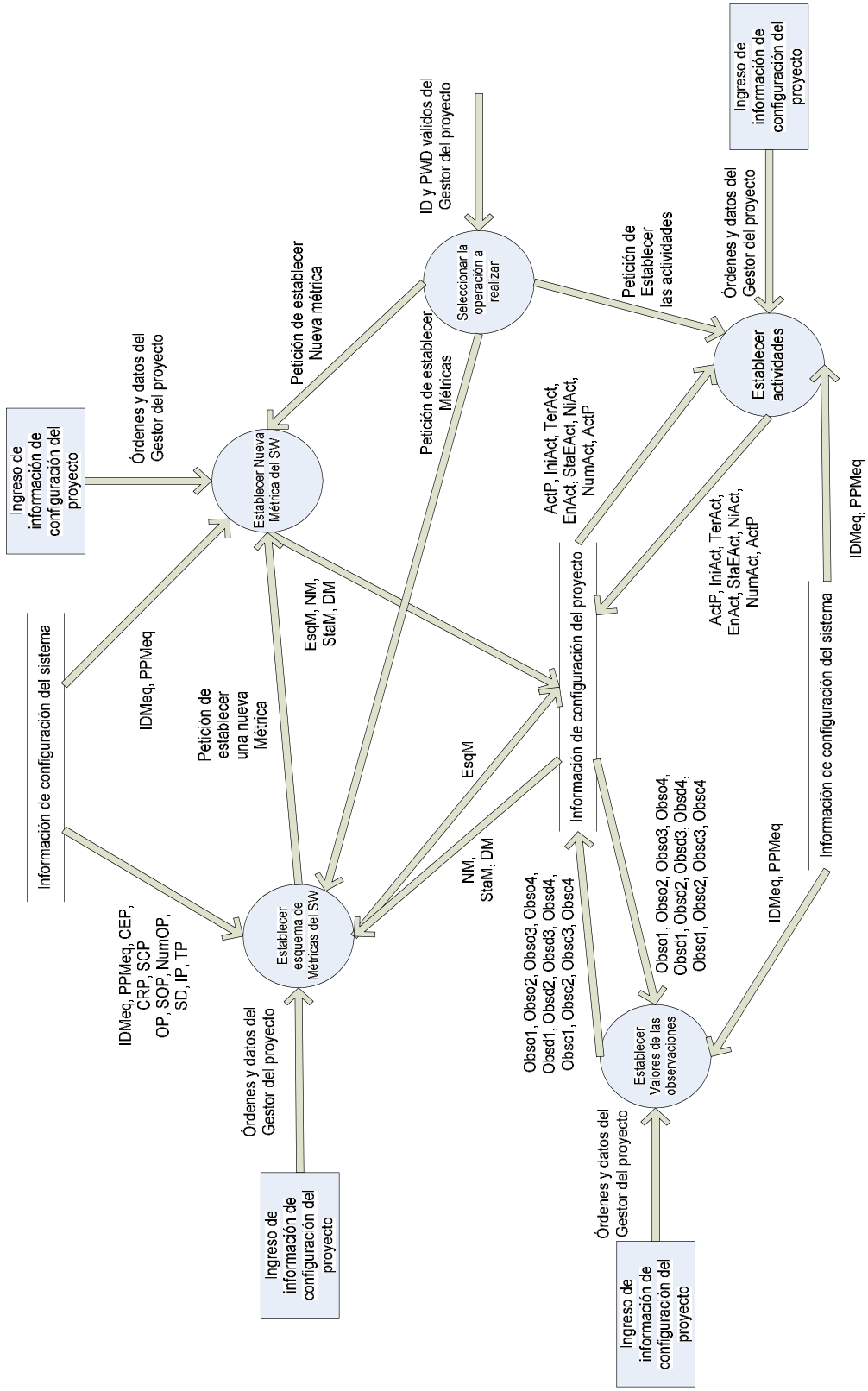


Figura 5.11. Diagrama DFD de nivel 2 que refina el proceso Configurar proyecto.

Diccionario de datos.

En el diccionario de datos se encuentran todos los elementos mencionados en los diagramas de flujo de datos, los cuales son:

- *Datos del gestor superior:* ID y PWD del superusuario sus definiciones se pueden revisar en las tablas 5.8 y 5.9 respectivamente.

- *Datos de la BD:* Nombre BD y UBD sus definiciones se pueden revisar en las tablas 5.10 y 5.11 respectivamente.

- *Datos de los miembros de equipo:* ID, PWD, horas que labora al día, días que labora, salario mensual, fecha de ingreso al la empresa, status del miembro del equipo del proyecto y proyectos a los cuales pertenece el miembro del equipo del proyecto. Las respectivas definiciones se presentan en las tablas 5.12, 5.13, 5.14, 5.15, 5.16, 5.17 y 5.18.

- *Datos del proyecto:* nombre, número y tipo de proyecto y tipo de producto de software. Sus definiciones se pueden consultar en las tablas 5.19, 5.20, 5.21 y 5.22.

- *Datos de la duración del proyecto:* Status de duración del proyecto, inicio del proyecto y término del proyecto. Sus definiciones se estudian en las tablas 5.23, 5.24 y 5.25.

- *Observaciones de la duración del proyecto:* Las cuales se ven en las tablas 5.26, 5.27, 5.28 y 5.29.

- *Datos de los objetivos del proyecto:* Objetivo del proyecto, status del objetivo del proyecto y número de objetivos del proyecto; sus respectivas definiciones se ven en las tablas 5.30, 5.31 y 5.32.

- *Observaciones de los objetivos del proyecto:* Contemplan las tablas 5.33, 5.34, 5.35 y 5.36.

- *Datos del costo del proyecto:* Costos estimado y real del proyecto y el status del costo del proyecto; cuyas definiciones se registran en las tablas 5.37, 5.38 y 5.39.

- *Observaciones de los costos del proyecto:* Las definiciones se pueden examinar en las tablas 5.40, 5.41, 5.42 y 5.43.

- *Datos de las Métricas:* Esquema de métricas, nombre, descripción, status y dato de métrica. Las definiciones que corresponden se analizan en las tablas 5.44, 4.45, 5.46, 5.47 y 5.48.

- *Datos de las actividades del proyecto:* Nombre, inicio y término de actividad, encargado de la actividad, status del encargado de la actividad, nivel de la actividad, número de actividad del proyecto y actividades que pertenecen al proyecto. Sus definiciones se estudian en las tablas 5.49, 5.50, 5.51, 5.52, 5.53, 5.54, 5.55 y 5.56.

Tabla 5.8. Definición del dato: ID del superusuario.

Nombre:	ID del superusuario
Alias:	IDGS
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida). 2. Mostrar menú general (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	IDGS = *un arreglo de 8 caracteres resultado de combinación de dígitos de 0-9 y letras de A-Z *
Información adicional:	Este dato permite acceder a la herramienta

Tabla 5.9. Definición del dato: PWD del superusuario.

Nombre:	PWD del superusuario
Alias:	PWDGS
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida). 2. Mostrar menú general (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	PWDGS = *un arreglo de 8 caracteres resultado de combinación de dígitos de 0-9 y letras de A-Z *
Información adicional:	Este dato permite acceder a la herramienta

Tabla 5.10. Definición del dato: nombre BD.

Nombre:	Nombre BD
Alias:	NBD
Dónde se usa/ Cómo se usa:	Instalar TOOLBDH (Salida y Entrada).
Descripción del contenido:	NBD= *un arreglo de no más de 15 caracteres resultado de combinación de dígitos de 0-9 y letras de A-Z *
Información adicional:	Este dato permite instalar a la herramienta

Tabla 5.11. Definición del dato: UBD.

Nombre:	UBD
Alias:	UBD
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada).
Descripción del contenido:	Ubicación de BD= *un arreglo de los caracteres suficientes de dígitos de 0-9 y letras de A-Z y [/|\] para indicar la ruta donde se guarda la BD*
Información adicional:	Este dato permite instalar a la herramienta

Tabla 5.12. Definición del dato: ID de Miembro del equipo del proyecto.

Nombre:	ID de Miembro del equipo del proyecto
Alias:	IDMeq
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Mostrar menú general (Entrada). 3. Configurar proyecto (Entrada). 4. Revisar información del proyecto (Entrada).
Descripción del contenido:	IDMeq = *un arreglo de 8 caracteres resultado de combinación de dígitos de 0-9 y letras de A-Z *
Información adicional:	Este dato permite acceder a la herramienta

Tabla 5.13. Definición del dato: PWD de Miembro del equipo del proyecto.

Nombre:	PWD de Miembro del equipo del proyecto
Alias:	PWDMeq
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Mostrar menú general (Entrada). 3. Configurar proyecto (Entrada). 4. Revisar información del proyecto (Entrada).
Descripción del contenido:	PWDMeq = *un arreglo de 8 caracteres resultado de combinación de dígitos de 0-9 y letras de A-Z *
Información adicional:	Este dato permite acceder a la herramienta

Tabla 5.14. Definición del dato: horas que labora al día el Miembro del equipo del proyecto.

Nombre:	Horas que labora al día el Miembro del equipo del proyecto
Alias:	HLMeq
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Configurar proyecto (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	HLMeq = *un secuencia numérica de dos dígitos de acuerdo al número de horas que labora al día el Miembro del equipo *
Información adicional:	(No aplica)

Tabla 5.15. Definición del dato: Días que labora el Miembro del equipo del proyecto.

Nombre:	Días que labora el Miembro del equipo del proyecto
Alias:	DLMeq
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Configurar proyecto (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	DLMeq = *un secuencia numérica de dos dígitos de acuerdo al número de días que labora el miembro del equipo al mes*
Información adicional:	Este dato sirve para el cálculo de métricas del SW.

Tabla 5.16. Definición del dato: Salario mensual del Miembro del equipo del proyecto.

Nombre:	Salario mensual del Miembro del equipo del proyecto
Alias:	SMMeq
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Configurar proyecto (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	SMMeq = *un secuencia numérica de cinco dígitos de acuerdo a la cantidad de dinero que recibe el Miembro del equipo al mes*
Información adicional:	Este dato sirve para el cálculo de métricas del SW

Tabla 5.17. Definición del dato: Fecha de ingreso al la empresa del Miembro del equipo del proyecto.

Nombre:	Fecha de ingreso al la empresa del Miembro del equipo del proyecto
Alias:	FIEMeq
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Configurar proyecto (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	FIEMeq = año+'-'+mes+'-'+día Año= *4 dígitos a partir 1970* Mes= *2 dígitos del 00 al 12* Día= *2 dígitos del 00 al 31*
Información adicional:	Este dato sirve para el cálculo de métricas del SW

Tabla 5.18. Definición del dato: Status del Miembro del equipo del proyecto.

Nombre:	Status del Miembro del equipo del proyecto
Alias:	SMeq
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Mostrar menú general (Entrada). 3. Configurar proyecto (Entrada). 4. Ingresar información del proyecto (Entrada). 5. Revisar información del proyecto (Entrada).
Descripción del contenido:	SMeq = * valor con dos posibilidades, es cero si no es Gestor del proyecto y uno si es Gestor del proyecto*
Información adicional:	Este dato permite acceder a la herramienta

Tabla 5.19. Definición del dato: Proyectos a los cuales pertenece el Miembro del equipo del proyecto.

Nombre:	Proyectos a los cuales pertenece el Miembro del equipo del proyecto
Alias:	PPMeq
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Mostrar menú general (Entrada). 3. Configurar proyecto (Entrada). 4. Ingresar información del proyecto (Entrada). 5. Revisar información del proyecto (Entrada).
Descripción del contenido:	PPMeq = IDMeq+','+SMeq+','+proyectos Proyectos=NP_1+','+NP_2+','…+','+NP_n *Todos los Nombres de los proyectos en los cuales participa*
Información adicional:	Este dato permite acceder a la herramienta

Tabla 5.20. Definición del dato: Nombre del proyecto.

Nombre:	Nombre del proyecto
Alias:	NP
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Configurar proyecto (Entrada). 3. Ingresar información del proyecto (Entrada). 4. Revisar información del proyecto (Entrada).
Descripción del contenido:	NP = *un arreglo de hasta 20 caracteres resultado de combinación de dígitos de 0-9 y letras de A-Z *
Información adicional:	(No aplica)

Tabla 5.21. Definición del dato: Número del proyecto.

Nombre:	Número del proyecto
Alias:	NumP
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Configurar proyecto (Entrada). 3. Ingresar información del proyecto (Entrada). 4. Revisar información del proyecto (Entrada).
Descripción del contenido:	NumP = *un arreglo de hasta 4 dígitos que se incrementa de acuerdo al número de proyectos que se archivan*
Información adicional:	Se auto incrementa cuando se agrega un proyecto.

Tabla 5.22. Definición del dato: Tipo de proyecto.

Nombre:	Tipo de proyecto
Alias:	TP
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	TP = (“desarrollo de concepto” |” desarrollo de una nueva aplicación”|” mejoras de aplicaciones”|” mantenimiento de aplicaciones”|”reingeniería”)
Información adicional:	(No aplica)

Tabla 5.23. Definición del dato: Tipo de producto de software.

Nombre:	Tipo de producto de software
Alias:	TPSW
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	TPSW = (“sistemas” |” tiempo real”|” gestión”|” ingeniería y científico”|”empotrado”|”basado en Web”|”SW de computadoras personales”|”inteligencia artificial”)
Información adicional:	(No aplica)

Tabla 5.24. Definición del dato: Status de duración del proyecto.

Nombre:	Status de duración del proyecto
Alias:	SD
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	SD = * valor con cuatro posibilidades, es cero si no se término, uno si se término fuera de tiempo, dos si se término en tiempo y tres si aún continua en desarrollo*
Información adicional:	Este dato sirve para el cálculo de métricas del SW

Tabla 5.25. Definición del dato: Inicio del proyecto.

Nombre:	Inicio del proyecto
Alias:	IP
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Configurar proyecto (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	IP = año+'-'+mes+'-'+día Año= *4 dígitos a partir 1980* Mes= *2 dígitos del 00 al 12* Día= *2 dígitos del 00 al 31*
Información adicional:	Debe ser menor al valor de Término del mismo proyecto. Este dato sirve para el cálculo de métricas del SW.

Tabla 5.26. Definición del dato: Término del proyecto.

Nombre:	Término del proyecto
Alias:	TP
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Configurar proyecto (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	TP = año+'-'+mes+'-'+día Año= *4 dígitos a partir 1980* Mes= *2 dígitos del 00 al 12* Día= *2 dígitos del 00 al 31*
Información adicional:	Debe ser mayor al valor de Inicio del mismo proyecto. Este dato sirve para el cálculo de métricas del SW.

Tabla 5.27. Definición del dato: Primera observación de la duración del proyecto.

Nombre:	Primera observación de la duración del proyecto
Alias:	Obsd1
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	Obsd1 = ("insuficiencia de SW,"|"suficiencia de SW,"|"el SW no afecta,")
Información adicional:	El valor de este dato tiene correspondencia con el status de duración del proyecto.

Tabla 5.28. Definición del dato: Segunda observación de la duración del proyecto.

Nombre:	Segunda observación de la duración del proyecto
Alias:	Obsd2
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	Obsd2 = (”insuficiencia de HW,”|”suficiencia de HW,”|”el HW no afecta,”)
Información adicional:	El valor de este dato tiene correspondencia con el status de duración del proyecto.

Tabla 5.29. Definición del dato: Tercera observación de la duración del proyecto.

Nombre:	Tercera observación de la duración del proyecto
Alias:	Obsd3
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	Obsd3 = (”insuficiencia de RH,”|”suficiencia de RH,”|”el RH no afecta,”)
Información adicional:	El valor de este dato tiene correspondencia con el status de duración del proyecto.

Tabla 5.30. Definición del dato: Cuarta observación de la duración del proyecto.

Nombre:	Cuarta observación de la duración del proyecto
Alias:	Obsd4
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	Obsd4 = (”insuficiencia de Dinero”|”suficiencia de Dinero”|”el Dinero no afecta”)
Información adicional:	El valor de este dato tiene correspondencia con el status de duración del proyecto.

Tabla 5.31. Definición del dato: Objetivo del proyecto.

Nombre:	Objetivo del proyecto
Alias:	OP
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	OP = *secuencia alfabética de hasta 200 caracteres*
Información adicional:	Tiene un status del objetivo del proyecto. Cuando es entrada para el proceso de revisar información del proyecto se presenta en una cadena que concatena de forma coherente todos los objetivos del proyecto.

Tabla 5.32. Definición del dato: Status del objetivo del proyecto.

Nombre:	Status del objetivo del proyecto
Alias:	SOP
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	SOP = * valor binario, es cero si no se logro y uno si se cumple *
Información adicional:	(No aplica)

Tabla 5.33. Definición del dato: Número de objetivos del proyecto.

Nombre:	Número de objetivos del proyecto
Alias:	NumOP
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	NumOP = *4 dígitos desde 0001 hasta 9999 *
Información adicional:	Se auto incrementa cuando se agrega un objetivo

Tabla 5.34. Definición del dato: Primera observación de los objetivos del proyecto.

Nombre:	Primera observación de los objetivos del proyecto
Alias:	Obso1
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	Obso1 = (”insuficiencia de SW,”|”suficiencia de SW,”|”el SW no afecta,”)
Información adicional:	El valor de este dato tiene correspondencia con el status de duración del proyecto.

Tabla 5.35. Definición del dato: Segunda observación de los objetivos del proyecto.

Nombre:	Segunda observación de los objetivos del proyecto
Alias:	Obso2
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	Obso2 = (”insuficiencia de HW,”|”suficiencia de HW,”|”el HW no afecta,”)
Información adicional:	El valor de este dato tiene correspondencia con el status de duración del proyecto.

Tabla 5.36. Definición del dato: Tercera observación de los objetivos del proyecto.

Nombre:	Tercera observación de los objetivos del proyecto
Alias:	Obso3
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	Obso3 = (”insuficiencia de RH,”|”suficiencia de RH,”|”el RH no afecta,”)
Información adicional:	El valor de este dato tiene correspondencia con el status de duración del proyecto.

Tabla 5.37. Definición del dato: Cuarta observación de los objetivos del proyecto.

Nombre:	Cuarta observación de los objetivos del proyecto
Alias:	Obso4
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	Obso3 = (”insuficiencia de Dinero”|”suficiencia de Dinero”|”el Dinero no afecta”)
Información adicional:	El valor de este dato tiene correspondencia con el status de duración del proyecto.

Tabla 5.38. Definición del dato: Costo estimado del proyecto.

Nombre:	Costo estimado del proyecto
Alias:	CEP
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	CEP = *Secuencia numérica de cualquier tamaño*
Información adicional:	(No aplica)

Tabla 5.39. Definición del dato: Costo real del proyecto.

Nombre:	Costo real del proyecto
Alias:	CRP
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	CRP = *Secuencia numérica de cualquier tamaño*
Información adicional:	(No aplica)

Tabla 5.40. Definición del dato: Status del costo del proyecto.

Nombre:	Status del costo del proyecto
Alias:	SCP
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	SCP = * valor con cuatro posibilidades, es cero si costo menos del costo estimado, uno si costo lo mismo, dos si costo más del costo estimado*
Información adicional:	(No aplica)

Tabla 5.41. Definición del dato: Primera observación de los costos del proyecto.

Nombre:	Primera observación de los costos del proyecto
Alias:	Obsc1
Dónde se usa/ Cómo se usa:	3. Instalar TOOLBDH (Salida y Entrada). 4. Revisar información del proyecto (Entrada).
Descripción del contenido:	Obsc1 = (”insuficiencia de SW,”|”suficiencia de SW,”|”el SW no afecta,”)
Información adicional:	El valor de este dato tiene correspondencia con el status de costos del proyecto.

Tabla 5.42. Definición del dato: Segunda observación de los costos del proyecto.

Nombre:	Segunda observación de los costos del proyecto
Alias:	Obsc2
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	Obsc2 = (”insuficiencia de HW,”|”suficiencia de HW,”|”el HW no afecta,”)
Información adicional:	El valor de este dato tiene correspondencia con el status de costos del proyecto.

Tabla 5.43. Definición del dato: Tercera observación de los costos del proyecto.

Nombre:	Tercera observación de los costos del proyecto
Alias:	Obsc3
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	Obso3 = (”insuficiencia de RH,”|”suficiencia de RH,”|”el RH no afecta,”)
Información adicional:	El valor de este dato tiene correspondencia con el status de costos del proyecto.

Tabla 5.44. Definición del dato: Cuarta observación de los costos del proyecto.

Nombre:	Cuarta observación de los costos del proyecto
Alias:	Obsc4
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	Obsc4 = (”insuficiencia de Dinero”|”suficiencia de Dinero”|”el Dinero no afecta”)
Información adicional:	El valor de este dato tiene correspondencia con el status de costos del proyecto.

Tabla 5.45. Definición del dato: Esquema de métricas.

Nombre:	Esquema de métricas
Alias:	Esq.
Dónde se usa/ Cómo se usa:	1. Configurar proyecto (Salida). 2. Ingresar información del proyecto (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	EsqM = Mm1 + Mm2 +…+Mmn Mmn= *Métrica n existente o n métrica nueva*
Información adicional:	El esquema se llena automáticamente con las métricas existentes. En el proceso de configurar proyecto se decide cuales son relevantes para el proyecto y cuales no, y se establecen las nuevas métricas. Este dato sirve para el cálculo de métricas del SW.

Tabla 5.46. Definición del dato: Nombre de la Métrica.

Nombre:	Nombre de la Métrica
Alias:	NM
Dónde se usa/ Cómo se usa:	1. Configurar proyecto (Salida y Entrada). 2. Ingresar información del proyecto (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	Nombre de la métrica= *un arreglo de hasta 20 caracteres resultado de combinación de dígitos del 0-9, letras de la A-Z y como separador de palabras caracteres blancos*
Información adicional:	Este dato sirve para el cálculo de métricas del SW.

Tabla 5.47. Definición del dato: Descripción de la métrica.

Nombre:	Descripción de la métrica
Alias:	DM
Dónde se usa/ Cómo se usa:	1. Configurar proyecto (Salida y Entrada). 2. Ingresar información del proyecto (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	Descripción de la métrica = *un arreglo de hasta 50 caracteres resultado de combinación de dígitos del 0-9, letras de la A-Z y como separador de palabras caracteres blancos*
Información adicional:	Este dato sirve para el cálculo de métricas del SW.

Tabla 5.48. Definición del dato: Status de métrica.

Nombre:	Status de métrica
Alias:	StaM
Dónde se usa/ Cómo se usa:	1. Configurar proyecto (Salida y Entrada). 2. Ingresar información del proyecto (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	StaM = *valor con tres posibilidades, es cero si es métrica existente, uno si es métrica básica, dos si es métrica nueva. La métrica básica sólo puede ser PF o LDC*
Información adicional:	Este dato sirve para el cálculo de métricas del SW.

Tabla 5.49. Definición del dato: Dato de métrica.

Nombre:	Dato de métrica
Alias:	DatM
Dónde se usa/ Cómo se usa:	1. Ingresar información del proyecto (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	DatM= (cad | num) Cad = *un arreglo de hasta 50 caracteres resultado de combinación de dígitos del 0-9, letras de la A-Z y como separador de palabras caracteres blancos* Num = *Secuencia numérica de cualquier tamaño*
Información adicional:	Este dato sirve para el cálculo de métricas del SW.

Tabla 5.50. Definición del dato: Nombre de la actividad del proyecto.

Nombre:	Nombre de la actividad del proyecto
Alias:	ActP
Dónde se usa/ Cómo se usa:	1. Configurar proyecto (Entrada y Salida). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	ActP = *un arreglo de hasta 20 caracteres resultado de combinación de dígitos del 0-9, letras de la A-Z y como separador de palabras caracteres blancos *
Información adicional:	Las actividades son las partidas de trabajo en las cuales se separa el proyecto. En el proceso de configurar proyecto se decide cuales son relevantes para el proyecto y cuales no. Este dato sirve para el cálculo de métricas del SW.

Tabla 5.51. Definición del dato: Inicio de Actividad.

Nombre:	Inicio de Actividad
Alias:	IniAct
Dónde se usa/ Cómo se usa:	1. Configurar proyecto (Entrada y Salida). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	IniAct = año+'-'+mes+'-'+día Año= *4 dígitos a partir 1980* Mes= *2 dígitos del 00 al 12* Día= *2 dígitos del 00 al 31*
Información adicional:	Debe ser menor al valor de Término de la misma actividad. Este dato sirve para el cálculo de métricas del SW.

Tabla 5.52. Definición del dato: Término de Actividad.

Nombre:	Término de Actividad
Alias:	TerAct
Dónde se usa/ Cómo se usa:	1. Configurar proyecto (Entrada y Salida). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	TerAct = año+'-'+mes+'-'+día Año= *4 dígitos a partir 1980* Mes= *2 dígitos del 00 al 12* Día= *2 dígitos del 00 al 31*
Información adicional:	Debe ser mayor al valor de Inicio de la misma actividad. Este dato sirve para el cálculo de métricas del SW.

Tabla 5.53. Definición del dato: Encargado de la actividad.

Nombre:	Encargado de la actividad
Alias:	EnAct
Dónde se usa/ Cómo se usa:	1. Configurar proyecto (Entrada y Salida). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	EnAct = ID del miembro del equipo del proyecto
Información adicional:	Este dato sirve para el cálculo de métricas del SW.

Tabla 5.54. Definición del dato: Status del encargado de la actividad.

Nombre:	Status del encargado de la actividad
Alias:	StaEnAct
Dónde se usa/ Cómo se usa:	1. Configurar proyecto (Salida). 2. Ingresar información del proyecto (Entrada). 3. Revisar información del proyecto (Entrada).
Descripción del contenido:	StaEnAct = *valor con dos posibilidades, es P si es responsable principal y A si es responsable de apoyo*
Información adicional:	Cada actividad tiene un solo responsable principal y varios responsables de apoyo. Este dato sirve para el cálculo de métricas del SW.

Tabla 5.55. Definición del dato: Nivel de la actividad.

Nombre:	Nivel de la actividad
Alias:	NiAct
Dónde se usa/ Cómo se usa:	1. Configurar proyecto (Entrada y Salida). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	NiAct = *4 dígitos desde 0001 hasta 9999 *
Información adicional:	Este dato sirve para el cálculo de métricas del SW.

Tabla 5.56. Definición del dato: Número de actividad del proyecto.

Nombre:	Número de actividad del proyecto
Alias:	Numact
Dónde se usa/ Cómo se usa:	1. Instalar TOOLBDH (Salida). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	Numact = *4 dígitos desde 0001 hasta 9999 *
Información adicional:	Se auto incrementa cada vez que se agrega una actividad. Este dato sirve para el cálculo de métricas del SW.

Tabla 5.57. Definición del dato: Actividades que pertenecen al proyecto.

Nombre:	Actividades que pertenecen al proyecto
Alias:	PA
Dónde se usa/ Cómo se usa:	1. Configurar proyecto (Salida y Entrada). 2. Revisar información del proyecto (Entrada).
Descripción del contenido:	PA = NP+’,’+actividades Proyectos=ActP_1+’,’+ActP_2+’,’…+’,’+ActP_n *Todos los Nombres de los proyectos en los cuales participa*
Información adicional:	Este dato permite acceder a la herramienta

5.3.5 Glosario de términos del documento de especificación de requisitos.

Base de Datos Histórica de proyectos BDH. Es una base de datos que almacena información histórica de proyectos.

Bases de datos BD. Conjunto de datos relacionados entre sí.

Sistema de Gestión de Bases de Datos SGBD *(DBMS, por las siglas en ingles de Database Management System).* Consiste en una base de datos y un grupo de programas para tener acceso a esos datos.

CENTOS. Llamado así por las siglas en inglés de *Community ENTerprise Operating System*. Es un clon a nivel binario de la distribución Red Hat Enterprise Linux, compilado por voluntarios a partir del código fuente liberado por Red Hat. La última versión estable es la 5.0.

Ciclo de Vida de los Sistemas de información CVSI. Es una herramienta que asiste en los proyectos de desarrollo de sistemas de información, para planearlos, ejecutarlos y controlarlos.

Diccionario de datos DD. Es un conjunto de datos estructurados y codificados que describen características de instancias conteniendo informaciones para ayudar a identificar, descubrir, valorar y administrar las instancias descritas. Contiene las características lógicas de los datos que se van a utilizar en el sistema que se programa, incluye nombre, descripción, alias, contenido y organización.

Estructura de División de Trabajo EDT. Es una forma de dividir un proyecto en piezas o partidas manejables para ayudar a asegurar que se identifiquen todos los elementos que se necesiten para completar el alcance del proyecto.

HTTP. El protocolo de transferencia de hipertexto (*HTTP, por las siglas en inglés de HyperText Transfer Protocol)* es el protocolo usado en cada transacción de la Web (WWW). El hipertexto es el contenido de las páginas web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceso a una página y la respuesta con el contenido. También sirve el protocolo para enviar información adicional en ambos sentidos, como formularios con campos de texto.

Licencia pública general*. (GPL por las siglas en inglés de General Public License).* Es una licencia creada por la Free Software Foundation a mediados de los 80's, y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

Sistemas de información SI*.* Sistema basado en computadoras que acepta datos como entrada, procesa los datos y produce información útil para lo usuarios.

MySQL. Es un sistema de gestión de base de datos relacional, multihilo y multiusuario. Por un lado lo ofrece bajo la GNU GPL, pero, empresas que quieran incorporarlo en productos privativos pueden comprar a la empresa una licencia que les permita ese uso.

PHP. Lenguaje de programación usado frecuentemente para la creación de contenido para sitios web con los cuales se puede programar las páginas html y los códigos de fuente. PHP es llamado así por las siglas en inglés de *PHP Hypertext Pre-processor* (inicialmente PHP Tools, o,

Personal Home Page Tools), y se trata de un lenguaje interpretado usado para la creación de aplicaciones para servidores.

PMBOK. Guía del PMBOK® (por las siglas en inglés de Project Management Body of Knowledge) es un estándar en la gestión de proyectos desarrollado por el Project Management Institute (PMI).

Samba. Es una implementación libre del protocolo de archivos compartidos de Microsoft Windows (antiguamente llamado SMB, renombrado recientemente a CIFS) para sistemas de tipo UNIX. Samba es una implementación de una docena de servicios y una docena de protocolos, entre los que están: NetBIOS sobre TCP/IP (NetBT), SMB (también conocido como CIFS), DCE/RPC o más concretamente, MSRPC, el servidor WINS también conocido como el servidor de nombres NetBIOS (NBNS), la suite de protocolos del dominio NT, con su Logon de entrada a dominio, la base de datos del gestor de cuentas seguras (SAM), el servicio Local Security Authority (LSA) o autoridad de seguridad local, el servicio de impresoras de NT y recientemente el Logon de entrada de Active Directory, que incluye una versión modificada de Kerberos y una versión modificada de LDAP.

Servidor HTTP Apache. Es un software (libre) servidor HTTP de código abierto para plataformas Unix (BSD, GNU/Linux, etc.), Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual.

Superusuario. Es el gestor superior.

6. CONCLUSIONES

Mediante la información presentada a lo largo de este trabajo se concluye que:

Es posible generar un documento de especificación de los requisitos para el desarrollo de un sistema de gestión de base de datos histórica de proyectos de software, con el modelo de ciclo de vida de los sistemas de información como un modelo de ciclo de vida generalizado. Siendo este documento el producto de la presente investigación.

Es posible recopilar información que permita el establecimiento de un grupo de métricas mínimas necesarias, para medir un sistema de información con aplicación a base de datos relacional. Éste es otro aporte del presente trabajo.

No se encontraron fuentes de información donde se relacione a las métricas del software con las etapas de ciclo de vida de los sistemas de información. En el presente escrito se propuso un esquema donde se relacionan estos conceptos.

Un sistema de información se puede conformar de una interfaz del usuario, una base de datos relacional y el código necesario para interconectarlos y hacer posibles las transacciones pertinentes.

La calidad de un producto de software debe ser un atributo inherente.

El correcto, conciso y suficiente establecimiento de los requerimientos y términos de referencia de un sistema de información con aplicación a base de datos relacional o de una aplicación es de vital importancia para el éxito del proyecto que se desarrolla.

De igual manera se recomienda que:

El sistema debe contemplar términos de referencia y Acuerdos de Nivel de Servicio (SLA). Éstos últimos indicando sus características, tales como:

- El cliente tiene el control sin necesidad de invertir tiempo.
- Se caracteriza por ser un proceso estructurado.
- Es una metodología universal, homogénea y común.
- Es un instrumento que promueve la convergencia organizacional.
- Es una herramienta para hacer benchmarking interno.
- Es una visión multidimensional de las relaciones entre servicios/cliente.
- Constituye un punto de referencia para el mejoramiento continuo (todo aquello susceptible de ser medido es susceptible de ser mejorado).

Además de los puntos a cubrir de forma contractual, tales como:

- Tipo de servicio
- Soporte a clientes y asistencia
- Provisiones para seguridad y datos
- Requerimientos de software o hardware
- Garantías del sistema y tiempos de respuesta
- Disponibilidad del sistema
- Conectividad
- Multas por servicios no suministrados

REFERENCIAS

1. Alhir, S. S. (1998). UML in a Nutshell. O'Reilly & Associates, Inc.

2. Barra, P. C. (2007). *Proceso y Proyecto de Ingeniería de Software.* Revista de Marina No. 1/98.

3. Barreiro, E. (2007). Medición y Métricas del Software. Universidad de Vigo. Escuela Superior de Ingeniería Informática. Ingeniería del software de gestión. Disponible en línea en: http://trevinca.ei.uvigo.es/~ebalonso/asignaturas/esx/guiones/esxClase26.pdf

4. Bedini, G. A. (2007). *Calidad Tradicional y de Software*. Universidad Técnica Federico Santa María. Chile.

5. Bendahan, M. (2007). *Desarrollo de Software*. Disponible en: http://cursos-gratis.emagister.com.mx/emag_users/modulo_transaccional/frame.cfm?Id_user=3000140579&id_centro=43204110021466565570676950524550&id_curso=5106904003225050515668 5466654555&url_frame=http://www.monografias.com/trabajos5/desof/desof.shtml Consultado el 15 de diciembre de 2007 a las 22:00.

6. Bieman, J. M. y Ott, L. M. (1994). Measuring Functional Cohesion. IEEE Trans. Software Engineering, vol. 20, Núm. 8, Agosto, pp. 308-320.

7. Boehm, B. W. (1981). Software Engineering Economics. Prentice-Hall.

8. Boehm, B. W. (1991). Software Risk Management: Principles and Practices. Institute of Electrical and Electronics Engineers Software.

9. Boehm, B. W. (1998). *A Spiral Model of Software Develpment and Enhancemen*t. The Institute of Electrical and Electronics Engineers.

10. Borges de Barros, P. H. (2002). Análisis experimental de los criterios de evaluación de usabilidad de aplicaciones multimedia en entornos de educación y formación a distancia. Tesis Doctoral. Universidad Politecnica de Catalunya. España.

11. Card, D. y Glass, N. R. L. (1990). Measuring Software Design Quality. Prentice Hall.

12. Carral, A. (1999). *La Industria del Software en México*. Asociación Mexicana de la Industria de Tecnologías de Información AMITI. Disponible en: http://www.bancomext.com/Bancomext/publicasecciones/secciones/421/acarralpp.ppt. Consultado el 4 de enero de 2008 a las 18:00.

13. Casares, C. (2007) *Modelo de Datos*. Programación en castellano. Disponible en línea en: http://www.programacion.net/bbdd/tutorial/moddatos/1/. Consultado el 13 de junio de 2008 a las 3:30.

14. Codd, E. (1990). The Relational Model for Database Management: Version 2. Addison-Wesley.

15. Date, C. J. (1990). Introducción a los Sistemas de Bases de Datos. Addison-Wesley Iberoamérica. Pp 648.

16. Davis, A. et al. (1993). *Identifying and Measuring Quality in Software Requirements Specification*. Proc. 1er Intl. Software Metrics Syposium, IEEE, Baltimore, USA, MD. Mayo, pp. 141-152.

17. DeMarco, T. (2007). Controlling Software Project. Tourdon-Press.

18. Dhama, H. (1995). Quantitative Models of Cohesion and Coupling in Software. Journal of Systems and Software, vol. 29, Núm.4, April.

19. Enciclopedia Británica, (2006). *Definición de software.* Disponible en: http://www.britannica.com/search?query=software&ct=&searchSubmit.x=4&searchSubmit.y=13. Consultado el 4 de diciembre de 2007 a las 15:00.

20. Favela, J. (2001) *¿Podemos correr el riesgo de no desarrollar la industria del software en México?* Disponible en línea en: http://gaceta.cicese.mx/ver.php?topico=articulos&ejemplar=120&id=25. Consultado el 4 de diciembre de 2007 a las 16:00.

21. Fenton, N. (1991). Software Metrics. Chapman & Hall.

22. Fernández, P. (2005). *China e India se están dando cuenta de que pueden hacer muchas cosas juntos*. Periódico El País. Disponible en línea en: http://www.elpais.com/articulo/internet/China/India/estan/dando/cuenta/pueden/hacer/muchas/cosas/juntos/elpeputec/20051026elpepunet_5/Tes?print=1. Consultado el 4 de diciembre de 2007 a las 17:00.

23. Futrell, R. T. et al. (2002). Quality Software Project Management. Prentice Hall PTR. Disponible en línea en: http://proquest.safaribooksonline.com/. Consultado el 22 de noviembre de 2007 a las 19:00.

24. Gaddis, Paul O. (1991). *The Project Manager*. Harvard Business Review. Estados Unidos de Norteamérica.

25. García E. A. (2004). *Por el Desarrollo de Software*. Clave Empresarial.

26. Gray, C. F. (2003). Project Management: The Managerial Process. McGraw Hill.

27. Guido, J. (2003). Administración Exitosa de Proyectos. Thomson.

28. Halstead, M. (1977). Elements of Software Science. Holland.

29. Henry, S. y Kafura, D. (1981). *Software Structure Metrics Based on Information Flow*. IEEE Trans. Software Engineering, vol. SE-7, Núm. 5. Septiembre, pp. 510-518.

30. Herrera, J. y Lizka, J. (2007). *Ingeniería de Requerimientos.* Disponible en línea en: http://www.monografias.com/trabajos6/resof/resof.shtml

31. Higuera, R., (1995). *Team Risk Management*. Crosstalk. US – Department of Defense. Enero. pp. 2 – 4.

32. ICC (2007). International Chamber of Commerce. The World Business Organization. Software de Código Abierto. Disponible en línea en: http://www.iccwbo.org/uploadedfiles/ICC/policy/e-business/Statements/373-66_OSS_ spanish.pdf. Consultado el 24 de febrero de 2008 a las 22:00.

33. IEEE (1987). IEEE Standard Glossary of Software Engineering Terminology ANSI/IEEE Std. 1058.1-1987.The Institute of Electrical and Electronics Engineers.

34. IEEE (1990). Standard Glossary of Software Engineering Terminology. ANSI/IEEE Std 610-1990.The Institute of Electrical and Electronics Engineers (IEEE) – Software.

35. IEEE (1993). IEEE Standard Collection: Software Engineering, IEEE Standard 610.12-1990. The Institute of Electrical and Electronics Engineers (IEEE) – Software.

36. IEEESGS (1993). IEEE Software Engineering Standards. Standard Glossary of Software Engineering. IEEE.

37. IFP (1994). Function Point Counting Practices Manual. Release 4.0. International Function Point Users Group (IFPUG).

38. Ince, D. et al. (1993). Introduction to Software Project Management and Quality Assurance. McGraw Hill.

39. ISLE (2000). The ISLE (International Standards for Language Engineering). Classification of Machine Translation Evaluations. Information Sciences Institute (ISI). Disponible en línea en: http://www.isi.edu/natural-language/mteval/. Consultado el 19 de febrero de 2008 a las 18:00.

40. ISO/IEC 15504 (2004). ISO/IEC 15504-1, Information Technology - Process Assessment - Part 1: Concepts and Vocabulary.

41. ISO/IEC 9126-1 (2001). International Standard Organization (ISO). ISO/IEC 9126, Software Engineering – Product quality. Part 1: Quality Model.

42. Jacobson, I. (1992). Object-Oriented Software Engineering. Addison-Wesley.

43. Jalote, P. (1997). An Integrated Approach to Software Engineering. Springer Verlag.

44. Johnson, J. (1999). *Turning CHAOS into SUCCESS*. Disponible en línea en: http://www.softwaremag.com/L.cfm?Doc=archive/1999dec/Success.html. Consultado el 24 de febrero de 2008 a las 22:00.

45. Kan, S. H. (2002). Software Quality Metrics Overview. Addison Wesley Professional. Disponible en línea en: http://www.informit.com/articles/article.asp?p=30306&seqNum=3&rl=1. Consultado el 12 de octubre de 2007 a las 22:00.

46. Kimmons, R. L. (1990). Project Management Basics: A Step-by-step Approach. New York, USA.

47. Klasky, H. (2003). A Study of Software Metrics. Graduate School-New Brunswick Rutgers. Tesis de Maestría. The State University of New Jersey. USA.

48. Korth, H. F. y Silberschatz A. (2006). Fundamentos de Bases de Datos. Ed. McGraw Gill.

49. Luna Lara, Luis Alfredo (2004). Plan para aplicar los principios de la Administración Moderna de Proyectos de Software en empresas que se dedican al desarrollo de software en el área metropolitana de Monterrey. Tesis de Maestría. Instituto Tecnológico de Estudios Superiores de Monterrey. México.

50. Marcianiak, J. J. (1994). Encyclopedia of Software Engineering. Wiley.

Marqués A. (2001). *Ciclo de Vida de los Sistemas de Información.* Disponible en línea en: http://www3.uji.es/~mmarques/f47/apun/node66.html. Consultado el 5 de diciembre de 2007 a las 18:00.

51. Masi H. (2007) *Industria del Software: Mercado, Tendencias y Oportunidades para el Software Libre.* Disponible en línea: http://www.softcatala.org/~jmas/conf/programari_lliure_economia-es.pdf. Consultado el 4 de diciembre de 2007 a las 12:00.

52. McCall, J. *et al.* (1977). Factors in Software Quality. 3 vols. NTIS AD-A049-014,015,055, Noviembre.

53. McConnell, S. (1997). Desarrollo y Administración de Proyectos Informáticos. McGraw Hill Interamericana.

54. Mills, H. (1980). The Management of Software Engineering. IBM Systems.

55. Olguín, H. R. (1997). Organización y administración de centros de cómputo. Facultad de Ingeniería. Universidad Nacional Autónoma de México. México.

56. Paulk, M. et al. (1993). Capability Maturity Model for Software. Software Engineering Institute. Carnegie Mellon University. USA.

57. Peñaloza B. M. (2002). *La Industria del Software, una Oportunidad para México*. Entérate. México. Disponible en línea: http://www.enterate.unam.mx/Articulos/2002/enero/software.htm. Consultado el 6 de diciembre de 2007 a las 22:00.

58. Perampalam, S. (2007). Software Metrics.

59. Pfleeger, S. L. (2002). Ingeniería de Software. Pearson Education. Pp. 729.

60. Piattini M. (2000). *Métricas para la Evaluación de la Complejidad de Bases de Datos Relacionales*. Computación y Sistemas. Vol. 3. No. 4. pp. 246-273. México.

61. PMI (2004). Project Management Institute. A Guide to the Project Management Body of Knowledge. Project Management Institute. USA.

62. Pressman, R. (2002) Ingeniería de Software: Un Enfoque Práctico. McGraw Hill.

63. Prince R. (2003). *RUP/UP: 10 Easy Steps*. X-tier SAE Inc. Disponible en línea en: http://hosteddocs.ittoolbox.com/RP092305.pdf. Consultado el 3 de marzo de 2008 a las 22:00.

64. Putman, L. (1992). Measures for Excellent. Yourdon Press.

65. Qiu, L. (2001). *Software Development and Trade*. Disponible en línea en: http://www.iips.org/Qui_paper.PDF. Consultado el 16 de marzo de 2008 a las 21:00.

66. Quiroz, J. (2003). El Modelo Relacional de Bases de Datos. Boletín de Política Informática Núm. 6, 2003. Disponible en línea en: http://www.inegi.gob.mx/inegi/contenidos/espanol/prensa/Contenidos/Articulos/tecnologia/relacional.pdf. Consultado el 26 de enero de 2008 a las 21:00.

67. Ragland, B. (1995). *Measure, Metric or Indicator: What's the difference?* Crosstalk, vol. 8, Núm. 3. Marzo. Pp 29-30.

68. Riggs, J. (1981). Production System Planning, Analysis and Control. Wiley.

69. Royce, W. W. (2002). Software Project Management: A Unified Framework. Addison Wesley, ISBN 0201309580. 416 páginas.

70. Royce, W. W. (1970). *Managing the Development of Large Software Systems: Concepts and Techniques*. 1970 WESCON Technical Papers, 14. Western Electronic Show and Convention.

71. Ruffinatti, A. F. (2007). *La Industria del Software en la India: ¿Un Éxito Casual?* Winred. Disponible en línea en: http://winred.com/management/la-industria-del-software-en-la-india-un-exito-casual/gmx-niv116-con2774.htm. Consultado el 3 de noviembre de 2007 a las 21:00.

72. Rumbaugh, J. et al. (1999). The Unified Modeling Language, User Guide. Addison Wesley.

73. Sánchez A. E. (1996). *El Mercado Mundial de la Industria de la Información*. Consultoría BIOMUNDI, CUBA.

74. Sears, A. (1993). *Layout Appropriateness: A Metric for Evaluating Users Interface Witged Layout.* IEEE Trans. Software Engineering, vol. 19. Julio. Pp. 707-719.

75. SEI (1987). Software Engineering Institute. *Models of Software Evolution: Life Cycle and Process, Curriculum Module SEI-CM-10-1.0*. Software Engineering Institute.

76. Senn, J. A. (1987). Análisis y Diseño de Sistemas de Información. México, McGraw Hill.

77. SML@b (2007). Software Measurement Laboratory. *CAME Tools Software Metrics*. Universidad de Magdeburg. Alemania. Disponible en línea en: http://www.smlab.de/CAME/CAME.p4.html. Consultado el 22 de enero de 2008 a las 18:00.

78. Sommerville, I. (2002). Ingeniería de Software. Pearson Educación.

79. SPME (2005). Subsecretaría de la Pequeña y Mediana Empresa. *Estudio de Producto/Mercado Software/América Latina.* Ministerio de Comercio. Argentina. Disponible en línea en: http://www.proargentina.gov.ar/documentos/bib_proargentina/Estudio_Producto_Software_Latinoamerica.pdf. Consultado el 10 de noviembre de 2007 a las 20:00.

80. The, L. (1993). Project Management Software That's IS Friendly. Datamation, Octubre. pp. 34-38.

81. Torres P. (2007). *Administración de Requisitos.* Universidad Autónoma de Guadalajara. Disponible en línea en: http://www.ewh.ieee.org/r9/guadalajara/boletin/marzo02/modelofurps.htm
82. UEAFIT (1998). Escuela de Administración, Finanzas y Tecnologías*. Guía Práctica: Desarrollo de un Sistema de Información Basado en Tecnologías de Información Computarizadas, SIBTIC.* Escuela de Administración, Finanzas y Tecnologías (EAFIT). Colombia.

83. UVP (2007). *Introducción al Proceso de Desarrollo de Software.* Universidad Politécnica de Valencia, Departamento de Sistemas Informáticos y Computación. España.

84. Veenendaal, E. V. et al. (2002). Measuring Software Product Quality. Software Quality Professional.

85. Vignaga, A. y Perovich, D. (2005). Enfoque Metodológico para el Desarrollo Basado en Componentes. Instituto de Computación. Facultad de Ingeniería. Universidad de la República. Uruguay. Disponible en línea: http://dis.um.es/~jsaez/dbc/curso0607/docs/art01_vp03.pdf. Consultado el 6 de enero de 2008 a las 17:00.

86. Wiegers, K. E. (2007). A Software Metrics Primer. Process Impact. Disponible en línea en: http://www.processimpact.com/articles/metrics_primer.pdf. Consultado el 21 de febrero de 2008 a las 22:00.

87. Xinhuanet (2007). *Industria China de software incrementa sus ingresos un 23% en primer semestre.* Disponible en línea en: http://www.spanish.xinhuanet.com/spanish/2007-07/29/content_464760.htm. el 5 de noviembre de 2007 a las 22:00.

88. Xuan, N. H. (1999). *Software Engineering.* Institute of Information Technology. National Center for Natural Science and Technology.

89. Zahran, S. (1998). Software Process Improvement. Practical Guidelines for Business Success. Addison Wesley.