UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES
CUAUTITLAN

"CALIDAD EN LA PRODUCCION DE SOFTWARE"

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO MECANICO ELECTRICISTA

P R E S E N T A :

OMAR GARCIA SAAVEDRA

ASESORA: M. EN C. FRIDA MARIA LEON RODRIGUEZ

CUAUTITLAN IZCALLI, EDO. DE MEX. NOVIEMBRE 1997

UNIVERSIDAD NACIONAL AVTNPMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN

UNIDAD DE LA ADMINISTRACION ESCOLAR

DEPARTAMENTO DE EXAMENES PROFESIONALES

U. N. A. M.
FACULTAD DE ESTUDIOS SUPERIORES-CUAUTITLAN

DEPARTAMENTO DE EXAMENES PROFESIONALES

ASUNTO: VOTOS APROBATORIOS

DR. JAIME KELLER TORRES
DIRECTOR DE LA FES-CUAUTITLAN
P R E S E N T E .

AT'N: Ing. Rafael Rodriguez Caballos
Jefe del Departamento de Examenes
Profesionales de la F.E.S. - C.

Con base en el art. 28 del Reglamento General de Exámenes, nos permitimos comunicar a usted que revisamos la TESIS TITULADA:

"Calidad en la producción de software"

que presenta el pasante: Omar García Saavedra

con número de cuenta: 8736483-9 para obtener el TITULO de:

Ingeniero Mecánico Electricista .

Considerando que dicha tesis reúne los requisitos necesarios para ser discutida en el EXAMEN PROFESIONAL correspondiente, otorgamos nuestro VOTO APROBATORIO.

A T E N T A M E N T E .
"POR MI RAZA HABLARA EL ESPIRITU"
Cuautitlán Izcalli, Edo. de Méx., a 31 de Octubre de 1996

PRESIDENTE	M. en C. Frida María León Rodríguez	
VOCAL	MVZ. José Fernando Altamirano Abarca	
SECRETARIO	Ing. Jorge Altamira Ibarra	
PRIMER SUPLENTE	QFB. Gabriela Ponce Anguiano	
SEGUNDO SUPLENTE	Ing. Rolando Cortés Montes de Oca	

UAE/DEP/VAP/98

DEDICATORIA

A Emilia García Lopez, como un homenaje a quien siempre confió en que lo lograría y contribuyó en gran parte de mi formación no profesional.

A mi madre y a mi hermana como testimonio de mi cariño.

AGRADECIMIENTOS

A la Universidad Nacional Autónoma de México, en especial a la Facultad de Estudios Superiores Cuautitlan por haber permitido mi formación profesional.

A mis profesores: Por haber compartido conmigo sus conocimientos.

A mi asesora: Por su empeño para el desarrollo y corrección del presente trabajo.

A mi madre, Ernestina Saavedra García: Por su cariño, confianza y apoyo invaluables.

A mi hermana, Verónica García Saavedra: Por que siempre me alentó a seguir adelante.

A mi padre y demás familiares: Por su apoyo moral.

A mis amigos: Carlos Aguilar, Enrique Leglisse, Alberto Vazquez, Claudia Hernández, Gloria Islas y Gabriela; por su amistad sincera.

Al Sr. Carlos Ochoa y a la Sra. Guadalupe Roldán, de quienes he recibido un excelente trato y un gran apoyo.

A Guadalupe Ochoa Roldán: Por que sin su compañía nada hubierá sido igual.

A mis compañeros de trabajo: Por su camaradería.

Al Lic. Antonio Cruz Benigno: Por su paciencia y apoyo brindado para mi formación al estar bajo sus ordenes.

A Ivonne Montiel Galicia: Por su compresión y ayuda.

CALIDAD EN LA PRODUCCIÓN DE SOFTWARE

INDICE

CAPITULO I

I.- INTRODUCCIÓN

1.1.- Antecedentes históricos de la ingeniería del software

A medida que la tecnología ha ido evolucionando no es difícil encontrarnos con computadoras cada vez mas novedosas, cuyas capacidades eran prácticamente insospechadas hace algunos años.

En la actualidad se puede decir que el software está aventajando al hardware como factor decisivo para la obtención y procesamiento de información de la manera más rápida, confiable y económica.

El software es el instrumento que nos permite darle vida a la computadora, ya que resultaría prácticamente de ninguna utilidad tener la computadora más moderna, con el procesador de más potencial de cálculo y velocidad del mercado, con una cantidad de almacenamiento y memoria exorbitantes, así como los periféricos más innovadores y costosos, si no se cuenta con el software adecuado que permita obtener el máximo rendimiento de las capacidades del hardware, es por esto que se dice que el software marca la diferencia.

En estos tiempos en que las empresas buscan ser cada día más competitivas "lo que diferencia a una compañía de su competidora es la suficiencia y oportunidad de la información dada por el software (y bases de datos relacionales)."

Evolución del Software

Resulta muy difícil hablar de la evolución que ha sufrido el software sin dedicar algunas líneas a la evolución del hardware, pues es de acuerdo a las capacidades y necesidades de éste, que se produce el software.

Básicamente podemos considerar 5 generaciones de computadoras:

1a. Generación:

La primera generación de computadoras marco el inicio de esta gran industria y sus primeras aplicaciones eran básicamente militares, se dio a partir de los finales de la década de los 50's

y hasta finales de la decada de los 60's. Su tecnología se basaba en los bulbos y requería de grandes sistemas de enfriamiento. Las dimensiones de una computadora alcanzaban las de un edificio por ejemplo la ENIAC o la UNIVAC, computadoras de bulbos cuya programación se efectuaba en lenguaje binario.

2a. Generación:

La segunda generación de computadoras se dio con la aparición de la tecnología del transistor la cual permitia tener en un espacio reducido el equivalente a una gran cantidad de bulbos. esta generación se dio durante los años 60's y durante ella se incrementó la capacidad de almacenamiento.

3a. Generación:

Durante la tercera generación de computadoras que comprende aproximadamente desde el final de la década de los 60's y principios de los 70's, la tecnología avanza de nuevo y esta vez nos trae el chip o circuito integrado, dispositivo que permite tener en un espacio muy reducido el equivalente a una gran cantidad de transistores.

4a. Generación:

Durante la cuarta generación de computadoras que comprende aproximadamente de los años 1972 a 1984, aparecieron otra vez nuevas tecnologías, en esta ocasión marco la nueva etapa la aparición del microprocesador, gracias al cual tuvimos la llegada de las computadoras personales, dicho microprocesador nos permite tener en un espacio muy reducido el equivalente de varios circuitos integrados.

5a. generación:

Es la generación que vivimos en la actualidad y en ella se manejan desde la arquitectura, diseños especiales, manejo de lenguaje natural, hasta sistemas de inteligencia artificial y sistemas expertos.

1a. Generacion	**CÓDIGO MÁQUINA** 10101110 10010001
2a. Generación	**LENGUAJE ENSAMBLADOR** 8085, Z80, 68000, ETC.
3a. Generación	**LENGUAJES DE ALTO NIVEL** Procedural (Instrucciones línea por línea) Pascal CORAL 66 BASIC etc. Declarativo (estado del problema) LISP Hope PROLOG (FORTH) ORIENTADO A OBJETOS (modelos) SMALLTALK C++ EIFFEL
4a. Generación	**BASES DE DATOS** CICS SQL
5a. Generación	**INTELIGENCIA ARTIFICIAL Y** **PROCESAMIENTO PARALELO**

Tabla I.1.- Rango de lenguajes para sistemas de software

El cuadro anterior nos muestra una visión de Mark Norris y Peter Rigby de como han evolucionado propiamente los lenguajes de programación a través de las 5 generaciones de computadoras.

Regresando a la evolución del software podemos decir que en los primeros años de desarrollo de software, éste se desarrollaba a medida para cada aplicación y prácticamente no se tenían métodos sistemáticos.

Para finales de los años 70's el software ya se desarrollaba para una amplia distribución y es entonces que aparecen empresas dedicadas plenamente al desarrollo de software.

Con la llegada de las computadoras personales, el hardware de éstas se ha ido convirtiendo en equipo estandarizado y lo que realmente marca una gran diferencia es el software que se

suministre con ellas; ahora es común que se gaste más en el software de aplicación que en la computadora misma.

Está avanzando con mayor rapidez el desarrollo de nuevas tecnologías que permiten un hardware de mayor potencial, sin que haya sido posible mantener el ritmo del desarrollo de software que explote a un 100% las capacidades de este nuevo hardware.

A continuación se presenta un análisis de la evolución del software presentado por Roger S. Pressman en su libro Ingeniería del Software en el que clasifica la evolución del software en 4 etapas:

Los primeros años	La segunda era	La tercera era	La cuarta era
• Orientación x lotes • Distribución limitada • Software a medida	• Multiusuario • Tiempo real • Bases de datos • Software como producto	• Sistemas distribuidos • Incorporación de "inteligencia" • Hardware de bajo costo • Impacto en el consumo	• Potentes sistemas de sobremesa • Tecnologías orientadas a objetos • Sistemas expertos • Redes neuronales artificiales • Computación paralela

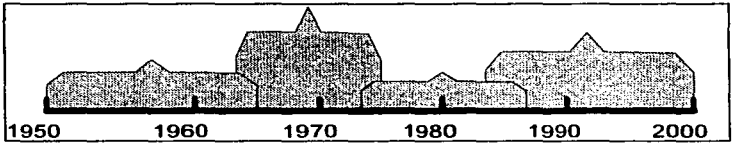


Figura I.1.- Evolución del software según Pressman

En la primera etapa dice Pressman, durante los primeros años de desarrollo de computadoras, el hardware sufrió continuos cambios, mientras que el software se contemplaba simplemente como un añadido, el desarrollo de software se realizaba virtualmente sin métodos sistemáticos y sin ninguna planificación. Durante este periodo se utilizaba una orientación por lotes. La mayor parte de hardware se dedicaba a la ejecución de un solo programa para una aplicación específica y lo normal era de esperarse que el hardware fuera de propósito general. Por otra

parte, el software se diseñaba a medida para cada aplicación y tenia una distribución relativamente pequeña.

El software como producto (es decir, programas para ser vendidos a uno o más clientes) estaba en su infancia, la mayoría del software se desarrollaba y era utilizado por la misma persona u organización. La misma persona lo escribía, lo ejecutaba y si fallaba lo depuraba. La documentación normalmente no existía.

En la segunda era tenemos la multiprogramación y los sistemas multiusuario. Los procesos de tiempo real podían recoger, analizar y transformar datos de múltiples fuentes, controlando así los procesos y produciendo salidas en milisegundos en lugar de minutos. Los avances en los dispositivos de almacenamiento en línea condujeron a la primera generación de sistemas de gestión de bases de datos.

El software ya se desarrollaba para tener una amplia distribución en un mercado multidisciplinario. Los programas se distribuían para computadoras grandes y para minicomputadoras a cientos e incluso miles de usuarios.

Todos esos programas tenían que ser corregidos cuando se detectaban fallas, modificados cuando cambiaban los requisitos de los usuarios o adaptados a nuevos dispositivos de hardware que se hubieran adquirido. Estas actividades se llamaron colectivamente mantenimiento del software. El esfuerzo gastado en el mantenimiento del software comenzó a absorber recursos en una forma alarmante.

La tercera era se caracteriza por la llegada de los microprocesadores y las computadoras personales, las computadoras personales han sido el catalizador del gran crecimiento de muchas compañías de software. Mientras que las compañías de software de la segunda era vendían cientos o miles de copias de sus programas, las compañías de software de la tercera era venden decenas e incluso cientos de miles de copias de sus programas.

La cuarta era del software de computadora es la actual, las técnicas orientadas a los objetos están teniendo mucho auge, los sistemas expertos y el software de inteligencia artificial se han trasladado del laboratorio a las aplicaciones prácticas, para un amplio rango de problemas del mundo real.

Conforme avanza esta era continúan intensificándose los problemas asociados con el software de computadoras:

- La sofisticación del hardware ha dejado desfasada nuestra capacidad de construir software que puede explotar el potencial del hardware.
- Nuestra capacidad de construir nuevos programas no puede dar abasto a la demanda de nuevos programas.
- Nuestra capacidad de mantener los programas existentes está amenazada por el mal diseño y el uso de recursos inadecuados.

Como respuesta a la crisis del software, muchas industrias están adoptando prácticas de ingeniería de software.

1.2.- Relatoria de la Tesis

Lo que se tratará en esta tesis es una propuesta de la forma en que se puede producir software de gran calidad.

CAPITULO I

En este capitulo se expone un breve historia de como ha seguido su curso la evolución del software desde el que se hacia a medida para las grandes computadoras hasta el que se produce en la actualidad con técnicas modernas como la programación orientada a objetos.

CAPITULO II

Se exponen básicamente las principales filosofías de calidad de los autores Edwards Deming y Joseph Juran con ejemplos de como es posible enfocarlas al desarrollo de software. Se presentan también las técnicas de QFD (despliegue de la función de calidad) aplicadas al análisis de requisitos para el desarrollo de software y por último se plantean los círculos de calidad y como implantarlos.

CAPITULO III

Se describen en general algunas de las técnicas de ingeniería de software empleadas en la actualidad en las diferentes fases de la producción del software como son el análisis de requisitos, diseño, prueba y mantenimiento de éste.

CAPITULO IV

Se describen en general algunas métricas de la calidad del software así como los factores que definen la calidad del software encaminados hacia el aseguramiento de la calidad, entrando por último en los sistemas de calidad como lo es ISO 9000 con su modelo para el aseguramiento de la calidad en diseño, desarrollo, producción, instalación y servicio.

CAPITULO V

Se presentan las conclusiones del trabajo de acuerdo a los objetivos planteados en un principio, así como las perspectivas futuras en la industria del desarrollo de software y las recomendaciones para la producción de software con gran calidad.

1.3.- Objetivos de la Tesis

Se pretende presentar una panorámica general de los métodos, técnicas, herramientas y estándares que pueden ser aplicados en la producción de software, lo anterior siempre con una fuerte orientación hacia la calidad; con la finalidad de que el lector, después de leer el presente trabajo se ubique en la materia de tal forma que tenga las nociones básicas de como desarrollar profesionalmente un software para cualquier campo de aplicación.

El presente trabajo es solamente una propuesta de calidad para el desarrollo de software, y pretende concientizar sobre las ventajas y limitaciones del uso de los métodos y técnicas aquí descritos, dando un especial énfasis al estándar ISO 9000, por que es el que se emplea con más frecuencia en la actualidad, además de ser el estándar con mayor aceptación, legalidad y legitimidad.

CAPITULO II

II.- PRINCIPALES FILOSOFÍAS DE CALIDAD COMO BASE

Autores como Feingenbaum, W. Edwards Deming, Joseph Juran, y Philib Crosby nos dieron a conocer sus filosofías de calidad que forman los antecedentes que constituyen bases sólidas para quien debe comprometerse con la calidad, es por esto que se expondrán las principales ideas de cada uno de ellos con un ejemplo de aplicación en el desarrollo de software.

A manera de historia podemos decir que después de la segunda guerra mundial se empieza en Japón con un enfoque de la administración total de calidad, cuyo método fué importado de Estados Unidos por el Dr. Edwards Deming y el Dr. Joseph Juran. Ambos investigadores encontraron su camino a Japón después de que sus trabajos habían sido rechazados en su propio país.

El Dr. Deming era un doctor en física que trabajaba para el gobierno estadounidense y fue enviado a Japón por la armada de Estados Unidos para ayudar con su censo y se le pidió que dirigiera la Unión Japonesa de Científicos e Ingenieros JUSE (por sus siglas en inglés Japanese Union of Sienctists and Engineers). Su método estaba basado en un enfoque estadístico para simplificar sus técnicas, pero el principal tema es un enfoque sistemático y riguroso de la calidad.

2.1 Filosofía de W. Edwards Deming:

Deming propone 14 puntos básicos a cubrir por la alta dirección:

1. Ser constantes en el propósito de mejorar el producto y servicio, con un plan de inicio en competitividad y negocios.

En la industria del desarrollo de software este punto se da casi forzosamente, siempre y cuando se quiera seguir en el mercado; los usuarios exigen cada vez nuevas mejoras y mejores funciones en el software que adquieren; para poder ser competitivos en este aspecto es de vital importancia entre otras cosas:

- Innovar:

Planes que consideren el desarrollo de nuevas aplicaciones de software que tengan mercado, nuevas habilidades, reentrenamiento del personal en las nuevas técnicas y herramientas para el desarrollo.

Además se debe considerar la aparición de nuevas tecnologías en el hardware cuyo potencial se incrementa cada vez más. No es difícil encontrarnos en el mercado con productos que se anticipen a la aparición de hardware; por ejemplo en sistemas operativos de red como lo es Netware de Novell que desde sus primeras versiones como la 2.X ya era un sistema operativo multitareas de 32 bits aunque todavía no se tenía el hardware que permitiera el uso de la tecnología de 32 bits.

- Dedicar recursos a la investigación y a la educación.

Es importante actualizar constantemente al personal para poder ser competitivos, sobre todo en el ambiente computacional en donde cada vez los productos se vuelven más rápidamente obsoletos.

En la industria de desarrollo de software la investigación es de vital importancia pues siempre se deben estar buscando nuevas formas de mejorar y estar un paso adelante de la competencia.

- **Mejorar constantemente el diseño del producto y el servicio**

El cliente debe ser la parte más importante del proceso de desarrollo, se debe actuar con el propósito de proporcionar software que satisfaga al cliente; es por esta razón que debemos poner un especial interés en la comunicación con el cliente para saber cuales son sus necesidades y cuales son sus expectativas del software a producir.

2. Adoptar la filosofía de la nueva era económica.

Philip Crosby cita "La calidad es opcional, usted no tiene que sobrevivir".

La competitividad va en aumento día tras día. Esto significa que a largo plazo solo permanecerán en el mercado las compañías o instituciones que a menor costo ofrezcan mayor calidad en sus productos o servicios; lo cual implica que se debe trabajar sin los errores que derivan en un aumento del costo de producción y por consiguiente en el precio del producto terminado.

En el desarrollo de software uno de los errores más comunes que afecta directamente en el costo de producción es una mala definición inicial o incompleta, es por esto que deben de quedar perfectamente definidos en el diseño aspectos fundamentales como son funciones, rendimiento, interfaces, criterios de validación, conectividad, etc.

3. No depender de la inspección masiva, exigir en su lugar evidencia estadística de que el producto o servicio se hace con calidad desde los primero pasos.

Es muy frecuente que se pierdan recursos en la inspección de software, es decir, cuando procedemos a efectuar pruebas del funcionamiento de este; sobre todo si tenemos que verificar varias veces el mismo error o cuando la corrección de un error especifico nos provoca otros mas.

Un ejemplo clásico es en el que se desarrolla un software sin tomar en cuenta alguna necesidad importante del cliente (volumen de información, variables a calcular, técnicas de impresión etc.,), la cual pasa inadvertida hasta el momento de entregar el sistema, y es hasta entonces cuando el cliente nos hace preguntas tales como: ¿En que módulo se calcula el subtotal de ventas por vendedor? o ¿Por que es tan lento el sistema? o ¿Por que no puedo

imprimir como de costumbre?. Es obvio que la implementación de cualquiera de estas especificaciones sería mucho más económica si es contemplada desde la fase de diseño.

4. El precio solo tiene sentido cuando hay evidencia de calidad. Se debe acabar con la práctica que usa como criterio de compra solo el bajo precio, lo importante es minimizar el costo total. Es preferible tratar con un número reducido de proveedores con los que se haya establecido una relación duradera, leal y confiable.

La política de hacer bajar el precio del artículo que se compra sin atender a la calidad puede poner fuera del mercado a vendedores de buenos productos y a quienes ofrecen buenos servicios.

El departamento de compras debe entender que en la compra de herramientas y equipos se trata de minimizar a largo plazo el costo de la producción o del servicio, y no el costo del instrumento mismo.

Recordemos que el hardware se vuelve cada vez más rápidamente obsoleto, es por eso que se debe tener presente la mejora continua del equipo.

5. Hay que estar mejorando constantemente el sistema de producción y de servicio, para mejorar la calidad y la productividad, para abatir así los costos.

El propósito de la calidad debe estar presente desde la etapa del diseño, sería demasiado tarde querer introducir la calidad en etapas posteriores. Por eso, es tan importante que el diseño del software sea el resultado de un trabajo en equipo. Hay que comprender cada vez mejor las necesidades de los consumidores y la forma como ellos van a usar el producto.

El mejoramiento del sistema significa mejorar día a día la calidad en cada una de las actividades: la ingeniería, los métodos, el mantenimiento, las ventas, métodos de distribución, la contabilidad, el servicio a los clientes.

Mejorar el proceso implica lograr un mejor aprovechamiento del esfuerzo humano, hacer una buena selección del personal y de la tarea que se le asigna, entrenarlo y ofrecerle la posibilidad de aumentar sus conocimientos y desarrollar sus aptitudes.

6.- Hay que poner en práctica métodos modernos de entrenamiento.

El personal deberá conocer a fondo la compañía. Uno de los despilfarros más grandes consiste en desaprovechar las habilidades del personal. Esto provoca frustración en las personas, lo cual afecta el rendimiento del trabajador.

7. Se debe administrar con liderazgo.

Como líderes auténticos, los jefes deben conocer el trabajo que supervisan a fin de ayudar a su personal a mejorar su propio desempeño.

8. Se debe eliminar el miedo en el trabajo.

No se puede rendir al máximo si cuando no se supera el miedo en cualquiera de sus manifestaciones: miedo de expresar sus propias ideas, de preguntar, etc.; el miedo implica siempre una pérdida económica por tanto se debe crear un ambiente que propicie la seguridad en el desempeño personal.

9. Deben eliminarse las barreras interdepartamentales.

El personal de diseño, de ingeniería, de soporte y de ventas, si trabajan en equipo, pueden realizar importantes mejoras en el diseño del producto, en el servicio, en la calidad y en la reducción de costos. A tales equipos se les podría denominar círculos de control de calidad a nivel gerencia.

10. No se deben proponer a los trabajadores metas numéricas, como también se deben evitar exhortaciones y amonestaciones.

En su mayoría los errores no provienen de los trabajadores sino del sistema mismo, por esto frecuentemente dichas amonestaciones generan frustración y resentimiento. Si se exige la terminación de un módulo en determinado tiempo, lo más probable es que este contenga errores cuya corrección llevará más tiempo del previsto inicialmente.

11.a.- Hay que eliminar las cuotas numéricas.

Las cuotas son un obstáculo para el mejoramiento de la calidad y productividad. En su lugar se debe instaurar un sistema eficiente de supervisión y fomentar que el operario se sienta orgulloso del trabajo realizado.

Si exigimos el desarrollo de determinado número de líneas de código al mes, es muy probable que estemos fomentando la generación de código de relleno, es decir código innecesario.

11.b.- Eliminar la administración por objetivos numéricos. Se debe administrar con liderazgo.

No deberán proponerse metas como aumentar en un 10% las ventas; Quien se inicia como administrador y quiere ser líder y promover el mejoramiento continuo debe aprender, entre otras cosas, que es lo que hace su gente y como lo hace. Este aprendizaje es más importante que revisar los reportes de calidad, de fallas, de ventas, etc. Fijar la atención en los resultados no es el camino efectivo para mejorar un proceso o una actividad.

Cuando se tiene un sistema estable, este trabaja a toda su capacidad, por consiguiente, sale sobrando especificar una meta numérica.

12. Derribar las barreras que impiden el orgullo de hacer bien un trabajo.

Nadie puede sentirse orgulloso de su trabajo si no sabe las condiciones que se necesitan para que su trabajo se considere bien hecho.

13. Instituir un fuerte programa de educación y readiestramiento.

14. Crear una estructura que fomente todos los días los trece puntos anteriores.

2.2 Filosofía de Joseph M. Juran:

El Dr. Juran llegó a Japón en 1954 con el objetivo básico de centrarse en la calidad en lugar del costo y de atender el rompimiento de la calidad. Sus ideas han estado bajo constante desarrollo en Japón desde entonces.

Juran encontró que el 90% de los problemas de calidad fueron causados por la administración y no por los trabajadores. Uno de los mayores beneficios se obtiene en hacer del administrador una persona que facilite el trabajo al usuario, en vez de un dictador.

Juran explica la calidad en 2 sentidos básicos.

- Comportamiento del producto y
- Ausencia de deficiencias.

En el sentido del comportamiento, la calidad se refiere a características tales como:

→ Millones de instrucciones por segundo (mips) de un ordenador.
→ Prontitud del proceso para cumplir con los pedidos de los clientes.
→ Velocidad de un programa para efectuar determinada función, por ejemplo ordenar una base de datos de determinado numero de registros.

Tales características son decisivas para el comportamiento del producto y para la satisfacción con el producto; dichas características compiten unas con otras en el mercado. Los usuarios finales comparan los comportamientos entre la competencia, y estas comparaciones se transforman en un factor de decisión sobre que producto se comprará.

Debido a la competencia en el mercado, el ser igual o superior en calidad entre los productos competidores es un objetivo prioritario para el comportamiento de cualquier producto.

La palabra calidad también se refiere a la ausencia de deficiencias, las cuales pueden adoptar la forma de:

→ Lentitud de operación del sistema
→ Fallos durante el servicio

- → Facturas incorrectas
- → Cancelación de contratos de ventas
- → Reprocesos
- → Cambio en la ingeniería del diseño

Las deficiencias dan como resultado quejas, reclamaciones, devoluciones, reprocesos y otros daños, que en su conjunto son las formas de insatisfacción con el producto.

Es bastante posible que un producto no presente deficiencias y aun así sea invendible porque algún producto de la competencia tenga un comportamiento mejor.

Se debe tener en cuenta que el término producto incluye bienes y servicios. En este caso nuestro producto es el software.

Clientes:

Son aquellas personas sobre quienes repercuten nuestros procesos y nuestros productos. Estas personas incluyen tanto los clientes internos como los externos.

Juran propone una trilogía muy interesante para el mejoramiento de la calidad que podemos extrapolar al desarrollo de software, dicha trilogía se expone a continuación:

- Planificación de la calidad:

Para la planificación de la calidad se planea a fin de desarrollar productos que satisfagan las necesidades del cliente a través de las siguientes etapas:

Identificar quienes son los clientes
Determinar las necesidades de los clientes
Traducir esas necesidades a nuestro lenguaje
Desarrollar un producto que pueda responder a esas necesidades
Optimizar las características del producto
Realizar auditorías preventivas

- Control de la calidad:

Establecer criterios que satisfagan el manual del control de calidad, revisar y actualizar dicho manual, y establecer estudios de factibilidad.

Muchas veces no se toman en cuenta costos de mantenimiento, costos por fallas imprevistas, costos por cambios en las especificaciones originales, etc.

- Mejora de la calidad:

En este punto entra la reducción de costos y la mejora de los beneficios, comprender las percepciones del trabajador, otorgar reconocimientos y servir en los equipos de proyectos.

El Dr. Juran agregó el concepto de rompimiento de la calidad. Esto en términos básicos, significa que usted no solo debe reducir el costo de la calidad sino la variación del control alrededor de la norma. En la siguiente figura se ilustra una reducción en el costo de la calidad y en la variación de la norma. El rompimiento se da por la reducción en el costo crónico como el trabajo doble, etcétera.

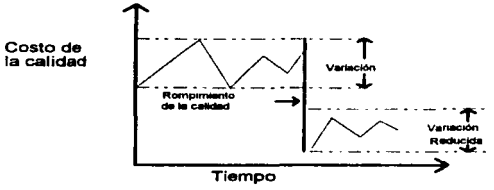


Figura II.1.- Gráfica de rompimiento de la calidad

2.3 QFD (Quality Function Deployment)

A continuación se presenta una condensación de un análisis del QFD publicado en Proyección, una revista de la Asociación Mexicana de Calidad A. C., en su No. 17

Definición de QFD del American Suplier Institute:

"QFD es un sistema para traducir la voz del cliente en requisitos de empresa para cada uno de las fases del ciclo de desarrollo"

Definición de la organización GOAL/QPC líder en estas técnicas en Estados Unidos:

"Es un sistema de diseño de productos o servicios, fundamentado en las necesidades del cliente, en el que involucran a todos los miembros de la cadena productiva"

La definición de QFD que se concluye en proyección es la siguiente:

"El QFD es un sistema ordenado de planificación que permite desplegar las necesidades del cliente en actividades concretas que se deben realizar en las diferentes áreas y funciones de una empresa para el correcto cumplimiento de las expectativas del cliente."

APLICACIONES DEL QFD

El QFD nació en el mundo industrial y sus primeras aplicaciones se han realizado en el sector del automóvil y electrónica de consumo. Sin embargo esta metodología de planificación se ha introducido con éxito en varios campos, como en el sector de servicios de salud, diseño de programas de estudios, desarrollo de software, etc.

METODOLOGÍA

La metodología del QFD tiene por objetivo determinar los procesos y características críticas del producto y sus parámetros importantes.

El QFD es una metodología orientada al diseño con calidad, que ayuda al propósito de trabajar en acciones preventivas en lugar de las correctivas. El QFD permite definir lo que hay que hacer y lo transforma progresivamente en procedimientos de como hacerlo, para satisfacer al cliente.

El enfoque comienza con la identificación de los requerimientos del cliente, los cuales son usualmente detalles de calidad establecidos confusamente, tales como: fácil de usar, buen desempeño, ultima moda o lujoso. Estos detalles son importantes para el cliente, pero son difíciles y ambiguos para que los técnicos puedan actuar.

En esta primera parte se entrevista al cliente potencial y se le solicitan sus necesidades y expectativas. A través de una reunión amplia y natural el cliente es entrevistado y escuchado cuidadosamente. Estas ideas se organizan a través de un Diagrama de Afinidad para estructurar la información recogida.

Con el fin de lograr un producto adecuado, es necesario convertir los requerimientos ambiguos del cliente, en requerimientos procesables dentro de la empresa, conocidos como requerimientos de diseño. Este proceso de traducción es realizado con la ayuda de una serie de tablas o matrices, siendo la Casa de la Calidad (Quality House) la más difundida.

ELEMENTOS BÁSICOS DE LA MATRIZ QFD

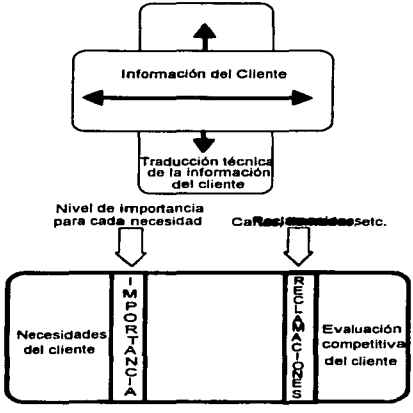


Figura II.2 Matriz QFD típica para traducción de requerimientos.

La matriz anterior contiene los siguientes elementos:

- Necesidades del cliente
- La forma como se puede lograr su satisfacción y sobre cuales parámetros debemos trabajar en nuestra empresa.
- El grado de importancia de cada una de las necesidades manifestadas por el cliente.
- Calificación que da el cliente sobre la forma como se siente con nuestro producto o servicio y para cada necesidad manifestada.
- Análisis competitivo. El cliente nos califica para cada necesidad con respecto a la competencia.
- Valores objetivo que si se alcanzan lograremos una total satisfacción del cliente.
- Otra información adicional que le facilite al equipo de trabajo la toma de decisiones.
- Correlaciones entre necesidades y formas de lograr cada una de ellas.

Los requerimientos de diseño obtenidos inicialmente de la traducción de la voz del cliente son globales, siendo necesario traducirlos a especificaciones detalladas para cada uno de los elementos constitutivos del producto o servicio. Esta actividad se realiza empleando tablas y matrices especialmente adaptadas para este propósito. El proceso de traducción secundario permite obtener las funciones esenciales de cada elemento del producto.

Una vez identificadas las funciones, se deben diseñar y desarrollar los procesos de producción para cada elemento o parte del producto. Nuevamente es necesario hacer uso de matrices y tablas QFD para traducir las características de cada elemento en las especificaciones que debe cumplir el proceso productivo. En este proceso se tienen en cuenta las limitaciones de empresa, tanto para los recursos físicos como humanos.

El empleo de la metodología QFD para el diseño de procesos productivos ayuda a determinar los métodos más adecuados en las operaciones para el logro de las caracteristicas críticas del producto o servicio. Con las matrices de QFD se pueden diseñar los mejores planes de inspección, control estadístico de proceso, programas de mantenimiento preventivo, entrenamiento e instrucción de operarios, así como la utilización de dispositivos de prevención de errores en los niveles operativos.

BENEFICIOS DEL QFD

El QFD promueve el desarrollo de productos previniendo los problemas en lugar de reaccionar ante ellos. Esta forma de trabajo permite reducir el tiempo de cambios que se debe realizar en el desarrollo y en la disminución de problemas que surgen en el proceso de diseño de productos, antes y después de comenzar su producción

Los pocos cambios de ingeniería que se presentan cuando se trabaja con la metodología QFD, en su mayoría (90%) se hacen mucho antes del inicio de la producción. Estos cambios se hacen con gastos mínimos, ya que se realizan sobre la mesa de diseño.

El QFD permite ahorrar tiempo y dinero. El tiempo del ciclo de introducción de productos, se reduce en una tercera parte y en ocasiones hasta la mitad.

El enfoque preventivo del QFD trae como resultado pocos problemas en la puesta en marcha del nuevo producto, especialmente en el comienzo de la producción. El sistema QFD ayuda a eliminar la causa de los problemas con anticipación antes de que estos se presenten.

El proceso del QFD puede ayudar en la solución de problemas difíciles, esta orientado hacia el cliente y las decisiones de ingeniería se dirigen en favor de estos, lo cual garantiza su satisfacción.

El QFD es un medio para aumentar el conocimiento técnico y su transferencia a nuevos empleados, evitándoles cometer los mismos errores del pasado. El uso de las tablas o gráficas, permiten acumular una gran cantidad de conocimientos en ellas.

Definitivamente concluye la publicación, el QFD es una buena herramienta para el desarrollo de organizaciones abiertas al aprendizaje o del siglo XXI.

A continuación se presenta la propuesta de como se puede aplicar el QFD al desarrollo de software:

Comenzaremos con nuestra propia definición de QFD enfocada al software, partiendo de la definición de la American Suplier Institute, podemos decir que:

El QFD es un sistema para traducir la voz del cliente en especificaciones técnicas o requisitos del diseño de software para cada una de las etapas del ciclo de desarrollo de éste.

Aplicando la matriz de QFD para el desarrollo de software tendríamos algo muy similar a lo siguiente:

MATRIZ QFD PARA DESARROLLO DE SOFTWARE

ELEMENTOS BÁSICOS DE LA MATRIZ QFD

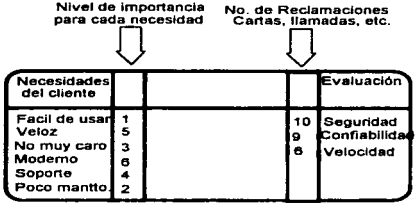


Figura II.2 Matriz QFD aplicada al desarrollo de software para la traducción de requerimientos.

Japón ha producido ahora sus propias filosofías como son la del profesor Ishikawa quien promueve los círculos de calidad y el diagrama Ishikawa, también conocido como el diagrama de "espinas de pescado".

Otro nombre bien conocido de Japón es el del Dr. Genichi Taguchi cuyo método inspira a la calidad y confiabilidad de un producto a ser construido en la etapa de diseño, en donde insiste en que uno no puede pintar la calidad después de que ha producido el producto, cuyo principio es bastante aplicable al software.

2.4 Círculos de calidad

A continuación se presenta un propuesta de círculos de calidad en la ingeniería de software basada en el tema "Cómo implantar los círculos de calidad" discutido en el libro "Implantar y Gestionar la Calidad Total" de los autores A. Bernillon y O. Cerutti.

LOS CÍRCULOS DE CALIDAD EN LAS EMPRESAS DE DESARROLLO DE SOFTWARE Y COMO IMPLANTARLOS

Los círculos de calidad constituyen un elemento importante en la calidad total y tienen su origen en Japón alrededor de los años 60, siendo en mayo de 1962 cuando se registra el primer círculo de calidad en JUSE (Unión de Científicos e Ingenieros Japoneses), organización creada en 1946 con la finalidad de investigación y desarrollo, enseñanza y entrenamiento, servicio internacional y patrocinio de conferencias técnicas anuales, publicaciones y servicios de extensión.

Los círculos de calidad se han desarrollado en Europa a partir de los años 80, mientras que en Japón se acrecentaron año con año, hasta lograr 800,000 a fines de 1981.

Por experiencia se sabe que los círculos no pueden subsistir en la empresa de forma duradera si no forman parte de un proceso global.

Los círculos no deben sustituir a la organización existente, sino integrarse en ella, afectándola en numerosos aspectos.

La utilidad de los círculos de calidad se da básicamente en dos ámbitos:

1.- Económico

- Permiten a la empresa movilizar todos sus recursos, facilitando la mejora de sus capacidades.
- Contribuyen a reducir la no calidad.
- Implican al empleado en la mejora de su propio trabajo.

2.- Social

- Facilitan la implicación del personal en la vida de la empresa permitiéndole de una parte expresar sus opiniones, y de otra, regular él mismo los problemas inherentes a su puesto de trabajo.
- Por el trabajo en grupo, los círculos facilitan los cambios en el interior de la empresa contribuyendo así a la mejora de las relaciones.

Definición: Un círculo de calidad agrupa de 5 a 10 personas de forma voluntaria. Es permanente, constituido por empleados pertenecientes a un mismo taller y ligados por las mismas preocupaciones de trabajo. Estudia las causas de los problemas de su actividad cotidiana, busca las soluciones, las propone a sus superiores y sigue su puesta en marcha.

Un círculo de calidad funciona siguiendo reglas bien definidas, anteriormente negociadas entre la dirección y el personal, consignadas en una carta difundida en toda la empresa. Esta carta precisa los derechos y los deberes de un círculo de calidad respecto de la organización, especifica en particular su campo de acción, comprendiendo básicamente:

- El análisis de los errores encontrados
- La reducción de pérdidas por no calidad
- La disposición de los puestos de trabajo
- La seguridad del puesto
- El estudio de la utilización del equipo

Un ejemplo de carta sería el siguiente:

Artículo 1

¿Que es un círculo de calidad?

- → Un círculo de calidad está formado por un grupo de 5 a 10 miembros. Es permanente.
- → Los miembros proceden del mismo equipo de desarrollo o están ligados por la misma preocupación de trabajo.
- → Estudia las causas de los problemas de su actividad cotidiana, busca las soluciones, las propone a sus superiores y sigue su puesta en marcha.

Artículo 2

¿Cómo se constituye un círculo de calidad?

El círculo de calidad está formado por voluntarios reunidos alrededor del animador. Se puede, en todo momento, ser candidato para entrar en un círculo de calidad o dejarlo. Una persona no puede pertenecer más que a un sólo círculo de calidad.

Artículo 3

¿Quién estimula un círculo de calidad?

Para poder estimular un círculo de calidad hay que haber seguido una formación específica, en principio los miembros de los círculos de calidad son estimulados por sus responsables jerárquicos.

Artículo 4

¿Cuándo y dónde se reune un círculo de calidad?

Un círculo de calidad dispone de una cuota de cuatro horas por mes. Planifica su reunión en función de la actividad y de acuerdo con el responsable del departamento. Las sesiones tendrán lugar en una sala del departamento.
Se reunen principalmente durante horas de trabajo. Si las necesidades de producción lo impiden, según su decisión , y después de que el responsable jerárquico lo acepte, podrá reunirse en horas recuperables.

Artículo 5

Organización de las sesiones de trabajo.

Periodicidad: según la apreciación del círculo de calidad, sin embargo se aconseja una periodicidad media de una sesión cada 2 semanas.

Duración: dos horas como máximo por sesión.

Asiduidad: teniendo en cuenta la importancia de los círculos de calidad, una vez la fecha y hora de la reunión decididas, cada miembro comienza a participar, salvo circunstancias excepcionales. En este caso, esta obligado a informar lo más pronto posible, al animador de su ausencia.

Artículo 6

¿Qué temas son abordados? .

Los temas tratados por el círculo de calidad son problemas propios del desarrollo o de la oficina, a los cuales se puedan aportar soluciones.

Por ejemplo, fuera del campo de competencia de los círculos de calidad están aquellos problemas que afectan a otras instancias como:

- Casos individuales.
- Salarios.
- Estatutos de la empresa.
- Gestión del personal.

Artículo 7

¿Cómo escoge sus temas el círculo de calidad?

El círculo de calidad escoge él mismo sus temas, respetando lo definido en el artículo 6. Los temas escogidos por el círculo de calidad serán definidos por decisión unánime.
Al principio, el círculo de calidad escogerá preferentemente temas simples, claramente definidos.

Artículo 8

¿Cómo estudia sus temas el círculo de calidad?

Los miembros del círculo de calidad trabajan siguiendo una metodología, para la cual habrán sido formados por su animador, y utilizando herramientas específicas.

Artículo 9

¿Qué hace el círculo de calidad cuando encuentra la solución?

El círculo de calidad propone sus soluciones a su jefe jerárquico con la ayuda de un informe explicativo. La decisión se toma por el nivel competente que se encarga de responder de manera rápida y argumentada. La respuesta será inscrita en el informe del problema.
Participa en la realización de las soluciones encontradas.

Artículo 10

¿Recibe una remuneración específica el círculo de calidad ?

La actividad del círculo de calidad forma parte integrante del trabajo de cada uno de los participantes y no da lugar a una remuneración adicional.

Artículo 11

¿Cuál es el papel de animador en el círculo de calidad ?

- Asegura la preparación y conduce las reuniones del círculo de calidad, trabajo para el que habrá sido formado previamente.
- Hacer respetar el espíritu y los principios de los círculos de calidad.
- Vela por la participación activa de todos los miembros del círculo de calidad.

Artículo 12

¿Quién coordina la actividad de los círculos de calidad ?

El facilitador

Artículo 13

¿Qué función tiene el grupo piloto?

- → Da las orientaciones generales.
- → Ayuda a la formación de los primeros círculos de calidad.
- → Define las acciones correctoras a aportar a la carta de principios de los círculos.
- → Asegura el desarrollo de los círculos de calidad.
- → Informa a la dirección.

Artículo 14

¿Cuál es la función del facilitador?

- → Asiste y aconseja a los animadores sobre sus peticiones.
- → Informa a los directivos o mandos del departamento.
- → Participa en el lanzamiento y desarrollo de los círculos de calidad.
- → Propone al grupo piloto la formación de futuros animadores.
- → Coordina los diferentes círculos de calidad.

Los círculos deben utilizar una metodología rigurosa de resolución de problemas, conduciéndolo a preguntarse sobre los problemas, buscar sus causas, las soluciones y escoger la mejor. El trabajo realizado debe ser concreto. Los círculos proponen al facilitador un cierto número de problemas que desean tratar. Este último conforme a su misión debe orientar al principio sobre temas fáciles y que tienen todas las posibilidades de ser solucionados. Según los problemas encontrados a lo largo del estudio, los círculos pueden, si lo precisan, recurrir a competencias exteriores.

Una etapa importante del funcionamiento de círculos es la presentación de propuestas a la jerarquía o a la dirección. Al fin de esta presentación, la dirección ha de autorizar la realización de la solución propuesta si es viable. Esta presentación simboliza el diálogo existente entre la dirección y los miembros del círculo y la importancia que la dirección da a los trabajos en grupo.

El último trabajo del grupo es el de seguir la realización de la solución y verificar que su eficacia es correcta. El círculo puede entonces estudiar el problema siguiente.

Las filosofías enunciadas anteriormente nos dan una visión general de en que aspectos debemos fijar nuestra atención, por ejemplo en involucrar a todo el personal de la compañía en el proceso de calidad.

Debemos de tener en cuenta que estos modelos de calidad no fueron hechos pensando en el software sino en la creación de un producto o servicio; sin embargo, la mayoría de los principios son aplicables al software.

CAPITULO III

III.- INGENIERÍA DE SOFTWARE

Definición de ingeniería de software: El desarrollo y uso de estrategias sistemáticas (a menudo basadas en software) para la producción de software de buena calidad dentro de los presupuestos y escalas de tiempo.

El desarrollo de software se produce mediante etapas sucesivas de especificación, diseño y modificación. Cada evaluación de una parte del software se hace por una revisión de la documentación que describe los requerimientos, especificación, diseño o, después, por pruebas al código y área usada del sistema realizado, lo cual da como resultado cambios.

3.1 Modelos de desarrollo de software de ciclo de vida (Paradigmas de la ingeniería del software)

Un ciclo de vida se refiere a la secuencia de estados o fases, desde el análisis de los requerimientos hasta el mantenimiento, involucrados en el desarrollo de software

El ciclo de vida proporciona un modelo conveniente que sirve para representar los procesos de concepción y producción en forma gráfica y lógica, además proporciona un marco de trabajo alrededor del cual las actividades de aseguramiento de calidad pueden ser construidas.

Los métodos de ingeniería del software indican como construir técnicamente el software; las herramientas de la ingeniería del software suministran un soporte para los métodos.

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 del desarrollo del software, llamado Ingeniería de Software Asistida por Computadora (del inglés CASE - Computer Aided Software Engeenering -). CASE combina software, hardware y bases de datos sobre ingeniería del software.

Los procedimientos de ingeniería del software constituyen la unión entre entre los métodos y las herramientas para facilitar el desarrollo racional y oportuno del software de computadora. Los procedimientos definen la secuencia en la que se aplican los métodos, las entregas de documentos, formatos, informes, los controles que ayudan a asegurar la calidad y coordinar los cambios, y las directrices que ayudan a los gestores del software a evaluar el progreso.

La ingeniería de software está compuesta por una serie de pasos que abarcan los métodos, las herramientas y los procedimientos antes mencionados; estos pasos se denominan frecuentemente paradigmas de la ingeniería del software. A continuación se presenta una visión general de los 4 más comunes:

A) El ciclo de vida clásico, a veces llamado modelo en cascada, el cual abarca las siguientes actividades:

Ingeniería y análisis del sistema.- Abarca los requisitos globales a nivel del sistema con una pequeña cantidad de análisis y de diseño a un nivel superior.

Análisis de los requisitos del software.- Para comprender la naturaleza de los programas que hay que construir, el ingeniero de software "analista" debe comprender el ámbito de la información del software, así como la función, el rendimiento e interfaces requeridos. Los requisitos, tanto del sistema como del software se deberán documentar y revisar con el cliente.

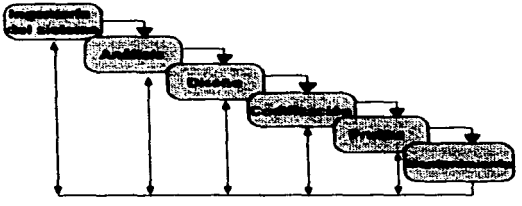


Figura III.1.- Ciclo de vida clásico

Diseño.- Es un proceso que comprende los siguientes cuatro aspectos del programa:

- Estructura de los datos
- Arquitectura del software
- Detalle de los procedimientos
- Caracterización de la interfaz

El proceso de diseño, traduce los requisitos en una representación del software que pueda ser establecida de forma que obtenga la calidad requerida antes de iniciar la codificación. El diseño deberá documentarse e integrar parte de la configuración del software.

Codificación:- Es la traducción del diseño en una forma legible para la máquina. Si el diseño se realiza de una manera detallada, la codificación puede realizarse mecánicamente.

Prueba.- Una vez que se ha generado el código, comienza la prueba del programa. La prueba se centra en la lógica interna del software, asegurando que todas las sentencias se han probado, y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce los resultados que realmente se requieren.

Mantenimiento.- El software, indudablemente sufrirá cambios después de que se entregue al cliente. Los cambios puede ser porque:

- Se hayan encontrado errores.
- El software deberá adaptarse a cambios del entorno externo (nuevo sistema operativo, o periférico).
- El cliente requiere ampliaciones funcionales o del rendimiento.

El mantenimieto del software aplica cada uno de los pasos anteriores del ciclo de vida a un programa existente en vez de a uno nuevo.

Desventajas del modelo clásico de ciclo de vida:

- Se requiere que el cliente establesca al principio todos los requisitos del sistema, lo cual es difícil para el cliente.
- El cliente debe tener paciencia. Hasta llegar a las etapas finales del desarrollo del proyecto, no estará disponible una versión operativa del programa.
- Un error no detectado hasta que el programa este funcionando puede ser desastroso.

El ciclo de vida clásico sigue siendo el modelo procedimental más ampliamente usado por los ingenieros del software. A pesar de sus inconvenientes, es significativamente mejor que desarrollar el software sin guías:

B) Construcción de prototipos.

La construcción de prototipos es un proceso que facilita al programador la creación de un módelo de software a construir y puede ser de tres formas:

1) Un prototipo en papel o un modelo basado en PC que describa la interacción hombre - maquina, de forma que facilite al usuario la comprensión de cómo se producirá tal interacción.

2) Un prototipo que implemente algunos subconjuntos de la función requerida del programa deseado.

3) Un programa existente que ejecute parte o toda la función deseada, pero que tenga otras características que deban ser mejoradas en el nuevo trabajo de desarrollo.

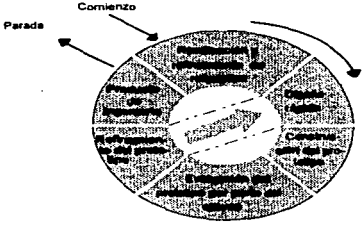


Figura III.2.- Creación de Prototipos

La construcción de prototipos comienza con la recolección de los requisitos, luego se produce un diseño rápido, enfocado sobre la representación de los aspectos del software visibles al usuario; este diseño conduce a la construcción de un prototipo, el cual será evaluado por el cliente/usuario y se usa para refinar los requisitos del software a desarrollar. Se produce un proceso interactivo en el que el prototipo es afinado para que satisfaga las necesidades del cliente, al mismo tiempo que facilita al que lo desarrolla una mejor comprensión de lo que hay que hacer.

Desventajas: El cliente ve funcionando lo que parece ser un primera versión del software, ignorando que por las prisas en hacer que funcione, no se han considerado los aspectos de la calidad o de mantenimiento del software a largo plazo. Cuando se informa que el producto debe ser reconstruido, el cliente solicita que se apliquen cuantas mejoras sean necesarias para hacer del prototipo un producto final que funcione. El gestor del desarrollo de software cede demasiado a menudo. Por las prisas en ocasiones se puede utiizar un sistema operativo o un lenguaje de programación inapropiados, simplemente por que ya está disponible y es conocido.

La construcción de prototipos es un paradigma efectivo para la ingeniería del software, la clave está en que el cliente y el técnico esten de acuerdo en que el prototipo se construya para servir sólo como un mecanismo de definición de los requisitos y que posteriormente ha de ser descartado (al menos en parte) y debe construirse el software real, con los ojos puestos en la calidad y en el mantenimiento.

C) Modelo en espiral

Se ha desarrollado para cubrir las mejores características de los 2 modelos anteriores añadiendo el análisis de riesgo. Este modelo define 4 actividades principales, representadas por los cuatro cuadrantes de la siguiente figura:

1.- Planificación: determinación de objetivos, alternativas y restricciones
2.- Análisis de riesgo: análisis de alternativas e identificación/resolución de riesgos
3.- Ingeniería: desarrollo del producto de "siguiente nivel"
4.- Evaluación del cliente: valoración de los resultados de la ingeniería

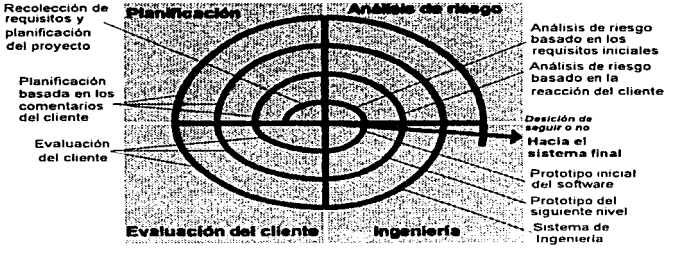


Figura III.3.- Modelo en espiral

Este modelo es actualmente el enfoque más realista para el desarrollo de software y de sistemas a gran escala; el modelo en si mismo es relativamente nuevo y no se ha usado tanto como el ciclo de vida o la creación de prototipos.

D) Técnicas de cuarta generación

El concepto abarca un amplio espectro de herramientas de software, que generan automaticamente el código fuente basandose en la especificación del técnico; entre mayor sea el nivel en el que se especifique el software, más rápido se podrá construir el programa.

Un entorno para el desarrollo de software que soporte el paradigma de técnicas de cuarta generación puede incluir todas o alguna de las siguientes herramientas: lenguajes no procedimentales para consulta a base de datos, generación de informes, manipulación de datos, interacción y definición de pantallas, generación de código, facilidades gráficas de alto nivel y facilidades de hoja de cálculo.

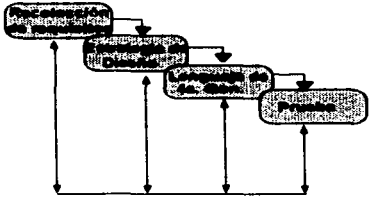


Figura III.4.- Técnicas de cuarta generación.

A este modelo se atribuyen reducciones drásticas en el tiempo del desarrollo del software y una mejora significativa en la productividad de la gente que construye el software.

Desventajas: Se dice que las herramientas actuales de técnicas de cuarta generación, no son más fáciles de usar que los lenguajes de programación; que el código fuente producido por tales herramientas es "ineficiente" y que el mantenimiento de grandes sistemas de software desarrollados así, es cuestionable.

En resumen podemos decir que ningún modelo de ciclo de vida no es ni perfecto ni genérico ya que todos ellos, son abstracciones idealizadas y cada uno toma diferentes puntos de vista.

Por lo general estan enfocados a aspectos técnicos del desarrollo y las relaciones con otros parámetros tales como el costo, tienen que crearse.

La adopción de un modelo de ciclo de vida apropiado es un paso importante hacia el mejoramiento de la calidad. Es el primer nivel para tener bajo control el proceso de desarrollo de software.

3.2 Análisis de requisitos

El análisis de requerimientos del sistema es una de las fases más importantes en un proyecto de desarrollo de software, en ella tanto el cliente como el desarrollador juegan un papel activo. El cliente intenta plantear detalladamente el concepto de la función y del comportamiento del software. El desarrollador actúa como interrogador, consultor y persona que resuelve problemas.

El análisis de requisitos facilita al ingeniero la especificación de la función y del rendimiento del software, proporciona además una representación de la información y de las funciones, que se puede traducir en un diseño de datos.

La especificación de los requisitos es la fase donde se alcanza un acuerdo explícito sobre las responsabilidades del comprador y del proveedor durante la ejecución del contrato, y suministra al técnico y al cliente un medio para valorar la calidad del software una vez que se haya construido.

La diferencia entre requerimientos y diseño es fácil de establecer:

Requerimiento: Lo que debe hacer el sistema.

Diseño: Cómo debe hacerlo el sistema.

Se sabe por experiencia que el costo de corregir o modificar un sistema después de la instalación, o incluso después de las primeras etapas de la fase de diseño, es alto y quizá mayor que el costo de preparar una especificación de requisitos.

Como regla empírica, el costo de reparar un error se incrementa en un factor de diez de una fase de desarrollo a la siguiente.

Técnicas para la captación y análisis de los requerimientos.- Se pueden considerar 3 categorías básicas:

a) Estándares

Una especificación ideal de requerimiento deberá tener las siguientes características:

- Sin ambigüedad.- Cada requerimiento deberá tener sólo una interpretación. Un glosario de términos puede ser de utilidad.
- Completo.- Deben incluirse todos los requerimientos significativos y los aspectos de su ambiente que se relacionen con funcionalidad, desempeño, restricciones de diseño, etcétera.
- Verificable.- El requerimiento debe de establecerse de forma que el sistema pueda probarse para asegurar que el requerimiento ha sido satisfecho.
- Consistente-.- No deben existir conflictos o contradicciones entre los requerimientos individuales.
- Modificable.- La especificación de requerimientos debe adaptarse a los cambios que los usuarios hacen a los requerimientos.
- Fácil de seguir.- Se deberan incluir mecánismos de referencia.
- Utilizable durante las fases de operación y mantenimiento.- Donde la falla de un elemento puede ser critica o donde se efectúan cambios temporales se debe poner atención a las necesidades de operación, de mantenimiento y a la especificación de requerimientos de actualización.
- Inteligible.- Los requerimientos tienen que ser leídos y entendidos por mucha gente (usuarios, analistas, compradores y proveedores). La especificación de requerimientos forma la base para la comunicación entre todas las clases diferentes de personas involucradas en un desarrollo.

b) Métodos

Los métodos de captación y análisis de requerimientos caen dentro de 3 categorias:

- Informales
- Semiformales y
- Formales.

c) Herramientas

Son auxiliares automatizados que por lo general soportan el control del volumen de la información, pruebas para asegurar la consistencia y pruebas para la totalidad.

Lo más difícil de la definición de requerimientos radica en la comunicación entre el cliente y el desarrollador. El primero no está siempre en posición de definir con precisión lo que se requiere; como resultado, al segundo por lo general no le es fácil producir una especificación con el detalle suficiente que permita asegurar la traducción de los requerimientos para el diseño.

Como resultado del análisis se desarrolla la especificación de requisitos del software y su revisión resulta esencial para asegurar que el realizador del software y el cliente tengan la misma percepción del sistema.

Una vez terminada la revisión, tanto el cliente como el técnico deben firmar la especificación de los requisitos del software. La especificación se convierte en la base del contrato para el desarrollo del software.

Cualquier sistema complejo necesita que se tomen en cuenta muchas opiniones y consideraciones. La técnica descrita requiere un análisis contra ciertos criterios. El alcance de este análisis con respecto a los criterios establecidos (por medio de la entrevista por ejemplo) se expone a continuación:

Análisis de relevancia.- Aquí se construye una tabla de los requerimientos y aquello a lo que afectan, se hace un intento de asegurar la relevancia de cada requerimiento a cada individuo o grupo involucrado. Puede asignarse una prioridad a los requerimientos de acuerdo con su relevancia total asegurada. Aquello a lo que afectan puede en forma similar, asignarse en un orden de prioridad de acuerdo con el número y extensión de los requerimientos que los afectan.

Análisis de calidad.- Los requerimientos se examinan respecto a un conjunto básico de atributos de calidad como son la eficiencia, la confiabilidad, etc.; luego debe examinarse su alcance para el cual los requerimientos afectan a esos atributos.

Análisis de interacción.- En algunos casos, cumplir un requerimiento puede tener un efecto perjudicial en otros.

En el análisis de requerimientos pueden emplearse también las técnicas de QFD mencionadas en la Sección 2.3.

Validación: Aún cuando los requerimientos sean comprendidos con claridad, deben ser escritos y en ocasiones refinados. En cualquier punto al escribir una descripción de un sistema, el texto tiene que se claro.

Lista de requerimientos: Dada la naturaleza de los requerimientos es fácil perder atributos importantes o esenciales que el sistema debe poseer. La única forma confiable para minimizar la ocurrencia de tales omisiones es asegurar que la gente con experiencia una y verifique los requerimientos del sistema.

Cuatro aspectos destacan en la captación y análisis de requerimientos, los cuales se enlistan a continuación con las preguntas claves que necesitan ser contestadas:

Información:

¿La información reunida incluye requerimientos no funcionales como seguridad, desempeño, utilidad, confiabilidad, etcétera?

¿Los puntos de vista de todos aquellos afectados por el sistema y su desarrollo han sido incluidos en el proceso de reunión de requerimientos?

¿Se han incluido los efectos más amplios de las funciones del sistema y sus cualidades (obligaciones contractuales, restricciones técnicas o limitaciones de tiempo y costo)?

Análisis:

¿Existen criterios para los requerimientos prioritarios?

Aseguramiento:

¿El cliente esta involucrado y comprometido a revisar el establecimiento de requerimientos?

El proceso de requerimientos:

¿Esta claro quien es el cliente?

¿Esta claro por que se necesita el sistema (los objetivos, términos de referencia y entregas esperadas, se conocen desde esta etapa)?

Si el costo es parte de la fase de requenmientos, ¿los efectos tales como mantenimiento, soporte, instalación, etc., son considerados?

3.3 Diseño e implementación del software

Una vez que se conocen los requirimientos que un sistema debe de satisfacer, se deberá comenzar con el diseño del sistema, el cual normalmente se divide en dos fases:

- Diseño preliminar.- En el cual se divide al sistema en pequeñas unidades individuales; el propósito de este diseño es descomponer el sistema en pequeños módulos que sean coherentes y fáciles de entender individualmente, además se determinan las relaciones entre estos.

- Diseño detallado.- En el cual se refinan las especificaciones de las unidades individuales.

El diseño es técnicamente la parte central de la ingeniería del software. Durante el diseño se desarrollan, se revisan y se documentan los progresivos refinamientos de las estructuras de datos, de la estructura del programa y de los detalles procedimentales. El diseño da como resultado representaciones del software cuya calidad se puede evaluar.

Existen 4 razones principales para dividir el sistema en módulos:

- Comprensibilidad.- Un sistema que no tiene una subestructura es difícil de entender.
- División del trabajo.- Se puede agilizar la implementación de algún módulo, dividiendolo en pequeñas partes.
- Respuesta al cambio.- Los cambios son una característica fundamental de los sistemas de software.
- Reusabilidad.- Cuando se construye un nuevo sistema, se puede ahorrar mucho tiempo, dinero y esfuerzo si es posible la utilización de módulos de un sistema anterior.

El diseño de software puede verse desde las perpectivas técnica y de gestión del proyecto. Desde el punto de vista técnico, el diseño comprende cuatro actividades:

- Diseño de datos:

Es la actividad más importante de las cuatro. El impacto de la estructura de datos sobre la estructura del programa y la complejidad procedimental, hace que el diseño de datos tenga una gran influencia en la calidad del software. Existen varias metodologías para el diseño

como son el diseño orientado al flujo de datos, el diseño orientado a objetos, el diseño orientado a los datos y el diseño de tiempo real.

En el proceso de diseño de datos según Wasserman, la actividad principal es la selección de las representaciones lógicas de los objetos de datos (estructuras de datos), identificados durante las fases de definición y especificación de requisitos. El proceso de selección puede implicar un análisis algorítmico de las estructuras alternativas, encaminado a determinar el diseño más eficiente, o puede simplemente implicar el uso de un conjunto de módulos que proporcione las operaciones deseadas sobre alguna representación de un objeto.

Una actividad importante durante el diseño es la de identificar los módulos del programa que deben operar directamente sobre las estructuras de datos lógicas. De esta forma puede restringirse el ámbito del efecto de las decisiones concretas de diseño de datos.

Independientemente de las técnicas de diseño usadas, los datos bien diseñados pueden conducir a una mejor estructura de programa, a una modularidad efectiva y a una complejidad procedimental reducida.

- Diseño arquitectónico:

Su objetivo es desarrollar una estructura de programa modular y representar las relaciones de control entre los módulos. Además mezcla la estructura de programas y la estructura de datos y define las interfaces que facilitan el flujo de los datos a lo largo del programa.

- Diseño procedimental:

En éste se emplean 3 construcciones lógicas que son la secuencia, la condición y la repetición. La secuencia implementa los pasos de procesamientos esenciales de la especificación de cualquier algoritmo, la condición da la posibilidad de seleccionar un procedimiento basado en alguna ocurrencia lógica y la repetición proporciona iteración. Estas tres construcciones son fundamentales en la programación estructurada que es una técnica de diseño dentro de la ingeniería del software.

Las construcciones estructuradas se propusieron para limitar el diseño procedimental del software a un pequeño número de operaciones predecibles; el uso de éstas reduce la complejidad de los programas y facilita la legibilidad, la prueba y mantenimiento. Cualquier

programa, independientemente del área de aplicación y de la complejidad técnica puede diseñarse e implementarse usando solo las tres construcciones estructuradas.

Diagrama de flujo: Es la representación gráfica más ampliamente usada en el diseño procedimental y consiste en un gráfico muy sencillo; para representar un paso de procesamiento se utiliza un cuadro, un rombo para representar una condición lógica y flechas para mostrar el flujo de control.

También es posible emplear el lenguaje de diseño de programas o pseudocódigo, el cual es un lenguaje que utiliza el vocabulario de un idioma como el español y la sintaxis de un lenguaje de programación estructurada.

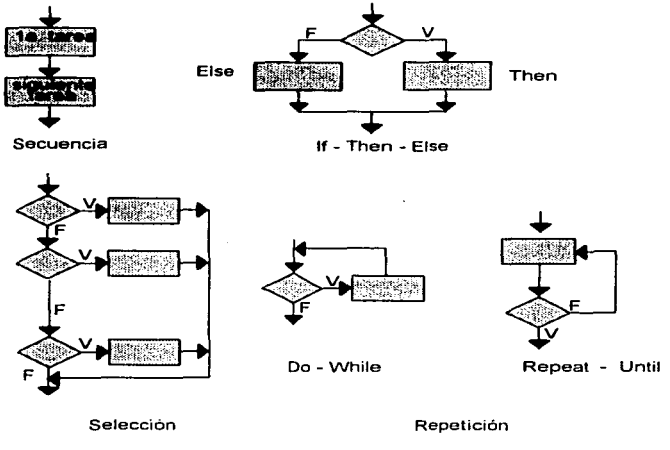


Figura III.5 Construcciones lógicas en diagramas de flujo.

Un lenguaje de diseño de programas puede ampliarse de modo que incluya palabras clave para procesamiento multitarea y/o concurrente, manejo de interrupciones, sincronización entre procesos y muchas otras características adicionales.

- Diseño de interfaces

Es común encontrar interfaces difíciles de aprender, complicadas de utilizar, confusas, imperdonables y, en muchos casos, totalmente frustrantes. Aun así alguien dedicó tiempo y energía construyendo cada una de esas interfaces y es casi seguro que el constructor no creó esos problemas intencionalmente.

A medida que se ha ido incrementando el uso de las computadoras, encontramos interfaces inteligentes cuando utilizamos una fotocopiadora, un horno de microhondas, un procesador de textos o un sistema de diseño asistido por computadora. Desde el punto de vista del usuario, lo que permite a un piloto pilotear un avión moderno, a un banquero transferir millones de dólares a través de continentes, es la interfaz. Ésta es en muchos sentidos el "envoltorio" del software de computadora. Si es fácil de aprender, simple de utilizar, directo y no muy estricto, el usuario podrá hacer buen uso de lo que hay dentro; si por el contrario, no tiene ninguna de estas características, aparecerán problemas invariablemente.

Si el diseño de la interfaz es bueno el usuario se adaptará a un ritmo normal de interacción, pero si no lo es, el usuario lo sabrá inmediatamente y se sentirá a disgusto con un modo de interacción poco amigable.

Las directrices para la interacción general sobrepasan, a menudo, las fronteras que la separan de la visualización de la información, la entrada de datos y el control global del sistema. Son por lo tanto, complementarias y se corre un gran riesgo ignorándolas. Las siguientes directrices se centran en la interacción general:

⇒ Ser consistente.- Se debe utilizar un formato consistente para la selección de menús, la entrada de órdenes, la visualización de datos y cientos de otras funciones que incorpora una interfaz hombre-maquina.

⇒ Ofrecer una realimentación significativa.- Se debe proporcionar al usuario una realimentación visual y auditiva para asegurar que se establece una comunicación bidireccional (entre el usuario y la interfaz).

⇒ Preguntar por la verificación de cualquier acción destructiva no trivial.- Si un usuario pide borrar un archivo, se pueden utilizar mensajes del tipo ¿Esta seguro?.

⇒ Permitir vuelta atrás fácil en la ejecución de la mayoría de las acciones.

⇒ Reducir la cantidad de información que debe ser memorizada entre acciones.

⇒ Buscar eficiencia en el diálogo, el movimiento y el pensamiento.- El número de pulsaciones debe ser minimizado, la distancia que un ratón debe recorrer entre dos pulsaciones también debe ser tenida en cuenta al diseñar el formato de presentación.

⇒ Perdonar los errores.- El sistema debe protegerse de los errores del usuario que pudiesen afectarle causándole un fallo.

⇒ Categorizar las actividades en base a su función y organizar la geografía de la pantalla convenientemente.

⇒ Proporcionar facilidades de ayuda sensible al contexto.

⇒ Utilizar verbos o frases de acción simples.

⇒ Mostrar solo aquella información que sea relevante en el contexto actual.- No abrumar al usuario con datos, los gráficos o esquemas deben reemplazar a las tablas de datos. Se pueden utilizar ventanas para modularizar los diferentes tipos de información.

⇒ Utilizar mayúsculas y minúsculas, tabulaciones y agrupaciones de texto para ayudar a la comprensión.

⇒ Minimizar el número de acciones de entrada de datos que debe realizar el usuario.

⇒ La interacción también debe ser flexible y estar ajustada al modelo de entrada preferido por el usuario (ratón o teclado por ejemplo).

⇒ Desactivar ordenes que sean inapropiadas en el contexto actual.- Esto evita que el usuario realice acciones que podrían conducir a un error.

⇒ Eliminar entradas innecesarias.- Por ejemplo tener que escribir el .00 en las entradas de dinero.

Desde el punto de vista de gestión del proyecto, el diseño va del diseño preliminar al diseño detallado.

A lo largo del proceso de diseño, la calidad del diseño resultante se evalúa mediante una serie de revisiones técnicas formales. Para evaluar la calidad de una representación del diseño se deben establecer criterios que determinen cuándo es bueno

Un diseño debe de:

- Exhibir una organización jerárquica que haga un uso inteligente del control entre los componentes del software.
- Ser modular.
- Contener representaciones distintas y separadas de los datos y de los procedimientos.
- Llevar a módulos (subrutinas o procedimientos) que exhiban características funcionales independientes.
- Llevar a interfaces que reduzcan la complejidad de las conexiones entre los módulos y el entorno exterior.
- Obtenerse mediante un método que sea reproducible y que esté conducido por la información obtenida durante el análisis de los requisitos del software.

Los primeros trabajos sobre diseño se centraron sobre los criterios para el desarrollo de programas modulares y los métodos para mejorar la arquitectura del software de una manera descendente. Los aspectos procedimentales de la definición del diseño evolucionaron hacia una filosofía denominada programación estructurada. Posteriores trabajos propusieron métodos para la traducción de la estructura de los datos en una definición de diseño. Nuevos enfoques para el diseño proponen un método orientado a objetos para la obtención del diseño.

Las diferentes metodologías de diseño coinciden en varias características comunes: 1) Un mecanismo para la traducción de la representación del campo de información en una representación de diseño; 2) Una notación para representar los componentes funcionales y sus interfaces; 3) Técnicas para el refinamiento y la partición; 4) Criterios para la valoración de la calidad.

Independientemente de la metodología de diseño usada el ingeniero de software debe aplicar un conjunto de conceptos fundamentales al diseño de datos, arquitectónico y procedimental.

Cabe señalar que en el diseño no hay que precipitarse, se debe de elaborar con el tiempo suficiente a fin de evitar errores que posteriormente se descubran y resulte más cara y laboriosa su corrección.

Codificación.- Es el proceso que transforma el diseño en un lenguaje de programación.

Cuando se considera como un paso de la ingeniería del software, la codificación es una consecuencia natural del diseño. Sin embargo, las características del lenguaje de programación y el estilo de programación pueden afectar profundamente a la calidad y al mantenimiento del software.

El paso de codificación traduce una representación del software, dada por un diseño detallado, a una realización en un lenguaje de programación. El proceso de traducción continua cuando un compilador acepta el código fuente como entrada y produce como salida un código objeto dependiente de la máquina. Posteriormente la salida del compilador es traducida a código máquina.

La interpretación equivocada de las especificaciones del diseño detallado puede conducir a un código fuente erroneo.

La características del lenguaje de programación tienen un impacto directo sobre la calidad y la eficiencia de la traducción.

Guía de estándares de programación:

Es común que unos programadores mantengan sistemas informáticos desarrollados por otros. Los creadores inciciales pueden haber cambiado de cometido o de empleo. Una característica apreciada en el código fuente es que sea predecible. Las tareas más comunes, tipo introducción de datos o pantallas de edición, deberían estandarizarse.

Un modo de hacer más predecible el código es definir una guía de estandares de programación, que describa los atributos que serán imprescindibles en todo programa desarrollado por la organización (que edita la guía). Los principales puntos de la guía de estándares de programación pueden ser los que siguen:

- **Legibilidad del código.-** Emplear mayúsculas y minúsculas, sangrado de las estructuras de control y el espaciado de las sentencias.

- **Convenios de elección de nombres.-** Recomendar la utilización de nombres racionalmente elegidos para variables, programas, archivos, apoyándose con ejemplos. Referir al diccionario de datos la normas aplicables a los nombres.

- **Documentación interna.-** Promover el uso de comentarios, tanto de línea completa como los adyacentes a las sentencias ejecutables. La guía debe contar con ejemplos para su aplicación. También debe de explicar la naturaleza propia de la documentación del sistema.

- **Técnicas estructuradas.-** Describir las normas y recomendaciones para la creación de diseños estructurados descendentes, la construcción de módulos del programa y de los archivos de procedimientos.

- **Software de utilidad para proyectos.-** Elaborar una lista de productos utiles como editores, generadores de código, formateadores de código, etc. Listar también sus características más interesantes y algunas breves instrucciones para su uso.

- **Librerías estándar de código.-** Los programadores acumulan con el paso del tiempo un amplio conjunto de programas y subrutinas de utilidad general. En la guía de estandares de programación deben aparecer referencias a estos valiosos recursos y puede incluir también descripciones de los programas de librería del sistema.

La guía de estandares debe de especificar qué se considera como buen estilo de programación, a fin de obtener un código más uniforme. La meta de todas estas sugerencias es la producción de más código, (claro y sencillo) en menos tiempo y de gran calidad.

3.4.- Depuración y prueba

La prueba no debe ser una actividad agregada al final del desarrollo de software. Para ser más efectiva debe llevarse a cabo como una parte en curso del proceso de desarrollo de software: los primeros errores son los más fáciles de encontrar y más baratos de corregir.

La opinión de que la prueba de software es una etapa individual del proceso de desarrollo después del código ha dejado la producción de software de pobre calidad. Las amplias prácticas actuales de la industria involucran pocas técnicas o herramientas de prueba. En lugar de estas, algunos conjuntos de datos de prueba se introducen en forma manual en un intento por demostrar que el código bajo prueba trabaja.

Existen muchas herramientas de prueba, técnicas y métodos disponibles en la actualidad. Si se ve más allá de los mejoramientos que pueden hacerse con sólo investigar la práctica actual, hay un número de áreas en las que la prueba puede volverse más efectiva, incluyendo medidas objetivas para la planeación de pruebas y el incremento de la automatización.

Quizá lo más importante en la dirección de la calidad es el movimiento de la detección de errores hacia la prevención de los mismos. Esto es soportado en el presente por el uso de revisiones y sesiones de revisión en etapas iniciales del ciclo de vida y, en potencia, por el empleo de técnicas de especificación más rigurosas.

Dentro de los diferentes tipos de prueba se tienen:

- La prueba de unidad la cual se centra en cada unidad o módulo del software, tal como está implementada en código fuente. Usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes, con el fin de descubrir errores dentro del ámbito del módulo. La complejidad relativa de las pruebas y los errores descubiertos está limitada por el alcance estricto establecido por la prueba de unidad y se puede llevar a cabo paralelamente para múltiples módulos.

- La prueba de integración cuyo foco de atención es el diseño y la construcción de la arquitectura del software. Una vez que todos los módulos han sido probados por separado es de esperarse que al ponerlos a interaccionar funcionen bien, sin embargo se pueden tener problemas tales como perdida de datos en una interfaz, un módulo puede tener un

efecto adverso e inadvertido sobre otro; las subfunciones, cuando se combinan, pueden no producir la función deseada y pueden presentarse problemas en las estructuras globales de datos.

La prueba de integración es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción.

- La prueba de validación es donde se validan los requisitos establecidos como parte del análisis de requisitos del software; se busca que el software funcione de acuerdo con las expectativas del cliente.

- La prueba del sistema, en la que se prueban como un todo software y otros elementos del sistema, realmente está constituida como una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora:

  - Prueba de recuperación: En algunos casos, un sistema debe ser tolerante a fallos, o sea, los fallos del procesamiento no deben hacer que cese el funcionamiento de todo el sistema. Esta prueba fuerza el fallo del software de muchas formas y verifica que la recuperación se lleva a cabo apropiadamente. Si la recuperación es automática (llevada a cabo por el propio sistema) hay que evaluar la corrección de la reinicialización, de los mecanismos de recuperación del estado del sistema, de la recuperación de datos y del rearranque. Si la recuperación requiere de intervención humana, hay que evaluar los tiempos medios de reparación para determinar si están dentro de límites aceptables.

  - Prueba de seguridad: En esta prueba se intenta accesar al sistema por diferentes medios para detectar puntos débiles por donde se pudiera colar alguna visita no deseada.

  - Prueba de resistencia: El encargado de la prueba intenta tirar abajo el programa. Se ejecuta un sistema de forma que demande recursos en cantidad, frecuencia o volúmenes anormales. Por ejemplo: (1) diseñar pruebas especiales que generen diez interrupciones por segundo, cuando las normales son una o dos; incrementar las frecuencias de datos de entrada en un orden de magnitud con el fin de comprobar cómo responden las funciones de entrada; (3) ejecutar casos de prueba

que requieran el maximo de memoria o de otros recursos; (4) diseñar casos de prueba que produzcan búsquedas excesivas de datos residentes en disco.

- Prueba de rendimiento: Se usa para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado.

Proceso de depuración: Comienza con la ejecución de un caso de prueba. Se evalúan los resultados y aparece una falta de correspondencia entre los esperados y los reales. En muchos casos los datos que no corresponden son un síntoma de una causa subyacente que todavía permanece oculta. El proceso de depuración intenta hacer corresponder el sistema con una causa, llevando así a la corrección del error. El proceso de depuración siempre tiene uno de dos resultados: (1) se encuentra la causa, se corrige y se elimina; o (2) no se encuentra la causa. En este último caso, la persona que realiza la depuración debe sospechar la causa, diseñar un caso de prueba que ayude a validar sus sospechas y el trabajo vuelve hacia atrás a la corrección del error de una forma iterativa.

Clasificación de errores

Las grandes organizaciones operan simultáneamente una gran variedad de sistemas, provocando conflictos a los directivos en la gestión de las tareas de desarrollo y mantenimiento. Una solución pasa por definir categorías de errores, ordenadas por su importancia, de modo que pueda determinarse la prioridad de cualquier problema que pueda surgir. A continuación se presenta una clasificación de seis niveles de errores:

Nivel 1: Errores estéticos pequeños.- Son errores de menor cuantía que no afectan al funcionamiento del sistema. Por ejemplo combinaciones de colores desafortunadas o datos mal alineados en pantallas o informes. Ni la utilización ni la exactitud del sistema son alteradas por estos detalles.

Nivel 2: Inexactitudes pequeñas.- Un error que afecta a la exactitud de un dato de una pantalla o informe. El usuario no considera que el problema sea grave, y el sistema puede ponerse en marcha, pese a lo cual el problema debe corregirse. Un ejemplo sería un sistema que en pantalla muestra cifras erróneas pero imprime las cifras correctas.

Nivel 3: Errores estéticos importantes.- Son más serios que los del nivel 1. Pueden irritar o cansar en exceso al usuario, pero no afectan a la utilización o exactitud. Por ejemplo

combinaciones de colores particularmente desagradables (como verde brillante sobre fondo rosa) que dan dolor de cabeza al mirarlas durante algún tiempo; o también, la mala colocación de las columnas en un informe, que influye negativamente en el usuario a la hora de comparar datos.

Nivel 4: Errores repetitivos.- Un error que deja al sistema fuera de combate cada cierto tiempo, sin que se produzca pérdida de información. Por ejemplo determinada combinación de opciones de impresión de informes que mete al sistema en un bucle infinito (Loop). El usuario debe rearrancar el sistema, pero no pierde más que tiempo.

Nivel 5: Inexactitudes importantes.- Errores que hacen aparecer información incorrecta en pantallas o informes, la equivocación es lo bastante seria como para impedir el funcionamiento normal del sistema. Por ejemplo no es útil un sistema de facturación que calcula mal los impuestos.

Nivel 6: Destrucción de datos.- La clase de error más importante, destruye o altera datos, que deben ser reintroducidos o recuperados desde su fuente. Por ejemplo una subrutina mal codificada que reescribe archivos de datos y destruye los registros de auditoría.

La clasificación de los problemas hace posible asignar prioridades a los errores, para tratar primero los más serios.

Estándares de prueba

ANSI: American National Standards Institute (Instituto americano nacional de estándares)
IEEE: Intitute of Electrical and Electronics Engineers, Inc. (Insituto de ingenieros eléctricos y electrónicos)
ISO: International Standard Organization. (Organización internacional de estándares)

La industria del software en general ha sido lenta para adoptar estándares de prueba, y es sólo en los últimos años que un estímulo coherente hacia la calidad en esta área ha tenido lugar. La adopción de estándares como IEEE Standard for Software Verification and Validation Plans, ANSI/IEEE Std 1012-1986 o IEEE Standard for Test Documentation, ANSI/IEEE Std 829-1983, no garantizarán la calidad del software producido bajo los mismos, pero va algo del camino hacia un enfoque disciplinado para la producción de software; los estándares deben verse como un nivel mínimo de logro más que como un cielo al cual aspirar.

El uso de estándares, en conjunción con otros mecanismos de control (tal como el control de versión), pueden proporcionar un marco para asegurar que:

- Los errores se detectan y corrigen tan pronto como sea posible en el ciclo de vida del software.
- Los efectos del riesgo del proyecto, costo y calendarización se minimizan.
- La calidad y confiabilidad del software resaltan.
- Los cambios propuestos y sus consecuencias se pueden apreciar con rapidez.

El requisito de que el software sea cada vez de mayor calidad demanda un planteamiento más sistemático de la prueba, de acuerdo con Dunn y Ullman [1] se requiere una estrategia global, que desborde el espacio estratégico de la prueba, tan deliberada en su metodología como lo fue el desarrollo sistemático en el que se basaron el análisis, el diseño y la codificación.

[1] Dunn R., R. Ullman. Quality Assurance For Computer Software. Ed. Mcgraw-Hill. 1982

3.5.- Documentación, entrenamiento y soporte

La documentación del sistema, la formación de los usuarios y el soporte del producto, son las características que a menudo distinguen un sistema utilizado satisfactoriamente de otros que nadie quiere utilizar. El esfuerzo dedicado a dichas tareas no necesita ser enorme para resultar efectivo. Un sistema puede quedarse sin utilizar por razones triviales: quizá nadie sabe cómo cargar un disco, formatearlo o quitar la protección de escritura; o tal vez no saben cómo arrancar el programa, configurarlo, o situarlo en el subdirectorio correcto. La mayor parte de la gente, incluso en países desarrollados, no tiene ni idea de informática, por extraño que pueda parecer a los que a diario trabajan con ella.

La documentación del usuario suele ser una guía de referencia (impresa o electrónica) de gran ayuda en el manejo de la aplicación. También puede constituir un recurso comercial: un comprador potencial se decidirá por un programa en concreto si éste tiene una documentación clara y atractiva. La buena documentación demanda un esfuerzo conjunto por parte de los programadores, los usuarios integrados en el equipo de diseño y otros usuarios que no conozcan el sistema de antemano. La documentación del usuario tiene muchos componentes como se muestra en la siguiente tabla:

Componente	Responsable
Descripción del sistema	U/I
Diagrama general	I
Disposición de pantallas e informes	I
Definiciones de datos	I/U
Situaciones especiales	U/I
Lista de comprobación de operaciones	U
Definiciones de términos	U/I

U = Usuario
I = Ingeniero de software

Tabla III.1 .- Componentes de la documentación del usuario

La documentación del sistema es aquella que sirve al programador que está a cargo del mantenimiento del sistema.

Sugerencias para una buena documentación:

- Adecuación del estilo al tipo de usuario.- El estilo de texto debe estar en consonancia con el nivel esperado del usuario que lo leerá. Un manual para principiantes requiere en general mayor dedicación porque la explicación debe ser muy clara e inteligible, a veces sumamente detallista. Un estilo muy utilizado en estas circunstancias es el llamado descripción tecla por tecla en el cual se define e ilustra cada pulsación. Para usuarios con más experiencia dicho estilo resulta tedioso.

- Facilidad de lectura.- Se recomienda utilizar la voz activa en lugar de la pasiva: es más simple, breve e interesante. Comparación entre las voces:

  Pasiva: una vez que se ha rellenado el impreso púlse la tecla RePág.

  Activa: rellene el impreso y después pulse la tecla RePág.

Evitar el uso de la jerga informática y de las siglas en la medida de lo posible. Muchas veces se da por sentado un conocimiento general de muchos términos que no corresponde con la realidad.

Para cambiar el contenido de un registro:

1. Pulse [M] para Modificar
2. Pulse la tecla Intro [Intro]
3. Haga los cambios oportunos
4. Pulse la tecla Av Pág [Av Pág] para guardar los cambios
5. Pulse [S] para Salir

Figura III.6.- Estilo descriptivo "tecla por tecla" para manuales de usuario.

Utilizar frases cortas, en ocasiones las frases largas y enrevesadas se entienden mal.

- Exactitud.- Comprobar que la información no da lugar a malas interpretaciones, se deben escoger las palabras con propiedad.

- Numeración de las páginas.- Se debe elegir un sistema de numeración que permita añadir páginas con facilidad.

- Encuadernación.- Si se pronostica la posibilidad de hacer cambios y adiciones será mejor poner la documentación en una carpeta de anillos.

- Identificación de versión. En el manual debe figurar la fecha y número de versión.

Explicación de los mensajes de error.- Conviene describir con toda exactitud los mensajes de error que el usuario puede encontrar durante la ejecución de la aplicación. Se deberá explicar la forma de corregir el problema. Un ejemplo de listado de mensajes de error es:

Mensaje	Explicación	Solución
Base de datos vacia	No hay ningún registro en la base de datos.	No existe problema alguno. Puede añadir registros con la opción "Añadir".
Registro duplicado	Hay un registro en la base de datos con el mismo número de transacción.	Introduzca el registro con un número de transacción diferente.

Tabla III.2.- Ejemplo de listado de mensajes de error

La documentación del usuario debe considerarse como un medio de comunicación. que debe indicar cómo emplear el sistema con la máxima eficacia. Existen varias formas de transmitir ideas y conceptos sobre un papel, siendo la más inmediata el texto descriptivo: otras son las tablas, listas, diagramas y demás clases de gráficos.

Pantallas de ayuda: Documentación en línea, "on - line".- Cuando se habla de documentación del usuario, normalmente se piensa en un manual impreso; sin embargo en la actualidad es frecuente documentar partes del sistema mediante pantallas de ayuda.

La documentación del sistema está destinada principalmente al programador que está a cargo del mantenimiento del sistema. Cabe señalar que el gasto máximo de recursos para la mayoría de las aplicaciones se hará durante la etapa de mantenimiento.

Algunos lenguajes proporcionan herramientas para documentación del sistema, por ejemplo Fox Pro incluye Fox Doc, el cual es una excelente herramienta capaz de producir casi todo lo necesario para confeccionar la documentación del sistema. Fox Doc creará por cada archivo de programa PRG un archivo con extensión ACT que contendrá el código fuente formateado con números de línea y diagramas de acción. Los diagramas resultan especialmente valiosos para aclarar la extensión y organización de las estructuras de control.

Un diagrama de árbol proporciona una buena panorámica general de la estructura del sistema. Es factible incluir en ellos los archivos de datos y de índices.

El resumen de procedimientos y funciones ofrece un registro detallado de la actividad desarrollada por los procedimientos, incluyendo el paso de variables y parámetros.

Entrenamiento:

Los buenos manuales y la ayuda electrónica resultan bastante insuficientes en muchas ocasiones: los usuarios necesitan además entrenamiento y formación (oficial o informal) para comprender un programa. Las personas poseen muy diversos casos de motivación, conocimientos técnicos y formas de aprendizaje. Un usuario fuertemente motivado y con amplios conocimientos técnicos será capaz de trabajar con una aplicación sin formación específica previa; en cambio, un trabajador de tipo medio con experiencia informática escasa o nula necesitará formación complementaria incluso para el sistema más sencillo. La siguiente figura muestra según su eficacia, los diferentes tipos de formación existentes.

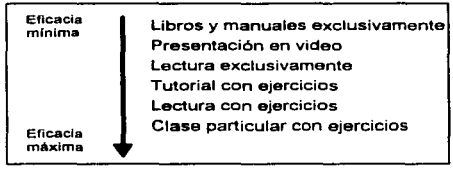


Figura III.7.- Métodos de formación ordenados según su eficacia

Soporte

Independientemente de lo fácil que sea utilizar el programa, de lo bien escrito que esté el manual y de lo buena que haya sido la clase de entrenamiento es muy posible que alguien tenga alguna duda y necesite de alguien a quien preguntar.

El soporte debe preeverse desde el principio y diseñarse durante la planificación del sistema.

Una posibilidad que puede ahorrar mucho tiempo y esfuerzo es el soporte a distancia. Para ello se necesita contar con modems, una línea telefónica y software de comunicaciones. Cuando el usuario llama al personal de soporte éstos conectan al PC del usuario de forma que puedan ver la misma pantalla que él y controlar el PC desde su teclado remoto.

El soporte a distancia puede ayudar a solucionar al momento muchos problemas o errores porque la persona de soporte puede ver cuál es el problema exacto.

La mayoría de los paquetes de comunicaciones permiten la transferencia de archivos; con esta capacidad es posible corregir un problema simplemente con la transferencia del archivo correcto al PC del usuario a través de la línea telefónica.

La documentación, el entrenamiento y el soporte son elementos clave para la correcta implantación de un sistema de información, que deben planificarse desde las primeras etapas de diseño. Muchos sistemas no satisfacen las expectativas o dejan de utilizarse a causa de errores triviales en estos aspectos. No debe escatimarse en ellos aunque el proyecto haya excedido su presupuesto o plazo de ejecución.

3.6.- Mantenimiento y verificación

La idea central del uso de técnicas de ingeniería del software, se centra en el desarrollo de sistemas cuyas necesidades de mantenimiento posterior a la puesta en funcionamiento sean mínimas.

Tipos de mantenimiento:

Existen 2 vertientes: La primera es el mantenimiento correctivo o reparación aun cuando el sistema haya superado todas las series de pruebas, todavía aparecerán en él errores inesperados. Los tipos más comunes son los errores lógicos y de sintaxis. En estos casos el programa no hace aquello que se espera de él y debe ser corregido. Otro tipo de error es el de especificación en el que por ejemplo el usuario olvidó alguna característica que debía incluirse en el diseño, o bien se produjo un fallo de comunicación entre el usuario y el desarrollador.

La segunda forma de mantenimiento tiene que ver con el cambio de las especificaciones. Es posible que las exigencias de la organización hayan evolucionado desde la época en la que el sistema fué diseñado, una situación muy común en la actualidad.

Es importante que se realice la asignación sistemática y objetiva de prioridades a las actividades correctivas de los sistemas, por ejemplo de acuerdo al error se propone la siguiente ponderación:

Nivel	Tipo	Peso
1	Errores estéticos pequeños	1
2	Inexactitudes pequeñas	5
3	Errores estéticos importantes	10
4	Errores repetitivos	20
5	Inexactitudes importantes	50
6	Destrucción de datos	100

Tabla III. 3 .- Ponderación de prioridades de mantenimiento de acuerdo al tipo de error.

Registro del mantenimiento

Cuando un sistema crece en complejidad, su mantenimiento crece igualmente en incomodidad. Una herramienta que puede ayudar a gestionar el mantenimiento es un registro de operaciones. Este sistema utiliza una base de datos para guardar la información básica de cada acción de mantenimiento. La estructura de la base de datos propuesta se muestra a continuación:

Registro del mantenimiento

Nombre del sistema [redacted]	Prioridad del sistema [redacted]
Cliente [redacted]	Solicitado por [redacted]
Fecha de solicitud 30_JUN_96	Reparación/Mejora R
Descripción del problema Memo	Nivel error 1
Fecha de Inicio 30_JUN_96	Prioridad 3.00
Acción propuesta Memo	Analista [redacted]
Fecha final 01_JUL_96	Horas estimadas 10.00
Descripción de la solución Memo	Horas reales 11.50

Figura III.8.- Ejemplo de pantalla de edición del sistema de registro del mantenimiento

Es importante resaltar que todo cambio debe de ser probado a fondo, tanto internamente (ingeniero de software) como externamente (usuario). Los datos de la etapa de prueba deben conservarse para su utilización en la etapa de mantenimiento.

Cuanto mayor sea el cambio realizado más rigurosa y extensa ha de ser su prueba, como es natural.

A lo largo de su ciclo de vida el sistema existirá en un mínimo de 2 computadoras, en la del desarrollador y en la del usuario. Es necesario registrar siempre todos los cambios realizados y todas las versiones generadas como fruto de éstos, de forma que se evite cualquier confusión o duda al respecto. Si esta cuestión puede dar problemas hasta con un solo usuario, el problema se agranda cuando se considera la posibilidad de varios miles. La identificación y gestión de los cambios, su seguimiento y distribución a los usuarios deben hacerse conforme a un plan sistemático. Algunas ideas para poner en práctica estos mecanismos son:

- Hacer que el número de versión y la fecha de ésta aparezcan en la pantalla de presentación del programa; otro sitio excelente es el programa de instalación del paquete.

- Preparar un registro de versiones que de cada cambio conserve su fecha, los archivos que resultan afectados y una breve descripción del mismo.

- El mantenimiento es tan necesario para los datos como para los programas. A lo largo de la vida de una base de datos es posible que se tengan que añadir campos, cambiar longitudes o incluso contenidos.

Ingeniería inversa y Reingeniería

El término "ingeniería inversa" tiene sus orígenes en el mundo del hardware. Una compañía desensambla un producto de hardware de la competencia en un esfuerzo por comprender los "secretos" del diseño y la fabricación del competidor. Esos secretos se podrían comprender fácilmente si se obtuvieran las especificaciones del diseño y de la fabricación del competidor. Pero esos documentos son privados y no están disponibles para la compañía que lleva a cabo la ingeniería inversa. Esencialmente una ingeniería inversa fructífera desemboca en una o más especificaciones de diseño y fabricación de un producto, obtenidas examinando ejemplares reales del producto.

La ingeniería inversa del software es bastante similar. Sin embargo, en la mayoría de los casos, el programa objeto de la ingeniería inversa no es un producto de la competencia sino un trabajo propio de la compañía realizado años atrás. Los secretos a comprender están ocultos porque no se desarrolló nunca una especificación. Por tanto, la ingeniería inversa del software es el proceso recuperación de diseño. Las herramientas de ingeniería inversa extraen la información del diseño de datos, arquitectónico y procedimental de un programa.

La reingeniería no solo recupera la información de diseño de un software existente, sino que usa esa información para alterar o reconstruir el sistema existente, en un esfuerzo por mejorar la calidad en general. En la mayoría de los casos el software resultante de la reingeniería reimplementa la función del programa existente. Pero, al mismo tiempo, el desarrollador añade nuevas funciones y/o mejora el rendimiento general.

La reingeniería produce un diseño que produzca la misma función con mayor calidad que la del programa original y por tanto, una mejor facilidad de mantenimiento en el futuro.

Las herramientas de ingenieria inversa y de reingenieria llegaran a ser una parte importante del proceso de mantenimiento a finales de los 90.

El mantenimiento constituye la última fase del proceso de ingenieria de software y se lleva la mayor parte del presupuesto destinado al software de computadora. A medida que se desarrollan más programas, surge la tendencia molesta de que la cantidad de esfuerzo y de recursos dedicados al mantenimiento crece. Al final algunas organizaciones de software pueden llegar a encontrar la barrera del mantenimiento, no siendo capaces de emprender nuevos proyectos porque tienen todos sus recursos dedicados al mantenimiento de programas antiguos.

CAPITULO IV

IV.- CALIDAD DEL SOFTWARE

4.1.- Métricas de la calidad del software

En la mayoría de los desafíos técnicos, la medición y las métricas nos ayudan a entender tanto el proceso técnico que se utiliza para desarrollar un producto, como el propio producto. El proceso se mide para intentar mejorarlo. El producto se mide para intentar aumentar su calidad. Frecuentemente la medición conlleva una gran controversia y discusión.

¿Cuales son las métricas apropiadas para el proceso y para el producto? ¿Cómo se deben utilizar los datos que se recopilan? ¿Es bueno usar medidas para comparar gente, procesos o productos? Estas y otras preguntas surgen siempre que se intenta medir algo que no se ha medido en el pasado. Tal es el caso de la ingeniería del software (el proceso) y del software (el producto).

Se han desarrollado métricas del software para proporcionar a los gestores y a los técnicos una mejor comprensión del proceso de la ingeniería del software y del producto que se genera.

La métricas del software se refieren a un amplio rango de medidas para el software de computadoras. Dentro del contexto de la planificación del proyecto de software, nos preocupamos principalmente por las métricas de productividad y de calidad (medidas del rendimiento de la "salida" del desarrollo del software como función del esfuerzo aplicado).

Para los propósitos de planificación y de la estimación, el interés es histórico: ¿Cuál fue la productividad del desarrollo de software en anteriores proyectos? ¿Cómo era la calidad del software producido? ¿Cómo se pueden extrapolar al presente los datos de productividad anteriores? ¿Cómo nos pueden ayudar a estimar más adecuadamente?

Medición del software

La medición es muy común en el mundo de la ingeniería, se miden potencias de consumo, pesos, dimensiones físicas, temperaturas, voltajes, señales de ruido, etcétera. Sin embargo en ingeniería de software encontramos dificultades en ponernos de acuerdo sobre qué medir y cómo evaluar las medidas.

Las razones por que medir el software son:

1) Para indicar la calidad del producto

2) para evaluar la productividad de la gente que desarrolla el producto;

3) para evaluar los beneficios en términos de productividad y calidad, derivados del uso de nuevos métodos y herramientas de ingeniería de software;

4) para establecer una línea de base para la estimación;

5) para ayudar a justificar el uso de nuevas herramientas o de formación adicional.

Las medidas del mundo físico pueden englobarse en dos categorías: medidas directas (por ejemplo, la longitud de un tornillo) y medidas indirectas (por ejemplo, la "calidad" de los tornillos producidos, medida por el porcentaje de tornillos rechazados). Las métricas del software pueden ser catalogadas en forma similar.

Algunas medidas directas del proceso de ingeniería del software son el costo y el esfuerzo aplicado.

Entre las mediciones directas del producto se encuentran las líneas de código (LDC) producidas, la velocidad de ejecución, el tamaño de memoria y los defectos observados en un determinado período de tiempo.

Para las medidas indirectas del producto se tiene la funcionalidad, calidad, complejidad, eficiencia, fiabilidad, facilidad de mantenimiento entre otras.

En la industria se utilizan tanto las métricas orientadas al tamaño como las orientadas a la función. Las métricas orientadas al tamaño utilizan la línea de código como factor de normalización para otras medidas como personas-mes o defectos.

Con la medición y conocimiento de algunos datos de un proyecto específico como el número de líneas de código, el esfuerzo invertido en un proyecto en personas - mes, el costo total, el número de errores encontrados después de entregar el sistema al cliente dentro del primer

año de utilización y el número de páginas de documentación desarrolladas; podemos obtener un conjunto de métricas sencillas de productividad y calidad orientadas al tamaño:

Productividad = KLDC / persona - mes

Calidad = errores / KLDC

Costo = Pesos / KLDC

Documentación = páginas de documentación / KLDC

Las métricas orientadas al tamaño son bastante polémicas y no están aceptadas universalmente como mejor modo de medir el proceso de desarrollo de software.

Las métricas de software orientadas a la función son medidas indirectas del software y del proceso por el cual se desarrolla y se centran en la funcionalidad o utilidad del programa.

Los puntos de función se calculan rellenando la tabla que se muestra a continuación, para lo cual se determinan cinco características del ámbito de la información y los cálculos aparecen indicados en la posición apropiada de la tabla. Los valores del ámbito de la información están definidos de la siguiente manera:

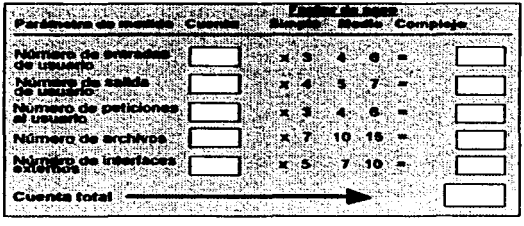


Figura III.9.- Cálculo de métricas de puntos de función.

Número de entradas de usuario.- Se cuenta cada entrada de usuario que proporciona al software diferentes datos orientados a la aplicación. Las entradas deben ser distinguidas de las peticiones, que se contabilizan por separado.

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

Número de peticiones al usuario.- Entradas interactivas que resultan de la generación de algún tipo de respuesta en forma de salida interactiva. Cada petición se cuenta por separado.

Número de interfaces externas.- Todas las interfaces legibles por la maquina por ejemplo, archivos de datos en cinta o disco, que son utilizados para transmitir información a otro sistema.

Una vez que han sido recopilados los datos anteriores, se asocia un valor de complejidad a cada cuenta, el cual es algo subjetivo.

Para calcular los puntos de función, se utiliza la siguiente relación:

$$PF = \text{Cuenta total} \times [\, 0.65 + 0.01 \times \Sigma\, (Fi)\,]$$

donde Cuenta total es la suma de todas las entradas PF obtenidas de la tabla anterior, Fi (i = 1 hasta 14) son los valores de ajuste de complejidad basados en las respuestas a las cuestiones de la siguiente tabla, los valores constantes de la ecuación anterior y los factores de peso aplicados a las cuentas de los ámbitos de información fueron determinados empiricamente.

Evaluar cada factor en una escala de 0 a 5:

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

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

Tabla III.4.- Cálculo de los puntos de función

Una vez calculados los puntos de función, se usan de forma análoga a las LDC como medida de productividad, calidad y otros atributos del software:

Productividad = PF / persona-mes

Calidad = errores / PF

Costo = Pesos / PF

Documentación = páginas de documentación / PF

El estándar del IEEE 982.1-1988 sugiere un índice de madurez del software

4.2.- Factores que determinan la calidad del software

McCall[1] propone una útil clasificación de los factores que afectan la calidad del software, los cuales se centran en tres aspectos importantes de un producto de software: sus características operativas, su capacidad de soportar los cambios y su adaptabilidad a nuevos entornos. Dichos factores aparecen en la siguiente figura:

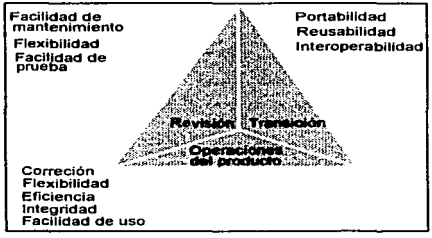


Figura III.10.- Factores de calidad del software de McCall

Para los factores mostrados McCall proporciona las siguientes descripciones:

Corrección.- El grado en que un programa satisface sus especificaciones y consigue los objetivos de la misión encomendada por el cliente.

Fiabilidad.- El grado en que se puede esperar que un programa lleve a cabo sus funciones esperadas con la precisión requerida.

Eficiencia.- La cantidad de recursos de computadora y de código requeridos por un programa para llevar a cabo sus funciones.

Integridad.- El grado en que puede controlarse el acceso al software o a los datos, por personal no autorizado.

Facilidad de uso.- El esfuerzo requerido para entrar a un programa, trabajar con él, preparar su entrada e interpretar su salida.

[1] McCall J:, P. Richards, G. Walkers. Factors in Software Quality. 1977

Facilidad de mantenimiento.- El esfuerzo requerido para localizar y arreglar un error en un programa.

Flexibilidad.- El esfuerzo requerido para modificar un programa operativo.

Facilidad de prueba.- El esfuerzo requerido para probar un programa de forma que asegure que realiza su función requerida.

Portabilidad.- El esfuerzo requerido para transferir el programa desde un hardware y/o un entorno de sistemas de software a otro.

Reusabilidad.- El grado en que un programa (o partes de un programa) se puede reusar en otras aplicaciones. Esto va relacionado con el empaquetamiento y el alcance de las funciones que realiza el programa.

Facilidad de interoperación.- El esfuerzo requerido para acoplar un sistema a otro.

Se define además un conjunto de métricas usadas para desarrollar expresiones para cada uno de los factores de acuerdo con la siguiente relación:

$$F_C = c_1 \times m_1 + c_2 \times m_2 + \ldots + c_n \times m_n$$

donde F_C es un factor de calidad del software, c_i son coeficientes de regresión y m_i son la métricas que afectan al factor de calidad, cuya escala de graduación va de 0 a 10, con las siguientes métricas:

Facilidad de auditoría.- La facilidad con que se puede comprobar la conformidad con los estandares.

Exactitud.- La precisión de los cálculos y del control.

Normalización de las comunicaciones.- El grado en el que se usan el ancho de banda, los protocolos y las interfaces estándar.

Completitud.- El grado en el que se ha conseguido la total implementación de las funciones requeridas.

Concisión.- Lo compacto que es el programa en términos de líneas de código.

Consistencia.- El uso de un diseño uniforme y de técnicas de documentación a lo largo del proyecto de desarrollo de software.

Estandarización en los datos.- El uso de estructuras de datos y de tipos estándar a lo largo de todo el programa.

Tolerancia de errores.- El daño que se produce cuando el programa encuentra un error.

Eficiencia en la ejecución.- El rendimiento en tiempo de ejecución de un programa.

Facilidad de expansión.- El grado en que se puede ampliar el diseño arquitectónico, de datos o procedimental.

Generalidad.- La amplitud de aplicación potencial de los componentes del programa.

Independencia del hardware.- El grado en el que el software es independiente del hardware sobre el que opera.

Instrumentación.- El grado en que el programa muestra su propio funcionamiento e identifica errores que aparecen.

Modularidad.- La independencia funcional de los componentes del programa.

Facilidad de operación.- La facilidad de uso de un programa.

Seguridad.- La disponibilidad de los mecanismos que controlen o protejan los programas o los datos.

Autodocumentación.- El grado en que el código fuente proporciona documentación significativa.

Simplicidad.- El grado en que un programa puede ser entendido sin dificultad.

Independencia del sistema de software.- El grado en que el programa es independiente de características no estándar del lenguaje de programación, de las características del sistema operativo y de otras restricciones del entorno.

Facilidad de traza.- La posibilidad de seguir la pista a la representación del diseño o de los componentes reales del programa hacia atrás, hacia los requisitos.

Formación.- El grado en que el software ayuda para permitir que nuevos usuarios apliquen el sistema.

La relación entre los factores de calidad del software y las métricas que se acaban de indicar se muestra en la siguiente tabla. Se debe recordar que el peso dado a cada métrica dependerá del producto en particular.

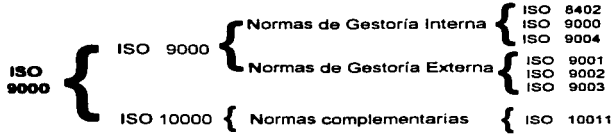
Factor de calidad \ Métrica de calidad del software	Corrección	Fiabilidad	Eficiencia	Integridad	Facilidad de manto	Flexibilidad	Facilidad de prueba	Portabilidad	Reusabilidad	Fácil interoperabilidad	Facilidad de uso
Facilidad de auditoría				X			X				
Exactitud		X									
Norm. de las comunicaciones										X	
Completitud	X										
Complejidad		X				X	X				
Consición			X		X	X					
Consistencia	X	X			X	X					
Estandarización en los datos										X	
Tolerancia de errores		X									
Eficiencia en la ejecución			X								
Facilidad de expansión						X					
Generalidad						X		X	X	X	
Independencia del hardware								X	X		
Instrumentación				X	X		X				
Modularidad		X			X	X	X	X	X	X	
Facilidad de operación			X								X
Seguridad				X							
Autodocumentación					X	X	X	X	X		
Simplicidad		X			X	X	X				
Indep. del sistema de software								X	X		
Facilidad de traza	X										
Formación											X

Figura III.11.- Relación entre las métricas y los factores de calidad del software

4.3- ISO 9000 como Directriz

La ISO.- Organización internacional de estándares (del inglés International Standard Organization), tiene su cede en Ginebra, Suiza y es una de las organizaciones que más se ha preocupado a nivel mundial por la estandarización. Existen otras organizaciones como la ANSI: American National Standars Institute (Instituto americano nacional de estándares) y el IEEE: Intitute of electrical and Electronics Engineers, Inc. (Insituto de ingenieros eléctricos y electrónicos) que tienen una gran cantidad de estándares y de publicaciones sobre la estandarización de diversos aspectos relacionados con hardware y software.

La familia de normas ISO 9000 se divide como sigue:



ISO 8402:1994 Administración de la calidad y aseguramiento de la calidad. Vocabulario
Esta norma es equivalente con la NMX-CC-001:1995

ISO 9000-1:1994. Administración de la calidad y aseguramiento de la calidad
Parte 1. Directrices para la selección y uso.
Esta norma es equivalente con la NMX-CC-002/1:1995

ISO 9001:1994 Sistemas de calidad - Modelo para asegurar la calidad en diseño/desarrollo, producción, instalación y servicio.
Esta norma es equivalente con la NMX-CC-003:1995

ISO 9002:1994. Sistemas de calidad - Modelo para asegurar la calidad en la producción, instalación y servicio.
Esta norma es equivalente con la NMX-CC-004:1995

ISO 9003:1994	Sistemas de calidad - Modelo para asegurar la calidad en la inspección y pruebas finales. Esta norma es equivalente con la NMX-CC-005:1995
ISO 9004-1:1994.	Administración de la calidad y elementos del sistema de calidad.- Directrices. Esta norma es equivalente con la NMX-CC-006/1:1995
ISO 9004-2:1994.	Administración de la calidad y elementos del sistema de calidad.- Directrices para servicios. Esta norma es equivalente con la NMX-CC-006/2.1995
ISO 10011-1:1993.	Directrices para auditar los sistemas de calidad. Parte 1. Auditorías. Esta norma es equivalente con la NMX-CC-007-1-1993
ISO 10011-2:1993.	Criterios de calificación para los auditores de los sistemas de calidad. Esta norma es equivalente con la NMX-CC-008-1993
ISO 10011-1:1993.	Directrices para auditar los sistemas de calidad. Parte 2. Administración de los programas de auditoría. Esta norma es equivalente con la NMX-CC-007-2-1993

El estándar ISO 9000 ha sido adoptado por los sistemas estandares de la mayoría de paises manufactureros alrededor del mundo. En la Comunidad Europea se conoce ahora como Euronorme EN29000-1987.

De manera básica, el ISO 9000 se refiere al aseguramiento de la calidad de las capacidades de la organización funcionales y consiste en documentar y estandarizar el proceso; representa en la actualidad la mejor práctica actual en administración y control cuando se aplica a la producción de software.

El sistema de calidad es necesario, toda vez que se debe de tener un plan básico para la calidad. Esto en realidad no sucede; solo es una forma de satisfacer de manera continua los requerimientos y expectativas de los clientes.

Un plan de calidad es un documento que establece los recursos de las prácticas específicas de calidad y actividades relevantes para un producto en particular, contratación de servicio o proyecto.

El sistema de calidad basado en ISO 9000 afecta a cualquiera asociado con la planeación, ventas, entrenamiento, suministro, manufactura, inspección pruebas, servicios al cliente e ingeniería; prácticamente a todos.

Los beneficios obtenidos con la implantacion de un sistema de calidad basado en ISO 9000 pueden dividirse en beneficios para el cliente y para la compañía:

Beneficios para el cliente:

- Un nivel conocido de calidad que es definido y puede ser medido.
- Un nivel de servicio que ha sido y continúa siendo auditado de manera independiente.
- Un desempeño que debe continuar mejorando.
- Un medio de seleccionar entre ofertas competitivas.
- Confidencialidad en los servicios proporcionados.

Beneficios para la compañía:

- Mejoras en la calidad a traves de :

  1) Incremento del conocimiento del personal.

  2) Consistencia en productos y servicios.

- Los empleados entienden su rol y objetivos al tener un sistema de administración documentado.
- Una moral incrementada, al desarrollar un sentido de orgullo al alcanzar objetivos y proporcionar satisfaccion al cliente.
- Productividad mejorada y ahorro en costos que hacen a la compañía más competitiva.
- Un sistema de administracion eficiente.

ISO 9001

El documento ISO 9001 es un modelo genérico para un sistema de administración de calidad El mismo documento puede usarse para construir un sistema de calidad en casi cualquier producto o servicio.

A continuación se comentan los 20 puntos que comprenden a ISO 9001:

1. Responsabilidad de la administración / política de calidad

Debe haber un programa definido para la calidad. La política debe ser comunicada y entendida a través de la compañía. La administración debe revisar de manera regular el sistema total de la calidad para asegurar su efectividad continua y conforme a ISO 9000. Debe haber administración de calidad efectiva con todas las responsabilidades definidas con claridad, por escrito, y suficientes recursos para hacer el trabajo.

- Se debe tener administración con autoridad y responsabilidad definida. (Organización).

2. Sistema de calidad

Todos los sistemas que directa o indirectamente afectan la calidad de nuestro producto y servicios deben ser documentados. Sin embargo, la documentación debe ser práctica, actual y controlada de manera efectiva. También debe corresponder a lo que pasa en la realidad.

- Es esencial que todos sepan qué es lo que se supone deben hacer.

3. Revisión del contrato

Se debe asegurar que:

a) Se sabe cuales son los requerimientos del cliente

b) Que esos requerimientos están documentados y revisados.

c) Todos los cambios a los requerimientos están resueltos.

d) Se tiene la capacidad de satisfacer los requerimientos.

e) Se está de acuerdo con el cliente.

- Los contratos y las órdenes deben ser revisados para garantizar que se continua satisfaciéndolos.
- Los clientes deben recibir los productos y servicios especificados.

4. Control del diseño

Este requerimiento puede asegurar que el diseño terminado de los productos o de los servicios satisface aquellas necesidades especificadas por el cliente.

- El diseño es el fundamento de la calidad. La secciones 3.2 y 3.3 tratan la necesidad de planear y revisar la estructura y la función del sistema en una etapa inicial.

5. Control de documentos y datos

La gente necesita saber qué se requiere que haga. Por lo tanto, las prácticas de trabajo deben estar documentadas, actualizadas, controladas y rápidamente disponibles. Los documentos obsoletos se deben retirar de uso con prontitud.

- ¿Estamos actualizados? ¿Todos utilizan la misma documentación?

6. Adquisiciones

Los materiales, partes y servicios que se compran a los proveedores deben corresponder con el propósito, satisfacer los propios requerimientos de calidad y ser controlados. Es necesario que las especificaciones de compra estén escritas con claridad para que un proveedor sepa lo que se desea con exactitud.

- El propósito es comprar sólo bienes y servicios que son de la calidad apropiada. Este requerimiento también incluye una estimación del vendedor y aseguramiento de subcontratos.

7. Control de productos proporcionados por el cliente

Los clientes pueden proveer materiales para utilizar en sus trabajos. Tales materiales deben ser almacenados de manera segura y es necesario usar algunos métodos para prevenir el deterioro o la pérdida.

8.- Identificación y rastreabilidad del producto

¿En cualquier momento es importante saber qué es cualquier ítem, a que pertenece, en qué etapa está, qué etapas ha completado ya, qué fue usado para construirlo, de dónde vinieron las partes y quién hizo qué?

Una vez que un ítem ha sido construido, es necesario que sepamos hacia dónde se dirige. Cuando sea apropiado, debe haber un seguimiento tanto hacia atrás como hacia delante.

- Si un elemento está equivocado en cualquier momento, su historia nos dice qué hacer para prevenir fallas futuras. Los sistemas grandes de software tienen muchos componentes, de ahí que sea esencial ser capaz de agrupar los correctos.

9.- Control del proceso

La producción e instalación debe ocurrir bajo condiciones controladas. Los empleados deben saber qué hacer y qué estándares se requieren. Es necesario que existan controles para asegurar que sólo se proporcionan productos de buena calidad. La operación puede ser compleja, de ahí que puede requerir verificaciones especiales, instrucciones adicionales, entrenamiento y calificaciones específicas. Las instrucciones adecuadas de trabajo así como los estándares a satisfacer se deberán escribir para asegurar que el trabajo se hace en forma correcta con las herramientas, partes, métodos y especificaciones apropiadas.

La calidad consistente requiere de procesos controlados. Siempre es necesaria una instrucción de trabajo; la ausencia de una afecta la calidad.

- Las instrucciones de trabajo deben ser fáciles de entender, prácticas, cortas y probadas.

10. Inspección y prueba

Los productos deben ser inspeccionados o verificados, y pasar el criterio de aceptación al momento de ser recibidos, trabajados y previamente a la distribución.
Todos los productos que no correspondan a esos estándares deberán segregarse para que una acción correctiva pueda llevarse a cabo. Es indispensable guardar los registros a fin de inspeccionar el trabajo efectuado en cada etapa.

- Asegurar que el producto cumple todas las especificaciones.

11. Control de equipo de inspección, medición y prueba

El equipo utilizado para la verificación o prueba de un producto tiene que ser capaz de desempeñar en forma correcta estas funciones. también se requiere que:

- Todo el equipo de prueba y medición esté identificado.
- Los registros muestren la frecuencia de calibración.
- Existan procedimientos de calibración de acuerdo a estándares.

12. Estado de inspección y prueba

- Distinguir entre inspeccionado y no inspeccionado (bueno y malo).

13. Control de productos no satisfactorios

- La reducción del desecho y la necesidad de volver a trabajar incrementa la eficiencia. ¿Tenemos software desechado?

14. Acción correctiva y preventiva

Todos los procedimientos, procesos y productos necesitan ser monitoreados y revisados para asegurar la entrega constante de calidad. Cualquier estándar no satisfecho debe ser documentado, analizado, identificar la causa y tomar acciones correctivas y preventivas.

15. Manejo, almacenamiento, empaque, conservación y entrega.

Los productos deben ser manejados con mucho cuidado a fin de prevenir daños. Deben ser almacenados en áreas seguras y controladas. Es necesario que todo el paquetaje esté en un estándar adecuado y, por lo tanto, prevenir el daño o deterioro de los productos mientras se encuentren en almacén o se entregan.

16. Control de registros de calidad

Los registros de calidad deben mantenerse para que puedan demostrar nuestro nivel de alcance contra los estándares requeridos y, de hecho probar la efectividad del sistema de calidad.

17. Auditorías de calidad internas

Los auditores del aseguramiento de la calidad interna llevan a cabo garantías y pruebas imparciales independientes para verificar si tanto los procesos como los procedimientos siguen siendo pertinentes, efectivos y satisfacen los requerimientos.

18. Entrenamiento

Para que los empleados desempeñen las tareas de manera satisfactoria y cumplan con los estándares de los requerimientos, se deben de capacitar en los métodos y técnicas correctos a utilizar.

- Se debe documentar el entrenamiento dentro y fuera del trabajo así como la experiencia derivada de ello. Debe haber un plan de cualquier entrenamiento futuro requerido.

19. Servicio

Donde el servicio de equipo es un requerimiento, debe estar bien especificado, llevado a cabo por una capacitación adecuada al personal y dentro de las escalas de tiempo acordadas.

- Un servicio bien hecho, a tiempo, por un equipo de trabajo capacitado de manera adecuada provee clientes satisfechos y felices.

20. Técnicas estadísticas

Se puede revisar o verificar la aceptación de los productos y procesos en una base ejemplo, pero el ejemplo debe ser representativo del universo. Los registros deben guardarse para identificar las tendencias y prevenir la asistencia en caso necesario.

- Aún no se tiene ninguna métrica aceptada a nivel universal que pueda ser utilizada para predecir o cuantificar el software; sin embargo podemos usar los métodos estadísticos básicos e intermedios en el análisis de datos obtenidos de las características propias de un software, para utilizarlos en el futuro como punto de comparación.

Estos 20 puntos explican lo que tiene que controlarse para instalar un sistema de calidad basado en el estándar ISO 9000.

4.4 Aseguramiento de la Calidad

Aseguramiento de la calidad.- Se refiere a garantizar al cliente que existen procedimientos efectivos para asegurar que todos los requerimientos del diseño serán satisfechos y que el producto resultante de la ingeniería total del diseño será satisfactorio.

Norris y Rigby definen el aseguramiento de la calidad como el negocio de confirmar que el buen software no es producto de la buena suerte sino de observar una buena práctica de administración.

Dada la importancia de los sistemas de software, el interés principal debe estar en el aseguramiento de la calidad, lo cual implica tener el control sobre la evolución técnica del software mismo, así como en el proceso de desarrollo.

El enfoque actual para el aseguramiento de la calidad del software es especificar un conjunto de procedimientos que definan las diferentes operaciones en el desarrollo que van a llevarse a cabo, tales como las convenciones para el formato y la documentación, estándares de código, etcétera; esto introduce alguna estabilidad básica al proceso de desarrollo de software, siendo la regla general:

- Diga lo que hace
- Haga lo que dice
- Sea capaz de probarlo

Este es el enfoque que sostiene todos lo estándares registrados para el aseguramiento actual de la calidad, tales como BS 5750 e ISO 9000. Cualquier organización que obtenga el registro o aprobación para esos estándares (o posea un esquema de peso equivalente) se considera que tiene un nivel básico de control sobre lo que produce.

La garantía de calidad del software (SQA) es un planificado y sistemático diseño de acciones que se requieren para asegurar la calidad del software. La responsabilidad de la garantía de calidad corresponde a muchos constituyentes de la organización, diganse ingenieros de software, gestores del proyecto, clientes y personas que trabajan dentro del grupo de aseguramiento de la calidad.

Dentro de las actividades de aseguramiento de la calidad de software se tienen:

1) aplicación de métodos técnicos,
2) realización de revisiones técnicas formales,
3) prueba del software,
4) ajuste a los estándares,
5) control de cambios,
6) mediciones,
7) registro y realización de informes.

La calidad del software debe estar diseñada para el producto o sistema, por esta razón el aseguramiento de la calidad comienza realmente con un conjunto de herramientas y métodos técnicos que ayudan al analista a conseguir una especificación de alta calidad y un diseño de alta calidad.

La adopción de un modelo apropiado de ciclo de vida para el desarrollo de software es un paso importante hacia el mejoramiento de la calidad. Es el primer nivel para tener bajo control el proceso de desarrollo de software.

Una vez que se ha creado una especificación (o prototipo) y un diseño, debe ser asegurada su calidad. La actividad central que permite garantizar la calidad es la revisión técnica formal, la cual es una especie de reunión del personal técnico con el único propósito de descubrir problemas de calidad. En muchas situaciones se ha visto que las revisiones son tan efectivas como la prueba para descubrir los defectos en el software.

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

El grado de aplicación de procedimientos y estándares en el proceso de la ingeniería del software varía de empresa a empresa. En muchos casos, los estándares vienen dados por los clientes o por mandamientos de regulación. En otras situaciones, los estándares se imponen por sí solos. Para los estándares formales (escritos), se debe establecer una actividad de aseguramiento de la calidad para garantizar que estos se siguen. La verificación del seguimiento de estándares puede ser llevada a cabo por los encargados del desarrollo de software como parte de una revisión técnica formal o, en situaciones en que se requiera una

verificación del seguimiento independiente por el grupo de aseguramiento de la calidad, mediante su propia auditoria.

Una de las principales amenazas para la calidad se deriva de los aparentemente benignos cambios. Cada cambio realizado sobre el software en potencia puede introducir errores o crear efectos laterales que propaguen errores. El formalizar el control de cambios mediante las peticiones formales de cambio, evaluar la naturaleza del cambio y controlar el impacto del cambio contribuyen directamente a la calidad del software.

La medición es una actividad integral para cualquier disciplina. Un objetivo importante para el aseguramiento de la calidad, es seguir la pista a la calidad del software y evaluar el impacto de los cambios de metodología y de procedimiento que intentan mejorar la calidad del software.

El registro de la información y la generación de informes para el aseguramiento de la calidad del software dan procedimientos para la recolección y divulgación de información acerca de la calidad. Los resultados de las revisiones, auditorias, control de cambios, prueba y otras actividades de aseguramiento de la calidad deben convertirse en una parte del registro histórico de un proyecto y deben ser divulgados a la plantilla de desarrollo para que tengan conocimiento de ellos.

CAPITULO V

V.- CONCLUSIONES

La ingeniería de software es una disciplina que integra métodos, herramientas y procedimientos para el desarrollo de software de computadora.

Se han propuesto varios modelos para el desarrollo de software distintos, cada uno con sus propias ventajas y desventajas, pero todos tienen una serie de fases genéricas en común, y es importante que se lleven a cabo y se les de la importancia que merecen a cada una de ellas para garantizar que el desarrollo del software se esta haciendo concientemente, con responsabilidad, con un nivel básico de control, y sobre todo con una gran inclinación hacia la calidad.

Cabe señalar que el solo hecho de cumplir con los pasos de las técnicas de ingeniería del software y con el modelo de aseguramiento de la calidad basado en ISO 9000 no nos asegura como resultado un producto de alta calidad, sin embargo nos proporciona un buen comienzo en la búsqueda de la calidad y nos garantiza que se cumplen los requisitos mínimos para alcanzar un resultado satisfactorio.

5.1 Perspectivas futuras

La medición, la disciplina y la preocupación continua por la calidad, darán como resultado un software que satisfaga las necesidades del cliente, eficiente y fácilmente mantenible; en una palabra un software mejor.

Considerando que la tecnología del software tiende a reaccionar ante los cambios en la tecnología del hardware, las perspectivas para la ingeniería de software estarán regidas por las tecnologías del hardware y del software. A medida que el software adquiera más fuerza en los problemas tales como inteligencia artificial, redes neuronales artificiales, sistemas expertos, etc., es muy probable que los enfoques evolucionen para afrontar dichos cambios.

A medida que vayamos entrando en el nuevo siglo, la ingeniería del software cambiará, pero a pesar de que los cambios sean radicales, es seguro que la calidad nunca perderá su importancia y que el diseño y el análisis efectivo así como una validación competente siempre tendrán un lugar en el desarrollo de sistemas basados en computadora.

5.2 Recomendaciones

Se debe tener presente que el software no es una ciencia. Es una actividad técnica con un peso relevante en la gente y su organización. Considerando que la mayoría de los ingenieros de software está conformada por individuos inteligentes e independientes, es muy difícil lograr que acepten un sistema de administración de la calidad; no obstante hay que considerar que se debe de involucrar a todos en el sistema de calidad; esto es, desde el personal de intendencia hasta el director de administración.

Para un desarrollo profesional de software con alta calidad, se recomienda adoptar las técnicas enunciadas en este trabajo, tanto de ingeniería del software como del modelo de aseguramiento de la calidad basado en ISO 9000.

Dichas técnicas comprenden la elección de un modelo para el desarrollo de software, la elaboración del análisis de requisitos, el diseño e implementación del software, depuración y prueba, documentación, entrenamiento y soporte, así como el mantenimiento y verificación, además de cumplir con los requerimientos de ISO 9000 para lo cual se requiere cumplir con ciertos requisitos tales como tener documentados los procesos, así como la elaboración de un manual de calidad en el cual se plasman las políticas de calidad y se describe el sistema de calidad de la organización.

Los sistemas de calidad están basados en el procedimiento y, por lo tanto requieren documentación y estándares, pero esta documentación debe ser apropiada al proyecto en curso y debe ser vista como una ayuda para el usuario.

La administración de la calidad de software es, en esencia una cuestión personal y no técnica. De aquí que la mayoría de las fallas y problemas se deba a la falta de comunicación. La mejor comunicación dentro del proyecto, con los clientes y proveedores, es la mejor calidad del producto final.

El grupo que lleva a cabo el aseguramiento de la calidad debe mirar el software desde el punto de vista del cliente. ¿Satisface en forma adecuada el software los factores de calidad? ¿Se ha realizado el desarrollo del software de acuerdo con los estándares preestablecidos? ¿Han desempeñado apropiadamente sus papeles las disciplinas técnicas como parte de la actividad

de aseguramiento de la calidad? El grupo de aseguramiento de la calidad intenta responder a éstas y otras cuestiones para asegurar que se mantiene la calidad del software.

La producción de un sistema de administración de la calidad diseñado de manera específica para el software puede servir de base para los sistemas futuros. Aunque es importante asegurar siempre que cualquier sistema está encaminado a satisfacer las necesidades del cliente y de la compañía, no sólo los requerimientos de los auditores.

CAPITULO VI

VI.- BIBLIOGRAFÍA

- Ingeniería de software explicada.
  Mark Norris - Peter Rigby
  Ed. Limusa S. A. de C. V.
  Grupo Noriega Editores. Megabyte, 1994

- Ingeniería del software. Un enfoque práctico.
  Roger S. Pressman
  Tercera Edición
  Ed. McGraw-Hill, 1993

- Aplicaciones de gestión con FOXPRO. Metodologías para el diseño de aplicaciones en empresas.
  Bill Chambers
  Ed. Anaya Multimedia America, S.A. de C.V., 1993

- Software engineering. Planning for change.
  David Alex Lamb
  Ed. Prentice Hall, 1988

- Redes de área local. La siguiente generación.
  Tomas W. Madron
  Ed. Limusa S. A. de C. V.
  Grupo Noriega Editores. Megabyte, 1992

- Auditoría en informática.
  José Antonio Echenique.
  Ed. McGraw Hill, 1991

- Implantar y gestionar la calidad total.
  A. Bernillon - O. Cerutti
  Ed. Gestión 2000, 1989

- Control total de la calidad: ingeniería y administración.
  A. U. Feigenbaum
  Ed. C.E.C.S.A., 1963

- Proyección.
  Revista de la Asociación Mexicana de Calidad A. C.
  No. 17, 1992

- Manual de control de calidad.
  Joseph M. Juran
  Ed. Reverte, 1983

- Como mejorar la calidad y la productividad con el método Deming: Una guía práctica para mejorar su posición competitiva.
  Howard S. Gitlow, Shelly J. Gitlow
  Ed. Norma, 1989