4
2 ejc

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
"ACATLAN"

PLANEACION Y ASEGURAMIENTO DE LA CALIDAD EN LOS PROYECTOS DE DESARROLLO DE SOFTWARE

TESIS PROFESIONAL

QUE PARA OBTENER EL TITULO DE:

Lic. en Matemáticas Aplicadas y Computación

P R E S E N T A :

AIMEE LILIAN ARDINES BUJANOS

Acatlán, Edo. de México 1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES "ACATLAN"

DIVISION DE MATEMATICAS E INGENIERIA
PROGRAMA DE ACTUARIA Y M.A.C.

SRITA. AIMEE LILIAN ARDINES BUJANOS
Alumna de la carrera de M.A.C.
P r e s e n t e .

Por acuerdo a su solicitud presentada con fecha 16 de febrero de 1993, me complace notificarle que esta Jefatura tuvo a bien asignarle el siguiente tema de Tesis: PLANEACION Y ASEGURAMIENTO DE LA CALIDAD EN LOS - PROYECTOS DE DESARROLLO DE SOFTWARE, el cual se desarrollará como sigue:

INTRODUCCION
CAP. I Conceptos Básicos
CAP. II Planeación de Proyectos de Software y sus métricas
CAP. III Estimación de costos y esfuerzo en el desarrollo de Software
CAP. IV Aseguramiento de la calidad de Software
CONCLUSIONES
ANEXOS
BIBLIOGRAFIA

Asimismo, fué designado como Asesor de Tesis el LIC. JORGE ARTURO LOPEZ PAREYON, Profesor de esta Escuela.

Ruego a usted tomar nota que en cumplimiento de lo especificado en la Ley de Profesiones, deberá presentar servicio social durante un tiempo mínimo de seis meses como requisito básico para sustentar examen profesional así como de la disposición de la Coordinación de la Administración Escolar en el sentido de que se Imprima en lugar visible de los ejemplares de la tesis el título del trabajo realizado. Esta comunicación deberá imprimirse en el interior de la misma.

A T E N T A M E N T E
"POR MI RAZA HABLARA EL ESPIRITU"
Acatlán, Edo. Méx., noviembre 18 de 1994.

ACT. LAURA MA. RIVERA ESPERRA
Jefe del Programa de Actuaría
y M.A.C.

JEFATURA DE

cg'

Por su comprensión, confianza, dedicación y especial forma de amar, por quien he llegado a ser la mujer que ahora soy, con amor a ...

... Mi Madre.

Ejemplo de estudio y honestidad profesional, mi mejor libro siempre abierto, con amor a ...

... Mi Padre.

Antiguo y futuro caballero águila, en camino hacia la verdadera dimensión del ser, mi hermano...

... José Luis.

Rebeldes, impulsivos, y por fortuna mis hermanos, no existe nadie como ellos pues son los mejores...

... Irving y Rogelio.

Mi fortaleza y mi debilidad, un regalo de Dios, mi amor...

... Pedro.

Con cariño y respeto al maestro y amigo, Lic. Jorge Arturo López Pareyón, por su apoyo incondicional en todo momento para concluir satisfactoriamente mi formación como profesional.

A la UNAM por todo lo que obsequia a sus estudiantes; a los profesores por su tiempo y noble tarea de enseñar, en especial al Fis. Mát. Jorge Luis Suárez Madariaga y al Ing. Pablo H. González Videgaray, por su amistad y valiosos consejos.

Por su lealtad, a mis amigos Andrea, Jorge A., Fernando, Sonia, Roberto, Pablo, Lucy, Edgar y Bety.

ÍNDICE DE CONTENIDO

INTRODUCCIÓN

El objetivo general de esta tesis es dar a conocer las técnicas de administración y de ingeniería encaminadas a finalizar a tiempo, de acuerdo al calendario de actividades, proyectos de desarrollo de software, así como el aseguramiento de calidad de los mismos y sin sobregiros en el presupuesto establecido durante la estimación, lo cual permitirá administrar correctamente y controlar el desarrollo de una aplicación de principio a fin.

La inquietud por desarrollar este trabajo surge de la experiencia profesional que he tenido, participando en la realización de proyectos de desarrollo de software. Generalmente el costo final de un software es muy superior a su costo estimado; por otra parte se fijan calendarios de actividades y fechas de terminación que pocas veces se cumplen y a medida que el sistema crece se da menor importancia a la calidad. Prácticamente nadie se preocupa por llevar un registro de datos de los proyectos de desarrollo de software, por lo que no se cuenta con documentos escritos que sirvan de guía para mejorar la productividad y calidad durante un desarrollo posterior.

Todas esta circunstancias no son nuevas, hace tiempo que fueron reconocidas y que en conjunto se les denominó *crisis del software*, y como respuesta a esta crisis surgió un conjunto de técnicas denominadas *ingeniería de software*. Estas técnicas tratan al software como un producto de ingeniería que requiere de planeación, análisis, diseño, implementación o codificación, pruebas y mantenimiento. El propósito de este trabajo no es mostrar un estudio conciso de cada uno de los procesos de la ingeniería de software, ya que existen muchos libros especializados en cada uno de ellos. Lo que se pretende es destacar la importancia de utilizar un modelo de planeación para administrar y controlar los sistemas de información y por otra parte dar a conocer los principales métodos y herramientas para la estimación de los costos, así como para el desarrollo de productos de alta calidad.

El capítulo I, compara la evolución del software con respecto al hardware y describe los problemas relativos a la crisis del software y a las soluciones basadas en la ingeniería de software. Por otra parte, incluye aspectos importantes que se deben tomar en cuenta para administrar correctamente un proyecto, con el fin de obtener productos de software de calidad.

El capítulo II ilustra la importancia de realizar una planeación adecuada a una organización determinada, de tal forma que las personas encargadas de administrar los sistemas de información, verdaderamente controlen el desarrollo de los diversos proyectos que se decidan implementar y para lo cual el analista o ingeniero de sistemas deberá poner en práctica algún modelo de planeación, similar al propuesto en este capítulo. Finalmente se destaca la importancia de medir durante el proceso de ciclo de vida de un producto de software y se proponen algunos modelos relativos a las métricas de productividad.

El capítulo III, está dedicado a explicar la utilización de algunas técnicas y herramientas automáticas de estimación de costos, que permitan controlar de manera real el presupuesto, el esfuerzo y el tiempo requerido para el desarrollo de un proyecto, evitando que el costo total de un proyecto sea sobrepasado y logrando que se administren correctamente los recursos.

El capítulo IV, tiene como finalidad ilustrar la manera de desarrollar software de calidad. Contiene una serie de lineamientos para la instrumentación de la calidad a lo largo del proyecto. Por otra parte, se identifican y clasifican características de calidad que debe poseer un producto y se presenta un enfoque para definir métricas para cada una de las características de calidad. Finalmente, se proponen algunos medios para lograr que las características de calidad estén presentes durante todo el ciclo de vida del software.

Capítulo I

CONCEPTOS BÁSICOS

1.1 Introducción

En este capítulo se revisan los conceptos básicos relativos al software, iniciando con una descripción de la evolución del software hasta la cuarta era, analizando las características del software así como sus áreas de aplicación y destacando los problemas que motivaron la creación y utilización de diversas técnicas que evitaran el retraso en el desarrollo de proyectos, así como los elevados costos del mismo y el poco rendimiento que ofrecían.

Posteriormente, se analizan los principales factores que determinan el éxito o fracaso de un proyecto, lo cual esta íntimamente ligado con el establecimiento de medidas a lo largo de un proyecto para tener el control del mismo.

Para finalizar el capítulo, en la última sección se trata a manera de introducción lo que implica la calidad del software, así como lo importante que es realizar una medición de la calidad a lo largo del proyecto pasa asegurar la calidad del producto.

1.2 La Importancia del Software [1]

Durante las tres primeras décadas de la informática, el principal desafío era desarrollar el hardware de las computadoras de forma que se redujera el costo de procesamiento y almacenamiento de datos. A lo largo de la década de los 80, los avances en microelectrónica dieron una mayor potencia de cálculo a la vez que una reducción del costo.

Actualmente el problema es diferente. El principal desafío es reducir el costo y mejorar la calidad de las soluciones basadas en computadoras, es decir, soluciones que se implementan con el software.

La capacidad de procesamiento y almacenamiento del hardware representa un gran potencial de cálculo. El software es el mecanismo que nos facilita utilizar y explotar este potencial.

1.2.1 La Evolución del Software

Los adelantos en el desarrollo del software están ligados a las cuatro décadas de la evolución de las computadoras. Un mejor rendimiento del hardware, un tamaño más pequeño y un costo más bajo, dieron lugar a equipos de cómputo más sofisticados. Nos trasladamos de los procesadores con tubos al vacío a dispositivos electrónicos.

La Figura 1.1 describe la evolución del software dentro de las áreas de aplicación de los sistemas basados en computadoras.

Durante los primeros años de desarrollo de las computadoras, el hardware sufrió cambios continuos, mientras que el software se desarrollaba de manera tradicional, es decir, mediante prueba y error del código. El desarrollo del software se realizaba prácticamente sin ninguna planificación. La mayor parte del hardware se dedicaba a la ejecución de un programa único que a su vez se dedicaba a una aplicación específica. El software se diseñaba a la medida para cada aplicación y tenía una distribución relativamente pequeña.

[1] [PRE87] pp. 1-33; [FAI85] pp. 1-25; [SOM85] pp. 3-12.

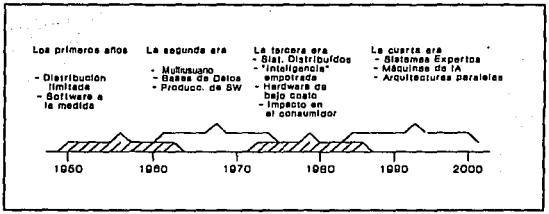

Figura 1.1 Evolución del Software.

La producción de software, es decir, de programas desarrollados para ser vendidos a uno o más usuarios, apenas iniciaba. La mayoría del software se desarrollaba y utilizaba por la misma persona u organización. La misma persona lo escribía, lo ejecutaba y si fallaba lo depuraba. Ya que la rotación de personal era baja, los ejecutivos estaban seguros de que esa persona estaría allí cuando se encontrara algún error. Debido a este desarrollo personalizado del software, el diseño era un proceso implícito, ejecutado en la cabeza de alguien y normalmente sin ningún tipo de documentación.

La segunda era de la evolución de los sistemas de computadora se extiende desde la mitad de la década de los 60 hasta finales de los 70. La multiprogramación, los sistemas multiusuarios, introdujeron nuevos conceptos de interacción hombre-máquina. Los avances en los dispositivos de almacenamiento en línea condujeron a la primera generación de sistemas de administración de base de datos.

La segunda era se caracterizó también por el uso del software como producto y la llegada de las "casas de software". El software se desarrollaba para una amplia distribución en un mercado multidisciplinario. Se distribuían los programas para computadoras grandes y minicomputadoras a cientos y a veces a miles de usuarios.

Conforme creció el número de computadoras, comenzaron a desarrollarse programas para computadoras. Las "casas" desarrollaron aplicaciones de decenas de miles de líneas de código fuente. Todos estos programas tenían que ser corregidos cuando se detectaban fallas o se detectaban cambios en los requerimientos del usuario o si tenía que adaptarse a un nuevo hardware que se hubiera comprado. Estas actividades se llamaron colectivamente mantenimiento del software. El mantenimiento comenzó a resultar muy caro, ya que los programas eran muy personalizados, es decir, como casi nadie usaba

métodos formales para el desarrollo, los programas eran difíciles de entender por una persona ajena la código desarrollado, lo cual los hacía prácticamente imposibles de mantener. Todo esto provocó la inversión de mayor tiempo y esfuerzo para realizar modificaciones, lo que afecta el costo total del sistema. Así comenzó una *crisis del software*.[2]

La tercera era de la evolución de los sistemas de computadoras comenzó a mediados de los 70. El sistema distribuido (computadoras múltiples, cada una ejecutando funciones concurrentemente y comunicándose con alguna otra), incrementó la complejidad de las aplicaciones a desarrollar. Redes de área local y global, comunicaciones digitales y la creciente demanda de acceso "instantáneo" a los datos crearon una fuerte presión sobre los desarrolladores de software. La tercera era también se caracteriza por la llegada y uso de los microprocesadores y computadoras personales.

La cuarta era en software de computadoras inicia a mediados de los 80. Autores como Feigenbaum y McCorduck[3] predijeron que "las computadoras de la quinta generación y su correspondiente software tendría un profundo impacto en el equilibrio de la potencia política e industrial de todo el mundo". Las técnicas de la cuarta generación y su software correspondiente cambiaron la forma en que algunas personas construyen los programas de computadora, con lo cual se busca evitar que la capacidad de dar mantenimiento a los programas existentes esta amenazada por el mal diseño y uso de recursos inadecuados.

Inicialmente los sistemas basados en computadoras se desarrollaban usando técnicas de administración orientadas al hardware. Los administradores del proyecto se centraban en el hardware, debido a que era el factor principal del presupuesto en el desarrollo del sistema. Exigían un análisis y diseño completo antes de que algo se construyera, medían el proceso para determinar donde se podían hacer mejoras, en general aplicaban controles, métodos y herramientas conocidas como ingeniería del hardware.

En cambio para la programación existían pocos métodos formales y pocas personas los usaban. El programador aprendía su oficio mediante prueba y error.

---

[2] [PRE87], [FAY85]
[3] [PRE87] p. 4.

Actualmente la distribución del costo para el desarrollo de sistemas informáticos ha cambiado. El software, en vez del hardware, normalmente es el elemento principal del costo. En las décadas pasadas los ejecutivos y muchos practicantes técnicos se habían hecho las siguientes preguntas:

- ¿Por qué lleva tanto tiempo terminar los programas?
- ¿Por qué es tan alto el costo?
- ¿Por qué no se pueden encontrar todos los errores antes de darle el software al usuario?
- ¿Por que es difícil medir el progreso conforme se desarrolla el software?

Estas y muchas otras preguntas, son una manifestación de las características del software y la forma en que se desarrolla, un problema que ha llevado a practicar la ingeniería del software.

1.2.2 Características del Software

Para poder comprender lo que es el software, es importante examinar las características del software que lo hacen diferente de otras cosas que los hombres pueden construir. Si se construye una nueva computadora, el boceto inicial, el establecimiento del diseño formal y prototipo de prueba, evolucionan a un producto físico con tarjetas de circuito, fuentes de potencia, etc.

El software es un elemento lógico en vez de físico del sistema. Por tanto el software tiene características considerablemente distintas a las del hardware.[4]

El software es desarrollado, no es fabricado. Aunque existen algunas similitudes entre el desarrollo de software y la construcción de hardware, las dos actividades son diferentes. En ambas actividades la buena calidad se obtiene mediante un buen diseño. Ambas actividades dependen de las personas, pero la relación entre la gente dedicada y el trabajo realizado es diferente para el software. Lo anterior se explicará con más detalle en el capítulo tercero.

[4] [PRE87] p. 6.

La **Figura 1.2** describe la proporción de fallas como una función del tiempo, para el hardware. La relación, llamada frecuentemente la "curva de bañera", indica que el hardware exhibe muchas fallas al principio de su vida. Estas fallas son atribuibles frecuentemente a defectos de diseño o de fabricación. Cuando se corrigen los defectos, caen las fallas hasta su nivel más bajo, en donde permanecen estacionadas durante un cierto período de tiempo. Sin embargo, conforme pasa el tiempo, las fallas vuelven a presentarse conforme los componentes del hardware sufren los efectos acumulativos de la suciedad, vibración, uso, temperaturas extremas y muchos otros males externos.

En otras palabras el hardware comienza a deteriorarse.

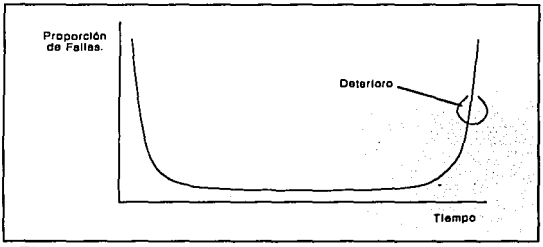

Figura 1.2 Curva de Fallas del Hardware.

El software no es afectado por males del ambiente como los que hacen que el hardware se deteriore. Por lo tanto, en teoría, la curva de fallas para el software tendría la forma mostrada en la **Figura 1.3**. Los defectos no descubiertos harán que falle durante las primeras etapas de la vida del programa. Sin embargo, una vez que estos se corrigen, suponiendo que no se introducen nuevos errores, la curva se aplanaría, como muestra la figura. Entonces *el software no se descompone, pero se deteriora*![5]

[5] [PRE87] p. 7.

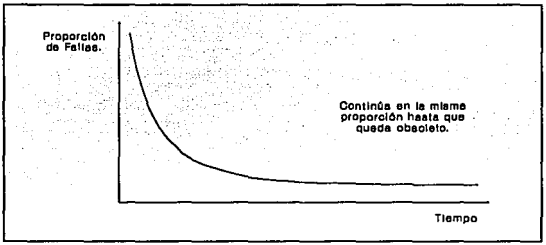

Figura 1.3 Curva de Fallas del Software. (Idealizada)

Lo anterior se puede comprender mejor considerando la **Figura 1.4**. Durante su vida el software sufre cambios (mantenimiento). Conforme se hacen los cambios, es probable que se introduzcan nuevos defectos, haciendo que la curva de falla tenga picos como muestra la Figura 1.4. Antes de que la curva pueda volver al estado original, se solicita otro cambio, haciendo que de nuevo se cree otro pico. Lentamente el nivel mínimo de fallas comienza a crecer; el software se va deteriorando debido a los cambios.[6]

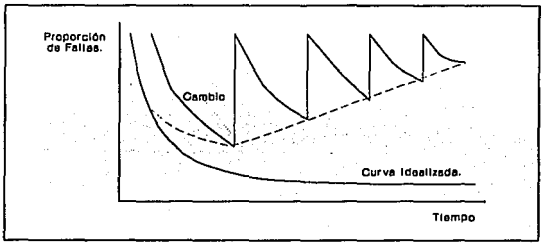

Figura 1.4 Curva de Fallas Reales del Software.

[6] IBM llevó a cabo un estudio en el que se realizaron inspecciones del software, concluyendo en un modelo de amplificación y eliminación de defectos. [PRE85] pp. 488 - 491.

Otro aspecto del deterioro ilustra la diferencia entre el hardware y el software. Cuando un componente del hardware se deteriora, se sustituye por una pieza de repuesto. No hay "piezas de repuesto" para el software. *Cada falla en el software indica un error en el diseño o en el proceso mediante el que se tradujo el diseño a un código ejecutable por la máquina.* Por tanto, el mantenimiento del software es mucho más complejo que el mantenimiento del hardware.

1.2.3 La Crisis del Software

La crisis del software se refiere a un conjunto de problemas encontrados en el desarrollo del software de computadoras. Los problemas no están limitados al software que "no funciona adecuadamente". Sino que la crisis del software abarca los problemas relacionados con cómo desarrollar el software, como mantener un volumen de software existente y como satisfacer la demanda creciente de software.

- Problemas

Existen muchos problemas que caracterizan la crisis del software dentro de los cuales se destacan los siguientes: (1) la planificación y estimación de costos es imprecisa y (2) la calidad del software a veces no llega a ser ni adecuada. Se ha sufrido el sobrepasar los costos en un orden de magnitud. Se ha errado en la planificación en meses o años. Se ha hecho muy poco para mejorar la productividad de los trabajadores en software. Los errores en los nuevos programas producen en los usuarios insatisfacción y falta de confianza. Tales problemas son solo las manifestaciones más visibles de otras dificultades del software:

No se tiene tiempo de recopilar datos sobre el proceso de desarrollo de software. Sin datos históricos como guía, la estimación no es buena y los resultados predichos muy pobres. Sin una indicación sólida de productividad, no se puede evaluar con precisión la eficacia de las nueva herramientas, técnicas o estándares.

La insatisfacción del usuario con el sistema "terminado" se produce frecuentemente. Los proyectos de desarrollo de software inician con solo una vaga indicación de los requerimientos del usuario. Normalmente la comunicación entre el usuario y el que desarrolla el software es muy escasa.

La calidad del software es cuestionable. Se ha empezado a comprender la importancia de la prueba sistemática y técnicamente completa del software.[7] Los conceptos cuantitativos como la confiabilidad del software y garantía de calidad, se tratarán en el capítulo cuarto.

El software existente puede ser muy difícil de mantener. La tarea de mantenimiento del software se lleva la mayor parte de todo el dinero invertido en el software. El mantenimiento no se ha considerado un criterio importante en la aceptación del software.

Todos los problemas descritos anteriormente pueden corregirse. La clave está en dar un enfoque de ingeniería al desarrollo de software, junto con la mejora continua de técnicas y herramientas.

- Causas

Los problemas asociados con la crisis del software se han producido por la complejidad del propio software y por los errores de las personas encargadas del desarrollo del mismo.

Como se mencionó anteriormente, el software es un elemento lógico en vez de físico; por tanto, el éxito se mide por la calidad de una entidad única en vez de por muchas entidades fabricadas. El software no se descompone. Si se encuentran fallas, existe una probabilidad alta de que se introdujeran inadvertidamente durante el desarrollo y no se detectaran durante la prueba. Se reemplazan las "partes defectuosas" durante el mantenimiento del software, pero se tiene muy pocas o incluso ninguna pieza de repuesto; es decir, el mantenimiento incluye normalmente la corrección o modificación del diseño.

Los problemas tratados anteriormente han sido causados por defectos humanos, es decir, los ejecutivos de nivel medio y alto sin conocimientos en software, frecuentemente son los responsables del desarrollo de software, entonces, el administrador de proyectos debe preocuparse por aprender las técnicas novedosas que pueden utilizarse para medir el desarrollo del proyecto, aplicar métodos efectivos de control, y llegar a conocer una tecnología que cambia rápidamente. El administrador debe poder comunicarse con todos los componentes implicados en el desarrollo del

[7][FRY91]

software -usuarios, desarrolladores, equipo de soporte y otros- . La comunicación puede romperse debido a que las características especiales del software, y si los problemas particulares asociados con su desarrollo son mal comprendidos, los problemas asociados con la crisis del software se multiplican.

Los programadores, que ahora tendrán que ganarse el titulo de ingenieros en software, no se preocupan por llevar a cabo un entrenamiento formal en las nuevas técnicas de desarrollo de software. Algunas personas desarrollan un método ordenado y eficiente de desarrollo del software mediante prueba y error, pero muchos otros desarrollan malos hábitos que dan como resultado una pobre calidad y mantenibilidad del software.

Generalmente todos nos resistimos al cambio. Sin embargo, es irónico, que mientras el potencial de cálculo (hardware) experimenta enormes cambios, la gente de software, responsables de aprovechar dicho potencial, se oponga a los cambios cuando se discuten, y se resistan al cambio cuando se introduce. Puede que esta sea la causa real de la crisis del software.

1.2.5 Ingeniería de Software

La crisis de software no desaparecerá de la noche a la mañana. El reconocimiento de los problemas y sus causas son los primeros pasos hacia las soluciones. Luego, las soluciones deben dar asistencia práctica al que desarrolla el software, mejorar la calidad del software y finalmente permitir al "mundo del software" emparejarse con el "mundo del hardware".

No hay un método completo que solucione la crisis del software. Sin embargo puede lograrse una disciplina para el desarrollo del software -ingeniería de software-, combinando métodos para todas las fases de desarrollo del software, utilizando mejores herramientas para automatizar estos métodos; y formar una filosofía predominante de coordinación, control y administración.

La primera definición de ingeniería de software fue propuesta por Fritz Bauer en la primera conferencia importante dedicada al tema en 1968[8]:

[8] [ZEL84] pp. 1-2.

"El establecimiento y uso de principios de ingeniería robustos, orientados a obtener económicamente software que sea confiable y funcione eficientemente sobre máquinas reales".

Abarca un conjunto de tres elementos claves -métodos, herramientas y procedimientos- que facilitan al administrador controlar el proceso de desarrollo del software y suministrar a los que practiquen dicha ingeniería las bases para construir software de alta calidad de una forma productiva.

Los métodos de la ingeniería de software suministran el "cómo" construir técnicamente el software, tales métodos abarcan las tareas de: planificación y estimación de proyectos, análisis de los requerimientos del sistema y del software, diseño de estructuras de datos, arquitectura de programas, algoritmos, codificación, prueba y mantenimiento.

Las herramientas de la ingeniería de software suministran un soporte automático o semiautomático para los métodos. Actualmente existen herramientas para soportar cada uno de los métodos mencionados anteriormente. Cuando se integran las herramientas de forma que la información creada por una herramienta pueda ser usada por otra, se establece un sistema para el soporte de desarrollo de software, llamado ingeniería de software asistida por computadora (CASE: computer-aided software engineering). CASE combina el software, hardware y bases de datos de la ingeniería de software (una estructura de datos que contenga la información relevante sobre el análisis, diseño, codificación y prueba) para crear un entorno de ingeniería de software análogo al diseño/ingeniería asistido por computadora (CAD/CAE: computer-aided design/engineering), para el hardware.

Los procedimientos de la ingeniería de software unen a los métodos y herramientas y facilitan un desarrollo racional del software. Los procedimientos definen la secuencia en la que se aplican los métodos y herramientas, así como los documentos, reportes, etc. que se requieren, y los controles que ayudan a asegurar la calidad y coordinar el desarrollo.

La ingeniería de software es interdisciplinaria, utiliza las matemáticas y la ingeniería entre otras cosas para analizar y verificar los algoritmos y estimar costos, y las ciencias administrativas para definir requerimientos, supervisar al personal y monitorear el avance del proyecto. El objetivo de la ingeniería de software es la construcción de grandes y complejos sistemas de una forma rentable, atendiendo problemas tales como baja productividad, desfasamiento en la calendarización de proyectos y productos no funcionales.

1.3 Administración de Proyectos

Las actividades técnicas y administrativas son igualmente importantes para el éxito de un proyecto de programación. El administrador controla los recursos y el ambiente en que las actividades técnicas se suceden; entonces el administrador tiene la responsabilidad de asegurar que los productos se entreguen a tiempo y dentro del presupuesto estimado, además de que los productos exhiban la funcionalidad y calidad que el usuario requiere.

1.3.1 El Fracaso en los Proyectos de Software

Cuando un proyecto falla no significa que los desarrolladores son incompetentes o que los administradores de software son menos capaces de administrar a su gente y sus tareas asignadas que otros administradores en otras disciplinas; sin embargo, "se ha comprobado que la mayoría del total de fallas en un proyecto de software se debe a la falta de capacidad y dirección, lo cual casi nunca ha sido considerado como la causa"[9]. Un buen administrador de proyectos debería poseer las siguientes características:

- Fuerte motivación en los miembros del equipo del proyecto.
- Claro entendimiento de la situación
- Comprensión adecuada de las nuevas tecnologías
- Evidente capacidad en la esfera política

Y aún cuando un administrador poseyera las características anteriores, un proyecto podría fallar. ¿Porqué?, ¿se hizo un diseño muy pobre, se codificó en forma muy lenta o se introdujeron demasiados errores?. En la mayoría de los casos, simplemente fallan por llevar a cabo las expectativas originales. Muchos proyectos fracasan no porque el equipo del proyecto falle, sino porque las expectativas son infladas e irrazonables y la consecuencia de ello es el fracaso sin importar la calidad lograda, ni la cantidad de trabajo que se haya hecho.

[9] [DEM82] p. 4.

1.3.2 El significado de "Control" en un Proyecto de Software [10]

"No se puede controlar lo que no se puede medir". Es necesario aprender a organizar los proyectos de software de manera que puedan ser medidos, es decir, aprender a reconocer aquello que se debe medir y cómo hacerlo, de tal forma que los indicadores medidos lleven a lograr un control significativo del proceso de desarrollo de software.

Se controla un proyecto en la medida que uno administra para asegurar que se presenten el mínimo número de sorpresas durante el avance del proyecto. El proyecto mejor controlado, no necesariamente es aquel que hace lo mejor o hace más trabajo, sino el que cumple con sus predicciones. Cuando hay desviaciones de lo que originalmente se propuso para entregar en un proyecto, esas desviaciones deben ser mínimas y ser identificadas tempranamente.

El mérito de un administrador de proyectos es en gran parte en función a qué tan bien este administrador permanece en control. Esto conduce a la siguiente paradoja: Si un administrador A planea mediocremente desde el inicio hasta el fin de su proyecto y lo entrega exactamente así, mientras que el administrador B predice grandes maravillas y entrega maravillas, pero muy escasas, entonces A es mejor que B. Esta conclusión no es muy alentadora, sin embargo es correcta.

La operación de cualquier empresa por pequeña que sea, depende más del control que de una maravilla ocasional. Por supuesto las dos necesidades no son mutuamente exclusivas. Permanecer en control significa asegurar que los resultados cumplan con las expectativas. Esto requiere de dos cosas:

1. Se debe administrar el proyecto de tal forma que el desempeño del equipo de trabajo permanezca dentro de un estándar razonable.

2. Se debe asegurar que las expectativas originales no permitan que se desperdicie esfuerzo en un proyecto, de acuerdo al estándar.

Pero, ¿de dónde vendrá ese estándar razonable?, ¿cómo calcular las limitantes que tal estándar debe ocasionar?, ¿cómo seguir el desempeño en el tiempo?, ¿qué pasos deben seguirse para ver qué desviaciones son identificadas tempranamente en el proyecto?. La respuesta a estas preguntas no es sencilla. Cualquier intento por responder a alguna de ellas, conduce a un área en la que se debe medir.

[10] [DEM82] p. 5.

1.3.3 La Importancia de Medir.

Un administrador de proyectos es un mal estimador cuando ni siquiera puede decir en que se excedió su estimación, ya que no tiene registros del tiempo y costos invertidos en el proyecto.

Hacer una estimación de tiempo y costos para un proyecto de software es una tarea complicada y las razones por las que se hacen mal son variadas y complejas. Aún si los problemas de un mal administrador mencionados anteriormente fueran considerados como triviales, siempre habrá un factor determinante en una buena estimación: que el administrador haya aprendido a estimar.

Para la estimación de un proyecto se deben tomar en cuenta los proyectos similares al que será estimado sin olvidar aquellos proyectos jamás terminados. Ignorar las lecciones del pasado puede ser común. Por supuesto que no debería ser así, se debería ir estudiando larga y detenidamente lo que ocasiona las fallas, analizando e investigando el camino que se siguió en el proyecto que fracasó.

"La única falla imperdonable es el fallar al aprender de las fallas pasadas" [11].

Esta regla se aplica particularmente a la estimación. El no aprender nada de una estimación mal hecha es imperdonable.

En un proyecto es muy importante el análisis, diseño y codificación. Pero no es suficiente. Al ver lo que queda de un proyecto pobremente completado o abortado, uno debe ser capaz de responder a una pregunta fundamental, ¿dónde esta el tiempo y dinero invertido?. Un administrador de proyectos de software debe instituir una política de medidas y registro de avance durante un proyecto, de tal forma que se tenga la seguridad de que todos los datos cruciales estarán disponibles al final. Estas medidas junto con el registro de avance es lo que DeMarco llama **Función Métrica**[12].

He aquí la importancia de medir para obtener destreza o habilidad durante la estimación de tiempo, costo y esfuerzo. La Función Métrica debería ser capaz de proporcionar respuestas a preguntas como éstas: ¿qué tan bueno fue el diseño?, ¿ la

[11] [DEM82] p. 5.
[12] [DEM82] p. 6.

especificación?, ¿el conjunto de pruebas?, ¿el código?. ¿Cuánto tiempo y esfuerzo se perdió y porqué?, ¿se obtuvo algún beneficio al invertir en entrenamiento?.

Cualquier aspecto del proyecto que esté fuera de control, igualmente está fuera de toda medición. El control es imposible sin retroalimentación. Si se acepta esta idea, entonces uno debiera darse cuenta de lo siguiente:

- La extensión del control es una función de la precisión al medir.
- Cualquier cosa que no se mida esta fuera de control.

Para lograr mantener en control el proceso de desarrollo del software, se debe instituir una política de medición cuidadosa y efectiva[13].

[13] [SOM85] pp. 323 - 325.

1.4 Calidad del Software [14]

La garantía de calidad del software (SQA) es una "actividad de protección" que se aplica a lo largo del ciclo de desarrollo del software.

Un producto de software de alta calidad satisface las necesidades del usuario, se apega a sus especificaciones de requisitos y diseño, y presenta una ausencia de errores. Las técnicas para evaluar y mejorar la calidad del software incluyen procedimientos sistemáticos de control de calidad, recorridos, inspecciones, depuración y pruebas de unidad, de integración y de aceptación. Cada técnica tiene sus puntos fuertes y débiles y ninguna técnica es suficiente por sí misma.

Lo más difícil es lograr la ausencia de errores. Es bien sabido que casi nunca se logra que un desarrollador de software produzca un programa sin fallas. Esto es aceptado como un triste hecho de la vida.

Un defecto es una desviación entre el resultado deseado y el resultado observado. Es parecido a lo que se conoce como falla. La diferencia es que una falla es algo que se acarrea, que poco a poco se introduce en el código y enreda las cosas. Esto ciertamente no es reflejo de uno mismo; puede sucederle a cualquiera. Pero un defecto es una falla que se comete eternamente. A nadie le gustan los defectos. Las fallas pueden ser aceptadas como un hecho de la vida; es necesario que los defectos no lo sean.

El primer paso en tratar de desarrollar productos de software de gran calidad es reconocer lo que son los defectos: *fallas individuales en el desempeño.*

1.4.1 Calidad de Software y el Concepto de Desperdicio.

La calidad no se logra asignando la mayor cantidad de dinero, del total estimado para un proyecto, a la fase de mantenimiento para corregir errores. Calidad no es el estado de actividades descrito en la **Figura 1.5.**

La Figura 1.5 expresa la inversión total en una pieza dada de software desde el inicio del proyecto a través de su desarrollo, del ciclo de mantenimiento, hasta el retiro del producto. Las porciones asignadas a cada categoría son tomadas de promedios

[14] [DEM82] pp. 197 - 204.

industriales[15]. El área sombreada en la figura muestra la porción del todo que debería ser considerada *"desperdiciada"*, esto es, esfuerzo dedicado al diagnóstico y remoción de las fallas que fueron introducidas durante el proceso de desarrollo. El desperdicio representa cerca del 55% del costo total de tiempo de vida del sistema.

No es extraño que se considere casi toda la fase de pruebas dentro de la categoría de "desperdiciada". Pero, ¿porqué se asignó tanto de la actividad de mantenimiento en esta categoría?. ¿No es verdad que la mayoría del dinero gastado en mantenimiento es por nuevos "aumentos"?. Indudablemente que sí, pero para un sistema común los aumentos representan el riesgo de incorporar una gran cantidad de fallas.

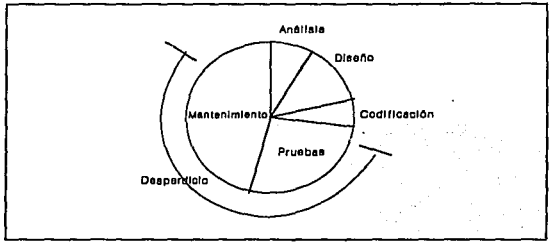

Figura 1.5 Costo de las Fallas.[16]

Muchos de los llamados aumentos son requerimientos que siempre existieron pero que no fueron tomados en cuenta durante el proceso de especificación. Esta es la razón por la que la curva del costo de mantenimiento crece en la **Figura 1.6**. Todos o la mayor parte de los cambios bajo la curva representan un error de análisis, entonces esto debería ser considerado como una fase desperdiciada.

[15] [DEM82] p. 199.
[16] [DEM82] p. 202

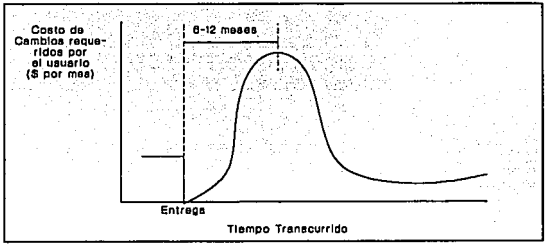

Figura 1.6 Evidencia de Fallas en el Análisis.[17]

El "desperdicio" representa el costo de fallas humanas en el proceso de desarrollo. Tales fallas humanas pueden incluir errores de codificación, prueba, diseño y análisis.

1.4.2 Medición de la Calidad a lo largo del Proyecto

La alta calidad no se puede lograr únicamente probando el código fuente. La mejor manera de minimizar el número de errores en un programa es encontrarlos y eliminarlos durante el análisis, especificación y diseño, de modo que se introduzcan pocos errores al código fuente.

Es posible que el administrador, o los programadores o cualquiera de los integrantes de un equipo de desarrollo de software no tengan una idea precisa de qué es lo que están haciendo mal, porque ellos no se preocupan por reunir datos acerca de los defectos, y el ignorar sistemáticamente la calidad del software solamente asegura que nunca mejorarán. Esto representa hacer un análisis continuo de la calidad a lo largo del proyecto hasta finalizar el sistema.

[17] [DEM82] p. 204

Capítulo II

PLANEACION DE PROYECTOS DE SOFTWARE Y SUS METRICAS

2.1 Introducción

En este capítulo se describe el modelo de planeación de la administración de sistemas de información, haciendo referencia a los principales problemas que se tratan de solucionar durante la planeación. Se propondrán diversas metodologías para implementar cada una de las cuatro etapas que componen el modelo presentado.

La planeación que se presenta aquí no es exclusivamente la requerida para un sistema grande de programación que necesita múltiples grupos para desarrollar el proyecto y poderlo terminar en un tiempo dado sin sobregiros en el presupuesto asignado. Principalmente se hará referencia a la planeación que implica estimar la necesidad de desarrollar un sistema, la viabilidad de producir tal sistema, de tal forma que se ajuste a las necesidades de una organización, y la asignación de prioridades y recursos al proceso de producción.

Por último se dará especial énfasis a la importancia de medir durante todo el proceso de desarrollo de un sistema de información, porponiendo dos modelos, uno para el proceso de recopilación de datos de cada proyecto y otro para el cálculo de las métricas de productividad.

2.2 Planeación Estratégica de la Administración de Sistemas de Información [18]

En este sección se analizará la planeación de la Administración de Sistemas de Información (ASI) y los problemas de la planeación. El modelo de cuatro etapas de planeación de la ASI consiste en planeación estratégica, análisis de requerimientos de información de la organización, asignación de recursos y planeación de proyectos. Se propondrán diversas metodologías para cada una de las cuatro etapas del modelo de planeación de la ASI. La clasificación de las metodologías se hizo de acuerdo a su uso y propósito.

El consumo de recursos excesivo durante los proyectos de desarrollo de aplicaciones, se convierte en el principal motivo para diseñar los primeros sistemas de planeación y control. Por ello se desarrollaron metodologías de desarrollo de sistemas y sistemas administradores de proyectos, que incluían divisiones en fases para cada proyecto, revisiones formales del usuario y estimaciones de tiempo. Técnicas tales como diseño estructurado y programación estructurada se utilizan para mejorar la administración del proceso de desarrollo de sistemas. Posteriormente también se da mayor importancia al hecho de completar adecuadamente los sistemas, es decir, que sean confiables y eficientes.

En la **Figura 2.2** se describe el modelo conceptual del proceso de planeación de la ASI[19].

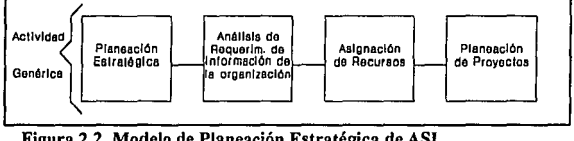

Figura 2.2 Modelo de Planeación Estratégica de ASI.

Las especificaciones del proceso de planeación de la ASI por supuesto varían de una organización a otra, es decir, las prioridades de un proyecto pueden ser determinadas por el administrador de la planeación de la ASI, por su superior, o por las políticas de la

[18] [DIC85] pp. 119 - 219
[19] [DIC85] p. 121

compañía. Una organización puede trabajar con funciones descentralizadas en la ASI y utilizar mecanismos de integración, tales como revisiones formales del avance de los proyectos y reuniones de todos los equipos que intervienen en el desarrollo para determinar el plan completo de la ASI.

2.2.1 Problemas a los que se enfrenta la Planeación de la ASI

Las principales dificultades que se tratan de resolver durante la planeación de la ASI son las siguientes[20]:

1) Alineación del plan de la ASI con las estrategias y objetivos de la organización.

2) Diseño de una arquitectura de sistemas de información, como una estructura dentro de la cual las aplicaciones serán diseñadas y desarrolladas.

3) Asignación de recursos para desarrollo y operación de los diversos sistemas de información que se realizarán.

4) Finalización de los proyectos a tiempo y de acuerdo al calendario de actividades.

5) Selección y uso de las metodologías para llevar a cabo los primeros tres puntos.

1) Alineación del Plan de la ASI con el Plan de la Organización.

El primer problema es asegurar que el proceso de planeación de la ASI identifique y seleccione las aplicaciones a desarrollar de tal forma que se ajusten a las prioridades ya establecidas por las necesidades de la organización. Sin embargo, a menudo las estrategias de la organización y sus planes no están escritos, o son formulados en términos que no son útiles para la planeación de los sistemas de información.

Por ello, resulta difícil determinar las estrategias y metas con las que el plan de desarrollo de sistemas de información deberá alinearse. Esta alineación es necesaria, ya que sin ella no se obtendrá apoyo de la organización a largo plazo.

---

[20] [DIC85] pp. 121 - 128

2) Diseño de la Arquitectura de un Sistema de Información.

El término "Arquitectura de Sistemas de Información" se refiere a toda la estructura del mismo. Esta estructura consiste de las aplicaciones para los niveles de la organización (operativo, gerencial y estratégico), así como las aplicaciones orientadas a diversas actividades administrativas (planeación, control y toma de desiciones). También incluye bases de datos y soporte de software.

Seleccionar una arquitectura para un sistema de información es difícil. Esta dificultad radica en dos procesos: la selección de la arquitectura (asumiendo un correcto y completo entendimiento de los requerimientos) y la determinación de tales requerimientos. Si los requerimientos de información de la organización están bien especificados, el número de alternativas en arquitectura será muy amplio. Si por el contrario los requerimientos son confusos y pobremente documentados, debido a que las organizaciones no tienen bien definidos sus procesos, las alternativas serán mínimas.

Si la organización no tiene bien definidos sus procesos durante la etapa de determinación de requerimientos, los administradores de un grupo o unidad, se preocuparán por definir exclusivamente sus necesidades de información, las cuales estan influenciadas por limitantes del proceso humano, causando por ejemplo, que un usuario fije su atención en el problema más reciente al cual se enfrentó como el más importante, y así llegue a conclusiones no garantizadas de un pequeño número de ocurrencias para algún evento.

Estas y otras limitantes humanas durante la definición de los requerimientos de información significan que cuando los administradores definen sus necesidades de información sin estar apoyados por un proceso sistemático para tal definición, el resultado será un conjunto de requerimientos que probablemente no sean completos o que su prioridad este determinada por sugerencias mal fundadas de los usuarios.

3) Asignación de Recursos para Desarrollo y Operación.

La distribución racional y óptima de los recursos de desarrollo entre los diversos grupos que intervienen en un proyecto es difícil, especialmente si el conjunto de aplicaciones potenciales no se ajusta al plan global de la organización y los requerimientos de cada grupo no se ajustan dentro de alguna estructura que establezca interrelación y prioridades.

4) Finalización de los Proyectos a Tiempo y de acuerdo al Calendario de Actividades.

Pocos proyectos de sistemas de información son completados dentro del calendario establecido, consecuentemente la planeación de la ASI pierde credibilidad. Los planes proyectados rara vez son exactos porque generalmente los requerimientos de tiempo y recursos son subestimados.

5) Selección de Metodologías.

Este es el mayor problema, seleccionar una metodología (especialmente aquellas para desarrollar aplicaciones y asignar recursos). En la literatura cada metodología se presenta como "la solución", pero digamos que cada una de ellas tiene una serie de circunstancias bajo las cuales es superior a las demás. No hay mucha información en la literatura que se pueda considerar como guía para hacer una selección, tomando en cuenta los diversos problemas que una organización enfrenta.

2.2.2 Planeación Estratégica de la ASI (PE)

Durante la etapa de planeación estratégica es crítico establecer las relaciones entre el plan estratégico de la ASI con el plan de la organización. Para lograr esto la organización debe:

- Apreciar los objetivos y estrategias de la misma.
- Establecer las políticas, estrategias y objetivos de la ASI.

En seguida se presenta un análisis de la única metodología aplicable a esta fase del modelo de planeación estratégica de la ASI.

Strategy Set Transformation
(Transformación de un Conjunto de Estrategias)

King[21] propone un acercamiento a la etapa estratégica de la planeación de ASI que él llama "Strategy Set Transformation". La estrategia de la organización es vista como un conjunto de información que consiste en la misión de la organización, sus objetivos y estrategias.

[21] [DIC85] pp. 128 - 130

El primer paso de esta metodología consiste en identificar y explicar el conjunto de estrategias de la organización. Si las estrategias no están escritas en ningun documento o tal documento no proporciona información suficiente para tomar desiciones, entonces se deberá construir un conjunto de estrategias. King describe el proceso de construcción del conjunto de estrategias como sigue:

1.- Descomponer en niveles la organización (dueño(s), gerentes, empleados, proveedores, clientes, acreedores, competidores, etc.)
2.- Identificar las metas de cada grupo.
3.- Identificar los propositos organizacionales y la estrategia relativa a cada grupo.

El siguiente paso es transformar las estrategias de la organización en un conjunto de estrategias de la ASI que consisten en objetivos del sistema, restricciones y principios de diseño para la estructura global de la ASI. Este proceso de transformación significa identificar los elementos estratégicos de la ASI para cada elemento dentro del conjunto de estrategias de la organización. Entonces, el analista de la información construye estructuras alternativas para la estructura global de la ASI, sujeta a los objetivos y restricciones enumerados, como el conjunto de estrategias de la ASI. Posteriormente las alternativas generales se presentan al administrador.

Esta metodología se enfoca exclusivamente a la etapa de planeación estratégica de la ASI. Para que resulte de utilidad, esta metodología requiere un entendimiento claro y conciso de las estrategias y objetivos organizacionales. Es la única metodología de la cual se sabe que está diseñada para proporcionar un enlace directo con el plan estratégico global de la organización.

2.2.3 Análisis de Requerimientos de Información de la Organización (ARIO) [22]

El primer paso en esta etapa consiste en apreciar las necesidades de información actuales y futuras para apoyar durante la toma de desiciones, y las operaciones de la organización. El segundo paso consiste en idear un plan maestro de desarrollo derivado de la arquitectura de la información, así como definir proyectos de sistemas de infonnación específicos y la prioridad de los mismos.

[22] [DIC85] pp. 130 - 135

Las metodologías listadas a continuación tienen relación con la etapa de análisis de requerimientos de la información, del modelo básico de planeación de la ASI:

- Planeación de Sistemas para Negocios.
- Factores Críticos de Exito.
- Análisis de Información para Negocios y Técnicas de Integración.

Business Systems Planning (BSP)
(Planeación de Sistemas para Negocios)

La metodología BSP fue desarrollada por IBM, se lleva a cabo por un equipo de planeación en el que intervienen usuarios y personal de la ASI; básicamente se compone de 2 fases.

Fase I

Se concentra en conocer y entender la organización, para apreciar cómo es actualmente la ASI en el negocio, especificando la red de sistemas de información requeridos para soportar la operación del negocio e identificar la prioridad de cada subsistema.

Esto se lleva a cabo mediante entrevistas realizadas a directivos y gerentes para determinar sus funciones, objetivos, toma de desiciones, problemas y necesidades de información. En general esta fase se concentra en analizar el proceso del negocio sin mirar la estructura de la organización.

Fase II

El objetivo es desarrollar un plan a largo plazo para el diseño, desarrollo e implementación de una red de sistemas de información que soporten el proceso del negocio identificado en la fase I. Este plan describe la arquitectura global de la red y define el esquema de implementación para los sistemas individuales dentro de la misma.

La metodología BSP también puede ser descrita como un proceso que elige, analiza y sintetiza la información para dar como resultado un plan:

1.- En la actividad de elección, se pretende descubrir las funciones organizacionales realizadas, desiciones tomadas y problemas de requerimientos de información.

2.- En la actividad de análisis el proceso de la organización se asocia con las necesidades de información y requerimiento de datos.

3.- La actividad de síntesis especifica aplicaciones para cubrir necesidades, requerimientos de bases de datos para soportar las aplicaciones y prioridades de desarrollo.

- Curso de Acción del BSP

1.- Formación de un comité (a nivel dirección).

2.- Preparación de un estudio preliminar (estudio de la información administrativa de la organización).

3.- Estudio exhaustivo de la información de la organización.
   3.1 Definir los problemas administrativos.
   3.2 Definir los datos del negocio.
   3.3 Definir la arquitectura de la información (establecer la relación entre los datos, la información y los sistemas de información.
4.- Analizar el soporte actual de sistemas (identificar áreas descuidadas).

5.- Entrevistas a ejecutivos.
   5.1 Verificar resultados.
   5.2 Determinar las necesidades de información.
   5.3 Definir problemas y prioridades.

6.- Definir hallazgos y conclusiones.

7.- Determinar las prioridades de la arquitectura de información.

8.- Revisar las políticas de administración de recursos de información.

9.- Desarrollar recomendaciones y un plan de acción.

10.- Reportar resultados (plan de sistemas de información).

11.- Actividades de seguimiento.

BSP utiliza un enfoque Top-Down (se analiza la organización como un todo y en base al analisis global se determinan los requerimientos de cada componente de la organización) para identificar y definir los requerimientos de información. Este enfoque puede ser efectivo en identificar los requerimientos actuales de información, pero si no se toman en cuenta los objetivos y el plan global estratégico, el plan resultante podría no ser viable a largo plazo. Otra desventaja es que para realizar entrevistas se necesita una gran cantidad de personal y tiempo para reunir y analizar la información, por lo que la síntesis de los datos dentro de un plan de sistemas de información puede resultar algo tardada.

Critical Success Factors (CSFs)
(Factores Críticos de Exito)

Una proposición de Rockart[23] afirma que los requerimientos de información de una organización pueden ser identificados a partir de factores críticos de éxito, es decir, las áreas claves para cualquier organización en las que el nivel de desempeño deba ser satisfactorio para que el negocio sobreviva y sea próspero.

Los factores críticos de éxito difieren entre las industrias y entre las firmas individuales que componen cada industria.

Como un ejemplo, Rockart cita los cuatro factores críticos de éxito (basado en la industria) de los supermercados como: una completa variedad de productos en cada tienda, correcta exposición de los productos, publicidad efectiva para atraer clientes y precios competitivos. Ya que éstas áreas determinan el éxito de un supermercado, el estado de desempeño de tales áreas debe ser medido y reportado continuamente.

De acuerdo a Rockart se identificaron tres fuentes primarias de factores críticos de éxito:

1.- Factores basados en la industria.
2.- Estrategia competitiva, posición de la industria y localización geográfica.
3.- Factores ambientales.

Los factores críticos de éxito se obtienen o definen a través de entrevistas. En una primera sesión cada directivo establece sus metas y los factores críticos de éxito que determinan las mismas.

[23] [DIC85] p. 131

Business Information Analysis and Integration Techniques (BIAIT)
(Análisis de Información de Negocios y Técnicas de Integración)

Esta metodología de análisis de información y planeación desarrollada por Burnstine[24] es diferente de la mayoría de las metodologías de planeación, las cuales tienden a usar preguntas ilimitadas a los usuarios para obtener información acerca de sus requerimientos y prioridades de información.

Burnstine, después de una extensa experimentación, obtuvo solo 7 preguntas que pueden ser usadas para determinar un conjunto normativo de requerimientos de información. De las respuestas a estas preguntas, teoricamente se pueden definir los requerimientos globales de información, independientemente de la organización, del tamaño del departamento o del producto o servicio que se ofrece.

El enfoque de esta clasificación se hace en base a "órdenes o solicitudes" y "proveedores". Proveedores son las personas, departamentos u organizaciones que responden a una orden. Ordenes son cualquier cosa que requiere de la respuesta de un proveedor, una cosa, un lugar, una técnica.

Las siete preguntas de BIAIT son las siguientes:

1.- Hace facturas a los clientes o recibe efectivo?
2.- Entrega productos o servicios inmediatamente o a futuro?
3.- Crea y mantiene el ambiente de compra de los clientes funcionalmente?
4.- Negocia los precios u opera en base a un precio fijo?
5.- Renta o vende sus productos o servicios?
6.- Retira y actualiza los productos que ofrece o actualiza sus servicios?
7.- Hace órdenes o suministra de la existencia del producto o servicio que ofrece?

Con esta metodología, la mayor parte de la planeación se basa en la elaboración de cuestionarios con preguntas específicas sobre procesos o entidades de la organización, las cuales comúnmente requieren de una respuesta binaria.

Las tres metodologías difieren en sus métodos. BSP es la metodología más completa, requiere de mucho trabajo y tiempo, pero genera información bastante extensa de los requerimientos de información de la organización. CSFs, requiere de menos trabajo

[24] [DIC85] p. 132

intenso y analiza los requerimientos de información en base a los requerimientos de los administrativos de alto nivel dentro de una organización. BIAIT es una metodología de ingeniería muy estructurada que parece ser eficiente en obtener un conjunto de requerimientos normativos a nivel operativo y gerencial, pero a nivel estratégico tal vez no sea aplicable para obtener los requerimientos de información necesarios para la toma de desiciones de alto nivel.

2.2.4 Asignación de Recursos [25]

La etapa de asignación de recursos consiste en seleccionar las plataformas de software, hardware y comunicaciones, personal y los planes financieros necesarios para llevar a cabo el plan maestro de desarrollo definido en la fase ARIO.

Las metodologías listadas a continuación tiene relación con la etapa de asignación de recursos, del modelo básico de planeación de la ASI:

- Chargeout.
- Presupuesto Basado en Cero.

Chargeout

En grandes organizaciones la función de la ASI se considera como una agencia que proporciona servicios de ASI a todas las subunidades de la organización. Se desarrolla un esquema de costos por servicio para cada unidad (velocidad del CPU, dispositivos de I/O, tiempo de programación, etc.) con el objeto de recobrar los gastos de ASI. En teoría son responsables del costo de los sistemas de información, más que de fomentar la planeación y control de los mismos.

Los sistemas basados en chargeout incluyen lineamientos, procedimientos y calendarización para dirigir la planeación, pero se enfoca a justificar los costos de nuevos sistemas de información propuestos, relativos a los beneficios. La decisión de llevar a cabo una planeación es descentralizada para los usuarios de los departamentos. Esto puede tender a limitar el alcance para los beneficios de los nuevos sistemas de información, especialmente con respecto a sistemas integrados que afectan a múltiples departamentos.

[25] [DIC85] pp. 135 - 137

Wetherbe y Dickson[26] identifican problemas asociados con la planeación basada en chargeout, incluyendo un alto costo (en términos administrativos y de proceso de cómputo), complejidad y dificultades con el desarrollo de sistemas multidepartamentales integrados.

La planeación de sistemas basados en chargeout difieren entre las organizaciones. Sin embargo, no existen mecanismos sistematicos que liguen este tipo de planeación con las estrategias y objetivos de la organización.

Zero Based Budgeting (ZBB)
(Presupuesto Basado en Cero)

Esta metodología fue desarrollada por Peter Pyhrr [27], como una alternativa para controlar el incremento del presupuesto.

Wetherbe y Dickson[28] sugieren el uso de ZBB como una planeación de la ASI, como herramienta de control y como una alternativa a los sistemas basados en chargeout. El primer paso en este proceso es reducir el presupuesto de todas las actividades de la ASI a cero. En seguida, se identifican y estructuran los sistemas de información potenciales de acuerdo a niveles de servicio.

Los beneficios que se esperan y los requerimientos de recursos de la ASI son listados para cada nivel de servicio. Se asignan prioridades a cada proyecto y en base a ello se alinean en algún nivel de servicio, se calculan los requerimientos de recursos y se asigna presupuesto.

El ZBB tiene un enfoque hacia la asignación de recursos, sin embargo, no hay un ciclo de planeación estratégica que tenga alguna liga con el proceso de planeación global de la organización.

La cantidad de tiempo de personal requerido para utilizar esta metodología es bastante. Durante el análisis de la información se debe dedicar una gran cantidad de tiempo con los usuarios para identificar los proyectos de sistemas de información y estructurarlos en niveles de servicio.

[26] [DIC85] p. 136
[27] [DIC85] pp. 136 - 137
[28] [DIC85] p. 136

Chargeout y ZBB caen dentro de la categoría de asignación de recursos. Chargeout aboga por descentralizar la desición de definir los requerimientos de la organización. Cada área que requiera de sistemas de información deberá pagar un costo por los recursos asignados. ZBB aboga por centralizar en un comité de planeación la decisión de asignar presupuesto a cada área de acuerdo al costo-beneficio de cada sistema de información.

2.2.5 Planeación de Proyectos

La planeación de proyectos consiste en evaluar el proyecto en términos de requerimientos y nivel de dificultad; se definen las actividades que deberán ejecutarse y finalmente se deben desarrollar estimaciones de tiempo y costo que serán utilizadas en la evaluación del avance del proyecto. La planeación es una fase esencial, a menudo ignorada, de un proyecto de software. La planeación podría dividirse en Definición del Producto y el Plan del Proyecto. Los componentes de una Definición de Producto son una descripción concisa del problema que se solucionará, una propuesta de los objetivos del sistema y el proyecto, la identificación de las características del usuario, una proposición de las funciones que el sistema realizará y la estrategia de solución.

Los componentes esenciales de un Plan de Proyecto incluyen el modelo del ciclo de vida que se usará, la estructura de la organización, las estimaciones preliminares de requerimientos de personal y de recursos, una estimación del costo preliminar, un calendario de desarrollo preliminar y las especificaciones para el monitoreo del proyecto y los mecanismos de control.

Las metodologías listadas a continuación tiene relación con la etapa de planeación de proyectos, del modelo básico de planeación de la ASI:

- PERT/CPM
- Diagramas de Gantt.

PERT y CPM [29]

Un proyecto se define como una combinación de actividades que deben ejecutarse en cierto orden antes de que el trabajo completo pueda terminarse. Las actividades están interrelacionadas en el sentido que algunas de ellas no pueden comenzar hasta que otras

[29] Apuntes de Teoría de Gráficas 1990. MAC - ENEP ACATLAN, UNAM.

hayan terminado. Una actividad en un proyecto usualmente se ve como un trabajo que requiere de tiempo y recursos para su terminación.

Los métodos PERT (Evaluación del Programa y Técnicas de Revisión) y CPM (Método de la Ruta Crítica) están orientados en el tiempo, en el sentido que ambos llegan a la determinación de un programa de tiempo. Aunque los dos métodos fueron desarrollados independientemente, ambos son muy similares; la diferencia más importante es que las estimaciones en el tiempo para las actividades que componen un proyecto se supusieron deterministicas en CPM y probabilisticas en PERT.

Ambas técnicas consisten en:

- Planeación : Descomponer el proyecto en las actividades que lo forman. Hacer estimaciones de tiempo para cada actividad. Construir un diagrama de red donde se muestra la dependencia entre las actividades.

- Programación : Ya teniendo el diagrama se identifican fechas de inicio y termino. Se encuentra la ruta crítica (actividades que requieren de atención especial para que el proyecto se termine a tiempo).

- Control : Análisis del diagrama de red para hacer reportes.

Gantt [30]

Los diagramas de Gantt son una técnica de planeación que como PERT y CPM define qué actividades se van a realizar, cuándo van a iniciar y cuándo se van a completar, sin embargo no muestra la dependencia secuencial de las actividades. En un diagrama de Gantt no se indica qué análisis se hizo antes iniciar el diseño del mismo, por ello es más fácil de preparar.

La ventaja de estos diagramas es que son fáciles de hacer y se puede descomponer una actividad en sub-actividades mostradas en otro diagrama de Gantt. Este sub-diagrama proporciona más detalle, permite la asignación específica de responsabilidades y permite estimar los requerimientos de tiempo.

[30] [DIC85] p. 139

PERT, CPM y Gantt son técnicas de planeación que caen dentro de la administración y planeación de proyectos. PERT se usa para predecir la duración de una actividad o donde las actividades involucran investigación y desarrollo. CPM determina la combinación de tiempo-costo, que satisfaga el tiempo programado de terminación del proyecto a un costo mínimo. Gantt es la técnica menos estructurada de las tres, define las actividades a ejecutarse en un proyecto, su inicio y su fin.

2.3.7 Lineamientos para llevar a cabo el Modelo de Planeación Estratégica de la ASI [31]

Anteriormente se identificaron los principales problemas a los que se enfrenta la planeación de la ASI como: Alineación de las Estrategias de la ASI con las Estrategias de la Organización, Desarrollo de una Arquitectura de Información, Asignación de Recursos, Finalización de los Proyectos a Tiempo y de acuerdo al Calendario de Actividades, y Selección de una Metodología para cada uno de los problemas anteriores.

Los primeros cuatro problemas corresponden a las cuatro etapas del modelo de planeación estratégica de la ASI. Dada la estructura del modelo se especificaron los métodos apropiados para cada etapa, lo que ayuda a elegir una metodología adecuada.

El modelo puede ayudar a reconocer los problemas de planeación que se presenten, así como la etapa de planeación a la que corresponden. Para llevar a cabo la planeación de la ASI la organización debe hacer una estimación de cada etapa del modelo de planeación estratégica para determinar la medida en la que cada una de ellas se ha realizado. Esto se puede hacer analizando las actividades y resultados principales de las cuatro etapas del modelo de planeación de la ASI. Después de que las necesidades de cada etapa del modelo de planeación estratégica se establezcan, se puede seleccionar una metodología apropiada.

Estimación de cada Etapa del Modelo de Planeación

Para conducir una estimación de la etapa Estratégica, una organización debe hacerse las siguientes preguntas:

[31] [DIC85] p. 141 - 144

I. ¿Existe una definición clara de los objetivos y estrategias de la organización?
   A. ¿Se ha revisado el plan estratégico de la organización?
   B. ¿Se identificaron los principales grupos de la organización así como sus objetivos?
II. ¿Se realizó un estudio de la ASI?
   A. ¿Se estimaron adecuadamente las capacidades de la ASI?
   B. ¿Se entendió el proceso actual del negocio?
   C. ¿Se definieron y documentaron las aplicaciones actuales?
   D. ¿Se conocen las técnicas utilizadas por el personal de la ASI.
III ¿Se establecieron las políticas, objetivos y estrategias de la ASI?
   A. ¿La organización de la ASI es apropiada para la organización global de la organizacion?
   B. ¿El enfoque de la tecnología de la ASI es apropiado al enfoque de la tecnología de la organización?
   C. ¿Los objetivos de asignación de recursos de la ASI son apropiados?
   D. ¿El proceso de la ASI es apropiado?

Si las respuestas a estas preguntas no satisfacen a los directivos de una organización, se debe poner en práctica una planeación estratégica. No se necesita una metodología formal para ello, pero puede ser útil la presentada anteriormente en este capítulo, Strategy Set Transformation.

Antes de llevar a cabo una estimación de la etapa ARIO la organización debe responder las siguientes preguntas:

I. ¿Se hizo una estimación adecuada de los requerimientos de información de la organización?
   A. ¿Se identificó la arquitectura global de la información?
   B. ¿Existe un buen entendimiento de las necesidades actuales de la organización?
   C. ¿Existe un buen entendimiento de las necesidades de información de la organización a futuro?
   D. ¿Se identificaron las prinicpales bases de datos así como sus relaciones?
II. ¿Existe un plan de desarrollo maestro de la ASI?
   A. ¿Se definieron los proyectos de la ASI?
   B. ¿Los proyectos se clasificaron por prioridad?

Si las respuestas a estas preguntas no satisfacen a los directivos de una organización, debe llevarse a cabo una planeación de ARIO. Las metodologías formales disponibles son BSP, CSFs, y BIAIT.

Para estimar el estado actual de la etapa de Asignación de Recursos la organización debe responder las siguientes preguntas:

I. ¿La organización tiene un plan de requerimientos de recursos?
   A. ¿Se identificaron las tendencias?
   B. ¿Existe un plan de hardware?
   C. ¿Existe un plan de Software?
   D. ¿Existe un plan de personal?
   E. ¿Existe un plan de comunicación de datos?
   F. ¿Existe un plan financiero?
II. ¿La organización tiene un procedimiento adecuado de asignación de recursos?

Si las respuestas a estas preguntas no satisfacen a los directivos de una organización, se debe implementar una planeación de Asignación de Recursos. Las metodologías disponibles son Chargeout, RI y ZBB.

Para evaluar el estado de la Planeación del Proyecto las organización debe hacerse las siguientes preguntas:

I. ¿Existe algún procedimiento para evaluar el proyecto en términos de dificultad y riesgo?
II. ¿Las actividades del proyecto usualmente se identifican adecuadamente?
III. ¿La estimación del costo del proyecto generalmente es exacta?
IV. ¿La estimación del tiempo de duración del proyecto generalmente es exacta?
V. ¿Se definieron puntos intermedios de chequeo para hacer un seguimiento del avance de los proyectos?
VI. ¿Los proyectos generalmente se terminan de acuerdo al calendario establecido?

Si las respuestas a estas preguntas no satisfacen a los directivos de una organización, se debe poner en práctica alguna técnica de Planeación de Proyectos como PERT, CPM y Gantt.

Selección de una Metodología

Con el modelo de planeación estratégica de cuatro etapas es posible preveer los posibles resultados y consecuencias de la planeación de la ASI. Esto permite seleccionar la metodología adecuada a la etapa de planeación en que una organización se encuentra. Por ejemplo puede ayudar a una organización para prevenirla de usar una metodología de Asignación de Recursos cuando en realidad es más apropiada una metodología de ARIO o de Planeación Estratégica. Sin embargo el modelo no indica cuál de las metodologías clasificadas por categoría anteriormente se debe seleccionar para cada etapa. Las organizaciones deben analizar las metodologías disponibles y escoger alguna de acuerdo a la situación o consecuencias que están enfrentando.

2.3 Métricas para la Planeación de un Proyecto de Software [32]

Una métrica es un número que se relaciona con una idea, específicamente, es una indicación mensurable de algunos aspectos cuantitativos de un sistema. Estos aspectos cuantitativos que requieren de métricas son principalmente el costo, el tiempo, la productividad y la calidad.

Las métricas son importantes, ya que los administradores tendrán la posibilidad de determinar la productividad en los diversos proyectos, así como la calidad, lo que servirá de base para hacer estimaciones más reales de nuevos proyectos, para perfeccionar los métodos y procedimientos actuales y futuros, acorde con su necesidades de mejoramiento de productividad y calidad, y sobre todo para controlar el proyecto. Por otra parte, las métricas también son útiles a los implementadores, ya que los datos recopilados de proyectos anteriores, les permitirán observar cuál ha sido su desempeño en cada uno de ellos, tomando en cuenta los diversos factores que influyeron durante el desarrollo de los mismos.

Un enfoque para controlar y mejorar el desarrollo de proyectos de software podría ser el establecer métricas para cada uno de los procesos que involucra el desarrollo de sistemas de información, entendiendose por proceso: "un processo es un conjunto de acciones requeridas para transformar en forma eficiente las necesidades de un usuario, en una solución de software efectiva"[33]; tal conjunto incluye herramientas, métodos y procedimientos para producir el producto. Por lo anterior, antes de definir las métricas lo conveniente sería seleccionar un modelo de procesos.

Diversos autores han tratado ampliamente cada uno de los modelos de procesos o modelos de ciclo de vida del software como, el de Cascada, el Incrementable, el de Prototipos, el Espiral, el Híbrido, el de Retroalimentación y otros. Un ejemplo de los componentes de un modelo de procesos sería: el Análisis de Requerimientos, Diseño Detallado y de la Arquitectura, Codificación, Pruebas de Unidad, Integración, Pruebas Formales y Operación.

Como se mencionó, algunos aspectos del software se mide por varias razones: para indicar la calidad del producto, para evaluar la productividad de la gente que desarrolla el producto, para asegurar los beneficios derivados de los métodos y herramientas de la ingeniería de software y para justificar el uso de nuevas herramientas.

[32] [DEM82] pp.49 - 61; [VON90] 553 - 620
[33] [HUM89] p. 5

En esta sección de tratarán las métricas que caen dentro del contexto de la planeación de un proyecto, es decir, las métricas de productividad, las cuales están orientadas a medir el tamaño del producto, el tiempo para llevar a cabo las actividades definidas, el esfuerzo requerido para finalizar el proyecto y el costo del mismo.

El costo y esfuerzo requerido para construir el software, el número de líneas de código y otras medidas son más fáciles de obtener que las relativas a la calidad y funcionalidad del software, o su eficiencia y facilidad de mantenimiento, las cuales sólo pueden ser medidas indirectamente.

Las métricas de productividad se centran en el rendimiento técnico y humano dentro de cada proceso o fase del ciclo de vida de la ingeniería de software mientras que las métricas de calidad proporcionan una indicación de cómo se ajusta el software a los requerimientos implícitos y explícitos del cliente. Estas últimas se tratarán de forma más amplia en el capítulo cuatro.

2.3.1 Métricas de Productividad [34]

La única manera de definir métricas, es a través de los datos que se conserven de los proyectos de desarrollo. Si se mantienen registros de los productos que se desarrollan, se puede crear una tabla (mucho más completa) de datos generales o de datos por proceso (fase de desarrollo) del proyecto, como la mostrada a continuación:

Proyecto	Tipo	Módulos	Esfuerzo	$ (10^3)	MLFE	Tiempo	Págs. Doc.	Errores	Gente
Axe-01	Científico	6	24	68	18.2	8	365	29	3
Axe-02	Científico	8	36	93	27.2	9	425	38	4
Clg-00	Educativo	3	2	58	07.8	2	128	13	1
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.

Tabla 2-1 Registro de Datos por Proyecto.

[34] [FRY91] pp. 3 - 90

La tabla anterior muestra una lista de proyectos de desarrollo de software y sus datos. Para el primer proyecto, Axe - 01, de tipo científico, con 6 módulos principales, se desarrollaron 18.2 MLFE (miles de líneas fuente entregadas), con 24 personas-mes de esfuerzo total, con un costo de $68,000.00 N.P., en 8 meses. Además de desarrollaron 365 páginas de documentación y se encontraron 29 errores después de entregárselo a cliente, dentro del primer año de utilización. Trabajaron tres personas en el desarrollo del proyecto Axe - 01.

Con estos simples datos se pueden desarrollar para cada proyecto un conjunto de sencillas métricas de productividad. Se pueden calcular promedios como por ejemplo:

Productividad = MLFE / persona-mes
Costo = $ / MLFE
Documentación = Págs. Docum. / MLFE

Conservando los resultados de estas y otras métricas, se estará formando una base histórica útil en la estimación de tiempo costo y esfuerzo de proyectos posteriores.

2.3.2 Recopilación de Datos de las Métricas

Para conseguir estimaciones precisas, se debe establecer una base histórica. Esta base histórica la conforman los datos obtenidos de proyectos de desarrollo de software anteriores y puede ser tan simple como la que se muestra en la tabla anterior. Para que los datos recopilados constituyan una verdadera ayuda durante las estimaciones de costo y esfuerzo, deben poseer los siguientes atributos:

1) Los datos deben ser absolutamente precisos, evitando suposiciones sobre proyectos anteriores.

2) Los datos deben obtenerse de tantos proyectos como sea posible

3) Las medidas tiene que ser consistentes (por ejemplo, las LFE tienen que ser interpretadas de forma consistente para todos el proyectos de los que se hayan obtenido datos)

4) El tipo de aplicación debe ser similar al trabajo que va a ser estimado.

En la **Tabla 2-2** se ilustra el modelo ejemplo para recopilar los datos y en la **Tabla 2-3** el modelo ejemplo para el cálculo de métricas. Lo ideal es recopilar los datos en forma directa, es decir, durante el desarrollo del proyecto, sin embargo, si esto no fue posible, será necesario hacer una investigación histórica de proyectos pasados para reconstruir los datos requeridos. Una vez que los datos han sido obtenidos (tal vez el paso más difícil), es posible el cálculo de métricas. Por último los datos tiene que analizarse, evaluarse y aplicarse en la estimación, ya sea para reestimar el proyecto actual de desarrollo o para uno nuevo.

Descripción	Unidades	Datos
Datos Orientados al Costo		
Costo Laboral	$ / personas/mes	$7,744.00
Año laboral	Horas-año	1560
Datos Orientados al Tamaño		
Nombre del proyecto	Alfanumérico	Axe-01
Personas en el Proy.	Personas	3
Esfuerzo	Personas-mes	39
Tiempo	Meses	13
Tamaño del Proyecto	MLFE	29.3
Docum. Técnica	Páginas	465
Docum. Usuario	Páginas	122
Número de Errores (1er. Año)	Errores	26
Esfuerzo de Mant. - Modificaciones	Personas-hora	810
Esfuerzo de Mant. - Aumentos	Personas-hora	560
Esfuerzo de Mant. - Errores	Personas-hora	740
Datos Orientados a la Función		
Número de Módulos	Módulos	8
Número de Salidas distintas	Salidas	28
Número de Interfaces del Sistema	Interfaces	1

Tabla 2-2 Ejemplo de un Modelo de Recopilación de Datos por Proyecto.

Descripción	Unidades	Datos
Costo de las LFE	$ / MLFE	$ 26,936.00
Productividad por Mes	Meses / MLFE	1.1
Productividad por Persona	MLFE / personas	9,766
Documentación	Páginas / MLFE	51
Documentación	Páginas / persona-mes	15
Defectos	Errores /MLFE	2.3
Costo de Errores	$ / errores	$ 141.00

Tabla 2-3 Ejemplo de un Modelo de Cálculo de Métricas.

Aplicando modelos similares a los mostrados en las dos tablas anteriores se puede establecer la base histórica, calcular las métricas necesarias, evaluarlas y finalmente aplicarlas en la estimación durante el desarrollo del proyecto o para estimaciones de nuevos proyectos.

A manera de conclusión diré que los lineamientos a seguir para administrar y controlar un proyecto serían:

- utilizar algún modelo para administrar los sistemas de información
- definir el modelo de procesos de ingeniería de software a seguir
- definir las métricas relativas a cada proceso
- recopilar los medidas de los datos involucrados en cada proceso
- analizar los datos para mejorar el proceso y el producto

Capítulo III

ESTIMACIÓN DE COSTOS Y ESFUERZO EN EL DESARROLLO DE SOFTWARE

3.1 Introducción

El objetivo del presente capítulo es explicar la utilización de diversas técnicas de estimación de tiempo, costos y esfuerzo, analizando los factores que intervienen en la determinación de los mismos, de tal forma que los costos no sean sobrepasados y se administren correctamente los recursos.

La primera sección hace referencia a los métodos básicos para calcular el tiempo, costo y esfuerzo de un proyecto, resaltando el valor que tiene realizar una buena planeación del proyecto, ya que la estimación del calendario de actividades, del costo de desarrollo y mantenimiento, así como el costo del personal involucrado, dependerán directamente del número y tipo de actividades en que se divida un proyecto.

En la segunda sección se describen los factores más importantes que se deberán tener en cuenta durante una estimación. Posteriormente, en la tercera sección, se analizan y explican algunas técnicas de estimación que en conjunto, contribuyen a evitar los retrasos en la entrega de un producto, así como el sobregiro en el presupuesto del proyecto.

Finalmente, se incluye una breve sección para analizar dos herramientas automáticas de estimación, las cuales se basan en dos de las técnicas incluidas en la sección tres.

3.2 Estimación de Tiempo, Costos y Esfuerzo del Software

La programación del tiempo y la estimación de costos y esfuerzo del software están relacionadas, ya que la mayor parte de los costos de un proyecto de software grande, son los relativos al pago de la gente que analiza, diseña, escribe y prueba el software. El costo del proyecto es directamente proporcional al número de personas-mes requeridas para terminar el trabajo. Desde luego hay otros costos, como los del hardware, capacitación, etc., pero estos son más fáciles de calcular. No son datos imponderables como la productividad de un analista o de un programador y la estimación del tamaño del programa fuente.

La técnica más utilizada para calcular los costos del software es estimar el tamaño del sistema de programación que se va a entregar y de ahí calcular el número de programadores-mes requeridos para construir el sistema, por medio de datos históricos de productividad, es decir, datos obtenidos de proyectos anteriores. El costo total del sistema se basa en esta cifra más los gastos varios.

El tamaño de la aplicación se calcula por medio de un estudio del sistema preliminar para establecer las partes que lo componen. Se hace una estimación del tamaño de cada unidad. Tal estimación a menudo se basa en la experiencia y en la intuición. Después se suman las estimaciones para obtener el tamaño total del sistema. Este método se basa en la suposición de que se puede predecir la productividad del programador para una parte dada del sistema. También supone que el diseño preliminar refleja con exactitud el sistema que se va a producir.

La estimación del tiempo de desarrollo para un proyecto es una las actividades más difíciles de la administración del software. La experiencia puede ser de mucha ayuda cuando se desarrolla un proyecto similar a un proyecto anterior, sin embargo para proyectos distintos a veces se usan diferentes lenguajes y metodologías de programación que complican más la tarea de estimar la planeación del tiempo del proyecto. Por tanto, la programación del tiempo es un proceso iterativo, i.e., se debe planear una estimación inicial del tiempo y conforme el proyecto avanza se da información a la persona que programa el tiempo y se modifica la estimación inicial.

La programación del tiempo del proyecto implica dividir éste en actividades y estimar cuándo se terminarán, lo cual depende en parte del número de personas que se van a dedicar al desarrollo. Bajo este concepto, se puede afirmar que un proyecto consiste de actividades particionables. Si varios individuos o grupos trabajan en un proyecto, algunas

de las actividades se podrán hacer en paralelo, por lo que el programador del tiempo del proyecto, debe coordinar tales actividades en paralelo y organizarlas de tal forma que el esfuerzo de trabajo sea óptimo.

Se podría pensar que una manera de calcular el tiempo total requerido por un proyecto es estimar el tamaño del sistema y dividirlo por la productividad esperada de una persona, a fin de obtener el número de personas-mes requeridos para terminar el proyecto. La cifra resultante sólo es aproximada, debido a las dificultades que implica estimar el tamaño del sistema y a las variaciones en la productividad de cada persona que interviene en un proyecto.

La productividad del programador se ve afectada por múltiples factores y no se puede estimar con precisión considerando el tamaño de cada unidad que se va a desarrollar. Otros factores como la experiencia del programador, la novedad y complejidad de la aplicación y las restricciones de la misma, también se deben tener en cuenta.

La estimación de la duración real de un proyecto no se puede hacer mediante la simple división del número de personas-mes por la cantidad de personas disponibles. Las razones son dos: primero, conforme el número de personas asignadas a un proyecto aumenta, la comunicación y la productividad disminuye, y segundo, algunas actividades son indivisibles, e independientemente del número de personas que trabajen en ellas, el tiempo requerido no se puede reducir. De nuevo se requiere experiencia para identificar las actividades indivisibles y para estimar la duración de un proyecto. En seguida se explicará lo anterior con más detalle.

El costo varía como el producto de número de hombres y número de meses, el tiempo no. Por tanto la relación persona-*mes* como una unidad de medición del tamaño de un trabajo es un mito engañoso[37]; esto implica que personas y meses no son intercambiables. Personas y meses son géneros intercambiables cuando una actividad puede ser particionada entre muchos trabajadores sin comunicación entre ellos. **Figura 3.1.**

[37] [BRO75] pp. 13 - 26

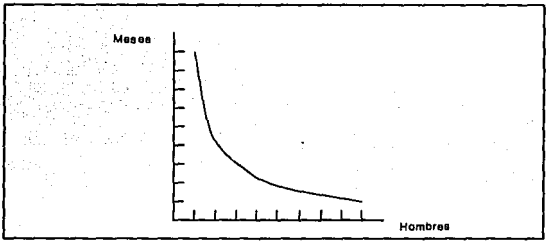

Figura 3.1 Actividades Particionables.

Cuando una tarea no puede ser particionada, el asignar más personas no tiene efecto sobre la planeación. **Figura 3.2**

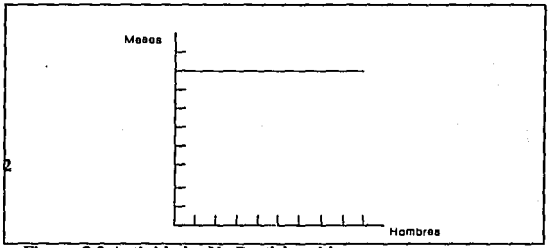

Figura 3.2 Actividades No Particionables.

En actividades que pueden ser particionadas, pero que requieren comunicación entre las subactividades, el esfuerzo de comunicación debe ser añadido a la cantidad de trabajo a realizar. **Figura 3.3**

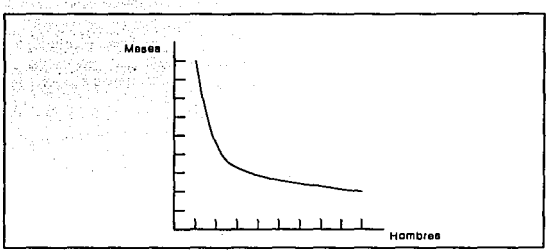

Figura 3.3 Actividades Particionables que Requieren Comunicación.

La carga de comunicación añadida consta de dos partes: adiestramiento e intercomunicación. Cada trabajador debe ser adiestrado en la tecnologia, las metas, la estrategia global y el plan de trabajo. Este adiestramiento no puede ser particionado, y esta parte del esfuerzo añadido varía linealmente con el número de trabajadores.

Si cada parte de la actividad se debe coordinar separadamente con cada una de las otras partes, el esfuerzo se incrementa como n(n-1)/2, donde n = no. de trabajadores, es decir, si tenemos una gráfica con tres componentes (o tres equipos de trabajo), donde el primer componente tiene dos vértices (o dos trabajadores), el segundo componente tiene 3 vértices y el tercer componente tiene 4 vértices, entonces, el número de líneas (de comunicación) entre los vértices de cada componente sería:

1er. componente = 2(2-1) / 2 = 1 línea de comunicación
2o. componente = 3(3-1) / 2 = 3 líneas de comunicación
3er. componente = 4(4-1) / 2 = 6 líneas de comunicación

Por tanto, tres trabajadores requieren el triple de líneas de comunicación que requieren dos; cuatro requieren seis veces la de dos. Si además se necesitan conferencias entre los equipos de trabajo (componentes), para el ejemplo anterior se tiene que el número máximo de líneas (de comunicación) sería ((n-k) (n-k+1)) / 2, con n = No. de trabajadores y k = No. de equipos de trabajo, por lo tanto,

((9-3) (10)) / 2 = 30 líneas de comunicación

El esfuerzo de comunicación añadido, puede contrarrestar completamente la división de la actividad original y conducimos a la situación de la **Figura 3.4.**

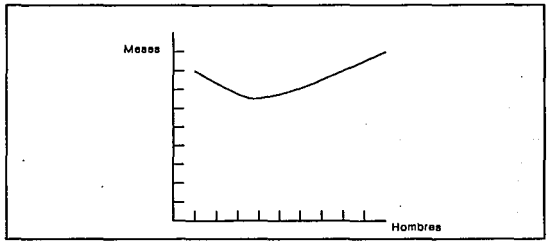

Figura 3.4 Actividades con Interrelaciones Complejas.

Con esto se establece que:

Añadir mano de obra a un proyecto de software retrasado lo retrasa aún más. (Ley de Brooks)[38].

El número de meses de un proyecto depende de sus restricciones secuenciales. El máximo número de personas depende del número de actividades independientes. De estas dos cantidades se pueden derivar los planes empleando menos personas y más meses cuando las actividades requieren de gran comunicación o mas personas y menos meses si las actividades se pueden dividir completamente si depender entre ellas, es decir, si no se requiere de gran comunicación.

Cada una de las técnicas o modelos de estimación de costos que existen actualmente, son tan buenos como lo sean los datos históricos que se utilizan para hacer la estimación. Si no existen datos históricos, las estimaciones proporcionadas, por la evaluación del costo, no tendrán un grado de riesgo aceptable, lo cual puede determinar la diferencia entre beneficios y pérdidas.

[38] [BRO75] p. 25

3.3 Factores que Influyen en el Costo de un Producto de Software

Los factores que determinan la precisión de una estimación se dividen en cuatro clases de acuerdo a Boehm[39]. Las siglas que aparecen entre paréntesis al final de cada uno de los siguientes factores serán utilizadas en el resto del capítulo para hacer más sencilla la referencia a cada uno de los mismos:

Atributos de Producto
- Confiabilidad Requerida del Software (CONF)
- Tamaño de la Base de Datos (DATO)
- Complejidad del Producto (CPLJ)

Características de la Máquina
- Limitantes en el Tiempo de Ejecución (TIEM)
- Restricciones de Almacenamiento en Memoria Principal (ALMC)
- Volatilidad en la Máquina Virtual (VIRT)
- Tiempo de Entrega de Programas (ENTR)

Características del Personal
- Capacidad de los Analistas (CAPA)
- Capacidad de los Programadores (CAPP)
- Experiencia en Programas de Aplicación (EXPA)
- Experiencia en Máquinas Virtuales (EXPV)
- Experiencia en Lenguajes de Programación (EXPL)

Características de Proyecto
- Uso de Técnicas Modernas de Programación (TMOD)
- Uso de Herramientas de Programación (HERR)
- Tiempo Requerido para el Desarrollo (DESA)

En seguida se presenta una descripción de cada uno de estos factores.

[39] [BOE81] pp. 371 - 473; [FAI85] pp. 66 - 75

Confiabilidad Requerida del Software (CONF)

El nivel de confiabilidad deseado de un producto de software debe establecerse durante la fase de planeación al considerar el costo de las fallas del programa. Se mide en una escala desde muy baja, en donde las fallas sólo ocasionan al usuario pequeñas inconveniencias; pasando por la nominal, en donde una falla originaría pérdidas moderadas y recuperables; hasta muy alta, donde las fallas implican un riesgo de vida humana.

Tamaño de la Base de Datos (DATO)

Se considera bajo, cuando el tamaño de la base de datos (en bytes) es inferior a 10 veces el número de instrucciones fuente entregadas (IFE); nominal, cuando el tamaño de la base de datos es entre 10 y 100 veces el tamaño del sistema; y muy alto, si la base de datos es superior a 1000 veces el programa.

Complejidad del Producto (CPLJ)

Varía en una escala que va desde muy baja hasta extra alta. De acuerdo a su nivel de complejidad los productos de software se dividen en tres clases:

1.- Programas de Aplicaciones (sistemas que desarrolla un programador basándose en los programas de apoyo). Tercer nivel de complejidad.

2.- Programas de Apoyo (utilerías, compiladores, ligadores, editores, depuradores, etc.). Segundo nivel de complejidad.

3.- Programas de Sistemas (sistemas operativos, DBM's, L4G). El mayor grado de complejidad.

Boehm considera a las tres clases de productos de software de la siguiente manera:

Programas de Aplicaciones	=	Modo Orgánico
Programas de Apoyo	=	Modo Semiseparado ó Semiindependiente
Programas de Sistemas	=	Modo Incrustado ó Incorporado

Limitantes en el Tiempo de Ejecución (TIEM)

Varían desde nominal hasta extra alta. Una clasificación nominal significa que se usa menos del 50 % del tiempo que se tiene o que fue asignado durante la planeación, para ejecutar o realizar las tareas de cada fase de desarrollo y una clasificación extra alta, que se debe usar el 95% del tiempo disponible para realizar las actividades de cada fase de desarrollo.

Restricciones de Almacenamiento en Memoria Principal (ALMC)

Varía de la misma manera que el tiempo de ejecución. El valor nominal significa que se usa menos de la mitad de la memoria de almacenamiento principal (disco duro de una computadora) disponible y una clasificación extra , que se debe usar el 95 % de la memoria de almacenamiento disponible.

Volatilidad de la Máquina Virtual (VIRT)

La máquina virtual es la combinación de hardware y software sobre la que se construye el producto de software. Una clasificación baja en este factor significa que sólo se modifica ocasionalmente (una vez al año), una clasificación nominal implica cambios mayores cada seis meses y una clasificación muy alta sugiere que la máquina virtual cambiará cada dos semanas.

Tiempo de Entrega de Programas (ENTR)

Este factor representa el tiempo de respuesta de la computadora, en horas. Varía desde bajo que implica el desarrollo de sistema interactivos o en línea (software que se hace de una forma conversacional entre el desarrollador y su terminal, con un tiempo de respuesta para operaciones simples entre un rango de 0 - 5 seg.), hasta muy alto, que significa que el tiempo de entrega de los programas es de más de 12 horas.

Atributos de Personal (*CAPA, CAPP, EXPA, EXPV, EXPL*)

Se consideran cinco atributos que reflejan la experiencia y capacidades del personal de desarrollo que trabaja en el proyecto. Estos atributos son ***Capacidad de los Analistas (CAPA), Capacidad de los Programadores (CAPP), Experiencia en Programas de Aplicación (EXPA), Experiencia en Máquinas Virtuales (EXPV), Experiencia en Lenguajes de Programación (EXPL).*** Todos ellos varían de muy bajo, que significa poca o ninguna experiencia; valor nominal, que implica al menos un año de experiencia; hasta muy alto, que significa más de tres años de experiencia.

Técnicas Modernas de Programación (TMOD)

Se refiere a lenguajes de cuarta generación y programación estructurada, análisis diseño descendente (top-down), bibliotecas de programas de apoyo, etc. Este atributo varía en una escala que va desde muy bajo, es decir, que no se utilizan tales técnicas; pasa por el valor nominal, que implica un uso esporádico y llega a muy alto, que significa que el uso de las técnicas modernas de programación es rutina y que el personal tiene experiencia en ellas.

Uso de Herramientas de Programación (HERR)

La disponibilidad de herramientas de software puede tener un efecto significativo en el esfuerzo que se necesita para desarrollar un sistema de software. Varía de muy bajo hasta extra alto. En seguida se muestra una lista de las categorías de acuerdo a Boehm:

Extra Alta	El mejor ambiente de desarrollo disponible, (CASE).
Muy Alta	Herramientas de Análisis, Diseño y Programación.
Alta	Herramientas de Mainframes, (depuradores, ligadores, editores, compiladores).
Nominal	Herramientas Básicas MINI/MAXI.
Baja	Herramientas MINI.
Muy Baja	Herramientas Básicas PC.

Tiempo Requerido para el Desarrollo (DESA)

Un valor muy bajo para este atributo significa que las actividades componentes de un proyecto deberán completarse en muy poco tiempo, mientras que un valor alto implica un calendario de actividades holgado o a largo plazo.

Si por alguna causa un proyecto tuviera que ser entregado antes de los previsto, por ejemplo, reduciendo el tiempo de entrega de 10 meses a 7.5 meses, lo que provocaría tal reducción de tiempo, sería un aumento en el esfuerzo. Por otra parte, si el tiempo aumenta, el nivel de esfuerzo disminuye hasta cierto punto, ya que se comprobó que si el tiempo es demasiado, el esfuerzo igualmente aumenta.

Los multiplicadores asociados con cada uno de estos 15 atributos se obtuvieron con el examen de datos de 63 proyectos de programación y mediante la técnica DELPHI (analizada posteriormente en la sección 3.4.3.2) entre un grupo de expertos de programación[40]. Los factores de esfuerzo obtenidos se muestran en la **Tabla 3-1** . **MB** significa muy bajo, **B** bajo, **N** nominal, **A** alto. **MA** muy alto y **EA** extra alto. Estos factores de esfuerzo son utilizados por el modelo de estimación de costos COCOMO (analizado más adelante en la sección 3.4.1.1).

[40] [FAI85] pp. 72 - 72

Atributo	Valores					
	MB	B	N	A	MA	EA
CONF	0.75	0.88	1.00	1.15	1.40	--
DATO	--	0.94	1.00	1.08	1.16	--
CPLJ	0.70	0.85	1.00	1.15	1.30	1.65
TIEM	--	--	1.00	1.11	1.30	1.66
ALMC	--	--	1.00	1.06	1.21	1.56
VIRT	--	0.87	1.00	1.15	1.30	--
ENTR	--	0.87	1.00	1.07	0.15	--
CAPA	1.46	1.19	1.00	0.86	0.71	--
CAPA	1.29	1.13	1.00	0.91	0.82	--
CAPP	1.42	1.17	1.00	0.86	0.70	--
EXPV	1.21	1.10	1.00	0.90	--	--
EXPL	1.14	1.07	1.00	0.95	--	--
TMOD	1.24	1.10	1.00	0.91	0.82	--
HERR	1.24	1.10	1.00	0.91	0.83	--
PROG	1.23	1.08	1.00	1.04	1.10	--

Tabla 3-1 Multiplicadores de Atributos de un Proyecto.

El atributo TMOD depende del tamaño del producto entregado en MLFE:

Tamaño del Producto (MLFE)	Valoración de TMOD				
	MB	B	N	A	MA
2	1.25	1.12	1.00	0.90	0.81
8	1.30	1.14	1.00	0.88	0.77
32	1.35	1.16	1.00	0.86	0.74
128	1.40	1.18	1.00	0.85	0.72
512	1.45	1.20	1.00	0.84	0.70

Tabla 3-2 Valores del Atributo TMOD.

3.4 Modelos de Estimación de Costos y Esfuerzo

En general se manejan dos tipos de estimación de costos del software[41]:

1.- TOP - DOWN
2.- BOTTOM - UP

1.- Top - Down, primero se enfoca a los costos del sistema en forma integral, es decir, se analiza el sistema como un todo, sin tomar en cuenta los módulos o subsistemas que lo compongan, y por otra parte también se ocupa de los costos de manejo de configuración, del control de calidad, de la integración del sistema, de entrenamientos, etc. Los costos de personal se estiman en base a los costos de proyectos anteriores similares.

2.- Bottom - Up, primero estima el costo del desarrollo de cada módulo o subsistema, posteriormente tales costos se suman para obtener un costo total. Se hace énfasis en los costos asociados con el desarrollo independiente de cada módulo, aunque puede fallar al no considerar los costos de control de calidad y del manejo de la configuración.

En la práctica ambas técnicas se deben desarrollar y comparar para que se eliminen las diferencias obtenidas. Boehm afirma que, para proyectos grandes, se deben usar en paralelo varias técnicas de estimación de costos. Si éstas predicen costos muy distintos, significa que no se dispone de suficiente información sobre ellos. En seguida se presenta una descripción de diversos modelos o técnicas de estimación de tiempo, costo y esfuerzo.

3.4.1 Modelos de Estimación Empíricos

Este tipo de modelos se puede construir por medio del análisis de los costos de proyectos terminados y establecer una relación entre los costos con el tamaño del proyecto, el número de programadores, etc.

[41] [FAI85] p. 75

3.4.1.1 Modelo COCOMO (Modelo Constructivo de Costos) [42]

El modelo constructivo de costos COCOMO, desarrollado por Boehm, es un modelo de costos por algoritmos o módulos, ya que los costos se estiman mediante la suma de los costos de cada uno de los módulos o subsistemas que conforman al sistema, de modo que esta técnica es del tipo jerárquica hacia arriba (bottom-up).

El COCOMO utiliza dos ecuaciones para proporcionar los valores nominales de la estimación de meses de programador y del calendario de desarrollo para cada unidad de trabajo, basándose en el número de líneas de código fuente entregadas de cada unidad; después se usan los factores multiplicadores de la tabla 3-1 para ajustar la estimación, de acuerdo con los atributos del producto, de la máquina, del personal y del proyecto.

Ecuación para calcular el número de personas-mes.

Fórmula: PM (Personas-Mes)

Aplicaciones	PM = (2.4) (MLFE) ^ 1.05
Apoyo	PM = (3.0) (MLFE) ^ 1.12
Sistemas	PM = (3.6) (MLFE) ^ 1.20

donde MLFE es Miles de Líneas Fuente Entregadas

Ecuación para calcular el tiempo de desarrollo de software.

Fórmula: TDES (Tiempo de Desarrollo)

Aplicaciones	TDES = (2.5) (PM) ^ 0.38
Apoyo	TDES = (2.5) (PM) ^ 0.35
Sistemas	TDES = (2.5) (PM) ^ 0.32

[42] [BOE81] pp. 57 - 156

Ejemplo:

Suponer que se calculó que el tamaño estimado de un programa del tipo apoyo es de 8000 líneas de código. Según la ecuación de esfuerzo, el número de personas-mes requeridas para este programa es:

PM = 3.0 (8) ^ 1.12 = 30.8 p.m.

Según la ecuación de programación del tiempo, el período requerido para terminar el programa es:

TDES = 2.5 (30.8) ^ 0.35 = 08.3 meses

La cantidad de personal necesario para terminar el proyecto en el tiempo requerido es:

NP = PM/TDES = 30.8 / 08.3 = 3.7 = 4 personas

donde NP es número de personas, divididas entre el tiempo de desarrollo.

El tiempo necesario para completar el proyecto es una función del esfuerzo total requerido por el proyecto y no una función del número de ingenieros de software que trabajan en él. Esto confirma la noción de que añadir más gente a un proyecto que sobrepasó el tiempo programado, es poco probable que ayude a recuperar el tiempo perdido.

En seguida una aplicación de los factores de esfuerzo, incluídos en la tabla 3-1, a las ecuaciones empíricas desarrolladas por Boehm.

Ejemplo:

Considerar que las ecuaciones empíricas produjeron los siguientes datos: un esfuerzo de 45 p.m. para desarrollar un sistema del tipo "incorporado" de software en una microcomputadora con un procesador de 16 bits que trabaja a 8 MHz y tiene 1Mb de memoria principal. Todos los multiplicadores de esfuerzo para el modelo de COCOMO tienen valores nominales, exceptuando los siguientes:

CONF 1.15 ALMC 1.21 TIEM 1.10 INST 1.10

Utilizando el COCOMO para estimar el esfuerzo de desarrollo, se obtiene la siguiente cifra:

PM = 45 * 1.15 * 1.21 * 1.10 * 1.10 = 76 p.m.

Si se supone que el costo por ingeniero de software es de N$ 6,000.00 al mes el costo total de desarrollo de este proyecto es:

C = 76 * 6000 = 456 000 nuevos pesos

A partir de los multiplicadores de esfuerzo, se puede observar que las restricciones del hardware tienen un efecto significativo sobre los costos totales del software. Suponer ahora que se hizo una propuesta para usar un procesador compatible de 16 MHz y 2Mb de memoria. Ya que esto requiere del desarrollo de interfaces especiales, el costo total del hardware adicional es de N$ 30,000.00. Además, esto podría significar que el atributo HERR fuera muy bajo (MB), en lugar de bajo (B), por otra parte, como el hardware es compatible, los atributos del personal no se alteran.

Con este nuevo procesador, los atributos TIEM y ALMC se reducen a valores nominales, así que el costo predicho del software es:

C = 45 * 1.24 * 1.15 * 6000 = 384 000 nuevos pesos

Por lo tanto se puede lograr un ahorro de N$ 42,000.00 si se invierte en un hardware superior.

Este ejemplo ilustra cómo incluso un modelo simple de costos puede ayudar a la administración a tomar desiciones.

Una de las principales ventajas de utilizar una técnica por módulos para la estimación de costos es que proporciona bases cuantitativas para tomar decisiones administrativas. Esto es muy valioso, aunque las estimaciones de los costos reales producidas por el modelo sean sólo aproximadas.

La implementación de esta técnica está en el producto COSTAR, el cual será descrito brevemente en la sección 3.5.1

3.4.1.2 Modelo de Putnam [43]

El modelo de estimación de Putnam, es utilizado para estimar el nivel de esfuerzo o de contratación de personal, a lo largo de un proyecto. Este modelo se basa en los estudios hechos por Lord Rayleigh y Norden.

La interpretación de la distribución del esfuerzo de proyectos de Putnam se muestra en la siguiente figura.

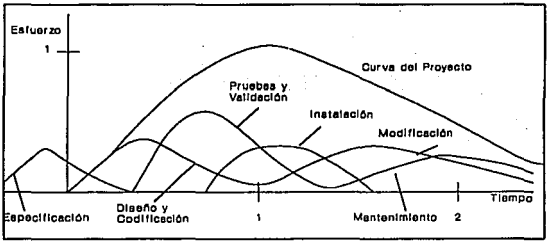

Figura 3.6 Interpretación de la Distribución del Esfuerzo de acuerdo a Putnam. (El Tiempo=1 ó 2 esta dado en años)

Las curvas mostradas en la figura 3.6 se basan en la descripción hecha por Rayleigh y en los datos empíricos del desarrollo de diversos sistemas, recopilados por Norden. En la **Figura 3.7** aparece la curva de Rayleigh, con Td = tiempo en que la curva alcanza su valor máximo y K = área total bajo la curva, que representa el esfuerzo total requerido por el proyecto. Por tanto, la distribución de esfuerzo mostrada en la figura 3.6 se denomina "Curva de Rayleigh - Norden". La fórmula de Rayleigh es $E = (K / Td^2)\ t\ e\ ^{\wedge}\ -(t^2 / 2Td^2)$.

[43] [PRE87] pp. 126 - 128; [FAI85] pp. 83 - 86.

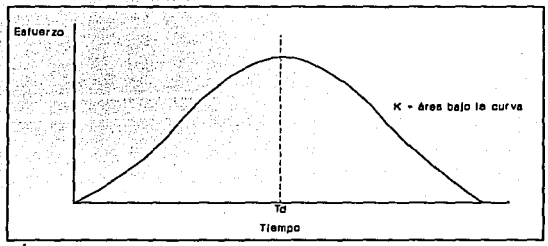

Figura 3.7 Curva de Rayleigh

Putnam estudió 50 proyectos de programación del ejército estadounidense y 150 más, para determinar la forma de utilizar la curva de Rayleigh para describir un ciclo de vida de desarrollo; observó que en muchos proyectos, el valor máximo del tiempo en la curva de Rayleigh, Td, se da en el momento de pruebas del sistema y entrega del proyecto, donde el área bajo la curva, en cualquier intervalo, representa el número de personal de tiempo completo utilizado en tal intervalo. Con esto obtuvo una ecuación que relaciona el número de líneas de código esperadas, L, con el esfuerzo, K, y el tiempo de desarrollo, Td :

$$L = C_k K^{1/3} Td^{4/3}$$

donde C_k es *una constante del estado de la tecnología* y refleja las "restricciones que frenan el progreso del programador" . Valores típicos pueden ser:

C_k = 2.0 ambiente pobre de desarrollo (por ejemplo, sin metodología, sin documentación, con revisiones escasas)
C_k = 8.0 buen ambiente de desarrollo (por ejemplo, buena metodología, documentación y revisiones adecuadas)
C_k = 11.0 ambiente excelente de desarrollo (por ejemplo, herramientas y técnicas automáticas)

Ahora para obtener una ecuación del esfuerzo de desarrollo o nivel de contratación, se readapta la ecuación anterior, obteniendo la siguiente expresión:

$$K = L^3 / (C_k^3 Td^4)$$

donde K es el esfuerzo empleado (en personas-año) durante el ciclo de vida completo de desarrollo y mantenimiento de software, y Td es el tiempo de desarrollo en años.

La implementación de esta técnica se encuentra en el producto SLIM, el cual será descrito brevemente en la sección 3.5.2.

3.4.1.3 Interpretación de Boehm de la Ecuación de Rayleigh

Boehm considera que la ecuación de Rayleigh es un estimador bastante exacto de la cantidad de personal requerido para el ciclo de vida de desarrollo, desde el diseño, hasta la implantación y pruebas del sistema, siempre y cuando se use la porción de la curva entre 0.3 Td y 1.7 Td[44]. La ecuación de Rayleigh formará la siguiente expresión:

$$PR = PM\ (\ W / X\ )\ e^{-(Y/Z)}$$

donde $W = 0.15\ TDES + 0.7\ T$, $X = 0.25(TDES)^2$, $Y = (0.15\ TDES + 0.7)^2$, $Z = 0.5(TDES)^2$, PM es el número estimado de meses de programador y TDES el tiempo estimado de desarrollo.

Con PM y TDES, la cantidad de personal requerido, PR, en el tiempo t, que se encuentra en el intervalo 0.3 Td a 1.7 Td, se puede obtener de la fórmula anterior.

3.4.2 Costos de Mantenimiento de Software [45]

Los costos de mantenimiento son difíciles de estimar con anticipación. Estudios hechos por Boehm muestran que los costos de mantenimiento son los más caros del desarrollo y uso del sistema, ya que estos costos fueron subestimados durante la planeación. Como ejemplo Boehm estimó que un sistema de la Fuerza Aérea de E.U. costó 30 dólares por instrucción durante el desarrollo, y 4000 dólares por instrucción de mantenimiento durante su tiempo de vida. Este sistema de control tenía una codificación muy compacta y tal vez su principal requisito era el rendimiento, lo que en muchas ocasiones, se logra sacrificando la comprensión, y por lo tanto la mantenibilidad de un programa.

---

[44] [FAI85] p. 84

[45] [FAI85] pp. 86 - 89; [SOM85] pp. 226 - 250

Existen otros factores que influyen en los costos de mantenimiento, como el tipo de sistema, la rotación del personal, la dependencia del programa de su ambiente externo, la estabilidad del hardware, la interdependencia de los módulos, el lenguaje de programación, las pruebas que se hagan del sistema y la calidad y cantidad de documentación del programa.

El mantenimiento de software se podría definir como, la modificación de un programa con el fin de corregir errores y proporcionar nuevas posibilidades, ya que hay tres categorías de mantenimiento de software: de perfeccionamiento, adaptativo y correctivo.

El mantenimiento de perfeccionamiento comprende cambios solicitados por el usuario (mayor eficiencia, mejor documentación, nuevos módulos), el mantenimiento adaptativo de debe a cambios en el ambiente que se desarrolla el sistema (información de entrada, equipo, sistema operativo, etc.) y el mantenimiento correctivo se refiere a la corrección de errores del sistema no descubiertos.

De acuerdo a los estudios hechos por Lientz y Swanson[46], se establece una regla útil para distribuir el esfuerzo de las actividades de mantenimiento, asignar 60% del tiempo a perfeccionamiento, 20% a adaptación y 20% a la depuración o corrección de problemas.

3.4.2.1 Estimación de los Costos de Mantenimiento

Lientz y Swanson[47] determinaron que un programador común en el ambiente de procesamiento de datos puede mantener hasta 32K instrucciones. En programas científicos como por ejemplo proyectos de aeronáutica, el número de líneas varía entre 8K y 10K. Entonces, la estimación del personal requerido en el mantenimiento de un proyecto de programación, se puede obtener dividiendo el número estimado de líneas de código que se mantendrán, entre el total de líneas que puede mantener un programador, es decir,

$$PRM = MLFE / 32K$$

Un ejemplo sería, estimar el número de personas que se necesitan para el mantenimiento de un programa con 64K líneas de código, donde cada programador puede mantener hasta 32K líneas de código. Aplicando la fórmula anterior se tiene,

[46] [FAI85] p. 86
[47] [FAI85] pp. 87 - 89

PMR = 64K / 32K = 2 personas para mantenimiento.

Por otra parte, Boehm calcula el costo de mantenimiento, en función de un cociente de actividad, que se define como el número de líneas fuentes que serán agregadas o modificadas, divididas entre el número total de líneas entregadas:

$$ACT = (LCF_{agregadas} + LCF_{modificadas}) / LFE_{totales}$$

El modelo de estimación de mantenimiento de Boehm utiliza el cociente de actividad y el esfuerzo de desarrollo por personas-mes (PM) o el esfuerzo de desarrollo por persona (NP), para determinar el esfuerzo requerido para el mantenimeinto de software. Este se calcula de la siguiente manera:

PMR = ACT * PM ó
PMR = ACT * NP

Por ejemplo, si se tiene un proyecto de 90K LCF del tipo aplicaciones, y se requiere aplicar un mantenimiento correctivo aproximado de 5K LCF y un mantenimiento de perfeccionamiento de 20 K LCF, el esfuerzo de mantenimiento básico sería:

PM = (2.4) (90) ^ 1.05 = 270.5 p.m.
TDES = (2.5) (PM) ^ 0.38 = 20.99 meses
NP = PM / TDES = 12.89 personas
ACT = (5 + 20) / 90 = 0.28

PMR = ACT * NP = 0.28 * 12.89 = 3.6092
= 4 personas para mant.

De acuerdo a Boehm, esta fórmula proporciona una aproximación general a los costos de mantenimiento, la cual sirve para calcular una cifra más precisa, basado en que los costos del software y en particular los costos de mantenimiento, dependen del tamaño del software, además de otros factores como la confiabilidad requerida, el tipo de software, las restricciones de hardware y la capacidad del personal de desarrollo entre otros. Utilizando los factores multiplicativos de la tabla 3-1 (que se encuentra en la seccion 3.2, donde se mencionan los principales factores que afectan a los costos del software), se puede precisar el costo de mantenimiento, seleccionando los multiplicadores adecuados al proyecto de desarrollo, y multiplicando el costo de mantenimiento por cada multiplicador, para dar una estimación revisada de los costos.

Por ejemplo, si se supone que en el sistema del ejemplo anterior los factores que tuvieron más efecto en los costos de mantenimiento fueron: la confiabilidad (CONF), experiencia en el lenguaje y en aplicaciones (EXPL y EXPA), que además fue alta, y el uso de técnicas modernas de programación para el desarrollo (TMOD), que fue muy alto. A partir de la tabla 3-1 de Boehm, se obtienen los siguientes multiplicadores para cada factor:

$$CONF = 1.10, \quad EXPA = 0.91, \quad EXPL = 0.95, \quad TMOD = 0.72$$

Aplicando estos valores a la estimación inicial, se obtiene el esfuerzo de mantenimeinto requerido revisado en la siguiente forma:

$$EMR = 4 * 1.10 * 0.91 * 0.95 * 0.72 = 2.73 = 3 \text{ personas}$$

La reducción en los costos estimados, se debió en parte a que se dispone de personal experimentado para el trabajo de mantenimeinto y a la utilización de técnicas modernas de programación durante el desarrollo del software. Si no se usaran técnicas modernas de programación y los otros factores permanecieran inalterados, la estimación de los costos sería la siguiente:

$$EMR = 4 * 1.10 * 0.91 * 0.95 * 1.40 = 5.32 = 5 \text{ personas}$$

Ya que Boehm estima que el uso de técnicas modernas de programación reduce los costos de desarrollo, la cifra de 5 personas, es una subestimación, porque los costos iniciales de desarrollo serían mayores si no se emplearan técnicas modernas.

Con esto se ilustra la importancia de adquirir experiencia para lograr que las estimaciones se apeguen lo más posible a la realidad. Las distintas partes de un sistema caerán dentro de las diferentes categorías de mantenimiento, por lo que para calcular una cifra más exacta, lo mejor sería estimar el esfuerzo de desarrollo inicial y el cociente de actividad para cada componente del sistema. Entonces, el esfuerzo de mantenimiento total, sería la suma de los esfuerzos de los componentes individuales.

Las fórmulas de estimación de mantenimiento de Boehm forman parte del modelo de COCOMO y permiten tomar desiciones de la misma manera que las fórmulas de estimación de esfuerzo de desarrollo, del COCOMO, lo hacen.

Por ejemplo, suponer que para el ejemplo anterior se decidió ahorrar dinero utilizando personal con menos experiencia para el mantenimeinto de software, con un costo de N$ 4500.00 por mes, comparado con N$ 5500.00 para los ingenieros de software con experiencia.

El emplear personal con experiencia origina los siguientes costos de mantenimiento:

$$CPM = 3 * 5500.00 = 16500.00$$

Si se utiliza personal sin experiencia, el esfuerzo requerido para el mantenimiento aumenta, porque los multiplicadores de la exprericncia del personal cambian:

$$EMR = 4 * 1.10 * 1.07 * 1.13 * 0.72 = 3.83 = 4 \text{ personas}$$

Entonces, los costos totales de el empleo de personal sin experiencia son:

$$CPM = 4 * 4500.00 = 18000.00$$

Por lo tanto, es más caro utilizar personal sin experiencia que ingenieros experimentados. Aunque los costos sean aproximados, el modelo demuestra como las decisiones administrativas afectan a los costos.

3.4.3 Otros Modelos de Estimación

En esta sección se tratarán las técnicas de estimación de costos más utilizadas, Juicio Experto, Delphi y Estructuras de División del Trabajo.

3.4.3.1 Juicio Experto[48]

Esta técnica es del tipo jerárquica hacia abajo y es la técnica más utilizada para la estimación de costos; se basa en la experiencia, en los conocimientos de administración, sistemas e ingeniería y en el sentido comercial de una o más personas dentro de la organización.

[48] [FA185] pp. 75 - 76

Por ejemplo, un experto podría llegar a una estimación de costos de la siguiente manera: El sistema que se desarrollará es de control de procesos, similar al que se elaboró el año pasado en un periodo de 10 meses, con un costo de mil nuevos pesos. El proyecto no produjo gran ganacia, pero si hubo ciertas utilidades, de modo que no se requieren modificaciones del proyecto. El nuevo sistema tiene funciones parecidas de control, pero requiere de un 25% más de este tipo de actividades; entonces, se le aumentarán costos y el tiempo de desarrollo en ese mismo porcentaje.

El sistema anterior fue el primero en su tipo que se desarrolló; además en esta ocasión se empleará el mismo equipo de cómputo y los mismos equipos de sensores y controles; así mismo se ocuparán muchos de los programadores que trabajaron en el otro proyecto, por lo que es posible reducir la estimación en un 20 %. Más aún, se puede volver a utilizar gran parte del código elemental del producto anterior, con lo que se reduce la estimación en otro 25%. El resultado final de estos ajustes da un 20% menos en la estimación, por lo que costará 800 dólares en 8 meses de desarrollo. Como se sabe que el cliente está dispuesto a pagar en concurso hasta un millón en un plazo de 10 meses, se puede dejar un márgen de seguridad y cotizar en 850 dólares en un periodo de nueve meses de desarrollo.

La desventaja de esta técnica es que el experto puede confiarse de que el proyecto sea similar a uno anterior, ya que podría haber olvidado algunos factores que ocasionen que el sistema nuevo sea considerablemente diferente, o quizás el experto que realiza la estimación no tenga experiencia en este tipo de proyectos.

Para compensar tales factores, los grupos de expertos tratan de llegar a un consenso de la estimación, lo cual tiende a minimizar las fallas individuales y la falta de familiaridad en proyectos particulares neutralizando las tendencia personales y el (posible inconsciente) deseo de ganar un contrato a través de una estimación optimista. Ahora bien, la estimación en grupo tiene un desventaja, el efecto que la dinámica interpersonal del grupo pueda tener en cada uno de los individuos; los miembros de un grupo pueden ser inocentes con respecto a factores de tipo político, a la presencia de alguna autoridad dentro del grupo, o al dominio de un miembro del grupo con una fuerte personalidad. La técnica Delphi puede ocuparse para resolver estas desventajas.

3.4.3.2 Delphi[49]

Es una técnica de estimación de costos, del tipo jerárquica hacia abajo, cuyo fin es obtener el consenso de un grupo de expertos, sin contar con los posibles efectos negativos de las reuniones de grupo. De forma general se basa en lo siguiente:

1.- Un coordinador organiza un equipo de personas con la mayor experiencia en el tipo de sistema que se desea desarrollar.

2.- El coordinador entrega a cada experto la documentación con la definición de sistema y se le solicita a cada uno que emita su estimación de acuerdo a su criterio personal y de forma anónima; los expertos pueden consultar con el coordinador, pero no entre ellos.

3.- El coordinador recibe los resultados y los sintetiza, entregando un resumen de las estimaciones efectuadas al equipo de expertos para una nueva revisión.

4.- Los expertos realizan otra estimación en base a los resultados de las estimaciones anteriores.

5.- El proceso se repite tantas veces como se juzgue necesario, hasta que la solución sea satisfactoria.

En seguida se presenta una variación de la técnica Delphi tradicional, en la que aumenta la comunicación, pero se sigue conservando el anonimato.

2.- El coordinador entrega a cada experto la documentación con la definición de sistema y se le solicita a cada uno que emita su estimación de acuerdo a su criterio personal y de forma anónima.

3.- El coordinador sintetiza las estimaciones anónimas proporcionadas por cada experto, sin incluír los razonamientos realizados por algunos de ellos.

4.- El coordinador solicita una reunión del grupo para discutir los puntos donde las estimaciones varíen más.

[49] [FAI85] pp. 76 - 77

5.- Una vez más los expertos efectúan sus estimaciones en forma anónima y el proceso se repite tantas veces como sea neceario, hasta que la estimación sea satisfactoria.

Es posible que después de hacer varias estimaciones, los expertos no lleguen a un consenso, por lo que el coordinador deberá analizar los razonamientos de cada experto para determinar las causas de las diferencias o recabar información adicional y presentársela a los expertos, con el fin de resolver las diferencias en los puntos de vista.

3.4.3.3 Estructuras de División del Trabajo [50]

Esta técnica conocida como Work Branch Structure (WBS), constituye un organigrama jerárquico hacia arriba, donde se establecen las diferentes partes de un sistema, reflejando una jerarquía de procesos.

El organigrama de procesos identifica las actividades de trabajo y sus interrelaciones. Utilizando esta técnica, los costos se estiman mediante la asignación del costo a cada componente individual en el organigrama y luego la suma de todos, **Figura 3.8.**

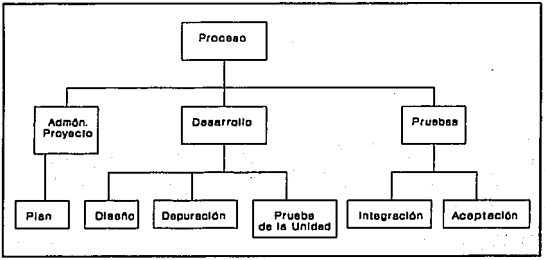

Figura 3.8 Estructura de División de Trabajo.

[50] [FA185] pp. 78 - 79

La ventaja principal de este modelo es que permite identificar y contabilizar los diversos procesos de un sistema, así como una visualización exacta de qué costos se incluyen en la estimación.

3.5 Herramientas de Estimación [51]

Existen herramientas automáticas de estimación que permiten al administrador estimar costos y esfuerzos, así como hacer más de un análisis con diversos datos variables de cada proyecto, como la fecha de entrega o la selección del personal. En esta sección se hablará de dos herramientas en las que se han implementado los modelos de COCOMO y Putnam.

3.5.1 COSTAR

Esta herramienta es un sistema automático de cálculo de costos, basado en el modelo de COCOMO. Toma en cuenta los siguientes factores para llevar a cabo una estimación:

1.- Capacidad de análisis y programación del personal de desarrollo
2.- El tipo de producto que se desea desarrollar
3.- El tipo de proyecto que se va a llevar a cabo
4.- El tipo de usuario que va a utilizar el producto
5.- Una estimación cuantitativa del tamaño del proyecto.
6.- El costo aproximado de cada fase del proyecto
7.- Una descripción del tipo de HW de desarrollo.
8.- Costo de mantenimiento

De estos datos, el modelo implementado por la herramienta automática de estimación proporciona estimaciones del esfuerzo requerido para completar el proyecto, los costos, el calendario de desarrollo y otros. En general Costar emite los siguientes reportes:

1.- Reporte Detallado (no. de personas requeridas para cada fase de desarrollo)
2.- Reporte de Calendarización (duración de cada fase)
3.- Reporte de Comparación de Estimaciones
4.- Reporte de Estructura (varias estimaciones para cada proyecto)
5.- Reporte de Actividades
6.- Histograma de Costos
7.- Histograma de Staff (incremento del no. de personas a lo largo del desarrollo del proyecto)
8.- Nombres de las Estimaciones

[51] [PRE87] pp. 131 - 132; [FRY91] pp. 34 - 35

3.5.2 SLIM

Esta herramienta automática de cálculo de costos, se basa en la curva de Rayleigh-Norden para el ciclo de vida del software y en el modelo de estimación de Putnam. Este sistema permite al administrador del proyecto realizar las siguientes funciones:

1.- Proporcionar al sistema datos históricos acerca del ambiente de desarrollo, como son las herramientas de desarrollo, el hardware utilizado, el tipo de producto a desarrollar, el tipo de proyecto, etc. Con ello será posible medir el ambiente actual de desarrollo de software.

2.- Crear un modelo de información del software, desarrollado a partir de las aracterísticas específicas del software, los atributos del personal, y las consideraciones del ambiente de desarrollo.

3.- Realizar un cálculo del tamaño del software.

Una vez que se establece el tamaño del software, SLIM calcula la desviación de tamaño (indicación de la incertidumbre de la estimación) un perfil de sensibilidad (indica la desviación del costo y del esfuerzo) y proporciona una distribución mes a mes del esfuerzo y el costo de tal forma que permita preveer los requerimientos de selección de personal y de liquidez.

Capítulo IV

ASEGURAMIENTO DE LA CALIDAD DE SOFTWARE

4.1 Introducción

El principal objetivo de este capítulo es hacer hincapié en que la calidad del software no es algo en lo que se empieza a pensar una vez que se generó el código. La garantía de calidad del software es una actividad de protección que se aplica a lo largo de todo el proceso de desarrollo del software.

Para considerar que un producto de software es de calidad, es necesario, pero no suficiente que satisfaga las necesidades del usuario, que se apegue a sus especificaciones de requisitos y diseño y que presente una ausencia de errores. La calidad de software es una mezcla de factores que varían de acuerdo a la aplicación y al usuario que la solicita. Aquellos factores que solo se pueden medir en forma indirecta (como la facilidad de uso o de mantenibilidad, su portabilidad, su integridad y otros), son los que determinan que un producto sea verdaderamente de calidad y por ello serán analizados en el presente trabajo.

En la primera sección se tratan los principales problemas a los que el equipo de desarrollo y el usuario se pueden enfrentar, a pesar de que el producto desempeñe sus funciones especificadas de manera correcta y se haya entregado a tiempo y dentro del presupuesto establecido. Posteriormente se presenta una serie de lineamientos para la instrumentación de la calidad del software a lo largo del proyecto. En seguida, se identifican y clasifican las características de calidad que debe poseer un producto para que sea considerado de calidad de acuerdo a Bohem y se proponen algunos medios para lograr que las características de calidad estén presentes durante todo el ciclo de vida del software. Por último se presentan dos casos prácticos del ciclo de vida del desarrollo de ambos sistemas.

4.2 Evaluación Cuantitativa de la Calidad de Software [53]

Es importante evaluar la calidad del software ya que, para considerar que un producto verdaderamente es de calidad, no es suficiente lograr que éste, se entregue a tiempo, se ajuste al presupuesto establecido y realice correcta y efectivamente todas las funciones especificadas durante la definición de requerimientos. Aun cuando todas estas características importantes, estuvieran presentes en un producto de desarrollo de software, los principales problemas a los que el usuario y el equipo de desarrollo se podrían enfrentar son los siguientes:

1.- El producto puede ser difícil de entender y/o de modificar. Esto provocaría un costo excesivo en el mantenimiento de software.

2.- El producto puede ser difícil de utilizar o fácilmente puede dejar de ser utilizado. Esto también provocaría un aumento en los costos de mantenimiento y por otra parte, significaría un gran desperdicio de dinero y esfuerzo, en el caso de no utilizar el software.

3.- El producto puede ser innecesariamente dependiente de una máquina específica, o difícil de integrarse con otros programas.

La calidad se obtiene mejor con una atención cuidadosa en los detalles de la planeación, el análisis, el diseño y la implantación. Para lograr ésto, es necesario reconocer las características que debe poseer un producto para que pueda ser considerado de calidad, además de especificar los lineamientos necesarios para la instrumentación de la misma. Una calidad alta no se puede probar dentro de un sistema concebido con equivocaciones y mal implantado.

4.2.1 Instrumentación de la Calidad de Software

De acuerdo a DeMarco "cualquier persona puede trabajar en forma efectiva para maximizar cualquier indicación de éxito observada como única" (Premisa Métrica)[54]. Weinberg y Schulman llevaron a cabo un experimento que confirma la propuesta anterior. Ellos dividieron un grupo de personas en cinco equipos de desarrollo y les asignaron un problema de desarrollo. A cada equipo se le encomendó una meta diferente. A un equipo,

[53] [BOE76] pp. 218 - 219
[54] [DEM82] p. 58-59.

por ejemplo, se le encomendó terminar el programa en el menor tiempo posible; a otro se le dijo que escribiera el código mas óptimo; y así con los demás equipos. El desempeño de los equipos se midió cuidadosamente y los resultados se presentan en la **Tabla 4-1**. Como se puede observar cada equipo sobresalió en la meta que se le asignó, pues sacrificaron cualquier otra meta de optimización por aquella que interpretaron como la más importante (la observada más cuidadosamente = 1).

Meta	Optimización del Desempeño 1 - Mejor				
	Tiempo de Fin.	Tamaño Prog.	Espacio p' Datos	Claridad Prog.	Manejo Amigable
Tiempo de Finalización	1	4	4	5	3
Tamaño del Programa	2-3	1	2	3	5
Espacio de Datos Utilizado	5	2	1	4	4
Claridad del Programa	4	3	3	1-2	2
Manejo Amigable para el Usuario	2-3	5	5	1-2	1

Tabla 4-1 Experimento de Weinberg y Schulman.

Por dar una importancia desproporcionada al tiempo de entrega (el cual casi siempre es una de las metas principales), los proyectos resultantes se entregan rápidamente pero no cumplen con todos los requerimientos, son difíciles de mantener y poco confiables. Tales productos en el análisis final cuestan más, que si se hubieran construido en forma cuidadosa.

La "Premisa Métrica" no quiere decir que se puede controlar todo lo que se puede medir, habrá cosas que a pesar de que se midan serán incontrolables, sin embargo, si se especifican, se miden y controlan ciertos aspectos de un proyecto, éste podrá terminarse con éxito, es decir, cumplirá con los objetivos planteados, y será un producto confiable, fácil de entender, de utilizar y de mantener. Para lograr esto, en seguida se presenta una lista de los lineamientos básicos para la instrumentación de la calidad:

1.- Preparar las especificaciones de calidad para el producto, es decir, formular una especificación de calidad de las funciones que se necesitan y del desempeño que éstas deben tener. Las especificaciones del producto podrían ser: seguridad de la

información, rapidez de proceso, facilidad de mantenimiento y facilidad para entenderlo, entre otras.

2.- Verificación del cumplimiento de las especificaciones de calidad del software.

3.- Realizar una negociación conveniente para compensar los costos de desarrollo y los costos operacionales. Esto es importante porque no tendría sentido desarrollar un sistema si el presupuesto y el tiempo es insuficiente, ya que repercutiría en la calidad del producto, pues la gente cuando trabaja bajo presión, no trabaja mejor, simplemente trabaja más rápido.

4.- Seleccionar un paquete de software. No todos las herramientas de desarrollo pueden proveer de todo lo que se requiere para un proyecto específico.

El evaluar y mejorar la calidad del producto es un concepto presente en todo el ciclo de vida, no solo una actividad que se desarrollará después de la implantación del sistema, pero por otra parte, el medir la producción de software sin medir la calidad del producto resultante, solamente asegurará que los costos de depuración "migrarán" fuera del proyecto hacia la actividad denominada como mantenimiento.

4.2.2 Puntos Claves en la Evaluación de la Calidad del Software

Algunos de los principales aspectos que se deben tomar en cuenta para llevar a cabo una evaluación de la calidad de software, son los siguientes:

1.- ¿Es posible establecer definiciones de las características de calidad de software que sean medibles y suficientemente diferenciables para permitir la evaluación del software?

2.- ¿Qué tan bien se puede medir integralmente la calidad del software o sus características individuales?

En las dos secciones siguientes se analizará la manera de identificar las características de calidad, así como las métricas asociadas y la forma de evaluarlas.

4.2.2.1 Identificación y Clasificación de las Características de Calidad [55]

Boehm desarrolló un conjunto jerárquico de características de calidad, así como de sus métricas asociadas. El enfoque que le dio fue el siguiente:

1.- Definir un conjunto de características que sean importantes para el software.

2.- Desarrollar métricas candidatas para estimar el grado en que el software posee las características definidas.

3.- Investigar las características y métricas asociadas para determinar su correlación con la calidad de software, los beneficios de su uso y la facilidad de automatización.

4.- Evaluar cada métrica candidata con respecto al criterio anterior y con respecto a su interacción con otras métricas: defectos, dependencias.

5.- Después de estas evaluaciones, refinar el conjunto de características en un conjunto que sea mutuamente excluyente y exhaustivo, el cual a su vez soporta la evaluación de la calidad de software.

6.- Refinar las métricas candidatas

El primer paso dio como resultado el siguiente conjunto inicial de características de calidad: Entendible, Completo, Conciso, Portable, Consistente, Mantenible, Evaluable, Utilizable, Confiable, Estructurado y Eficiente.

Como segundo paso se comenzaron a definir medidas candidatas de las características anteriores, con lo que se observó que cualquier medida de la Entendibilidad también era una medida de la Mantenibilidad, ya que para cualquier tipo de mantenimiento, se requiere que la persona que dará el mantenimiento, entienda lo que va a mantener. Por otra parte, se encontraron medidas de Mantenibilidad, que no tenían nada que ver con la Entendibilidad. Por ejemplo, las herramientas para evaluar la relación entre las entradas y salidas, son importantes para reevaluar la función de Mantenimiento del software, pero no está relacionado con la Entendibilidad.

[55] [BOE76] pp. 220 - 222; [FRY91] pp. 78 - 90

De esta manera, se determinó que las características estaban relacionadas en un tipo de árbol estructurado, en el que la dirección de las flechas representan una implicación lógica: si un programa es Mantenible, éste debe ser necesariamente Entendible y Evaluable, **Figura 4.1**.

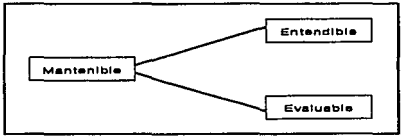

Figura 4.1 Relación entre Características de Calidad

Posteriormente se encontró que existía otro nivel de conceptos más primitivos, bajo el nivel de Entendible y Evaluable. Por ejemplo, si un producto es Entendible, a su vez, necesariamente es Estructurado, Consistente y Conciso, además de Legible y Autodescriptivo. Finalmente, a medida que se fueron generando otras características y se fueron revisando y evaluando sus métricas asociadas, se llegó al paso seis, representando el conjunto entero de características de calidad en un árbol estructurado, como el que se ilustra en la siguiente figura.

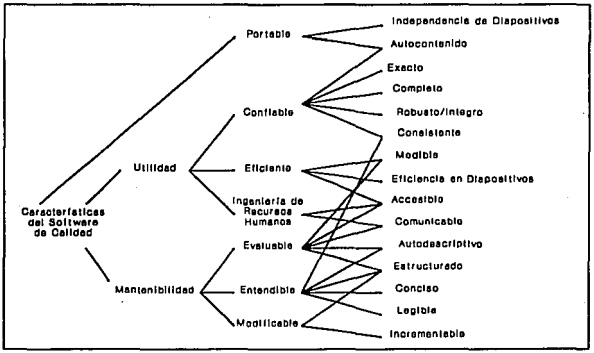

Figura 4.2 Características de Calidad

En el árbol de la figura 4.2 se puede observar que el nivel más alto se compone de tres características, Portabilidad, Utilidad y Mantenibilidad. La Utilidad por ejemplo, requiere que el programa sea Confiable, Eficiente y que se aprovechen correctamente los Recursos Humanos, pero no requiere que el usuario evalúe el programa, o que entienda sus procesos internos, que lo modifique o que trate de usarlo. Un producto Mantenible requiere que el usuario sea capaz de entender, modificar y evaluar el programa, lo que no depende de la Confiabilidad actual del programa, o de su Eficiencia y Portabilidad.

El nivel más bajo de la estructura se forma por un conjunto de características que también se diferencian unas de otras, pero que se combinan en conjuntos de condiciones necesarias para las características de nivel intermedio, por ejemplo:

- Un programa que es Independiente de Dispositivos y Autocontenido pero no es Exacto, Completo, Robusto, Consistente, Medible, Eficiente de Dispositivos, Accesible, Comunicable, Autodescriptivo, Estructurado, Conciso, Legible e Incrementable, aún así, satisface la definición de Portabilidad.

En seguida se incluye una breve descripción de las características de calidad:

Portabilidad.- indica que el código no es dependiente de una máquina o software específico, es decir, que el sistema se puede operar fácilmente y bien con otra configuración.

Independencia de Dispositivos.- Significa que la aplicación puede ejecutarse en diferentes configuraciones.

Autocontenido.- de manera sencilla proporciona al usuario textos de ayuda y operación de cada módulo del sistema.

Utilidad.- significa que es confiable, que los recursos humanos son eficientes.

Confiable.- significa que la aplicación se ejecuta correctamente, produciendo resultados exactos y que continuará operando satisfactoriamente.

Exacto.- significa que la aplicación produce salidas suficientemente precisas que cumplen con el objetivo de sus utilización.

Completo.- significa que la aplicación tiene todas sus partes especificadas y cada una de ellas esta totalmente desarrollada.

Robusto/Integro.- significa que el programa podrá ejecutar sus funciones a pesar de que se le proporcionen entradas inválidas.

Consistente.- consistencia interna significa que es uniforme en el uso de la notación, terminología; consistencia externa implica una correspondencia entre el diseño del programa y la codificación.

Eficiente.- significa cumplir con sus propósitos sin desperdiciar recursos.

Medible.- significa que la eficiencia de la aplicación (recursos, memoria y tiempo entre otros) se pueda medir.

Eficiencia en Dispositivos.- Se refiere a implementar una configuración adecuada al tipo de aplicación.

Accesible.- se refiere a la facilidad de utilizar de manera selectiva cada parte de la aplicación, por ejemplo, funciones, procedimientos, programas.

Ingeniería de Recursos Humanos.- se refiere al cumplimiento de los propósitos de la aplicación, sin desperdiciar el tiempo y esfuerzo de los usuarios.

Comunicable.- facilita la especificación de las entradas que alimentan el sistema, de tal forma que los resultados obtenidos sean confiables.

Mantenible.- facilita la actualización para satisfacer con nuevos requerimientos o corregir las deficiencias identificadas.

Evaluable.- facilita el establecer criterios de evaluación del desempeño de la aplicación.

Autodescriptivo.- contiene suficiente información para que una persona determine y verifique los objetivos, I/O, componentes y restricciones.

Estructurado.- posee una jerarquía organizacional de sus partes independientes, utiliza la secuenciación, iteración y selección para programar cualquier parte de la aplicación.

Entendible.- significa apreciar los objetivos de cada uno de los componentes de la aplicación de manera clara, es decir, los nombres de variables, términos, funciones y programas se usan de forma consistente y los módulos son autodescriptivos.

Conciso.- implica que el programa no se dividió excesivamente en módulos, funciones y subrutinas, ni que una misma secuencia de código se repitió en numerosos lugares en lugar de definir una subrutina o macro.

Legible.- significa que la función que desempeña el código, se puede identificar tan solo leyéndolo.

Modificable.- facilita la incorporación de cambios.

Incrementable.- se refiere a que el código de una aplicación puede crecer, tanto interna (rango de variables) como externamente (módulos).

Las características primitivas o de nivel más bajo, permiten definir métricas cuantitativas, las cuales se pueden utilizar para determinar la medida en que un producto, posee tanto las características de nivel más alto, como las de nivel más bajo. Esto permite evaluar la utilidad del software y apreciar las necesidades de perfeccionamiento del mismo. La siguiente sección se refiere a la manera de definir las métricas asociadas a las características de calidad identificadas.

4.2.2.2 Medición de la Calidad de Software [56]

Dado que ya se definió el conjunto de características de calidad que debe poseer un producto, ahora se procede a definir métricas para cada una de las características identificadas, tal que:

1.- Dado un programa arbitrario, la métrica proporcione una medida cuantitativa del grado en que el programa posee la característica asociada.

2.- La calidad del software de manera global, se puede definir como una función de los valores de las métricas.

[56] [BOE76] pp. 222 - 224

La definición de las métricas consistentes y de carácter global, para la evaluación de la calidad de software, puede ser una tarea difícil, ya que la mayoría de las características primitivas (de nivel más bajo) de calidad del software están en conflicto, por ejemplo, a mayor eficiencia, menor portabilidad, exactitud, entendibilidad y mantenimiento y una mayor exactitud implica probable conflicto con la portabilidad (i.e. independencia del tamaño de palabra) y si es conciso puede crear conflicto con la legibilidad. Además otro problema es que las métricas son generalmente medidas incompletas de sus características asociadas. Por tanto en resumen se tiene lo siguiente:

1.- Las características de calidad de un producto de software, varían de acuerdo a las necesidades y prioridades del usuario.

2.- Por lo tanto, no existe una métrica única que pueda otorgar una medida universal de la calidad de software.

3.- En el mejor de los casos, un usuario puede recibir un software de calidad, si se proporciona el grado de calidad deseado del sistema a través de prioridades y se verifica su cumplimiento con el uso de listas de chequeo.

4.- Sin embargo, ya que las métricas no son exhaustivas, el resultado de su aplicación sería más sugestivo que conclusivo.

5.- Por lo tanto, lo mejor es utilizar las métricas como indicadores individuales de fallas, de tal forma que sirvan como una guía de desarrollo de software, de planeación de pruebas y de mantenimiento.

4.3 Aplicación de las Características de Calidad en el Proceso de Ciclo de Vida del Software.

El proceso de ciclo de vida de un proyecto de desarrollo de software en general se divide en: análisis de requerimientos, especificación, diseño, codificación, prueba, operación y mantenimiento. Los principales medios encontrados para usar las características de calidad en el mejoramiento del proceso de ciclo de vida del software son [57]:

- Establecer los objetivos y prioridades de calidad de software
- Utilizar listas de chequeo de calidad de software
- Especificar claramente en qué consiste la actividad de aseguramiento de calidad
- Usar métodos y herramientas enfocadas al desarrollo de productos de calidad

1.- Establecer los Objetivos y Prioridades de Calidad de Software. El experimento de Weinberg y Shulman presentado en la sección 4.2.1, muestra que el grado de calidad que una persona logra, esta relacionado con los objetivos y prioridades de calidad de software que se le encomiendan. Entonces, si un usuario requiere portabilidad y mantenibilidad más que eficiencia del código, es importante decirle ésto al desarrollador, de tal forma que el usuario pueda determinar la medida en que estas cualidades están presentes en el producto final.

2.- Utilizar Listas de Chequeo de Calidad de Software. Las métricas de calidad desarrolladas por Boehm, se pueden usar como base de un conjunto de listas de chequeo de calidad de software, las cuales se utilizan para hacer revisiones, recorridos, inspecciones y estimaciones del producto de desarrollo. Estas listas de chequeo se han usado principalmente para apoyar los objetivos de confiabilidad del software, pero se pueden utilizar para otros objetivos de calidad de software; por ejemplo, la **Tabla 4-3** presenta una porción de una lista de chequeo para la característica de autodescriptivo de un programa de computadora.

[57] [BOE76] pp. 226 - 228; [PRE87] pp. 480 -514

Lista de Chequeo Parcial para Determinar
si un Programa es Autodescriptivo

Un producto de software es autodescriptivo, en la medida que contiene suficiente información para que el inspector que evaluará el producto, determine o verifique sus objetivos, suposiciones, restricciones, entradas, salidas, componentes y estado de revisión. Lista de chequeo:

a) ¿Cada módulo del programa contiene una cabecera de comentarios en la que se describe el nombre del programa, fecha efectiva, requerimientos de exactitud, propósito, limitaciones y restricciones, historia de las modificaciones, entradas y salidas, métodos y suposiciones?

b) ¿Se definieron adecuadamente las funciones de los módulos, así como las entradas y salidas, de tal forma que sea posible una evaluación por módulo?

c) ¿Los comentarios proporcionan información suficiente para seleccionar valores de entrada específicos que permitan evaluar el desempeño del programa?

d) ¿Cuando existe dependencia de módulos, ésto se especifica claramente en algún comentario, en la documentación del programa o esta implícito en la propia estructura del programa?

e) ¿Los nombres de las variables describen claramente la propiedad física o funcional que representan?

3.- Especificar Claramente en qué consiste la Actividad de Calidad de Software. De acuerdo a Boehm, tanto la calidad del producto como la optimización de los costos se incrementa en forma notable, a medida que se lleva a cabo la actividad de aseguramiento de calidad en los proyectos de desarrollo.

Un proyecto se divide en actividades, las cuales se determinan en base al tamaño y al alcance del mismo, es decir, el aseguramiento de calidad de un proyecto depende en gran medida de los requerimientos de calidad que el usuario especifique y en particular del tipo de aplicación a desarrollar.

Las responsabilidades de la actividad de asegurar la calidad incluyen:

- Planeación. Preparación de un plan de aseguramiento de calidad de software en el que se tomen en cuenta los requerimientos de calidad del programa, la asignación de actividades, el calendario del proyecto y las responsabilidades organizacionales.

- Desarrollo de Procedimientos y Políticas. Preparación de manuales de estándares de calidad para todas las fases del desarrollo del proyecto.

- Aseguramiento de la Calidad de Software. Desarrollo de manuales y procedimientos automáticos para verificar que se cumpla con los estándares de calidad establecidos y con los requerimientos y funciones especificadas del software.

- Auditorías. Revisión de la documentación y de los procedimientos del proyecto, con el fin de que se lleven a cabo los estándares del plan de desarrollo de software. Documentar todas las acciones correctivas, así como el resultado obtenido de la auditoría.

- Revisión de las Pruebas. Reportar los problemas encontrados en el software y sus causa, asegurándose de corregirlos adecuadamente.

- Registro. Conservar un registro del análisis, diseño, de los reportes de problemas en el software, de los casos y datos de prueba, así como de las revisiones del aseguramiento de calidad.

- Control de los Medios Físicos. Inspección de los discos, cintas, tarjetas y otros medios de almacenamiento de información, para asegurar que ésta, así como la transmisión de la misma sea segura, de tal forma que disminuya el riesgo de ser alterada o destruida por el medio ambiente u otros factores.

Aunque la mayoría de estas responsabilidades se enfocan a la confiabilidad del software y al cumplimiento de los estándares y especificación de requerimientos del proyecto, las actividad de aseguramiento de calidad, también se puede usar para asegurar que el producto posea otras características de calidad, como mantenibilidad o portabilidad.

4.- Usar Métodos y Herramientas enfocadas al Desarrollo de Productos de Calidad. Cada método y herramienta de software como, el análisis y diseño estructurado, los generadores de código, los procedimientos de administración de la configuración y otros, se desarrollaron con el fin de proporcionar apoyo a las diversas fases del ciclo de vida de

un proyecto de desarrollo de sistemas, con un grado de calidad elevado. Tales métodos y herramientas, son relativas al menos, a una característica de calidad.

Métodos y Herramientas Enfocadas al Análisis y Especificación

El objetivo del análisis, es especificar de manera completa, detallada y sin ambigüedades, los requisitos recopilados en la planeación inicial, para poder definir las características que el producto de programación debe tener. Se han propuesto muchos métodos de análisis de requerimientos. En la siguiente tabla se presenta una lista de algunos métodos y herramientas.

Método/Herramienta	Fuente de Información	Categoría *
Desarrollo de Sistemas Estructurados	(WAR74), (ORR77), (ORR81)	M, A
Desarrollo de Sistemas de Jackson	(JAC83)	M, A
Análisis de Sistemas Estructurados	(DEM79), (GAN82)	M, A
Design Aid	Nastec Corporation	A
Excelerator	Index Technology, Inc.	A
PSL/PSA	ISDOS Inc.	F, A
Software Requirements Engineering Methodology (SREM) and (SYSREM)	(ALF85), (WHI85)	F, A
Structured Analysis and Design Technique (SADT)	Softtech, Inc.	M, A
Tecnology for Automated Generation of Systems (TAGS)	(SEI85)	F, A
Herramientas Matemáticas para el Análisis de Sistemas	(MCC85)	M

* M = Técnicas Manuales; A = Herramienta Automática; F = Leng. de Especificación Formal

Tabla 4-4 Métodos y Herramientas de Análisis de Requerimientos

Técnicas Enfocadas al Diseño

La etapa de diseño se ocupa del refinamiento de los requisitos especificados durante el análisis. Se definen la estructura interna del sistema, los procesos a realizar, los detalles de los algoritmos, las estructuras de datos, las pantallas, los formatos de reportes, las entradas y salidas de datos y los planes de prueba. Además del establecimiento de las relaciones e interconexiones entre los procesos, los datos y el almacenamiento de los mismos. Se han desarrollado muchas metodologías que se utilizan en diferentes aplicaciones. En la siguiente tabla se presenta una lista de algunas de ellas.

Método/Herramienta	Fuente de Información	Categoría *
Niveles de Abstracción	(DIJ68)	M
Desarrollo Integrado Descendente	(FAI85)	M
Diseño Estructurado	(YOU78), (PAG80)	M
Programación Estructurada de Jackson	(JAC83)	M
Structured Analysis and Design Technique (SADT)	Softtech, Inc.	M, A
Método de Diseño de Sistemas de Tiempo Real (DARTS)	(GOM84)	M
Tecnology for Automated Generation of Systems (TAGS)	(SEI85)	F, A

* M = Técnicas Manuales; A = Herramienta Automática; F = Leng. de Especificación Formal

Tabla 4-5 Métodos y Herramientas de Diseño de Sistemas

Métodos y Herramientas Enfocadas al Código

Durante esta etapa se traducen las especificaciones de diseño a código fuente, además de desarrollar una documentación interna que permita verificar que el código cumple con las especificaciones y que por otra parte permita de manera sencilla llevar a cabo depuraciones, pruebas y modificaciones. A continuación se muestra algunos métodos y herramientas que facilitan la instrumentación del sistema.

Método/Herramienta	Fuente de Información	Categoria *
Modelo Sintáctico/Semántico	(SHN80)	M
Administradores de la Configuración (SCCS), (MAKE), (CADES), (RCS)	(ROC75), (FEL79), (MCG79) (BER84)	A, M
Programación Estructurada	(LIN79), (YOU75)	M
Editores Especializados para Programación	Cheetah, Inc.	A
Generadores de Código	Zachary, Inc.	A
Prototipos	(BRO75), (WAS82), (BAL82) (BOE84)	F, A
Administradores de Bases de Datos	(PRE87), (FAI85)	A
Lenguajes de Tercera Generación	(PRE87)	F
Lenguajes de Cuarta Generación	(CHO86), (MAR85)	F

* M = Técnicas Manuales; A = Herramienta Automática; F = Leng. de Especificación Formal

Tabla 4-6 Métodos y Herramientas de Codificación

Métodos y Herramientas Enfocadas al Proceso de Prueba

La prueba del software es un elemento importante, cuyo objetivo es mejorar la calidad del software y representa un repaso de las especificaciones, del diseño y de la codificación. En seguida, algunas herramientas y técnicas que apoyan la etapa de prueba del sistema.

Método/Herramienta	Fuente de Información	Categoría *
Técnicas de Verificación y Validación	(MYE79), (FAI85)	M
DAVE (para Fortran)	(OST76), (MIL81), (DUN84)	A
LINT (para lenguaje C)	(RIT78)	A
Sintetizador de Programas de Cornell (para PL/1)	(TEI81)	A

* M = Técnicas Manuales; A = Herramienta Automática

Tabla 4-7 Métodos y Herramientas para el Proceso de Prueba

Métodos y Herramientas Enfocadas al Mantenimiento

El término "mantenimiento" se aplica al proceso de modificar un programa cuando ya se ha entregado y está en uso. Esas modificaciones pueden implicar cambios sencillos para corregir errores de codificación, cambios mayores para corregir errores de diseño o reescrituras drásticas para corregir errores de especificación o introducción de nuevos requerimientos. En el capítulo tres se presentaron algunos métodos y herramientas utiles durante la estimación de los costos de mantenimiento, los cuales tambien sirven para realizar revisiones y validaciones del mismo. Otras herramientas automáticas que sirven de apoyo durante la fase de mantenimiento son, Editores de Texto, Ayudas de Depuración, Generadores de Referencia Cruzada, Editores de Enlace, Comparadores, Calculadores de Métricos de la complejidad, Sistemas de Control de Versión y Configuración de Base de Datos .

4.4 Ejemplos

En esta sección se presentarán dos ejemplos comparativos, uno donde prácticamente no existió planeación del proyecto debido a diversos factores que serán mencionados más adelante y otro ejemplo en el cual se utilizó una metodología para la planeación del proyecto.

4.4.1 Sistema de Inscripciones del Registro Agrario Nacional.

El Registro Agrario Nacional es un órgano desconcentrado de la Secretaría de la Reforma Agraria, el cual fue creado con el fin principal de dar solución al rezago de trámites de tipo agrario, solicitados por los ejidatarios y comuneros de nuestro país.

4.4.1.1 Objetivo del Sistema

Facilitar la inscripción en el Registro Agrario Nacional (R.A.N.) de los acuerdos de asamblea tomados por los ejidatarios de cada ejido, comunidad o sociedad rural de la República Mexicana generando en forma rápida los documentos o "inscripciones" que avalen dichos acuerdos; por otra parte optimizar la emisión de certificados de derechos parcelarios y de uso común, así como la de títulos de solares urbanos y de propiedad de los ejidatarios y comuneros, para lo cual será necesario llevar el control principalmente en relación con las siguientes preguntas:

1.- ¿Cuáles? y ¿Cuántos?
   a) municipios existen por estado
   b) ejidos o comunidades existen por municipio
   c) sociedades rurales existen en cada ejido o comunidad
   d) parcelas son propiedad del ejido o comunidad
   e) parcelas posee cada ejidatario o comunero
   f) títulos de solar urbano y de propiedad posee cada ejidatario o comunero

2.- ¿Quiénes? y ¿Cuántos?
   a) ejidatarios y comuneros existen por ejido o comunidad
   b) ejidatarios integran cada una de las sociedades rurales
   c) ejidatarios y comuneros poseen certificados de derechos parcelarios, de uso común
   d) ejidatarios y comuneros poseen títulos solares o de propiedad
   e) son los representantes de cada ejido, comunidad o sociedad

3.- ¿Cuánto? (en hectáreas, áreas y centiáreas, así como en metros)
   a) es la superficie del ejido
   b) del total de la superficie del ejido pertenece a la zona denominada parcelaria
   c) del total de la superficie del ejido pertenece a la zona denominada de uso común
   d) del total de la superficie del ejido pertenece a la zona denominada urbana
   e) es la superficie de cada parcela o solar urbano de cada ejidatario
   f) es la superficie de cada parcela o solar urbano propiedad del ejido

4.- ¿Cuándo?
   a) se creó el ejido
   b) se tomó el acuerdo en que se asignan derechos parcelarios o de uso común a los integrantes de un ejido o comunidad
   c) se inscribieron o registraron en el R.A.N. cada uno de los acuerdos de asamblea
   d) se generaron físicamente cada uno de los certificados de derechos parcelarios y y de uso común que se encuentran ya inscritos en el R.A.N.
   e) se generaron físicamente cada uno de los títulos de solar y de propiedad que se encuentran ya inscritos en el R.A.N.

Aparentemente el sistema cumple con cada una de las cuestiones anteriores, sin embargo a continuación se muestra un análisis general del mismo, con el cual será posible determinar si en realidad es tan bueno como parece.

4.4.1.2 Metodología empleada en la solución

Resultó imposible determinar si se utilizó alguna metodología en el desarrollo del Sistema de Inscripciones del Registro Agrario Nacional (S.I.R.A.N.) pues no existen datos históricos que permitan hacer un seguimiento de la planeación que se realizó. Sin embargo mi participación en el desarrollo de la cuarta versión del sistema me hace suponer que no se aplicó ninguna metodología en las versiones anteriores, ya que ni siquiera fue posible identificar la fase en que uno se encontraba durante el avance de la 4a. versión del proyecto, pues constantemente se modificaban las indicaciones (ya que no existen especificaciones de diseño) hechas por el administrador a los integrantes del equipo de desarrollo, para automatizar los diversos procesos que conformaron la cuarta versión. Las únicas fases que fueron bien especificadas fueron la instalación y puesta en operación.

4.4.1.3 Métricas

El desarrollo del Sistema de Inscripciones del Registro Agrario Nacional (SIRAN), dio inicio a principios del mes de octubre de 1992. No se cuenta con registros de datos que indiquen el avance del proyecto durante el desarrollo de las 3 primeras versiones del mismo, ni se cuenta con los programas correspondientes a cada versión.

Durante el período comprendido entre el 1o. de abril de 1994 y 15 de mayo de 1994, por primera vez después de 3 versiones del sistema se trató de implementar un plan de pruebas, el cual consistía en asignar una única actividad a 5 personas: probar el sistema en todas y cada una de sus opciones. Con lo anterior se descubrió que el sistema estaba plagado de errores, se registraron 195 errores de programación, y se corrigieron 124 errores, al resto fue imposible darles atención, ya que era necesario dar inicio al desarrollo de la siguiente versión, pues esta incluiría principalmente nuevos requerimientos solicitados por el R.A.N. El resto de los errores se fueron corrigiendo durante la fase de mantenimiento de la cuarta versión, así como los errores generados por las correcciones hechas durante los meses de abril y mayo, ya que lamentablemente de los 124 errores corregidos, se generaron 31 errores más de programación.

Los datos que se presentan a continuación fueron recopilados de la última versión del sistema la cual se llevó a cabo en 4 semanas. Estos datos serán utilizados para definir las métricas de productividad y de calidad. Por otra parte, a pesar de que en realidad no existió una división real de las fases de desarrollo del proyecto, por simple organización, me tomé la libertad de dividir en fases los datos observados.

ANÁLISIS

Fecha de inicio de la fase	Abril - 94
No. de diagramas de flujo de información	0
Personas que intervinieron en el análisis	1
Duración de la fase antes de pasar a la fase de diseño (aprox.)	10 días
No. de catálogos a desarrollar	0
No. de procesos (inscripciones) a desarrollar	16
No. de reportes a desarrollar	32
No. de consultas de inscripciones a desarrollar	16

DISEÑO

Fecha de inicio de la fase de diseño	Mayo - 94
Modelos de entidad - relación	0
Personas que intervinieron en el diseño	1
Duración de la fase antes de pasar a la fase de programación (aprox.)	8 días
Cantidad de bases de datos nuevas	6
No. de índices por bases de datos (promedio)	4
No. de campos no utilizados por base de datos (promedio)	4
No. de formatos de diseño de reportes	0
No. de diagramas de flujo de los procesos automatizados	0

PROGRAMACION

Fecha de inicio de la fase de programación	09-Mayo-94
No. de personas que programaron	4
Lenguaje de programación utilizado	Fox ProLan
Experiencia de los programadores en el lenguaje utilizado	1 año
Experiencia en programación para red	1 año
Tiempo que duró la programación antes de las pruebas	4 días
Duración de la programación a la par de las pruebas	7 días
Inscripciones programadas por persona	4
Reportes programados por persona	4
Consultas programadas por persona	4
Reportes elaborados por un programador extra	16
Programas documentados	0
Sueldo promedio de un programador o instalador (mensual)	3,000.00
Sueldo del líder, analista, diseñador y administrador del proyecto	30,000.00
Existe algún líder de programadores	No

PRUEBAS

Fecha de inicio de la fase de pruebas	16-Mayo-94
Tiempo de duración de las pruebas antes de la instalación	5 días
Persona que realizaba las pruebas de unidad (de cada programa)	el desarrollador
Persona que realizaba las pruebas de integración	un instalador
Errores encontrados por inscripción, reporte o consulta (promedio)	2
Tiempo asignado para corregir todos los errores encontrados	5 días
No. de instaladores que intervinieron en las pruebas	5
¿Se realizó una prueba de operación antes de la instalación? (con usuarios finales)	No

INSTALACIÓN

Fecha de inicio de la fase de instalación	23-Mayo-94
Duración de la instalación	12 días
No. de estados de la República Mexicana donde se instaló	32
Duración de la instalación por cada estado de la república	3 días
No. de programadores que fungieron como instaladores	3
No. de instaladores que actualizaron la versión del sistema	5
Tiempo de capacitación a los instaladores	5 días
Tiempo de capacitación al personal usuario de cada estado	2 días
Costo por viaje	1,000.00

DOCUMENTACIÓN

Existe documentación actualizada de:

La operación del sistema	No
La instalación del sistema	No
Bases de datos e índices correspondientes	Si
No. de programas desarrollados	Si
Módulos del sistema	No
Diagramas de flujo de información	No
Errores encontrados	No
Errores corregidos	No

MANTENIMIENTO

Fecha de inicio del mantenimiento	06-Junio-1994
Fecha de término de mantenimiento de la 4a. versión	30-Julio-1994
No. de viajes realizados en el período de mantenimiento	10
Costo de cada viaje	300.00
No. de envíos por mensajería de las correcciones a cada estado	64
Costo promedio por envío	40.00
No. de programadores que realizaban el mantenimiento	4
No. de instaladores que realizaban las pruebas de integración	3
No. de errores de programación	15
No. de cambios en los programas por error de diseño	10
No. de cambios en la bases de datos por error de diseño	6

En seguida se presentan algunas métricas de productividad basadas en los datos recopilados del desarrollo de la 4a. versión del sistema.

Costo del desarrollo

5 programadores + 5 instaladores = 10 * $ 3,000.00 = $ 30,000.00 de sueldo mensual
1 administrador del proyecto = $ 30,000.00 de sueldo mensual
32 viajes * $ 1,000.00 de viáticos por viaje = $ 32,000.00 de costo por 32 viajes
Costo de transporte = $ 20,000.00 por transporte

Si se suma el sueldo mensual de los programadores e instaladores, más el sueldo del administrador, más el costo de viáticos y transporte se tiene un total de 82,000.00 nuevos pesos por el desarrollo e instalación en 32 estados de la 4a. versión, en un lapso de 22 días.

Costo del mantenimiento del desarrollo

Duración del mantenimiento : 2 meses aproximadamente
5 programadores + 5 instaladores = 10 * $ 3,000.00 = $ 30,000.00 de sueldo mensual
1 administrador del proyecto = $ 30,000.00 de sueldo mensual
10 viajes * $ 400.00 de viáticos y transporte = $ 4,000.00 por viajes
64 envíos por mensajería * $ 40.00 = $ 2560.00 por envíos

Si se suma el sueldo bimestral de los programadores e instaladores, más el sueldo del administrador en dos meses, más el costo de viáticos y transporte, más el costo de envíos por mensajería, se tiene un total de 126,560.00 nuevos pesos por mantenimiento y actualización en 32 estados de la 4a. versión, en un lapso aproximado de 2 meses

A continuación se presenta el conjunto de métricas de calidad o indicadores individuales de fallas del sistema en relación con las características de calidad de nivel más bajo.

EXACTO

De 20 inscripciones capturadas, todas presentan diferencias de texto y de cantidades numéricas cuando se emite un reporte o cuando se consultan en pantalla.

De 20 reportes emitidos, 15 presentan datos falsos, como superficies de parcelas erróneas, fechas de captura y de impresión erróneas, acumulados de superficie de parcelas por ejidatario, por comunero, por ejido, por comunidad, por municipio y por estado erróneas.

Permite emitir certificados de parcelas que fueron cancelados con anterioridad.

COMPLETO

Todos los módulos que componen el sistema no están operando actualmente.

Se realizaron 30 tipos de trámite o inscripciones que puede solicitar un ejidatario o comunero y de todas ellas ninguna funciona cuando se trata de comuneros.

AUTOCONTENIDO

No existe modulo o alguna simple tecla programada que proporcione al usuario algún tipo de orientación con relación a cada uno de los procesos que realiza el sistema y como se deben operar.

ROBUSTO/INTEGRO

Durante la captura de cada una de 10 inscripciones, al presionar la tecla F5 o F9 presenta un error en pantalla, suspende la operación del sistema y envía el control al menú principal del sistema.

Durante la captura de la superficie de una parcela, si se introduce alguna letra o se presiona una tecla de función, se presenta un error en pantalla y la operación del sistema se suspende totalmente, enviando el control al sistema operativo.

Si se cometen errores de captura durante la delimitación de un ejido, es decir durante la captura de la superficie que abarca cada una de las zonas en que se divide un ejido, como lo son la zona de urbanización, la zona parcelaria, la zona de uso común y la zona por delimitar, es imposible hacer correcciones desde el mismo sistema pues este no lo contempla; la única manera de solucionar tales problemas es accesando directamente las bases de datos.

INDEPENDENCIA DE DISPOSITIVOS

La aplicación no funciona si en la red de computadoras no se encuentra instalado el manejador de bases FOX PROLAN V.2.0.

Es necesario que en el directorio donde se encuentran las bases de datos, existan físicamente todos los programas (compilados en la versión de fox mencionada) que componen la aplicación, ya que no existe ningún ejecutable del sistema.

CONSISTENTE

De 30 programas analizados de un total de 150, 25 tienen título y 3 tienen todas sus variables documentadas y a pesar de que el lenguaje es estructurado, el código no lo es en más de la mitad de los 30 programas analizados (cabe aclarar que los 30 programas analizados se eligieron tratando de abarcar la mayor parte del sistema, por ello se seleccionaron 6 programas dentro de cada uno de los 5 módulos principales del sistema).

No existen estándares de presentación en pantalla, ni de reportes, ni de menús, entre los módulos que componen el sistema; al parecer cada programador le puso su toque de distinción.

MEDIBLE

Cuando solo una persona trabaja en el sistema, los datos son almacenados inmediatamente, un reporte de catálogo o de certificados se tarda 3 seg., una consulta en pantalla de una inscripción o de los datos de un ejido o de un ejidatario tarda 3 seg., sin embargo cuando trabajan más de 6 personas con el mismo proceso, el tiempo mencionado anteriormente se triplica.

En las bases de datos se incluyen campos que jamás se utilizan.

De 10 programas que emiten reportes, 8 abren y cierran una misma base de datos más de dos veces, lo cual invariablemente incrementa el tiempo de respuesta e interfiere con el desempeño de otros procesos, ya que la base de datos permanece más tiempo en uso exclusivo; en disco existen índices que fueron creados y posteriormente se decidió que no eran necesarios, sin embargo permanecen ocupando espacio.

ACCESIBLE

Cada uno de los módulos que componen la aplicación se pueden utilizar en forma individual , ya que el manejador de las bases de datos permite compilar y ejecutar cada programa en forma independiente del resto de los programas, de tal forma que la ejecución y seguimiento del código de cada programa es sencillo.

EFICIENCIA DE DISPOSITIVOS

Se instaló una red para veinte estaciones, con un servidor cuya capacidad de almacenamiento es de 2 Gigabytes, así como dos impresoras láser y dos de matriz para cada estado. Ya que el sistema fue diseñado para que operara en forma conjunta por un grupo de personas además de que es posible seleccionar cualquiera de las impresoras previamente instaladas y dado que el volumen promedio de información que actualmente existe en cada delegación es de 50 Megabytes, considero que la configuración implementada es la adecuada, pues en un año y medio no se han presentado problemas de funcionamiento del sistema en relación con los dispositivos programados.

COMUNICABLE

No existen comentarios de ayuda durante la captura de cada uno de los datos de solicita el sistema.

En 6 de 10 sub-módulos probados de uno de los módulos principales del sistema, después de terminar una captura, o de solicitar una consulta o emisión de algún reporte, el usuario no sabe que hacer pues el sistema no envía ningún tipo de mensaje, simplemente deja en pantalla lo que hasta ese momento había, y después de un segundos envía el control al menú principal.

AUTODESCRIPTIVO

El nombre de cada módulo permite determinar el tipo de procesos a realizar, así como los datos a capturar, sin embargo no existe ningún módulo o siquiera un manual que confirme las suposiciones de operación.

ESTRUCTURADO

Ya que el lenguaje que se utilizó es FOX, es de esperarse que la programación es estructurada, ya que así funciona FOX, aunque como se mencionó en páginas anteriores, el código de SIRAN no es estructurado en una gran cantidad de programas.

CONCISO

En 5 menús existen opciones que indican en pantalla que se encuentran en desarrollo o las cuales ya fueron desarrolladas pero se incluyeron en otro módulo.

Existen 4 sub-módulos que realizan el mismo proceso, con la única diferencia que uno solicita al usuario un número de ejido para trabajar y otro sub-módulo solicita un número de comunidad.

No existe alguna biblioteca de funciones, por ejemplo, la función que pregunta si los datos son correctos, existe en todo y cada uno de los programas que componen la aplicación.

LEGIBLE

No es posible saber que almacena cada variable, cuando se lee el código de los programas, pues en primer lugar los nombres de variables no son alusivos a su contenido y en segundo lugar no se encuentran documentadas..

A pesar de que el lenguaje es estructurado, cuando el código no lo es, se dificulta hacer un seguimiento del mismo.

INCREMENTABLE

Ya que el diseño modular no ha requerido de cambios considerables, no es difícil ampliar el número de opciones del sistema, así como la cantidad de programas que maneja; en cuanto al número de variables de memoria no existe problema siempre y cuando se configuren adecuadamente cada una de las computadoras donde se encuentre instalado el sistema.

4.4.2 Sistema de Control y Seguimiento Administrativo.

El sistema de control y seguimiento administrativo (SCYSA) es propiedad de Ericsson, empresa de origen sueco, cuyo giro en nuestro país son las telecomunicaciones. El SCYSA fue desarrollado por personal mexicano que labora en la institución y uno de los problemas al cual se buscaba darle solución fue al desperdicio de tiempo y recursos de cómputo por parte del personal del área administrativa de la Dirección de Sistemas de Radio de Ericsson.

El desperdicio consistía en la utilización (por unas cuantas personas) de las computadoras como simples máquinas de escribir, ya que con ellas se realizaban reportes (utilizando procesadores de texto y hojas de cálculo) a la dirección y al resto de las áreas de la misma, de las ventas reales y de las ventas presupuestadas, así como de los anticipos y saldos del costo de los equipos de radio-comunicación (radiobases telefónicas, radiobases mobitex, por mencionar algunas), de los materiales (tarjetas de circuitos, cables, enchufes, etc.) y de las instalación de ambos.

4.4.2.1 Objetivo del Sistema

Apoyar a la Dirección de Sistemas de Radio en la toma de decisiones, de tal forma que sea posible controlar el cumplimiento de los objetivos y por otra parte evaluar la eficiencia de los procedimientos establecidos, así como de la productividad del personal administrativo, para lo cual será necesario automatizar el trabajo rutinario de los procesos que se enumeran a continuación:

1.- "Backlog" o seguimiento del comportamiento de los costos presupuestados, de los costos reales y de los saldos pendientes por concepto de instalación de cualquier equipo o material de radio-comunicación utilizado en diversos proyectos.

2.- "Backlog" o seguimiento del comportamiento de los costos presupuestados, de los costos reales y de los saldos pendientes por concepto de venta de equipo y material de radio-comunicación utilizado en uno o más proyectos.

3.- "Backlog" o seguimiento del comportamiento de los saldos pendientes en relación con los descuentos a los costos reales por concepto de anticipo de venta de equipo o material de radio-comunicación.

4.- Facturación de venta e instalación de equipos y materiales de radio-comunicación.

4.4.2.2 Metodología empleada en la solución

El desarrollo del Scysa es el resultado de la planeación realizada por el personal de Ericsson encargado de la automatización de la Dirección de Sistemas de Radio; dicha planeación inició con el análisis del manual de métodos y procedimientos de calidad encaminados a cumplir con los objetivos ya establecidos para la dirección. Dado que los objetivos, métodos y procedimientos de la dirección ya estaban definidos, se procedió a diseñar una estructura de sistemas de información que cumpliera con dichos objetivos y además optimizará los recursos de cómputo, así como de tiempo y esfuerzo del personal.

Una vez definida la estructura de los sistemas informáticos a desarrollar, se dio prioridad al SCYSA, ya que los informes de costos y saldos por venta e instalación de equipo y material de radio-comunicación, llegaban tarde a manos de los gerentes y directivos del área, lo cual impedía llevar a cabo un seguimiento eficiente del costo real de los proyectos.

Hasta el punto anterior no se siguió ninguna metodología de planeación estratégica, simplemente utilizando la experiencia del desarrollador se diseñó la arquitectura, se asignaron prioridades a los sistemas y se dio inicio al desarrollo del sistema SCYSA.

Una vez que se decidió comenzar con la automatización de los procedimientos administrativos del área, se pensó entonces en aplicar alguna metodología que garantizara la interrelación sencilla entre todos los sistemas definidos en la arquitectura, que acelerara la construcción de cada uno de los sistemas y que evitara la dependencia, por parte de la institución, hacia una o varias personas para que se encargaran del mantenimiento, es decir, lo que se buscaba era construir una aplicación fácil de usar y de modificar.

Por lo anterior se optó por una metodología estructurada automatizable que definiera un enfoque de ingeniería para todos o algunos aspectos del desarrollo y mantenimiento del software, tal metodología es CASE. Esta metodología se basa en el uso de un conjunto de herramientas integradas que han sido diseñadas para trabajar juntas y para automatizar una fase del ciclo de vida del software o una clase particular de trabajo del software.

Debido a la falta de experiencia del personal de desarrollo en herramientas automáticas de análisis y diseño de un sistema, no fue posible automatizar dichas fases del ciclo de vida del Scysa, únicamente se utilizó una herramienta de software la cual permite almacenar las especificaciones de diseño, de tal forma que a medida que se alimenta a la herramienta con tales especificaciones, al mismo tiempo se van creando las bases de datos, sus relaciones, los reportes, las pantallas de captura y de consulta y el diccionario de datos, para finalmente indicar a la herramienta que genere el código de los programas necesarios que compondrán la aplicación, así como las bases de datos y sus correspondientes índices.

Una vez generada la aplicación, el manual de operación se puede obtener también de la herramienta, ya que ésta se basa en el código que generó y automáticamente imprime en archivo o en papel dicho manual.

La herramienta generador de aplicaciones a la que se ha hecho mención se conoce como "Zachary" y el código que genera es Clipper 5.0.

4.4.2.2 Métricas

El desarrollo del sistema de control y seguimiento administrativo (SCYSA), dio inicio a principios del mes de septiembre de 1993. Solo se ha desarrollado una versión del sistema, ya que prácticamente todos los requerimientos fueron cubiertos durante el análisis y diseño del sistema, por lo que la modificaciones que se han requerido no han sido de fondo, sino de forma, es decir, cambios en el diseño de pantallas, en el diseño de los reportes y nuevos reportes, lo cual no ha representado ninguna complejidad, pues absolutamente todas las especificaciones de diseño se encuentran almacenadas y documentadas en la herramienta Zachary, lo cual ha facilitado las modificaciones y la incorporación de los nuevos reportes.

Los datos que se presentan a continuación fueron recopilados de la primera y única versión del sistema la cual se llevó a cabo en 4 meses; están divididos por fases y serán utilizados para definir las métricas de productividad y de calidad:

ANÁLISIS

Fecha de inicio de la fase	Septiembre - 93
No. de diagramas de flujo de información	10
Personas que intervinieron en el análisis del Scysa	1
Duración de la fase antes de pasar a la fase de diseño	20 días
No. de catálogos a desarrollar	4
No. de procesos a desarrollar	4
No. de reportes a desarrollar	4

DISEÑO

Fecha de inicio de la fase de diseño	Octubre - 93
Modelos de entidad - relación	2
Personas que intervinieron en el diseño	1
Duración de la fase antes de pasar a la fase de programación	20 días
Cantidad de bases de datos	12
No. de índices por bases de datos (promedio)	2
No. de campos no utilizados por base de datos (promedio)	0
No. de formatos de diseño de reportes	4
No. de diagramas de flujo de los procesos automatizados	0
Herramienta de diseño utilizada	Zachary

PROGRAMACIÓN

Fecha de inicio de la fase de programación	Noviembre - 93
No. de personas que programaron	1
Lenguaje de programación utilizado	Clipper 5.0
Experiencia de los programadores en el lenguaje utilizado	4 años
Experiencia en programación para red	3 años
Tiempo que duró la programación	20 días
Programas documentados	todos
Sueldo del desarrollador	3,000.00
Herramienta de programación utilizada	Zachary

PRUEBAS

Fecha de inicio de la fase de pruebas	Diciembre - 93
Tiempo de duración de las pruebas antes de la instalación	15 días
Persona que realizaba las pruebas de unidad (de cada programa)	el desarrollador
Persona que realizaba las pruebas de integración	el desarrollador
Tiempo de duración de las pruebas de unidad e integración	5 días
Errores encontrados por proceso, reporte o consulta	0
¿Se realizó una prueba de operación antes de la instalación? (con usuarios finales)	Si
Tiempo de capacitación a los usuarios para realizar la prueba	5 días
No. de usuarios que intervinieron en la prueba de operación	3
Duración de la prueba de operación	5 días
Si surgieron nuevos requerimientos durante la prueba de operación, ¿de qué tipo fueron?	Diseño de reportes y pantallas.
Tiempo requerido para realizar los nuevos requerimientos	5 días

INSTALACIÓN

Fecha de inicio de la fase de instalación	Enero - 94
Duración de la instalación	2 días
No. de redes donde se instaló el Scysa	1
No de personas que instalaron el sistema	1
Tiempo de capacitación al usuario en el sistema ya instalado	5 días

DOCUMENTACIÓN

Existe documentación actualizada de:

La operación del sistema	Si
La instalación del sistema	Si
Bases de Datos e índices correspondientes	Si
Programas desarrollados	Si
Módulos del sistema	Si
Diagramas de flujo de información	No
Errores encontrados	No
Errores corregidos	No

MANTENIMIENTO

En cuanto al mantenimiento, no tuve la oportunidad de obtener información detallada del mismo, ya que la persona quien me facilitaba el acceso a la información del Scysa (el desarrollador), dejó de prestar servicios a Ericsson en el mes de Marzo de 1994, por lo que solo pude observar que en el tiempo posterior a la instalación del sistema y antes de la fecha mencionada, la operación de éste no sufrió contratiempos ni surgieron nuevos requerimientos.

En cuanto a las métricas de productividad solo mencionare una basada en los datos recopilados durante el desarrollo del SCYSA:

Costo del desarrollo

1 programador * Tiempo de duración del proyecto * sueldo mensual =
1 * 4 * $ 3,000.00 = $ 12,000.00

Debido a que no hubo necesidad de invertir en hardware y software para desarrollar y dado que una sola persona realizo todas las fases del ciclo de vida del proyecto, entonces el costo del sistema se reduce al sueldo del programador durante cuatro meses de duración del desarrollo del SCYSA.

A continuación se presenta el conjunto de métricas de calidad o indicadores individuales de fallas del sistema en relación con las caracteristicas de calidad de nivel más bajo.

EXACTO

El sistema es exacto, ya que en todas las salidas que produce (reportes en pantalla y en papel) la información es precisa. Tuve la oportunidad de probar todos los reportes impresos que genera el sistema y verificar su exactitud con las consultas en pantalla, con las facturas emitidas por el mismo sistema, con los costos de equipo y material almacenados en los catálogos del sistema y con las listas de precios proporcionadas por el personal encargado de utilizar el sistema.

COMPLETO

Todos los módulos diseñados y especificados para componer el sistema están completamente desarrollados y en operación.

AUTOCONTENIDO

El sistema presenta "ventanas" con texto de ayuda de operación del mismo con solo presionar la tecla <F1> en el momento que el cursor se encuentra posicionado en cualquiera de los campos que requieren la captura de datos por parte del usuario.

ROBUSTO / INTEGRO

Cualquier dato que el usuario captura es analizado por el sistema, enviando un mensaje de aviso por error de tipo de dato capturado, si la información proporcionada por el usuario es incorrecta. Es posible corregir desde el sistema la precisión de los datos reportados, ya que la única forma en que pudieron ser producidos es por la captura de datos falsos en el sistema.

INDEPENDENCIA DE DISPOSITIVOS

Este sistema fue desarrollado únicamente para la Dirección de Sistemas de Radio, la cual contaba con el acceso a una red de computadoras con sistema operativo Novell, por lo anterior, la configuración planeada fue la requerida para aprovechar los recursos mencionados. El funcionamiento del Scysa no requiere de ningún software en particular, ni la presencia física de algún manejador de bases de datos, ni de los propios programas compilados que componen la aplicación, ya que existe un archivo ejecutable del sistema.

CONSISTENTE

El sistema presenta un estándar en las pantallas de presentación, así como en los reportes, los menús y código, ya que todo esto fue creado por el generador de aplicaciones Zachary, el cual proporciona un estándar de presentación en pantalla, independientemente de la facilidad de diseñar uno propio para cada aplicación, así como de reportes y de menús.

MEDIBLE

El tiempo de respuesta del sistema no fue diferente al ser utilizado por un usuario y por cuatro usuarios a la vez. Las bases de datos están totalmente normalizadas, no existen campos sin uso alguno, ni campos innecesariamente repetidos y en promedio cada base de datos tiene dos índices.

ACCESIBLE

Debido a que la aplicación fue creada con ayuda del generador de aplicaciones 'Zachary', la necesidad de ejecutar cada módulo en forma individual resulta laborioso, ya que el generador almacena absolutamente todas las especificaciones del sistema en forma conjunta e interrelacionada, entonces para ejecutar un módulo o programa desde el generador en forma individual sería necesario cancelar las especificaciones no requeridas y posteriormente volver a definirlas.

A pesar de lo anterior el acceso a los programas, así como su ejecución no dependen del generador, ya que utilizando cualquier editor de programas o algún manejador de bases de datos como FOX, es posible accesar cada programa, función o procedimiento generado por Zachary y con el compilador de Clipper 5.0 o mayor, la ejecución de cada uno de ellos es relativamente muy sencilla.

EFICIENCIA DE DISPOSITIVOS

En la Dirección de Sistemas de Radio se contaba con el acceso a una red de computadoras antes de desarrollar la aplicación, así que desde un principio se buscó explotar al máximo los recursos con que se contaba, por lo que el sistema se adaptó a la configuración ya existente.

COMUNICABLE

Desde el momento que se accesa al sistema, se presentan mensajes indicativos al usuario, ya sea durante una consulta, una captura, una emisión de reportes o durante la simple 'navegación' por los menús. Por otra parte orienta al usuario cuando éste comete errores de captura y siempre le solicita verifique todos los datos capturados antes de actualizar la información almacenada en los archivos o bases de datos.

AUTODESCRIPTIVO

El usuario fácilmente puede determinar el proceso que va a realizar en cada una de las opciones que presentan los menús del sistema, así como los datos que requiere para utilizar cada una de dichas opciones.

ESTRUCTURADO

El lenguaje utilizado fue clipper, por lo que se facilita el uso de programación estructurada y por otra parte ya que el código fue generado por Zachary absolutamente todos los programas presentan un código altamente estructurado.

CONCISO

Debido a que la aplicación se creó con ayuda del generador Zachary, el código se optimizó hasta donde fue posible, no existen rutinas repetidas y las funciones creadas específicamente para el sistema se almacenaron en un solo archivo. Por otra parte los módulos diseñados fueron únicamente los requeridos, por ello no existe redundancia.

LEGIBLE

En cualquiera de los programas generados es posible identificar los procesos que realiza, ya que todas las funciones y variables están documentadas, aunado a esto lo estructurado del código facilita hacer un seguimiento de las operaciones que realiza cada programa.

INCREMENTABLE

El sistema es totalmente incrementable ya sea en módulos o en variables pues clipper no presenta límites en cuanto al número de módulos, además de que el sistema solo cuenta con cuatro; y en cuanto al número de variables, de ello se encarga Zachary pues optimiza al máximo el uso de las mismas en cada programa.

4.4.3. Comparación.

Aunque las dos aplicaciones presentadas son de tipo administrativo existe una diferencia básica entre ambas, el SIRAN en enfoca a la inscripción o registro en el R.A.N. de los acuerdos agrarios de tipo legal entre ejidatarios o comuneros y el SCYSA se encarga de cuestiones contables como saldos y facturación.

Después de probar ambos sistemas y analizar la documentación, programas y bases de datos respectivos, considero que el SIRAN implica un mayor nivel de complejidad ya que el diseño de las 'inscripciones' de los acuerdos de asamblea ejidal o comunal deben alinearse a lo establecido en la ley agraria, lo cual representó el principal reto de automatización debido a los siguientes factores: primero, el administrador del sistema fue el encargado de diseñar los formatos de inscripción para cada uno de los posibles acuerdos de asamblea a tramitar en el R.A.N; y segundo, el administrador es ingeniero, no abogado, por ello fue mayor la necesidad de discutir con el cliente, en este caso grupo de abogados del R.A.N., las inscripciones una vez diseñadas, lo que implicó numerosas modificaciones durante el avance del proyecto, principalmente debido a que en la interpretación de la ley influye mucho el criterio de cada abogado.

En cuanto al SCYSA la complejidad es mucho menor comparada con el SIRAN, debido a que trata problemas tipicos en cualquier organización como lo son el control de saldos de los clientes, la emisión de facturas, por venta e instalación de equipo de radio comunicación en este caso, y la generación de presupuestos para diversos proyectos de radio-comunicación.

Aún cuando el SIRAN es considerablemente mayor tanto en dimensiones como en complejidad el desarrollador solo asignó 10 días a la fase de análisis de la cuarta versión, lo que representa la mitad del tiempo asignado para el análisis del SCYSA.

Para las fases de diseño y programación ocurrió lo mismo que en el análisis, la mitad del tiempo de diseño y programación asignado para el SCYSA fue el tiempo asignado para el SIRAN. En la fase de programación hubo diferencias en el número de desarrolladores, lo cual es lógico por las dimensiones del SIRAN, sin embargo una diferencia notable en contra del SIRAN es la falta de experiencia tanto en el lenguaje utilizado, como en la programación para Red.

En relación con las pruebas, el tiempo asignado fue mayor para el SCYSA, además de que lo probaron usuarios finales antes de su instalación, a diferencia del SIRAN que fue probado solo por el equipo de desarrollo y además muy poco tiempo en relación con su tamaño y con el SCYSA.

Durante la fase de instalación no hubo problemas con ningún sistema, la diferencia radica en el tiempo requerido para llevarla a cabo pues el SIRAN se instaló en 32 redes de cómputo en comparación con el SCYSA que solo se instaló en una red, lo que en tiempo, dinero y esfuerzo resultó mucho mayor para el SIRAN.

El SCYSA tiene documentación de todas las fases de desarrollo y el SIRAN solo de las bases de datos y del No. y nombre de programas desarrollados.

El mantenimiento del SIRAN requirió de una considerable inversión económica, así como de tiempo y esfuerzo principalmente para corregir errores de programación no detectados durante las pruebas del sistema; en cuanto al SCYSA, éste no requirió mantenimiento correctivo ni adaptativo en un lapso de 2 meses a partir de su puesta en operación.

A pesar de que el SIRAN se desarrolló e instaló aproximadamente en 1 1/2 meses, éste costó más que el SCYSA por el número de personas que participaron en su creación y por los gastos de transporte, hospedaje y alimentación de 8 personas durante 12 días, además de los costos generados a causa del mantenimiento correctivo principalmente; en cambio el SCYSA se desarrollo en 4 meses, con un costo mínimo y sin gastos de mantenimiento al menos a corto plazo.

En seguida se presenta un análisis global de las características de calidad de segundo nivel tomando en cuenta las métricas o indicadores individuales de fallas mencionados para cada característica primitiva o de nivel más bajo de cada aplicación:

PORTABLE

SIRAN
Considero que después de mencionar las fallas en relación con lo autocontenido del sistema y su independencia de dispositivos, éste no se considera portable; sin embargo ya que el sistema esta diseñado y desarrollado a la medida para el Registro Agrario Nacional, no es de gran importancia que sea portable, pues desde un principio se pensó en un sistema que funcionara en ambiente de red de computadoras y así se diseño, además de que no es una aplicación que pueda ser comercializada por el tipo de procesos tan particulares que solo se realizan en el R.A.N.

SCYSA
El sistema no es portable si se piensa instalarlo en equipo mainframe, pues fue diseñado para PC y redes, sin embargo esto no es de importancia pues solo será utilizado para controlar el área encargada de dar seguimiento a los costos, presupuestos y saldos relacionados con la venta e instalación de equipos y materiales de radio-comunicación de Ericsson.

CONFIABLE

SIRAN
La confiabilidad del sistema es bastante dudosa, ya que ni siquiera un reporte o una consulta en pantalla refleja información correcta, pues los cálculos no son exactos, y los nombres de ejidatarios, ejidos, comunidades y estados entre otros se presentan alterados, es decir, incompletos, equivocados, y hasta totalmente ausentes del reporte o texto en pantalla. Por otra parte no es posible hacer modificaciones por error de captura en procesos tan utilizados e importantes como lo son la delimitación de un ejido o comunidad, así como la inscripción de parcelas y solares urbanos.

SCYSA
El sistema es confiable ya que la aplicación ejecuta todos los procesos en forma correcta, produce salida precisas y controla el error de captura, por otra parte, controla el acceso directo a los archivos ya que estos se encuentran almacenados en un área del disco duro conocida únicamente por el administrador de la red.

EFICIENTE

SIRAN
De los dispositivos configurados para trabajar con la aplicación, todos se utilizan, por ello considero que la inversión hecha en equipo de cómputo es recuperable, sin embargo existe desperdicio en tiempo del usuario y en espacio de almacenamiento, es decir, cuando se trabaja en forma conjunta, el tiempo de respuesta se triplica y en cuanto al espacio, las bases de datos tienen campos que no se utilizan y por consiguiente se ocupa espacio innecesariamente.

SCYSA
El Scysa no representa en ningún momento desperdicio de espacio de almacenamiento pues las bases de datos están normalizadas; gracias a lo anterior y a que el código generado por Zachary es el más óptimo posible, el tiempo de respuesta es inmediato, ya sea durante la captura y actualización de información o durante las consultas en pantalla y en las emisiones de reportes impresos.

INGENIERÍA DE RECURSOS HUMANOS

SIRAN
El sistema omite la presencia de mensajes, ya sea de tipo informativo de los procesos que en un momento dado realiza la máquina, o del tipo indicativos al usuario de tal forma que él mismo sepa que hacer en todo momento. Por otra parte el esfuerzo y tiempo de cada una de las personas que utiliza el sistema, cuando trabajan en forma conjunta, se desperdicia ya que el tiempo de respuesta es lento comparado con el tiempo necesario durante la operación de la aplicación con tres o menos personas. En general el sistema cumple con los objetivos planteados, pero en forma imprecisa, lo cual provoca la necesidad de verificar manualmente los datos proporcionados en pantalla y en forma impresa, lo que implica un desperdicio de tiempo y esfuerzo.

SCYSA
La aplicación cumple con los objetivos planteados, ya que evita el uso excesivo de hojas de cálculo y procesadores de texto para realizar los reportes del avance de los proyectos, de tal forma que es posible dar seguimiento a éstos de manera instantánea, sin necesidad de tener hojas y hojas de cálculo, así como archivos de texto para cada proyecto de venta e instalación de equipo. Lo anterior representa un ahorro de tiempo para los usuarios ya que solo será necesario actualizar al sistema con las cantidades correspondientes a los anticipos y pagos realizados por los clientes, para que automáticamente se genere una

factura, se actualicen los saldos y se emita un reporte del estado actual del proyecto en cuanto al costo.

EVALUABLE

SIRAN
A pesar de que el sistema no incluye comentarios al usuario del procedimiento que debe seguir dentro de cada módulo, pienso que si se puede evaluar ya que me fue posible determinar el promedio de campos e índices no utilizados por base de datos, verificar la exactitud con ayuda de un calculadora de las operaciones aritméticas que realiza el sistema tanto en la captura de superficies como en los datos presentados en pantalla o en reportes y ya que es estructurado pude determinar, con un poco de trabajo, que los procesos de búsqueda para las consultas y emisiones de reportes es bastante lento cuando se utiliza el comando set filter de FOX, o cuando abren y cierran una misma bases de datos más de dos veces en el mismo programa y cuando trabajan en forma conjunta más de tres personas.

SCYSA
Se pudo observar el tiempo de respuesta del sistema así como el espacio ocupado por los archivos (aprox. por proyecto), realizando pruebas de operación bajo diversas cargas de trabajo; por otra parte haciendo un seguimiento al código de cada programa se pudo observar lo estructurado y accesible de cada proceso; finalmente ya que el sistema es comunicable y autodescriptivo considero que la aplicación es evaluable.

ENTENDIBLE

SIRAN
He mencionado que el diseño modular del sistema es acorde a los objetivos de la aplicación, sin embargo esto no es suficiente para que el código del mismo pueda ser entendido por cualquier persona. Los programas que componen esta aplicación tiene el inconveniente de que fueron realizados por 5 personas sin ningún tipo de supervisión, lo que dio origen principalmente a: que pocos programas estén documentados, a que una función este repetida en todos los programas, a que existan programas casi iguales y variables cuyos nombres no tienen relación con su contenido y principalmente a que existan programas escritos en una forma desordenada, es decir, sin alguna clase de identación, lo cual los hace especialmente difíciles de entender.

SCYSA
La aplicación es totalmente entendible hasta por una persona ajena al tipo de procesos automatizados que realiza, ya que el sistema es totalmente autodescriptivo y conciso, además en cuanto al código, éste es estructurado y fácilmente legible.

MODIFICABLE

SIRAN
A pesar de que el código de los programas es difícil de entender, es posible modificarlos con solo invertir mayor tiempo y esfuerzo ya que el lenguaje es estructurado y FOX no presenta límites en el número de módulos o en el tamaño de cada programa.

SCYSA
Ya que el código es estructurado y legible, éste puede ser modificable ya sea en forma directa sobre el programa, o desde la herramienta Zachary, lo cual es mucho más sencillo y recomendable pues se trabaja directamente con las especificaciones de diseño, con las bases de datos y con el diccionario de datos, lo cual representa un ahorro de tiempo y esfuerzo del personal encargado del mantenimiento, pues en un solo paso mientras se actualizan las especificaciones, automáticamente se están modificando los programas asociados, así como el manual de operación.

Para finalizar a continuación se presenta el análisis de las dos características de calidad de primer nivel para ambos sistemas.

UTILIDAD

SIRAN
Después de analizar cada una de las características de calidad, es obvio que el sistema no es nada confiable, tanto que en cuatro delegaciones en las que tuve oportunidad de trabajar, pude observar que verificaban los datos impresos en los reportes con una calculadora y si estaban mal a veces preferían hacerlo a mano antes de enviarlos a su jefe inmediato. Por otra parte esta claro que existe desperdicio en el tiempo y esfuerzo de los usuarios pues prácticamente siempre utilizan el sistema más de seis personas; y en cuanto al desempeño de la aplicación también existe desperdicio pues las bases de datos no están normalizadas.

SCYSA
El sistema es de gran utilidad para el personal encargado del "bakclog" o seguimiento a los proyectos, pues es confiable y eficiente, no existe desperdicio de tiempo y esfuerzo por parte de los usuarios, y además de producir salidas precisas y optimizar el espacio, cumple con los objetivos planteados de controlar y evaluar los procedimientos y productividad del personal administrativo de la dirección de Sistemas de Radio.

MANTENIBILIDAD

SIRAN
A pesar de que el código es difícil de entender por lo desordenado de su escritura y por la falta de documentación del mismo, considero que el sistema es mantenible, pues fue posible identificar las fallas que tiene, las cuales poco a poco se han ido corrigiendo. Sin embargo, a medida que se han ido corrigiendo la mayoría de los errores identificados antes de dar inicio al desarrollo de la cuarta versión, se han provocado otros sin pensar, principalmente por falta de documentación, ya que esta es necesaria pues los programadores iniciales del sistema ya no trabajan en la empresa que desarrollo el sistema. Por lo anterior, al modificar un programa sin actualizar aquellos programas con los que se encuentra interrelacionado, lentamente el no. de errores disminuye, pero nunca desaparece.

SCYSA
Dado que el código de cada uno de los programas que integran esta aplicación es evaluable, entendible y modificable, entonces en sí el sistema cumple con las tres características mencionadas, por ello considero que el sistema es fácilmente mantenible.

CONCLUSIONES

Los problemas asociados con el desarrollo del software son múltiples, y no existe ningún método o técnica que sea perfecto, de tal forma que pueda ser aplicado al desarrollo de distintos tipos de aplicaciones y que a su vez minimice dificultades tan comunes como el desfasamiento en el calendario de actividades, sobregiro en el presupuesto asignado para su desarrollo y falta de calidad en el producto terminado.

A pesar de lo anterior, existen técnicas y herramientas que se pueden utilizar en conjunto para lograr el éxito de un producto de software. Sin embargo, antes de utilizar dichas técnicas y herramientas será necesario reconocer la necesidad de cambiar y actualizar la forma de desarrollar los sistemas. Algunos puntos básicos que requieren de especial atención por parte de los administradores de un proyecto se presentan a continuación:

- La rotación constante del personal que integra un equipo de desarrollo provoca un aumento de tiempo en el ciclo de vida de un sistema, ya que será necesario invertir tiempo en capacitación de los nuevos integrantes.

- Por lo anterior la fase del ciclo de vida de desarrollo más afectada es la de mantenimiento, ya que es entonces cuando se requiere mayor inversión de tiempo para entender y mantener todos aquellos programas creados por personas que ya no forman parte del equipo de desarrollo.

- El mantenimiento correctivo y adaptativo de programas genera nuevos errores si no existe documentación actualizada que indique la interrelación de programas, así como de las bases de datos que son afectadas por dichos programas.

- La mayoría de los errores encontrados en un sistema informático son causados por mala programación y permanecen en el sistema mientras no se ponga en práctica algún tipo de plan de pruebas que permita detectar todos los errores de programación antes de entregar la aplicación al usuario final.

Los problemas anteriores se veran solucionados cuando se acepte la idea de que conforme se asigne mayor tiempo a las fases de análisis, diseño y pruebas, el tiempo dedicado a la programación y mantenimiento, así como los costos, disminuirán considerablemente, pues las especificaciones de diseño no sufrirán cambios constantes lo cual evitará la modificación igualmente constante de los programas desarrollados, generando así una aplicación tan completa y eficiente que únicamente requerirá de cambios de forma y no de fondo.

A pesar de que la calidad de un sistema puede ser bastante cuestionable, éste permanecerá en operación si cumple con el objetivo principal de su desarrollo (en el caso del Siran, la emisión de certificados parcelarios y de uso común, así como la de títulos de solar y de propiedad) y permanecerá en uso con mayor razón si el administrador tiene suficiente habilidad para conducirse en el terreno político del cliente, de tal forma que obtenga de éste todo el apoyo para que el sistema se siga utilizando así (el tiempo necesario, así sean años), mientras de corrigen los "detalles" encontrados en el sistema y a la vez se incorporan nuevos requerimientos.

Pienso que el origen de la mayoría de los errores en un sistema lo constituyen las personas encargadas de administrar un proyecto, ya que si no están conscientes de que para ello es necesario estar al tanto del avance de la tecnología, así como de los nuevos métodos y procedimientos relativos a la administración, nunca lograrán obtener un producto de calidad, pues seguirán desperdiciando tiempo en la programación de procesos tan comunes como los relativos a un simple catálogo, lo cual fácil y rápidamente puede ser generado por una herramienta automática; seguirán dejando como última prioridad la documentación y registro de avance de cada fase del ciclo de vida del sistema dado que "no hay tiempo que perder"; seguirán creyendo que no hay tiempo de pensar en estándares en programación, siendo que un generador de código automáticamente puede controlar la estandarización en una aplicación; seguirán asignando un mínimo de tiempo a la fase de pruebas pues no existen controles que certifiquen la prueba de todos y cada uno de los procesos que realiza cada programa que forma parte de la aplicación y en general seguirán pensando que su mente es una computadora tan perfecta que siempre tiene en línea las especificaciones de la aplicación así como los errores cometidos y corregidos, por lo que el registro de datos históricos del avance de un proyecto no es necesario.

Por lo anterior se destaca la importancia de tomar conciencia de que la calidad se logra mejor con una atención cuidadosa en la planeación, análisis, diseño, implantación y pruebas, para lo cual será necesario reconocer las características de calidad que debe poseer un software para que pueda ser considerado de calidad.

BIBLIOGRAFIA

[BOE73] Boehm, B. W., J. R. Brown, H. Kaspar, M. Lipow, G. J. MacLeod, M. J. Merritt, Characteristics of Software Quality, TRW Software Series, 1973.

[BOE76] Boehm, B. W., J. R. Brown, M. Lipow, "Quantitative Evaluation of Software Quality", IEEE 2nd. intertnational Conference on Software Engineering, Octubre 1976.

[BOE81] Boehm B. W., Software Engineering Economics, Prentice Hall, 1981.

[BRO75] Brooks, F. P., The Mythical Man-Month, Addison Wesley, 1975.

[DEM82] DeMarco, T., Controlling Software Projects, Yourdon Press, 1982.

[DIC85] Dickson and Wetherbe, The Management of Information Systems, Mc Graw Hill, 1985.

[FAI85] Fairley, R., Sotware Engineering Concepts, Mc Graw Hill, 1985.

[FRY91] Fry, L. A., Introduction to the Software Metrics, Technology Training S.A. de C.V., 1991.

[HUM89] Humphrey, W. S., Managing the Software Process, Addison Wesley, 1989.

[KIN84] King and Kramer, Evolution and Organitational Information Systems: An Assesment of Nolan's Stage Model, Communication of the ACM, Vol. 27, No. 5, 1984.

[PRE87] Pressman, R. S., Software Engineering - A Practitioner's Approach, Mc Graw Hill, 1987.

[SOM85] Sommerville, I., Ingeniería de Software, Addison Wesley Iberoamericana, 1985.

[VON90] Von Mayrhauser, A., Software Engineering, Methods and Management, Academic Press, Inc., 1990.

[ZEL84] Zelkowitz, M. V., Software Engineering, Auerbach Publishers Inc., 1984

La lista de libros que se presentan a continuación no se consultaron para la realización del presente trabajo, simplemente se incluyen en apoyo a la sección 4.3

[ALF85] Alford, M., SREM at the Age og Eight; The Distributed Computing Design System, Computer, Vol. 18, 1985.

[BAL82] Balzer, R. M., Goldman, N. M., and Wile, D. S., Operational Specification As the Basis for Rapid Prototyping, ACM software Engineering Notes, 1982.

[BER84] Bersoff, E. H., Elements of Software Configuration Management, IEEE Software Engineering, 1984.

[BOE84] Boehm, B. W., Prototyping vs. Specifying: Some Experimental Results and Software Life Cycle Implications, 1st. Software Process Workshop, Reino Uniod, 1984.

[DEM79] DeMarco, T., Structured Analysis and Systems Specification, Prentice Hall, 1979.

[DIJ68] Dijkstra, E., The Structure of the "THE" Multiprogramming System, ACM, vol. 18, 1968.

[DUN84] Dunn, R., Software Defect Removal, McGraw Hill, 1984.

[FEL79] Feldman, J., MAKE, A Program for Maintainig Computer Programs, Software Practice and Experience, 1979.

[GAN82] Gane, T. and C. Sarson, Structured System Analysis, McDonnell Douglas, 1982.

[GOM84] Gomaa, H., A Software Desing Method for Real Time Systems, CACM, vol. 27, 1984.

[JAC83] Jackson, M. A., System Development, Prentice Hall, 1983

[LIN79] Linger, R. C., Mills, H. D., and Witt, B. I., Structured Programming - Theoty and Practice, Addison Wesley, 1979.

[MAR85] Martin, J., Fourth Generation Languages, vol. I, Prentice Hall, 1985.

[MCC85] McCabe, T. J., Structured Real Time Analysis and Design, COMPSAC-85, IEEE, 1985.

[MCG79] McGuffin, R. W., Elliston, A. E., Tranter, B. R., and P. N. Weatmacott, CADES, Software Emgineering in Practice, 4ht. International Conference Software Emgineering, 1979.

[MYE79] Myers, G., The Art of Software Tersting, Wiley. 1979.

[ORR77] Orr, K. T., Structured Systems Development, Yourdon Press, New York, 1977.

[ORR81] Orr, K., Structured Requirements Definition, Ken Orr and Associates, Inc., Topeka, 1981.

[OST76] Osterweil, L. and L. Fosdick, DAVE, A Validation Error Detection and Documentation System for FORTRAN Programs, Software-Practice and Experience, 1976.

[PAG80] Page Jones, M., The Practical Guide to Structured Systems Design, Yourdon Press, New Yoek, 1980.

[RIT78] Ritchie, D. M., Johnson, S. C., Lesk, M. E., and Kernighan, B. W., The C Programming Language, Bell Systems Tech., 1978.

[ROC75] Rochking, M., The Source Code Control Systems, IEEE Software ENgineering, 1975.

[SEI85] Seivert, G. E. and T. A. Mizelli, Specification-based Software Engineering with TAGS, Computer, Vol. 18, 1985

[SHN80] Shniederman, B., Software Psychology, Winthrop Publishers, 1980.

[TEI81] Teitelbaum, T. and T. Reps, The Cornell Program Synthesizer: A syntax-directed programming enviroment, ACM, 1981.

[WAR74] Warnier, J. D., Logical, Construction of Programs, Van Nostrand Reinhold, 1974.

[WAS82] Wasserman, A. I., and Shewmake, D. T., Rapid Prototyping of Interactive Information Systems, ACM Software Engineering Notes, 1982.

[WHI85] White, S. M., and J. Z. Lavi, Embedded Computer System Requirements Workshop, Computer, vol. 18, 1985.

[YOU75] Yourdon, E., Techiniques of Program Structure and Design, New Jersey, Prentice Hall, 1979.

[YOU78] Yourdon, E. and L. Constantine, Structured Design, Yourdon Press, 1978.