UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

ADMINISTRACION DE PROYECTOS PARA EL DESARROLLO DE SOFTWARE DE CALIDAD

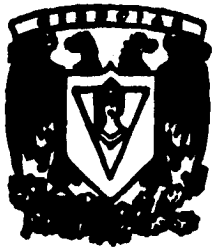
T E S I S

QUE PARA OBTENER EL TITULO DE:

M A T E M A T I C O

P R E S E N T A :

ANASTACIO NUÑEZ GUZMAN



DIRECTOR DE TESIS:

M.C. MA. GUADALUPE ELENA IBARGÜENGOITIA GONZALEZ

1 9 9 6

VNIVERSIDAD NACIONAL
AVPNOMA DE
MEXICO

M. en C. Virginia Abrín Batule
Jefa de la División de Estudios Profesionales de la
Facultad de Ciencias
P r e s e n t e

Comunicamos a usted que hemos revisado el trabajo de Tesis:

Administración de proyectos para el desarrollo de software de calidad

realizado por Anastacio Núñez Guzmán

con número de cuenta 8929742-2 , pasante de la carrera de Matemáticas

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	M. en C. Ma. Guadalupe Elena Hernández González	
Propietario	M. en C. Gustavo Arturo Márquez Flores	
Propietario	Fis. Juan Jesús Gutiérrez García	
Suplente	Mat. Ana Luisa Solís González Cosío	
Suplente	Act. Jorge Flores Maldo	

Consejo [illegible] de Matemáticas

FACULTAD DE CIENCIAS
CONSEJO DEPARTAMENTAL

A DIOS:

Por haberme dado la vida, por permitirme estar cerca de tí y darme todo lo necesario para poder llegar hasta este momento.

A MIS PADRES:

Por todo su amor, esfuerzo y constante apoyo en todos los momentos de mi vida, especialmente en los más difíciles, - como es éste que hoy logramos juntos.

A MI ESPOSA:

Por todo tu amor, comprensión y apoyo, que sé nunca me faltarán

A MI BEBE, que esta en camino.
Espero que algún día este tra
bajo te sirva como un estímulo
para que tus sueños se convier
tan en realidades, y no impor-
tando cuan difíciles sean tus
metas, yo estaré allí para apo
yarte y aplaudir tus triunfos.

A MIS HERMANOS:
Por su fortaleza, comprensión y cariño aún en los momentos - más difíciles.

AGRADECIMIENTOS

Al director de este trabajo

M. C. Ma. Guadalupe Elena Ibargüengoitia González

A mis sinodales

M. C. Gustavo Arturo Márquez Flores

Fís. Juan Jesús Gutiérrez García

Mat. Ana Luisa Solís González Cosío

Act. Jorge Flores Maldonado

CONTENIDO

INTRODUCCION

La inquietud por desarrollar este trabajo surge de la experiencia profesional que he tenido, participando en la realización de proyectos de desarrollo de software. Generalmente no se cuenta con un manual de procedimientos estandarizados para el desarrollo del software; por otra parte los jefes únicamente les interesa dar resultados lo que implica el poco interés en como se desarrollan los sistemas y mucho menos su calidad. Normalmente pocas veces se cumplen las fechas de terminación de un proyecto de software debido a que no se cuenta con una metodología de administración de proyectos.

El objetivo general de esta tesis es dar a conocer las técnicas de la administración y de la ingeniería del software con el propósito de otorgar asistencia práctica a las personas que se ocupan de los proyectos de desarrollo de software, con el único fin de elaborar un producto de calidad.

En el capítulo I, se da una breve explicación de lo que comprende la ingeniería del software, así como una descripción de cada una de las etapas que forman el ciclo de vida de un proyecto de software incluyendo las actividades que se deben seguir para obtener productos de calidad. Finalmente, se determina la importancia de los estándares en el desarrollo de los productos de software.

En el capítulo II, se habla de uno de los estándares de calidad más populares ISO 9000 y sus componentes. Se explica lo que es esta norma y lo que se requiere para obtener la certificación, así como el estado en que se encuentra el proceso de normalización en México. Por último, se presenta el modelo CMM el cual está orientado al logro de la calidad de las empresas.

En el capítulo III, se habla de como se organiza una prueba del software, las técnicas y estrategias a seguir durante su aplicación; también se presentan algunos tipos de pruebas que pueden ser aplicados a un software. Así mismo se discute el proceso de revisión que debe ser aplicado durante el desarrollo de un producto.

Por último en el capítulo IV, propongo algunas técnicas, herramientas y procedimientos de la administración de proyectos, las cuáles pienso deben ser tomadas en cuenta si se quiere desarrollar software con calidad.

CAPITULO I.

CALIDAD DEL SOFTWARE

1.1. INGENIERIA DEL SOFTWARE

Antes del siglo veinte, la garantía de calidad era responsabilidad única de la persona que construía el producto. La primera función de control de garantía de calidad formal fue introducida por los laboratorios Bell en 1916 y se extendió rápidamente por todo el mundo de las manufacturas. Hoy en día, cada empresa tiene un mecanismo que asegura la calidad de sus productos. De hecho, durante la pasada década se ha usado ampliamente, como táctica de mercado, la declaración explícita de mensajes que ponían de manifiesto la calidad ofrecida por las empresas.

La historia de la garantía de calidad en el desarrollo de software ha sido paralela a la historia de la calidad en la fabricación de hardware. Durante los primeros años de la informática en los 50 y 60, la calidad era responsabilidad únicamente del programador. En la década de los 70 se introdujeron estándares de garantía de calidad para el software en los contratos militares de desarrollo de software y se han extendido rápidamente en los desarrollos del mundo comercial .

La ingeniería de software tiene como objeto de estudio al software, apoyándose en los resultados teóricos y estudios empíricos que tienen un impacto potencial en la construcción, análisis y administración del software. Estos proyectos generalmente requieren un desarrollo por equipos de trabajo y no por individuos.

En particular, la ingeniería de software comprende [9]:

- Los métodos y modelos para el desarrollo y mantenimiento, por ejemplo, técnicas y principios para la especificación, diseño, implementación de sistemas de software incluyendo notaciones y modelado de los procesos. Los métodos de la ingeniería del software indican "cómo" construir técnicamente el software. Los métodos abarcan un amplio aspecto de tareas que incluyen: planificación y estimación de proyectos, análisis de los requisitos del sistema y del software, diseño de estructuras de datos, arquitecturas de programas y procedimientos algorítmicos, codificación, prueba y mantenimiento. Los métodos de la ingeniería del software introduce frecuentemente una notación gráfica orientada a un lenguaje y un conjunto de criterios para la calidad del software.

- Las herramientas y ambientes, por ejemplo, herramientas específicas, ambientes integrados incluyendo sus arquitecturas, bases de datos y características de procesamiento paralelo y distribuido. Las herramientas de la ingeniería del software suministran un soporte automático o semiautomático para los métodos. Hoy existen herramientas para soportar cada uno de los métodos mencionados anteriormente. Cuando se integran las herramientas de forma que la información creada por una herramienta pueda ser usada por otra,

se establece un sistema para el soporte del desarrollo del software, llamado "ingeniería del software asistida por computadora" (CASE). La ingeniería del software asistida por computadora combina software, hardware y bases de datos sobre la ingeniería del software (una estructura de datos que contenga la información relevante sobre el análisis, diseño, codificación y prueba).

- Los métodos de evaluación, por ejemplo, pruebas de software y validación, modelos de confiabilidad, procedimientos de pruebas y diagnósticos, redundancia en el software y diseño para el control de errores y mediciones y evaluaciones de varios aspectos de los procesos. Los procedimientos de la ingeniería del software son el pegamento que junta los métodos y las herramientas y facilitan un desarrollo racional y oportuno del software de computadora. Los procedimientos definen la secuencia en la que se aplican los métodos, las entregas (documentos, informes, formas, etc.) que se requieren, los controles que ayudan a asegurar la calidad y coordinar los cambios de directrices que ayudan a los gestores del software a evaluar el progreso.

- La administración de los proyectos de software, por ejemplo, modelos de costo de los factores de productividad, problemas de calendarización y organización, selección de estándares.

- El software como parte de un sistema, balance entre implementación hardware-software.

1.2. EL CICLO DE VIDA CLASICO DEL SOFTWARE

Al igual que los otros sistemas a gran escala, los grandes sistemas de software requieren un tiempo considerable para su desarrollo y permanecen en uso durante un tiempo aún mayor. En este período de desarrollo y uso pueden identificarse varias etapas, que juntas construyen lo que se llama el ciclo de vida del software.

Es probable que el modelo inicial del ciclo de vida del software lo hubiera propuesto por primera vez Royce [11] en 1970, y desde entonces el modelo ha tenido muchos perfeccionamiento y variaciones. Todos ellos pueden incluirse en un "marco" modelo del ciclo de vida clásico el cual es ilustrado en la figura 1.1. El paradigma del ciclo de vida exige un enfoque sistemático y secuencial del desarrollo del software que comienza en el nivel del sistema y progresa a través del análisis, diseño, codificación, prueba y mantenimiento.

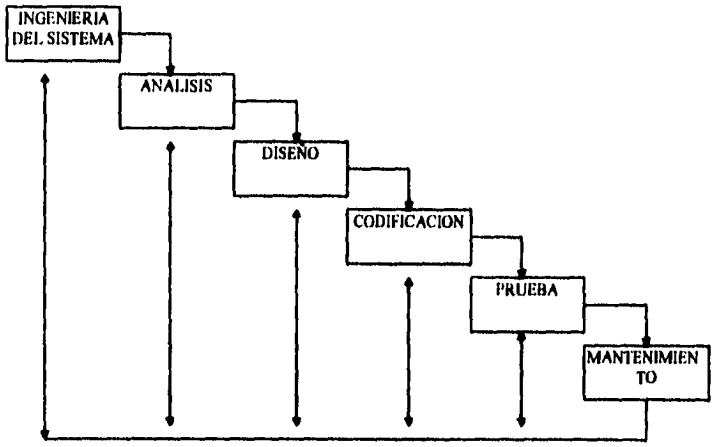


Figura 1.1. El ciclo de vida clásico

- Ingeniería del sistema. Debido a que el software es siempre parte de un sistema mayor, el trabajo comienza estableciendo los requisitos de todos los elementos del sistema y luego asignado algún subconjunto de estos requisitos al software. Este planteamiento del sistema es esencial cuando el software debe interrelacionarse con otros elementos, tales como personas, hardware, y bases de datos.

- Análisis de los requisitos del software. El proceso de recopilación de los requisitos se centra e intensifica especialmente para el 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 y las interfaces requeridas. Los requisitos tanto del sistema como del software, se documentan y se revisan con el cliente.

- Diseño. El diseño del software es realmente un proceso multipaso que se enfoca sobre cuatro atributos distintos del programa: la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. El proceso de diseño traduce los requisitos en una representación del software que puede ser establecida de forma que obtenga la calidad requerida antes de que comience la codificación. Al igual que los requisitos, el diseño se documenta y forma parte de la configuración del software.

- Codificación. El diseño debe traducirse en una forma legible para la máquina. En el paso de codificación se realiza esta tarea. 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 probadas, 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 ocurrirán debido a que se hayan encontrado errores, que el software deba adaptarse a cambios del entorno externo (Por ejemplo, un cambio solicitado debido a que se tiene un nuevo sistema operativo o dispositivo periférico), o debido a que el cliente requiera amplificaciones funcionales o del rendimiento.

El ciclo de vida clásico es el paradigma más antiguo y más ampliamente usado en la ingeniería del software. Sin embargo, con el paso del tiempo, se han producido críticas al paradigma, incluso por seguidores activos, que cuestionan su aplicabilidad a toda las situaciones. Entre los problemas que se presentan algunas veces, cuando se aplica el paradigma del ciclo de vida se encuentran:

- Los proyectos reales raramente siguen el flujo secuencial que propone el modelo. Siempre hay iteraciones y se crean problemas en la aplicación del paradigma.

- Normalmente, es difícil para el cliente al principio establecer explícitamente todos los requisitos. El ciclo de vida clásico lo requiere y tiene dificultades en acomodar posibles incertidumbres que pueden existir al comienzo de muchos proyectos.

- 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 importante no detectado hasta que el programa esté funcionando puede ser desastroso.

Cada uno de estos problemas es real. Sin embargo, el paradigma clásico del ciclo de vida tiene un lugar definido e importante dentro del trabajo realizado en ingeniería del software. Suministra una plantilla en la que pueden colocarse los métodos para el análisis, diseño, codificación, prueba y mantenimiento. 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.

1.3. CALIDAD DEL SOFTWARE

La calidad del software se define como:

'Es la Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente [9].

La anterior definición sirve para hacer incapie en tres puntos importantes:

1. Los requisitos del software son la base de las medidas de la calidad. La falta de concordancia con los requisitos es una falta de calidad.

2. Los estándares especificados definen un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería del software. Si no se siguen esos criterios, casi siempre habrá falta de calidad.

3. Existe un conjunto de requisitos implícitos que a menudo no se mencionan (p.ej.: el deseo de un buen mantenimiento). Si el software se ajusta a sus requisitos explícitos pero falla en alcanzar los requisitos implícitos, la calidad del software queda en entredicho.

La calidad del software es una compleja mezcla de ciertos factores que varían para las diferentes aplicaciones y los clientes que las solicitan.

1.3.1. FACTORES QUE DETERMINAN LA CALIDAD DEL SOFTWARE

Los factores que afectan a la calidad del software se pueden clasificar en dos grandes grupos:

- Factores que pueden ser medidos directamente (p. ej.: errores, unidad de tiempo).

- Factores que sólo pueden ser medidos indirectamente (p.ej.: facilidad de uso o de mantenimiento).

En ambos casos es posible medir. Debemos comparar el software (documentos, programas, etc.) con alguna referencia y llegar a una indicación de la calidad.

McCall [5] ha propuesto una útil clasificación de los factores que afectan a la calidad del software. Estos factores de calidad del software, que aparecen en la figura 1.2, 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.

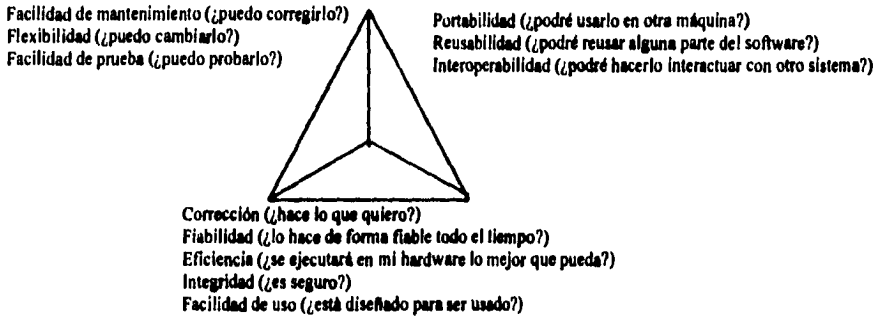


Figura 1.2. Factores de calidad del software de McCall

Características operativas:

- 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 aprender un programa, trabajar con él, preparar su entrada e interpretar su salida.

Capacidad de cambio:

- Facilidad de mantenimiento. El esfuerzo requerido para localizar y arreglar un error en un programa.
- Flexibilidad. El esfuerzo requerido para modificar un programa.
- Facilidad de prueba. El esfuerzo requerido para probar un programa de forma que se asegure que realiza su función requerida.

Adaptabilidad:

- 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 pueden 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.

Es difícil, y en algunos casos imposible, desarrollar medidas directas de los factores de calidad anteriores. Por tanto, se define un conjunto de métricas usadas para desarrollar expresiones para cada uno de los factores de acuerdo con la siguiente relación:

$$Fc = c1 * m1 + c2 * m2 + ... + cn * mn = \sum_{i=1}^{n} ci * mi$$

Donde *Fc* es un factor de calidad del software, *ci* son coeficientes de regresión y *mi* son las métricas que afectan al factor de calidad. Desgraciadamente, muchas de las métricas definidas por McCall sólo pueden ser medidas de forma subjetiva. Las métricas pueden estar en forma de lista de comprobaciones, usadas para "obtener el grado" de los atributos específicos del software. El esquema de graduación propuesto por McCall va en una escala de 0 (bajo) a 10 (alto). En el esquema de graduación se usan las siguientes métricas:

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

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

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

- Completes. El grado en 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. Uniformidad en el diseño y documentación a lo largo del proyecto.

- Estandarización en los datos. El uso de estructuras de datos y de tipos estándares 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 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 operación de un programa.

- Seguridad. La disponibilidad de 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 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 tabla 1.1. Debe insistirse en que el peso dado a cada métrica dependerá de los productos particulares, así como de otros aspectos.

Hewlette Packard ha desarrollado un conjunto de factores de calidad del software cuyas siglas son FURPS. Los factores de calidad FURPS se han obtenido libremente de trabajos anteriores y se definen los siguientes atributos para cada uno de los cinco factores principales:

- La funcionalidad. Se obtiene mediante la evaluación del conjunto de características y de posibilidades del programa, la generalidad de las funciones que se entregan y la seguridad de todo el sistema.

- La facilidad de uso. Se calcula considerando los factores humanos, la estética global, la consistencia y la documentación.

- La fiabilidad. Se calcula midiendo la frecuencia de fallos y su importancia, la eficacia de los resultados de salida, el tiempo medio entre fallos, la posibilidad de recuperarse a los fallos y la previsibilidad del programa.

- El rendimiento. Se mide mediante la evaluación de la velocidad de proceso, el tiempo de respuesta, el consumo de recursos, el rendimiento total de procesamiento y la eficiencia.

- La capacidad de soporte. Combina la posibilidad de ampliar el programa (extensibilidad), la adaptabilidad y la utilidad (estos tres atributos representan un término más común -facilidad de mantenimiento-), además de la facilidad de prueba, la compatibilidad, la posibilidad de configuración, la posibilidad de organizar y controlar elementos de la configuración del software, la facilidad con la que se puede instalar un sistema y la facilidad con la que se pueden localizar los problemas.

1.3.2. GARANTIA DE CALIDAD DEL SOFTWARE

La garantía de calidad del software esta relacionada con las actividadess de verificación y validación realizadas en cada una de las etapas del ciclo de desarrollo del software. En algunas organizaciones no se hace una distinción entre estas actividades. Sin embargo, la garantía de calidad, la verificación y la validación son actividades diferentes y se realizan en forma separada.

Simplemente, la garantía de calidad es una función administrativa mientras que la verificación y la validación son funciones técnicas en el proceso de desarrollo del software. Otra importante distinción es que la verificación y la validación

Tabla 1.1. Factores y métricas de calidad

Métrica de calidad del software \ Factor de calidad	Corrección	Fiabilidad	Eficiencia	Integridad	Facilidad de mantenimiento	Flexibilidad	Facilidad de pruebas	Portabilidad	Reusabilidad	Facilidad de interoperación	Facilidad de uso
Facilidad de auditoría				X			X				
Exactitud		X									
Norm. de las comunicaciones										X	
Completes	X										
Complejidad		X				X	X				
Concisión			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				
Ind. del sistema de software								X	X		
Facilidad de traza	X										
Formación											X

detectan fallas, mientras que la garantía de calidad se remite más allá de la falla del software, y trata asuntos como la seguridad, el mantenimiento, la portabilidad, la readaptación, etc.

En una organización la garantía de calidad del software es responsabilidad de muchas personas - Ingenieros de software, gestores de proyectos, clientes, comerciantes y personas que trabajan dentro de un grupo independiente de garantía de calidad del software, quien debe reportar directamente a un líder de proyecto. El grupo de garantía de calidad no debe tener ninguna relación con los desarrolladores por ser el responsable de la garantía de calidad de todos los proyectos en una organización, figura 1.3.

La garantía de calidad del software es un diseño de acciones. El grupo encargado de la garantía de calidad sirve como representante del cliente. Es decir, la gente que lleva a cabo la garantía de calidad del software debe mirar el software desde el punto de vista del cliente y preguntarse lo siguiente:

- ¿Satisface de forma adecuada el software los factores de calidad?
- ¿Se ha realizado el desarrollo del software de acuerdo con estándares preestablecidos?
- ¿Han desempeñado apropiadamente sus papeles las disciplinas técnicas como parte de la actividad de la garantía de calidad del software?

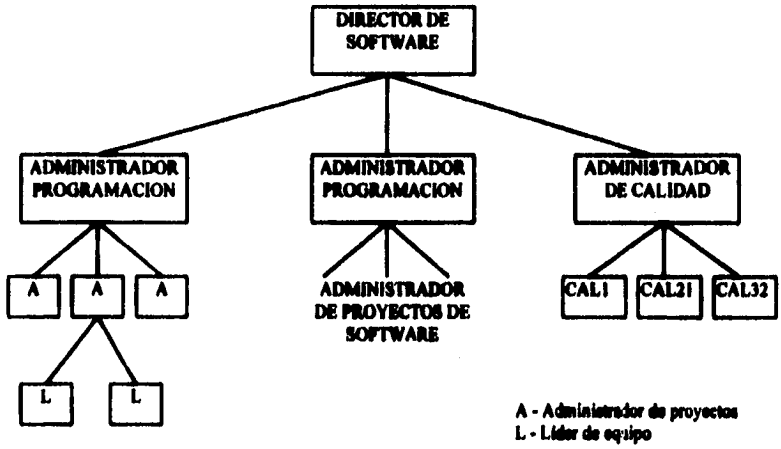


Figura 1.3 Organización de proyectos de software

Una definición de garantía de calidad del software fué dada por Bersoff [1] en 1984, en la cuál estableció que la garantía de calidad consiste en la aplicación de procedimientos, técnicas y herramientas por profesionales para asegurar que un software alcance o exceda los estándares establecidos durante el ciclo de desarrollo; y sin especificar los estándares establecidos, la garantía de calidad asegura que un producto alcance o exceda un nivel de excelencia industrial y/o comercial.

La garantía de calidad del software es una "actividad de protección" que se aplica a lo largo de todo el proceso de ingeniería del software, la garantía de calidad del software engloba:

1. Métodos y herramientas de análisis, diseño, codificación y prueba.
2. Revisiones técnicas formales que se aplican durante cada paso de la ingeniería del software.
3. Una estrategia de prueba multiescalada.
4. El control de la documentación del software y de los cambios realizados.
5. Un procedimiento que asegure un ajuste a los estándares del desarrollo del software.
6. Mecanismo de medida de información.

1.3.3. ACTIVIDADES PARA LOGRAR LA GARANTIA DE CALIDAD DEL SOFTWARE

La garantía de calidad del software comprende una gran variedad de tareas, asociadas con siete actividades principales:

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 garantía de la calidad del software comienza realmente con un conjunto de herramientas y métodos técnicos que ayudan al analista a conseguir una especificación y un diseño de alta calidad.

Una vez que se ha creado una especificación (o prototipo) y un diseño, debe ser garantizada su calidad. La actividad central que permite garantizar la calidad es la revisión técnica formal. La revisión técnica formal (RTF) 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 defectos en el software.

La prueba del software combina una estrategia de múltiples pasos con una serie de métodos de diseño de casos de prueba que ayudan a asegurar una efectiva detección de errores. Muchos grupos de desarrollo de software usan la prueba del software como una red de "seguridad" para la garantía de calidad. Esto es, asumen que mediante la prueba descubrirán la mayoría de los errores, mitigando así la necesidad de otras actividades de la garantía de calidad del software. Desgraciadamente, la prueba incluso cuando se realiza adecuadamente, no es tan efectiva como desearíamos para todas las clases de errores.

El grado de aplicación de procedimiento y estándares en el proceso de 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. Si existen estándares formales (escritos), se debe establecer una actividad de la garantía de calidad del software para garantizar que se sigan. La garantía de seguimiento de estándares puede ser llevada a cabo por los encargados del desarrollo del software como parte de una revisión técnica formal o, en situaciones en que se requiera una verificación del seguimiento independiente, por el grupo de la garantía de calidad del software mediante su propia auditoría.

Cada cambio realizado sobre el software en potencia puede introducir errores o crear efectos laterales que propaguen errores. El proceso de control de cambios contribuye directamente a la calidad del software, al formalizar las peticiones de cambio, evaluar la naturaleza de cambio y controlar el impacto del cambio. El control de cambio se aplica durante el desarrollo del software y, posteriormente durante la fase de mantenimiento del software.

La medición es una actividad integral para cualquier disciplina. Un objetivo importante de la garantía de calidad del software es seguir la pista a la calidad del software y evaluar el impacto de los cambios de metodología y de procedimiento que intenta mejorar la calidad del software. Para conseguir esto, se deben recolectar métricas del software.

El registro de información y la generación de informes para la garantía de calidad del software dan procedimiento para la recolección y divulgación de información de garantía de calidad del software, los cuales 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. Por ejemplo, los resultados de cada revisión técnica formal (RTF) de un diseño procedimental se registran y se guardan en una "carpeta" que contenga toda la información técnica y de garantía de calidad del software sobre cada módulo.

1.4. ESTANDARES DEL SOFTWARE

Uno de los más importantes papeles del grupo encargado de la calidad del software es el de desarrollar productos y procesos estandarizados. Un producto estándar es aquel que todos sus componentes han sido probados y un proceso estándar es el que determina como se debe desarrollar un software. Un ejemplo de un producto estándar es un lenguaje de programación estandarizado el cual determina como debe ser usado, y un ejemplo de un proceso estándar es el que determina como se debe realizar y documentar el diseño.

Los estándares son importantes por las siguientes razones:

- Determinan la estrategia para continuar el desarrollo de un proceso evitando caer en errores que anteriormente se habían tenido, lo que es importante para cualquier organización.

- Proporcionan un serie de procedimientos que garantizan la calidad.

- Facilitan la continuidad de los trabajos, cuando un trabajo es empezado por una persona y terminado por otra, lo que implica que el esfuerzo al reiniciar un trabajo se reduzca.

Una vez que una organización decide institucionalizar la garantía de calidad del software, se debe establecer un plan y se deben adquirir o desarrollar unos estándares. El IEEE ha desarrollado un formato estándar para un plan de garantía de calidad del software, que se muestra en el siguiente esquema:

Estándares ANSI/IEEE 730-1984 Y 983-1986 plan de garantía de calidad del software

1.- Propósito del plan
2.- Referencias
3.- Gestión
    A. Organización
    B. Tareas
    C. Responsabilidades
4.- Documentación
    A. Propósito
    B. Documentos de ingeniería del software requeridos
    C. Otros documentos
5.- Estándares, prácticas y convenios
    A. Propósito
    B. Convenios
6.- Revisiones y auditorías

A. Propósito
B. Requisitos de la revisión
B.1. Revisión de los requisitos del software
B.2. Revisiones del diseño
B.3. Revisiones de validación y verificación del software
B.4. Auditoría funcional
B.5. Auditoría física
B.6. Auditorías en proceso
B.7. Revisiones de gestión
7.- Gestión de configuración del software
8.- Información sobre problemas y acciones correctivas
9.- Herramientas, técnicas y metodologías
10.- Control del código
11.- Control de los medios
12.- Control del distribuidor
13.- Agrupación, mantenimiento y retención de registros

El plan de garantía de calidad del software proporciona un mapa para institucionalizar la garantía de calidad del software. La siguiente tabla presenta una lista de estándares relativos a la garantía de calidad del software que pueden servir como guía para el desarrollo de procedimientos técnicos para alcanzar la calidad del software.

Estándares de garantía de calidad del software

DOD-STD-2167A	Ingeniería del software
DOD-STD-2168	Estándar de evaluación de calidad del software
FAA-STD-018	Estándar de garantía de calidad del software para FAA
IEEE-std. 730-1984	Planes de garantía de calidad del software
IEEE-std. 983-1986	Planificación de la garantía de calidad del software
IEEE-std. 1028-1988	Revisiones y auditorías del software
IEEE-std. 1012-1986	Planes de validación y verificación del software

Los ingenieros en software tienen un cierto rechazo a los estándares, los ven como un proceso burocrático e irrelevante para el desarrollo de software. Están de acuerdo en que los estándares tienen en general un gran valor, pero a menudo encuentran una razón para determinar que no son apropiados para algunos proyectos en particular.

Desafortunadamente algunos estándares son tediosos en su seguimiento y en su verificación, debido a que son escritos por grupos independientes a los que desarrollan software y no están enterados de prácticas modernas.

Para evitar estos problemas, el grupo encargado de la calidad del software debe contar con los suficientes recursos y realizar los siguientes pasos:

- Involucrar a los ingenieros en software en el desarrollo de productos estandarizados. Ellos deben saber el motivo por el cual se desarrollan los estándares y formar parte de una comisión.

- Revisar y modificar los estándares constantemente en base a los proyectos para que reflejen los cambios tecnológicos. Estar al pendiente de que cuando se desarrolle un procedimiento estándar éste quede establecido en un manual de estándares y una vez incluido en el manual, difícilmente deben ser cambiados. Un manual de estándares debe ser un documento dinámico y no estático.

Los procesos estandarizados causan dificultad por el hecho de que son implantados partiendo de la idea de que sólo es un proceso, y nosotros nos limitamos a este. Hay diferentes enfoques en la producción de software pero cada proyecto es desarrollado de acuerdo a un único proceso. Además, muchos de esos procesos son implícitos y mal entendidos y no se sabe como expresarlo.

Otros procesos estandarizados son interpretados favorablemente por los líderes de proyectos. No hay un punto que indique si la forma de desarrollar un proyecto es la adecuada. Cada líder de proyecto tiene la autoridad de modificar los procesos estandarizados de acuerdo a las circunstancias.

CAPITULO II.

ESTANDARES DE CALIDAD

2.1. EVALUACION DEL SOFTWARE

En el proceso de desarrollo de software se hace necesaria una serie de evaluaciones orientadas tanto a la estimación de esfuerzos y recursos necesarios para el desarrollo de un producto, como a la medición de resultados.

Estas evaluaciones se requieren en todas las etapas de desarrollo de software y son de gran importancia para la administración del proyecto. Sólo cuando se monitorean indicadores del funcionamiento del proceso de construcción de software es posible iniciar una gestión hacia la calidad.

Los diferentes colegios de profesionales que existen en el área técnica, arquitectos o ingenieros, han tratado de establecer estándares y medidas de calidad. Los principales motivos para ello son: brindar un mejor desempeño del producto vendido, poder sistematizar la producción en tiempo y costo así como saber cuánto cobrar por un producto.

En el caso de la computación, no es fácil definir estándares ni métricas de calidad, pero se han realizado varios intentos. Al respecto, se tienen dos enfoques: uno busca asegurar la calidad a través de todo el desarrollo del producto y el otro se orienta a evaluar el producto en sí, casi todos los esfuerzos se han concentrado en el primer enfoque, en contraste con otras ramas de la ingeniería. De hecho, cuando se habla de calidad en el software casi siempre es a través de la opinión de los mismos que lo producen, sin referencia a los usuarios finales y sus opiniones y sin estándares relativos a las características de los productos finales.

En este capítulo se verán algunas normas de calidad en el proceso de desarrollo de software.

2.2. CALIDAD DURANTE EL PROCESO

Hace poco más de una década, las empresas involucradas en el desarrollo de software preocupadas por el hecho de que sus productos siempre se terminaban en más tiempo y con un costo mayor del presupuestado, y que incluso la realización de un producto similar a otro construido con anterioridad presentaba grandes diferencias en esfuerzo, costo y tiempo comenzaron a buscar soluciones. Al tratar de mejorar la calidad de sus productos a la vez dejar satisfechos a los administradores en lo relativo a tiempo y recursos gastados, notaron que carecían de elementos objetivos para estimar la calidad y con muy pocas métricas confiables para medir los esfuerzos requeridos. En general, tampoco se contaba con información histórica utilizable.

En forma más o menos convergente, diversas empresas grandes comenzaron proyectos internos orientados a establecer estándares y definir métricas adecuadas, a la vez que se recopilaba información aprovechable para estudiar la calidad. En general estos esfuerzos, quizá por su mismo origen dentro de las empresas, se orientaron al proceso de desarrollo mismo, más que a los productos en sí.

Los partidarios de éste enfoque consideran que si todo el proceso se realiza con cuidado, siguiendo estándares, el resultado naturalmente será bueno y además se cumplirán propósitos de ahorro en el costo de desarrollo.

2.3. ESTANDAR ISO 9000

La Organización Internacional para la Estandarización (ISO), conformada por representantes de los cuerpos normalizadores de aproximadamente 100 países, fue establecida con el objeto de lograr lo siguiente:

- Promover la estandarización internacional, de tal manera que se facilitara el intercambio internacional de bienes y servicios así como el desarrollo científico y tecnológico.

- Desarrollar un código mínimo de prácticas de administración, aplicable a todo tipo de empresa, de aseguramiento y administración de calidad.

Esta lista de prácticas debía representar lo que una empresa estaba obligada a hacer como mínimo para poder responder a los requerimientos de un mercado competitivo. Desde este punto de vista, también significaba una base para poder establecer acuerdos sobre las responsabilidades de proveedores y compradores respecto a la calidad de los bienes o servicios intercambiados.

El ISO 9000 consiste de cuatro partes 9001, 9002, 9003 y 9004 las cuales serán tratadas en las siguientes secciones.

2.3.1. ISO 9001

El estándar ISO 9001, abarca desde el diseño del producto o servicio hasta su entrega y soporte al cliente, junto con otros complementos, busca garantizar la calidad asegurando que la empresa se comprometa como un todo y de manera que ese compromiso se concrete en cada uno de los pasos necesarios para la elaboración del software que se produzca.

En forma más o menos convergente, diversas empresas grandes comenzaron proyectos internos orientados a establecer estándares y definir métricas adecuadas, a la vez que se recopilaba información aprovechable para estudiar la calidad. En general estos esfuerzos, quizá por su mismo origen dentro de las empresas, se orientaron al proceso de desarrollo mismo, más que a los productos en sí.

Los partidarios de éste enfoque consideran que si todo el proceso se realiza con cuidado, siguiendo estándares, el resultado naturalmente será bueno y además se cumplirán propósitos de ahorro en el costo de desarrollo.

2.3. ESTANDAR ISO 9000

La Organización Internacional para la Estandarización (ISO), conformada por representantes de los cuerpos normalizadores de aproximadamente 100 países, fue establecida con el objeto de lograr lo siguiente:

- Promover la estandarización internacional, de tal manera que se facilitara el intercambio internacional de bienes y servicios así como el desarrollo científico y tecnológico.

- Desarrollar un código mínimo de prácticas de administración, aplicable a todo tipo de empresa, de aseguramiento y administración de calidad.

Esta lista de prácticas debía representar lo que una empresa estaba obligada a hacer como mínimo para poder responder a los requerimientos de un mercado competitivo. Desde este punto de vista, también significaba una base para poder establecer acuerdos sobre las responsabilidades de proveedores y compradores respecto a la calidad de los bienes o servicios intercambiados.

El ISO 9000 consiste de cuatro partes 9001, 9002, 9003 y 9004 las cuales serán tratadas en las siguientes secciones.

2.3.1. ISO 9001

El estándar ISO 9001, abarca desde el diseño del producto o servicio hasta su entrega y soporte al cliente, junto con otros complementos, busca garantizar la calidad asegurando que la empresa se comprometa como un todo y de manera que ese compromiso se concrete en cada uno de los pasos necesarios para la elaboración del software que se produzca.

Algunos puntos relevantes del estándar son:

- Requiere documentación escrita de todos los procesos.
- Debe haber un compromiso escrito acerca de la calidad y de cómo lograrla.
- El compromiso debe comenzar por la administración.
- Debe asegurarse que el personal y los recursos disponibles sean suficientes.
- Acerca del control de calidad, debe asegurarse que exista un plan.
- Si se subcontrata a otra empresa, debe haber un control que asegure que ésta reúna los requisitos de calidad, compromiso y documentación. Además debe contar con personal y recursos suficientes.
- Debe haber un control del diseño del producto.
- Debe haber un control de la documentación.
- Debe haber un control de los procesos.

2.3.2. ISO 9002 Y 9003

Estos estándares abarcan únicamente las actividades de producción, instalación, inspección y prueba del producto antes de entregarse al cliente.

Algunos puntos relevantes de estos estándares son:

- Debe haber un control de los productos fuera de especificación.
- Debe haber un proceso de inspección y prueba durante el desarrollo de un producto.
- En cada empresa debe existir un equipo independiente de inspección, medición y pruebas.
- Debe haber un control del estátus del producto en función a inspección y pruebas.

2.3.3. ISO 9004

En esta norma se listan los elementos de un sistema de administración de calidad, con la intención de que las empresas lo apliquen internamente y de manera voluntaria para establecer o fortalecer sus propios sistemas de calidad.

En esta norma se define que se debe implementar un sistema de calidad, basado en la filosofía de aseguramiento de calidad y documentado en un manual de calidad. Este sistema debe incluir todas las políticas, procesos y procedimientos necesarios para asegurar la calidad. Así mismo también indica que para garantizar la calidad de los productos y servicios se debe estructurar un plan de calidad que incluya todas las inspecciones, pruebas y verificaciones necesarias a lo largo de su ciclo de vida, es decir desde su concepción y diseño hasta su instalación y servicio, así como su proceso de producción.

Algunos puntos relevantes del estándar son:

- Debe haber un proceso de revisión de contratos.
- Debe haber un control de adquisiciones.
- Debe haber un control de productos suministrados por los clientes.
- Debe haber acciones preventivas y correctivas al sistema de calidad.
- Debe haber un registro de la calidad de los productos.
- Debe haber un proceso de auditorías internas de calidad.
- Debe haber programas de capacitación.

2.4. NORMAS ESPECIFICAS PARA EL SOFTWARE

La aplicación de las normas ISO 9000 al desarrollo y mantenimiento de software está soportada por la ISO 9000-3, y por el esquema británico de certificación TickIt.

La ISO 9000-3 es una guía que describe los elementos de un sistema de calidad orientado a software. Se incluye algunos temas que no se encuentran en las normas ISO 9000 genéricas, tales como administración de la configuración o planeación de proyectos. Sería poco probable lograr resultados de calidad en un proyecto de desarrollo de software de tamaño mediano, sin haber tomado las provisiones necesarias para el control de configuración. Esto implica que para

ciertos productos o servicios, la especificación de requerimientos contenida en las normas genéricas ISO 9000 no es suficiente para asegurar la calidad, y esto justifica la necesidad de otras normas o guías más específicas.

El esquema TickIt es una iniciativa del gobierno británico que consiste en un esquema de certificación basado en la norma ISO 9000-3, que pone especial énfasis en la experiencia y el entrenamiento que deben tener los auditores calificados, tanto en el área de desarrollo de software como en la evaluación de los criterios específicos de la norma.

2.5. EL PROCESO DE CERTIFICACION

A continuación se describen las fases del proceso recomendado para alcanzar la certificación en ISO 9000:

1- Objetivos estratégicos. El camino para lograr la certificación en ISO 9000 empieza por establecer los objetivos estratégicos de la empresa:

¿ Para qué se requiere la certificación ?
¿ Qué se busca con la certificación ISO 9000 ?
¿ Es necesaria la certificación en ISO 9001 o sólo con ISO 9003 es suficiente ?

La respuesta a estas preguntas depende de las circunstancias específicas de cada empresa.

2- Evaluación inicial. Una vez aclarados los objetivos, se procede a realizar un análisis del estado actual del sistema de calidad de la empresa. Se requiere determinar en que grado existe, si está documentado y si las previsiones de ese sistema son realmente efectivas para prevenir y asegurar la calidad de los productos o servicios.

3- Plan de trabajo. Con los resultados de la fase anterior, se puede hacer un plan para desarrollar o completar el sistema de calidad de la empresa. Normalmente el sistema de calidad debe desarrollarse de arriba hacia abajo, es decir, primero los documentos de mayor nivel como las políticas generales de calidad, y al final los documentos que describen en detalle las operaciones.

4- Auto auditoría. Cuando el sistema de calidad está terminado, se debe realizar una auditoría interna del sistema de acuerdo a lo planeado, completa y rigurosa en todo detalle, para verificar los resultados del proyecto de desarrollo. Se deben priorizar los problemas encontrados y corregirlos antes de pasar a la siguiente fase.

5- Certificación. Una vez verificado y corregido el sistema de calidad, se puede proceder a la certificación. Un punto importante es la selección del organismo certificador que cumpla con el perfil identificado en el inicio, cuando se definieron los objetivos.

6- Seguimiento. El organismo certificador hará auditorías de seguimiento, cada 6 meses ó 1 año, dependiendo del organismo y de otras circunstancias, como cuál norma se escogió para la certificación y qué tipo de productos o servicios fueron certificados.

2.6. LA CERTIFICACION EN MEXICO

En México, con un rezago aún de por lo menos 50 años en materia de normalización y certificación, sólo alrededor de 300 empresas ya cuentan con certificados de calidad y de producto, de las cuales por lo menos 50 por ciento son maquiladoras. Esto se atribuye a varios problemas, en donde predomina la falta de cultura del empresario mexicano en materia de normalización y certificación. No reconocen a las normas como una herramienta de competitividad y por lo tanto no las ven como negocio y no invierten en ellas, no están acostumbrados a documentar sus operaciones.

Se avanza lentamente en materia de normalización con el riesgo de que las compañías mexicanas tengan que depender de normas extranjeras para competir en el exterior y en su propio mercado. De esta forma, nuestro país con sus nueve organismos de normalización y certificación, se encuentra en lucha contra el tiempo ante las negociaciones de reconocimiento mutuo que tendrá que enfrentar con otros países.

A nivel general, México ya cuenta con alrededor de mil normas diferentes, contra más de 15 mil en Estados Unidos y Canadá y más de 17 mil en Francia. Esto significa tener abierta una gran compuerta que facilitaría la entrada de empresas y productos extranjeros al país y una llave que a cuentagotas permitiría la salida de negocios y productos mexicanos hacia el exterior.

Es importante hacer un mayor esfuerzo para contar con sistemas de normalización y certificación sólidos para poder competir con otros países y que las compañías mexicanas no tengan que jugar con las reglas, los tiempos y costos que impongan organismos de certificación foráneos.

Empresas certificadas por organismos mexicanos

Xerox Mexicana	Mabe
Sappel	Vitro
General de Cables	Black & Decker
Schneider Electric	Phillips
Ameridata Global	Condumex
Canon Mexicana	Lusa
Pemex	Roger
Pirax Sarco	Vicino
Herco	Le Grand

No implica que no cuenten con ISO-9000. No están incluidas todas las empresas con certificación.

Empresas con certificación ISO-9000

AT&T	Vitrocrisa
Alcatel-Indetel	TMM
Telecom	Schneider
Celanese	Condumex
Girsa	Hewlet Packard
Rohm and Hass	IBM
Sicartsa	Cía. Hulera Tornel
Alcomex	Nal. de Conductores
Kodak Mexicana	Electric

No significa que no hayan sido certificadas por organismos nacionales. No están incluidas todas las empresas con certificación.

2.7. MODELO CMM

Hace poco más de una década, la mayoría de los estándares y esfuerzos por comprender el desarrollo del software corrían en dos vertientes con muy poca comunicación entre sí: la académica, que a veces llevaba a resultados elegantes pero inútiles, y la pragmática, a nivel de empresa o desarrollador. A mediados de los 80, se comenzó un esfuerzo para reunir puntos de vista tanto de industria como de academia lo que llevó al modelo CMM (Modelo de Madurez de Capacidades) que es una propuesta del Instituto de Ingeniería de Software de la Universidad de Carnegie-Mellon en el cual participan académicos y representantes de la industria. El modelo también se orienta al logro de la calidad al asegurar la calidad del proceso de desarrollo de software. En forma semejante al ISO 9001, este modelo también involucrar a toda la empresa, no sólo a los desarrolladores del software.

El modelo CMM considera que las empresas pueden estar en uno de cinco estados, en relación a la producción de software:

- Inicial: es un estado inestable, en que la calidad depende de la capacidad de los individuos y en su acierto al elegir métodos de trabajo. Por lo tanto, cualquier cambio de personas o de tipo de trabajo llevan a resultados impredecibles. A veces se le llama "heroico", en honor a los desarrolladores. En este estado, costos y tiempos de entrega usualmente rebasan las metas, y presentan una enorme variabilidad.

- Repetible: En este estado se documentan aciertos y errores en los proyectos, por lo que planeación y administración se basan en proyectos anteriores. Se pueden trazar costos, funcionalidad, planeaciones. Más aún, se establecen y se cumplen los estándares necesarios, por lo que se puede decir que los procesos están disciplinados. En este estado la probabilidad de que costos y tiempos de entrega tengan una distribución más centrada por lo que hace a las metas es mayor, pero aún presenta una variabilidad grande.

- Definido: En este estado ya se tiene información que permite establecer procesos típicos, consistentes, los cuales se documentan. Todo el desarrollo del software se integra en un proceso de software estándar. En este estado se reduce la variabilidad en estimaciones de costo y duración.

- Administrado: Este cuarto estado se caracteriza porque cuenta con una base de datos global. La empresa establece metas cualitativas y cuantitativas para productos y procesos, con medidas consistentes. Se puede medir calidad y productividad. Los procesos, de este modo, resultan predecibles. En este estado se continúa reduciendo la variabilidad de las estimaciones.

- Optimizando: Al llegar a este estado, la empresa entra en un ciclo de mejoras continuas a los procesos que ha establecido. Cabe suponer que para entonces la variabilidad en estimaciones será mínima y que la misma empresa irá fijando sus metas de superación.

Para obtener un reconocimiento oficial del nivel en que se halla una empresa, se debe pasar por un proceso de certificación semejante al del ISO 9000. Como este proceso involucra una auditoría, se requiere realmente el soporte de la empresa como un todo.

2.8. METRICAS DE CALIDAD DEL SOFTWARE

El área de métricas de software se preocupa de la medición y predicción en ingeniería de software. Las medidas de software indican el grado de calidad de procesos y productos de software. Las métricas de software son indispensables para la administración de las tareas de desarrollo de software, y el mejoramiento de procesos y productos.

La investigación en esta área continúa en forma muy rápida, principalmente tratando de validar teorías en la práctica e implantando métodos y herramientas que apoyan la recolección y presentación de resultados. El uso de métricas se comienza a llevar a cabo desde las primeras etapas del ciclo de vida del software hasta el mantenimiento.

Indices de calidad del diseño

Para calcular el índice de calidad de la estructura del diseño (ICED) se tienen que averiguar los siguientes valores:

S1 = Número total de módulos definidos en la arquitectura del programa.
S2 = Número de módulos cuya correcta función depende de la fuente de los datos de entrada o que produce datos que se usan en cualquier parte, en general los módulos de control (entre otros) no se van a considerar como parte de S2.

S3 = Número de módulos cuya correcta función depende del procesamiento previo.

S4 = Número de elementos de una base de datos (incluye los objetos de datos y todos los atributo que definen objetos).

S5 = Número total de elementos de base de datos únicos.

S6 = Número de segmentos de base de datos (registros diferentes u objetos individuales).

S7 = Número de módulos con una sóla entrada y una sóla salida (el procesamiento de excepciones no se considera como una salida múltiple).

Una vez determinados los valores S1 a S7 para un programa de computadora, se pueden calcular los siguientes valores intermedios:

Estructura del programa: D1, que se define de la siguiente forma: Si el diseño arquitectónico se desarrolló usando un método característico (p. ej.: diseño orientado al flujo de datos o diseño orientado a los objetos), entonces D1 = 1; en caso contrario D1 = 0.

Independencia de módulos: D2 = 1 - (S2 / S1).

Módulos no dependientes del procesamiento previo: D3 = 1 - (S3 / S1).

Tamaño de la base de datos: D4 = 1 - (S5 / S4).

Compartimiento de la base de datos: D5 = 1 - (S6 / S4).

Característica de entrada/salida del módulo: D6 = 1 - (S7 / S1).

Habiendo determinado esos valores intermedios, el ICED se calcula de la siguiente manera:

$$ICED = \Sigma\, Pi\, Di$$

donde i varía de 1 a 6, Pi es el peso relativo de la importancia de cada uno de los valores intermedios y la $\Sigma\, Pi = 1$ (si todos los Di tienen el mismo peso, entonces $Pi = 0$)

El estándar del IEEE sugiere un índice de madurez del software (IMS), que proporciona una indicación de la estabilidad de un producto de software (basada en los cambios que se producen en cada versión del producto). Se determina la siguiente información:

Mt = Número de módulos en la versión actual
Fm = Número de módulos en la versión actual que han sido modificados
Fa = Número de módulos en la versión actual que han sido añadidos.
Fe = Número de módulos de la versión anterior que se han eliminado en la versión actual

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

$$IMS = \frac{Mt - (Fa + Fm + Fe)}{Mt}$$

A medida que el *IMS* se aproxima a 1, el producto comienza a estabilizarse. El *IMS* también se puede utilizar como métrica para la planificación de actividades del mantenimiento del software. El tiempo medio para producir una versión de un producto de software puede tener correlación con el *IMS* y se pueden desarrollar modelos empíricos para el esfuerzo de mantenimiento.

En la actualidad el índice de calidad para las otras etapas del ciclo de vida del software no están suficientemente documentos, por lo que no son tratados en este trabajo.

CAPITULO III.

ESTRATEGIAS DE PRUEBA

3.1. ESTRATEGIAS DE PRUEBA DEL SOFTWARE

La prueba es un conjunto de actividades que se pueden planificar por adelantado y llevar a cabo sistemáticamente. Por esta razón se debe definir una plantilla para la prueba del software que consiste en un conjunto de pasos en los que podemos situar las técnicas específicas de diseño de casos de prueba y los métodos de prueba.

Se han propuesto varias estrategias de prueba de software en distintos libros, todas proporcionan al ingeniero del software una plantilla para la prueba y todas tienen las siguientes características generales:

- La prueba comienza en el nivel de módulo y trabaja "hacia fuera", hacia la integración de todo el sistema basado en computadora.
- Diferentes técnicas de prueba son apropiadas en diferentes momentos.
- La prueba la lleva a cabo el que desarrolla el software o (para grandes proyectos) un grupo de prueba independiente.
- La prueba y la depuración son actividades diferentes, pero la depuración se puede incluir en cualquier estrategia de prueba.

Una estrategia para la prueba del software debe acomodar pruebas de bajo nivel que verifiquen que cada pequeño segmento de código fuente se ha implementado correctamente, así como pruebas de alto nivel que muestren la validez de las principales funciones del sistema frente a los requisitos del cliente.

Los individuos que forman parte del equipo de desarrollo son humanos y como tales cometen errores en las especificaciones, interpretaciones, conclusiones e implementaciones. Es perdonable que se cometan errores, pero es imperativo darse cuenta de su existencia.

El área de las pruebas de software se ha desarrollado con el fin de proporcionar un nivel de confianza adecuado de que el software es correcto. La meta de las pruebas es detectar errores en los programas. Se detecta un error cuando al ejecutar un programa los resultados obtenidos no corresponden con los resultados especificados. Por ejemplo, un modelo del proceso de prueba puede ser el siguiente:

- Se genera un conjunto de pruebas (datos de entrada del programa y datos de salida esperados). Los datos de entrada pueden obtenerse usando casos límites o simplemente generándolos en forma aleatoria.

- El programa se ejecuta usando el conjunto de pruebas y las salidas se comparan con las salidas esperadas. También se hace una revisión para determinar si el conjunto de pruebas es "adecuado".

- Si al menos hay un caso de prueba que detecta un error, el programa es corregido y se realizan pruebas de regresión; si no se detecta ningún error en las pruebas de regresión pero el conjunto de pruebas es "inadecuado", se incluyen pruebas adicionales en dicho conjunto.

- El proceso continúa hasta que el programa se ha ejecutado usando un conjunto de pruebas "adecuado" que no es capaz de exponer ningún error.

En este punto el programa se entrega. Aunque no está garantizado de ser correcto entre "mejor" sea el criterio que determina si el conjunto de pruebas es "adecuado" más es la confianza de que el programa sea correcto.

Es importante observar que un error se detecta cuando hay una diferencia entre la salida "especifica" y la salida obtenida al ejecutar el programa.

Una estrategia de prueba del software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que dan como resultado una correcta construcción del software.

3.1.1. ORGANIZACION PARA LA PRUEBA DEL SOFTWARE

El que desarrolle el software siempre es responsable de probar las unidades individuales (módulos) del programa, asegurándose de que cada uno lleva a cabo la función para la que fue diseñada. En muchos casos también se encargará de la prueba de integración. Sólo una vez que la arquitectura del software esté completa entra en juego un grupo independiente de prueba.

El papel del grupo independiente de prueba (GIP) es eliminar los problemas inherentes asociados con el hecho de permitir al constructor que pruebe lo que ha construido. Una prueba independiente elimina el conflicto de intereses que, de otro modo, estaría presente.

Sin embargo, el que haya desarrollado el software no cede simplemente el programa al GIP y se desentiende. El desarrollador y el GIP trabajan estrechamente a lo largo del proyecto de software para asegurar que se realizan pruebas exhaustivas. Mientras se dirige la prueba, el desarrollador debe estar disponible para corregir los errores que se van descubriendo.

El GIP es una parte del equipo del proyecto de desarrollo de software en el sentido de que se ve implicado durante el proceso de especificación y sigue implicado a todo lo largo del proyecto. Sin embargo, en muchos casos, el GIP informa a la organización de garantía de calidad del software, consiguiendo de este modo un grado de independencia que no sería posible si fuera una parte de la organización de desarrollo de software.

3.1.2. CRITERIOS PARA LA REALIZACION DE LA PRUEBA

Cada vez que se trata la prueba del software surge una pregunta clásica: una vez que hemos realizado la prueba, ¿cómo saber que hemos probado lo suficiente?. Tristemente, no hay una respuesta definitiva a esta pregunta, pero hay algunas respuestas prácticas y nuevos intentos con bases empírica.

Una respuesta a la pregunta anterior es: la prueba nunca termina, ya que el desarrollador carga o pasa el problema al cliente. Cada vez que el cliente/usuario ejecuta un programa de computadora, dicho programa se está probando con un nuevo conjunto de datos. Este importante hecho subraya la importancia de otras actividades de garantía de calidad del software. Otra respuesta es, se termina la prueba cuando se agota el tiempo o el dinero disponible para tal efecto.

Aunque algunos profesionales se sirvan de estas respuestas como argumento, un ingeniero de software necesita un criterio más riguroso para determinar cuándo se ha realizado la prueba suficiente. Musa y Ackerman [6] sugieren una respuesta basada en un criterio estadístico: "No, no podemos tener la absoluta certeza de que el software nunca fallará, pero en base a un modelo estadístico de corte teórico y validado experimentalmente, hemos realizado las pruebas suficientes para decir, con un 95 por ciento de certeza, que la probabilidad de funcionamiento libre de fallo de 1000 horas de CPU, en un entorno definido de forma probabilística, es al menos 0.995".

Mediante modelización estadística y teoría de fiabilidad del software, se pueden desarrollar modelos de fallos del software (descubiertos durante la prueba) como una función del tiempo de ejecución. Una versión del modelo de fallos, denominado modelo logarítmico de Poisson de tiempo de ejecución, toma la siguiente forma:

$$f(t) = \frac{1}{p} * ln(lo * p * t + 1)$$

donde $f(t)$ = número acumulado de fallos que se espera que se produzcan una vez que se ha probado el software durante una cierta cantidad de tiempo de ejecución t.

lo = la intensidad de fallos inicial del software (fallos por unidad de tiempo) al principio de la prueba.

p = la reducción exponencial de intensidad de fallo a medida que se encuentran los errores y se van haciendo correcciones.

La intensidad de fallo instantánea, $l(t)$, se puede obtener mediante la derivada de $f(t)$:

$$l(t) = \frac{lo}{(lo * p * t + 1)} \qquad (2.1.)$$

Mediante la relación de la ecuación (2.1.), los que realizan la prueba pueden predecir la disminución de errores a medida que avanza la prueba. La intensidad de error real se puede trazar junto a la curva predecida Figura 2.1. Si los datos reales recopilados durante la prueba y el modelo logarítmico de Poisson de tiempo de ejecución están razonablemente cerca unos de otros, sobre un número de puntos de datos, el modelo se puede usar para predecir el tiempo de prueba total requerido para alcanzar una intensidad de fallos aceptablemente baja.

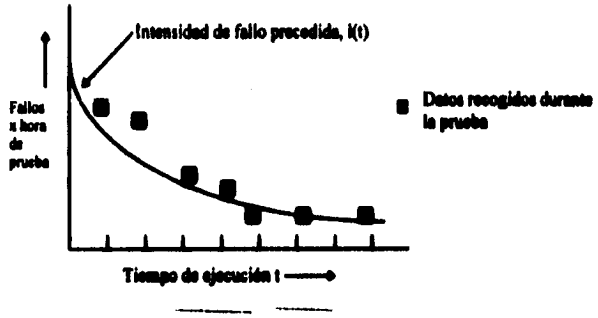


Figura 2.1. Intensidad de fallos como una función de tiempo de ejecución

Mediante la agrupación de métricas durante la prueba del software y haciendo uso de los modelos de fiabilidad del software existentes, es posible desarrollar guías importantes para responder a la pregunta: ¿cuándo terminamos la prueba? Hay pocas dudas sobre que todavía queda mucho por hacer antes de que se puedan establecer reglas cuantitativas para la prueba, pero los enfoques empíricos que existen actualmente son considerablemente mejores que la pura intuición.

3.2. TECNICAS DE PRUEBA DEL SOFTWARE

Fundamentos de la prueba del software

La prueba presenta una interesante anomalía para el ingeniero de software. Durante las fases anteriores de definición y de desarrollo, el ingeniero intenta construir el software partiendo de un concepto abstracto y llegando a una implementación tangible. A continuación, llega la prueba. El ingeniero crea una serie de casos de prueba que intenta "demoler" el software que ha sido construido. De hecho, la prueba es uno de los pasos de la ingeniería que se puede ver (por lo menos, psicológicamente) como destructivo en lugar de constructivo.

Objetivos de la prueba

Glen Myers [7] establece una serie de reglas que sirven acertadamente como objetivos de prueba:

1. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.
2. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
3. Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Los objetivos anteriores suponen un cambio dramático de punto de vista. Nos quita la idea que, normalmente, tenemos de que una prueba tiene éxito si no descubre errores. El objetivo es diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

Si la prueba se lleva a cabo con éxito (de acuerdo con el objetivo anteriormente establecido), descubrirá errores en el software. La prueba demuestra hasta que punto las funciones del software parecen funcionar de acuerdo con las especificaciones y parecen alcanzarse los requisitos de rendimiento. Además, los datos que se van recogiendo a medida que se lleva a cabo la prueba proporcionan una buena indicación de la fiabilidad del software y, de alguna manera, indican la calidad del software como un todo.

Diseño de casos de prueba

Recordando el objetivo de la prueba, debemos diseñar pruebas que tengan la mayor probabilidad de encontrar el mayor número de errores con la mínima cantidad de esfuerzo y tiempo.

Cualquier producto de ingeniería (y de otros muchos campos) puede ser probado de una de dos formas: (1) conociendo la función específica para la que fue diseñado el producto, se pueden llevar a cabo pruebas que demuestren que cada función es completamente operativa; (2) conociendo el funcionamiento del producto, se pueden desarrollar pruebas que aseguren que "todas las piezas encajan"; o sea, que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada. El primer enfoque de prueba se denomina prueba de la caja negra (Ver tema 3.4.) y la segunda prueba de la caja blanca (Ver tema 3.3.).

La prueba de la caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa (p. ej.: archivos de datos) se mantiene. Una prueba de la caja negra examina algunos aspectos del modelo fundamental del sistema sin tener mucho en cuenta la estructura lógica interna del software.

La prueba de la caja blanca se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se pueden examinar el "estado del programa" en varios puntos para determinar si el estado real coincide con el esperado o afirmado.

La prueba de la caja blanca, sin embargo, no se debe desechar como impracticable. Se puede elegir y ejercitar una serie de caminos lógicos importantes. Se pueden comprobar las estructuras de datos más importantes para ver su validez. Se pueden combinar los atributos de la prueba de la caja blanca así como los de la caja negra, para llegar a un método que valide la interfaz del software y asegure selectivamente que el funcionamiento interno del software es correcto.

3.3. PRUEBA DE LA CAJA BLANCA

La prueba de la caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar los casos de prueba. Mediante los métodos de prueba de la caja blanca, el ingeniero del software puede obtener casos de prueba que (1) garanticen que se ejercitan por lo menos una vez todos los caminos independientes de cada módulo; (2) ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa; (3) ejecuten todos los bucles en sus límites y con sus límites operacionales y (4) ejerciten las estructuras internas de datos para asegurar su validez.

Cualquier producto de ingeniería (y de otros muchos campos) puede ser probado de una de dos formas: (1) conociendo la función específica para la que fue diseñado el producto, se pueden llevar a cabo pruebas que demuestren que cada función es completamente operativa; (2) conociendo el funcionamiento del producto, se pueden desarrollar pruebas que aseguren que "todas las piezas encajan"; o sea, que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada. El primer enfoque de prueba se denomina prueba de la caja negra (Ver tema 3.4.) y la segunda prueba de la caja blanca (Ver tema 3.3.).

La prueba de la caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa (p. ej.: archivos de datos) se mantiene. Una prueba de la caja negra examina algunos aspectos del modelo fundamental del sistema sin tener mucho en cuenta la estructura lógica interna del software.

La prueba de la caja blanca se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se pueden examinar el "estado del programa" en varios puntos para determinar si el estado real coincide con el esperado o afirmado.

La prueba de la caja blanca, sin embargo, no se debe desechar como impracticable. Se puede elegir y ejercitar una serie de caminos lógicos importantes. Se pueden comprobar las estructuras de datos más importantes para ver su validez. Se pueden combinar los atributos de la prueba de la caja blanca así como los de la caja negra, para llegar a un método que valide la interfaz del software y asegure selectivamente que el funcionamiento interno del software es correcto.

3.3. PRUEBA DE LA CAJA BLANCA

La prueba de la caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar los casos de prueba. Mediante los métodos de prueba de la caja blanca, el ingeniero del software puede obtener casos de prueba que (1) garanticen que se ejercitan por lo menos una vez todos los caminos independientes de cada módulo; (2) ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa; (3) ejecuten todos los bucles en sus límites y con sus límites operacionales y (4) ejerciten las estructuras internas de datos para asegurar su validez.

Las razones para llevar a cabo las pruebas de la caja blanca son:

- Los errores lógicos y las suposiciones incorrectas son inversamente proporcional a la probabilidad de que se ejecute un camino del programa. Los errores tienden a reproducirse en nuestro trabajo cuando diseñamos e implementamos funciones, condiciones o controles que se encuentran fuera de lo normal. El procedimiento habitual tiende a hacerse más comprensible (y bien examinado), mientras que el procedimiento de "casos especiales" tiende a caer en el caos.

- A menudo creemos que un camino lógico tiene pocas posibilidades de ejecutarse cuando, de hecho, se puede ejecutar de forma regular. El flujo lógico de un programa a veces no es nada intuitivo, lo que significa que nuestras suposiciones intuitivas sobre el flujo de control y los datos nos pueden llevar a tener errores de diseño que sólo se descubren cuando comienza la prueba del camino.

- Los errores tipográficos son aleatorios. Cuando se traduce un programa a código fuente en un lenguaje de programación, es muy probable que se den algunos errores de escritura. Muchos serán descubiertos por los mecanismos de comprobación de sintaxis, pero otros permanecerán sin detectar hasta que comience la prueba. Es igual de probable que haya un error tipográfico en un oscuro camino lógico que en un camino principal.

3.3.1. PRUEBA DEL CAMINO BASICO

La prueba del camino básico es una técnica de prueba de la caja blanca propuesta inicialmente por Tom MaCabe [4]. El método del camino básico permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Notación de grafo de flujo

Antes de considerar el método del camino básico se debe introducir una sencilla notación para la representación del flujo de control, denominada grafo de flujo (o grafo del programa). El grafo de flujo representa el flujo de control lógico mediante la notación ilustrada en la figura 3.2.

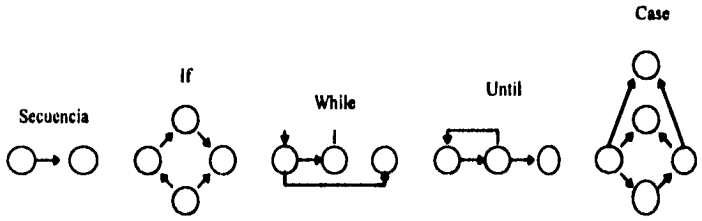


Figura 3.2. Notación de grafo de flujo

Cada construcción estructurada tiene su correspondiente símbolo en el grafo de flujo.

Para ilustrar el uso de un grafo de flujo, consideremos la representación del diseño procedimental de la figura 3.2.a.:

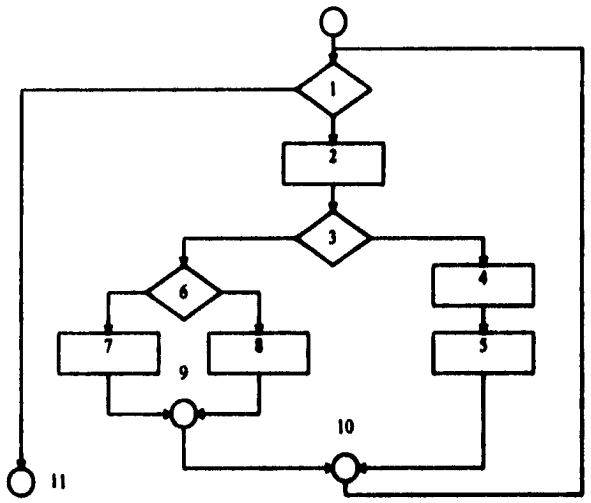


Figura 3.2.a. Diagrama de flujo

En ella, se usa un diagrama de flujo para representar la estructura de control de programa. En la figura 2.4.b., se muestra el grafo de flujo correspondiente al diagrama de flujo anterior.

En la figura, cada círculo, denominado nodo del grafo de flujo, representa una o más sentencias procedimentales. Un sólo nodo puede corresponder a una secuencia de cuadros de proceso y a un rombo de decisión. Las flechas del grafo de flujo, denominadas aristas, representan flujo de control y son análogas a las

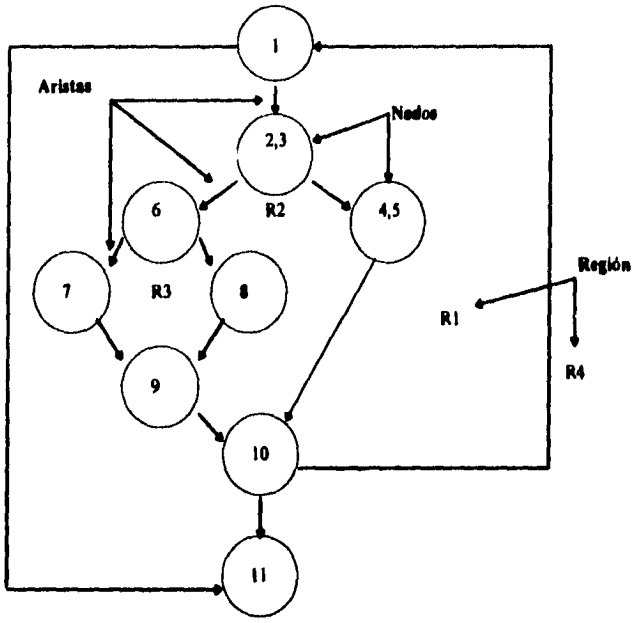
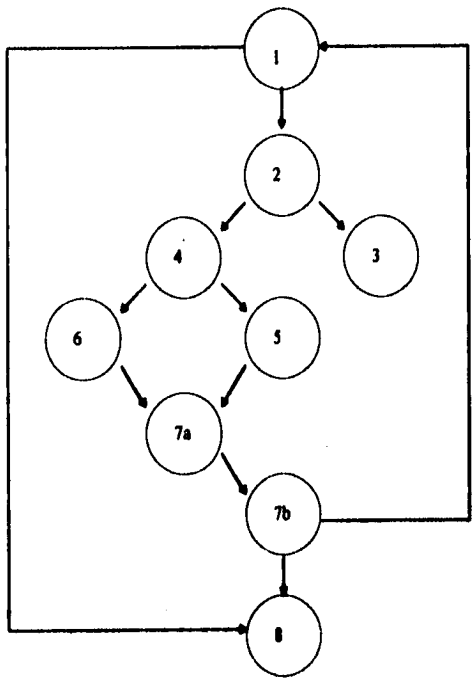


Figura 2.4.b. Grafo de flujo

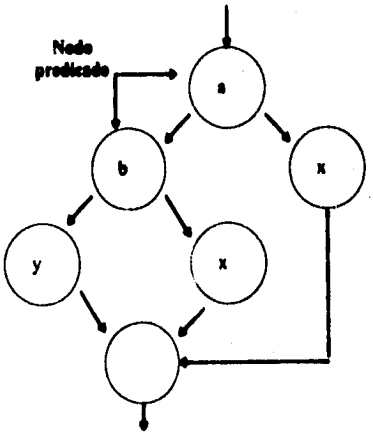
flechas del diagrama de flujo. Una arista debe terminar en un nodo. Las áreas delimitadas por aristas y nodos se denominan regiones. Cuando contabilizamos las regiones incluimos el área exterior del grafo, contando como otra región más. Cualquier representación del diseño procedimental se puede traducir a un grafo de flujo, como en la figura 2.5. En donde se muestra un segmento de lenguaje de diseño de programa y su correspondiente grafo de flujo. Se puede observar que se han numerado las sentencias y que en el grafo de flujo se usa la misma numeración.

Cuando en un diseño procedimental se encuentran condiciones compuestas, la generación del grafo de flujo se hace un poco más complicada. Una condición compuesta se da cuando aparecen uno o más operadores lógicos (OR, AND, NAND, NOR) en una sentencia condicional. En la figura 2.6.




```
Procedimiento: ordenar
1:  do while queden registros
        leer registros
2:      if campo 1 del registro=0
3:          then procesar registro;
               guardar en buffer;
               incrementar contador
4:          elseif campo 2 del registro=0
5:              then reinicializar contador;
6:              else procesar registro;
               guardar en archivo;
7a:         endif
        endif
7b: enddo
8: end
```

Figura 2.5. Traducción de código a grafo de flujo




```
IF a OR b
   then procedimiento x
   else procedimiento y
ENDIF
```

Figura 2.6. Lógica compuesta

el código se traduce en el grafo de flujo anexo. Se crea un nodo aparte por cada una de las condiciones a y b de la sentencia IF a OR b. Cada nodo que contiene una condición se denomina nodo predicado y está caracterizado por que dos o más aristas emergen de él.

Complejidad ciclomática

La complejidad ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa en el contexto del método de prueba del camino básico, el valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez.

Un camino independiente es cualquier camino del programa que introduce por lo menos un nuevo conjunto de sentencias de procesamientos o una nueva condición. En términos del grafo de flujo, un camino independiente se debe mover por lo menos por una arista que no haya sido recorrida anteriormente a la definición del camino. Por ejemplo, para la figura 2.4.b., un conjunto de caminos independientes sería:

camino 1 : 1-11
camino 2 : 1-2-3-4-5-10-1-11
camino 3 : 1-2-6-8-9-10-1-11
camino 4 : 1-2-3-6-7-9-10-1-11

Cada nuevo camino introduce una nueva arista. El camino

1-2-3-4-5-10-1-2-3-6-8-9-10-1-11

no se considera un camino independiente, ya que es simplemente una combinación de caminos ya especificados y no recorre ninguna nueva arista.

Los caminos 1, 2, 3 y 4 definidos anteriormente componen un camino básico para el grafo de flujo de la figura 2.4.b., O sea, si se pueden diseñar pruebas que fuercen la ejecución de esos caminos (un conjunto básico), se garantizará que se habrá ejecutado en sus vertientes verdadera y falsa. Se debe hacer incapié en que el conjunto básico no es único. De hecho, de un mismo diseño procedimental se pueden derivar varios conjuntos básicos diferentes.

¿Cómo sabemos cuantos caminos hemos de buscar? El cálculo de la complejidad ciclomática nos da la respuesta.

La complejidad ciclomática está basada en la teoría de grafos y nos da una métrica del software extremadamente útil. La complejidad se puede calcular de tres formas:

1. El número de regiones del grafo de flujo coincide con la complejidad ciclomática.
2. La complejidad ciclomática, V(G) de un grafo de flujo G se define como: V(G) = A - N + 2, donde A es el número de arista del grafo de flujo y N es el número de nodos del grafo de flujo.
3. La complejidad ciclomática, V(G) de un grafo de flujo G también se define como: V(G) = P + 1, donde P es el número de nodos predicado contenidos en el grafo de flujo G.

Refiriéndonos de nuevo al grafo de flujo de la figura 2.4.b., la complejidad ciclomática se puede calcular mediante cualquiera de los anteriores algoritmos

1. El grafo de flujo tiene cuatro regiones
2. V(G) = 11 aristas - 9 nodos + 2 = 4
3. V(G) = 3 nodos predicado + 1 = 4

Por tanto, la complejidad ciclomática del grafo de flujo de la figura 2.4.b., es 4.

Más importante, el valor de V(G) nos da un valor límite para el número de caminos independientes que componen el conjunto básico y, consecuentemente, un valor límite para el número de pruebas que se deben diseñar y ejecutar para garantizar que se cubren todas las sentencias del programa.

Derivación de casos de prueba

El método de prueba del camino básico se puede aplicar a un diseño procedimental detallado o a un código fuente. Usaremos el procedimiento media, representado a continuación para ilustrar cada paso del método de diseño de casos de prueba.

```
Procedure media;
INTERFACE RETURNS media, total.entrada, total.válido;
INTERFACE ACCEPTS valor, mínimo, máximo;
TYPE valor(1:100) IS SCALAR ARRAY; TYPE j IS INTEGER;
TYPE media, total.entrada, total.válido, mínimo, suma IS SCALAR;
j=1; total.entrada=total.válido=0; suma=0;
DO WHILE valor(j)<>-999 AND total.entrada<100
    incrementar total.entrada en 1;
    IF valor(j)>=mínimo AND valor(j)<=máximo
      THEN incrementar total.válido en 1;
           suma=suma+valor(j)
      ELSE ignorar
    ENDIF
    incrementar j en 1;
ENDDO
IF total.válido>0
  THEN media=suma/total.válido
  ELSE media=-999;
ENDIF
```

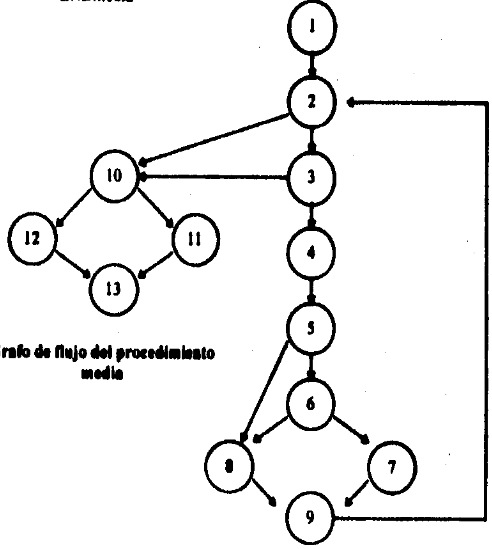
El procedimiento anterior calcula la media de 100 o menos números que se encuentran entre unos límites; también calcula el total de entradas y el total de válidos.

Se puede ver que media aunque con un algoritmo extremadamente simple, contiene condiciones compuestas y bucles.

- Usando el diseño o el código como base, dibujamos el correspondiente grafo de flujo:

```
        Procedure media;
        INTERFACE RETURNS media, total.entrada,total.válido;
        INTERFACE ACCEPTS valor, mínimo, máximo;
        TYPE valor(1:100) IS SCALAR ARRAY;
        TYPE media, total.entrada, total.válido, mínimo, suma IS SCALAR;
        TYPE j IS INTEGER;
   1    j=0; total.entrada=total.válido=0; suma=0;          2

        DO WHILE valor(j)<>-999 AND total.entrada<100       3
               4   incrementar total.entrada en 1;
   5 ------->      IF valor(j)>=mínimo AND valor(j)<=máximo    6
                      THEN incrementar total.válido en 1;
   7                        suma=suma+valor(j)
                      ELSE ignorar
   8               ENDIF
                   incrementar j en 1;
   9    ENDDO
        IF total.válido>0   10
   11      THEN media=suma/total.válido
           ELSE media=-999;   12
   13   ENDIF
        ENDmedia
```



Grafo de flujo del procedimiento media

- Determinamos la complejidad ciclomática del grafo de flujo resultante.
  V(G) = 6 regiones
  V(G) = 17 aristas - 13 nodos + 2 = 6
  V(G) = 5 nodos predicado + 1 = 6

- Determinamos un conjunto básico de caminos linealmente independientes.
  Camino 1: 1-2-10-11-13
  Camino 2: 1-2-10-12-13
  Camino 3: 1-2-3-10-11-13
  Camino 4: 1-2-3-4-5-8-9-2.....
  Camino 5: 1-2-3-4-5-6-8-9-2.....
  Camino 6: 1-2-3-4-5-6-7-8-9-2.....

Los puntos suspensivos (.....) que siguen a los caminos 4, 5 y 6 indican que cualquier camino del resto de la estructura de control es aceptable. Normalmente merece la pena identificar los nodos predicado para que sea más fácil derivar los casos de prueba. En este caso, los nodos 2, 3, 5, 6 y 10 son nodos predicado.

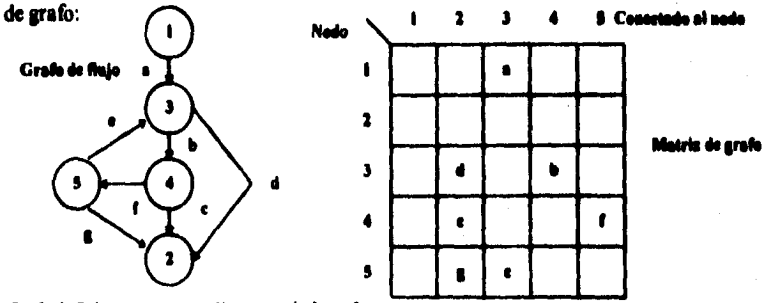
- Preparamos los casos de prueba que forzarán la ejecución de cada camino del conjunto básico.

Debemos escoger los datos de forma que las condiciones de los nodos predicado estén adecuadamente establecidas, con el fin de comprobar cada camino.

Ejecutamos cada caso de prueba y comparamos los resultados con los esperados. Una vez terminados todos los casos de prueba, el probador podrá estar seguro de que todas las sentencias del programa se han ejecutado por lo menos una vez.

Matrices de grafos

Una matriz de grafo es una matriz cuadrada cuyo tamaño (o sea, el número de filas y columnas) es igual al número de nodos del grafo de flujo. Cada fila y cada columna corresponde a un nodo específico y las entradas de la matriz corresponden a las conexiones (aristas) entre los nodos. En la siguiente figura se muestra un sencillo ejemplo de un grafo de flujo con su correspondiente matriz de grafo:



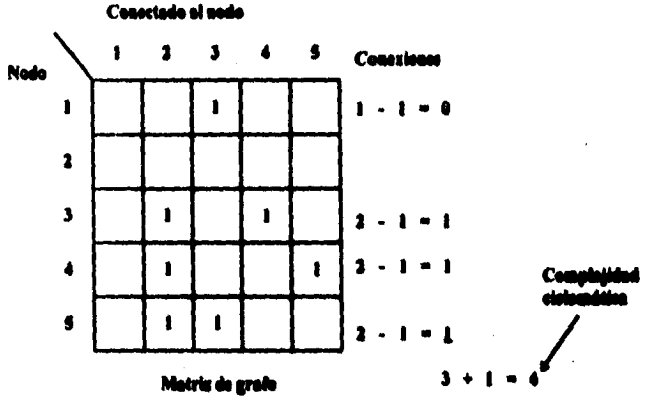
Grafo de flujo y su correspondiente matriz de grafo

Respecto de la figura, cada nodo del grafo de flujo está identificado por un número, mientras que cada arista lo está por su letra. Se sitúa una entrada en la matriz por cada conexión entre dos nodos. Por ejemplo, el nodo 3 está conectado con el nodo 4 por la arista b.

Hasta aquí, la matriz de grafo no es nada más que una representación tabular del grafo de flujo. Sin embargo, añadiendo un peso de enlace a cada entrada de la matriz, la matriz de grafo se puede convertir en una potente herramienta para la evaluación de la estructura de control del programa durante la prueba. El peso de enlace nos da información adicional sobre el flujo de control. En su forma más sencilla, el peso de enlace es 1 (existe una conexión) ó 0 (no existe conexión).

Para ilustrar esto, usaremos la forma más simple de peso, que indica la existencia de conexiones (0 ó 1).

La matriz de grafo de la figura anterior se rehace tal como se muestra en la siguiente figura:



Matriz de conexiones

Se ha reemplazado cada letra por un 1 indicando la existencia de una conexión. Cuando se representa de esta forma, la matriz se denomina matriz de conexiones.

En la figura, cada fila con dos o más entradas representa un nodo predicado. Por tanto, los cálculos aritméticos que se muestran a la derecha de la matriz de conexión nos dan otro nuevo método de determinación de la complejidad ciclomática.

3.4. PRUEBA DE LA CAJA NEGRA

Los métodos de prueba de la caja negra se centran en los requisitos funcionales del software. O sea, la prueba de la caja negra permite al ingeniero del software obtener conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de la caja negra no es una alternativa a las técnicas de prueba de la caja blanca. Más bien se trata de un enfoque complementario que intenta descubrir diferentes tipos de errores que los métodos de la caja blanca.

La prueba de la caja negra intenta encontrar errores de las siguientes categorías: (1) funciones incorrectas o ausentes; (2) errores de interfaz; (3) errores en estructuras de datos o en accesos a bases de datos externas; (4) errores de rendimiento y (5) errores de inicialización y de terminación.

Las pruebas se diseñan para responder a las siguientes preguntas:

- ¿Cómo se prueba la validez funcional?
- ¿Qué clases de entrada compondrán unos buenos casos de prueba?
- ¿Es el sistema particularmente sensible a ciertos valores de entrada?
- ¿De qué forma están aislados los límites de una clase de datos?
- ¿Qué volúmenes y niveles de datos tolerará el sistema?
- ¿Qué efectos sobre la operación del sistema tendrán combinaciones específicas de datos?

Mediante las técnicas de prueba de la caja negra se deriva un conjunto de casos de prueba que satisfacen los siguientes criterios: (1) casos de prueba que reducen, en un coeficiente que es mayor que uno, el número de casos de prueba adicionales que se deben diseñar para alcanzar una prueba razonable y (2) casos de prueba que nos dicen algo sobre la presencia o ausencia de clases de errores en lugar de un error asociado solamente con la prueba, en particular que se encuentre disponible.

3.4.1. PARTICION EQUIVALENTE

La partición equivalente es un método de prueba de la caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores (p. ej.: procesamiento incorrecto de todos los datos de caracteres) que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que hay que desarrollar.

El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencias para una condición de entrada. Una clase de equivalencia para una condición de entrada. Una clase de equivalencia representa un conjunto de estados válidos o inválidos para condiciones de entrada. Típicamente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica. Las clases de equivalencias se pueden definir de acuerdo con las siguientes directrices:

1. Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos inválidas.
2. Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos inválidas.
3. Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una inválida.
4. Si una condición de entrada es lógica, se define una clase válida y una inválida.

Como ejemplo, consideremos los datos contenidos en una aplicación de automatización bancaria. El usuario puede "llamar" al banco usando su microcomputadora, dar su contraseña de seis dígitos y continuar con una serie de órdenes clave que desencadenarán varias funciones bancarias. El software proporcionado por la aplicación bancaria acepta datos en la siguiente forma:

Código de área	-	en blanco o un número de tres dígitos
Prefijo	-	número de tres dígitos que no comience por 0 o 1
Sufijo	-	número de cuatro dígitos
Contraseña	-	valor alfanumérico de seis dígitos
Ordenes	-	"comprobar", "depositar", "pagar factura", etc.

Las condiciones de entrada asociadas con cada elemento de la aplicación bancaria se pueden especificar como:

Código de área:	condición de entrada, lógica "el código de área puede estar o no presente" condición de entrada, rango "valores definidos entre 200 y 999, con excepciones (p. ej.: Sin valores > 905) y requisitos (P. Ej.: todos los códigos de área tienen un 0 o un 1 sin valores > 905) y requisitos (p. ej.: todos los códigos de área tienen un 0 o un 1 como segundo dígito).
Prefijo:	condición de entrada, rango "valor especificado > 200"
Sufijo:	condición de entrada, valor "longitud de cuatro dígitos"

Contraseña:	condición de entrada, lógica "la palabra clave puede estar o no presente"
Orden:	condición de entrada, conjunto "contenida en las órdenes listadas anteriormente"

Aplicando esas directrices para la obtención de clases de equivalencia, se pueden desarrollar y ejecutar casos de pruebas para cada elemento de datos del campo de entrada. Los casos de prueba se seleccionan de forma que se ejercite el mayor número de atributos de cada clase de equivalencia a la vez.

3.5. PRUEBA DE INTEGRACION

Algunos piensan que una vez que todos los módulos han sido probados por separado, estos deben funcionar cuando se junten. Por supuesto que no necesariamente, el problema es "ponerlos juntos" interaccionando. Los datos se pueden perder en una interfaz, un módulo puede tener un efecto adverso e inadvertido sobre otro; las subfunciones, cuando se combinan pueden no producir la función principal deseada; la imprecisión aceptada individualmente puede crecer hasta niveles inaceptables; y las estructuras de datos globales pueden presentar problemas.

A menudo hay una tendencia a intentar una integración no incremental; o sea, a construir el programa mediante un enfoque de "big bang". Se combinan todos los módulos por anticipado. Se prueba todo el programa en conjunto. Normalmente se llega al caos. Se encuentra un gran conjunto de errores. La corrección se hace difícil, puesto que es complicado aislar las causas al tener delante la vasta extensión del programa entero. Una vez que se corrigen esos errores aparecen otros nuevos y el proceso continúa en lo que parece ser un ciclo sin fin.

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. El objetivo es coger los módulos probados en unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.

Integración descendente:

La integración descendente es un planteamiento incremental a la construcción de la estructura de programas. Se integran los módulos moviéndose hacia abajo por la jerarquía de control, comenzando con el módulo de control principal (programa principal). Los módulos subordinados al módulo de control principal se van incorporando en la estructura, bien en la forma primero en profundidad, bien primero de forma primero en anchura.

Refiriéndonos a la figura 2.2., primero en profundidad integra todos los módulos de un camino de control principal de la estructura. La selección del camino principal es, de alguna forma, arbitraria y dependerá de las características específicas de la aplicación. Por ejemplo, si se elige el camino a mano izquierda, se integrarán primero los módulos M1, M2 y M5. A continuación se integrará M8 o M6 (si es necesario para un funcionamiento adecuado de M2). Acto seguido se construyen los caminos de control central y derecho. La integración primero en anchura incorpora todos los módulos directamente subordinados a cada nivel, moviéndose por la estructura de forma horizontal. Según la figura 2.2., los primeros módulos que se integran son M2, M3 y M4 (reemplazando al resguardo S4). Sigue el siguiente nivel de control, M5, M6, etcétera.

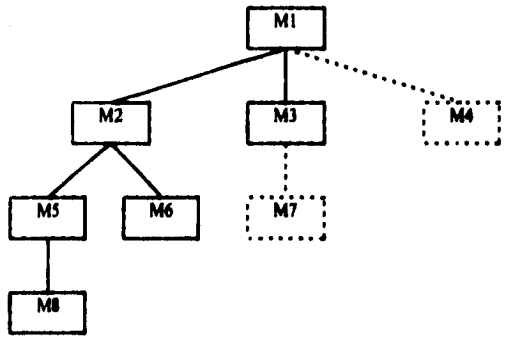


Figura 2.2. Integración descendente

El proceso de integración se lleva a cabo en una serie de cinco pasos:

1. Se usa el módulo de control principal como conductor de la prueba, disponiendo resguardos para todos los módulos directamente subordinados al módulo de control principal.
2. Dependiendo del enfoque de integración elegido (o sea, primero en profundidad o primero en anchura) se van sustituyendo los resguardos subordinados uno a uno por los módulos reales.
3. Se llevan a cabo pruebas cada vez que se integra un nuevo módulo.
4. Tras terminar cada conjunto de pruebas, se reemplaza otro resguardo con el módulo real.
5. Se hace la prueba de regresión (o sea, todas o algunas de las pruebas anteriores) para asegurar que no se han introducido nuevos errores.

El proceso continua desde el paso 2 hasta que se haya construido la estructura del programa entero, la figura 2.2., ilustra el proceso.

Suponiendo que se sigue un planteamiento primero en profundidad y que la estructura está parcialmente construida, el resguardo S7 es el siguiente a ser reemplazado con el módulo M7. El propio M7 podría tener resguardos para ser reemplazados con los correspondientes módulos. Es importante insistir en que en cada reemplazamiento se llevan a cabo prueba para verificar la interfaz.

La estrategia de integración descendente verifica los puntos de decisión o de control principales que aparecen más pronto en el proceso de prueba. En una estructura de programa bien fabricada, la toma de decisiones se da en los niveles superiores de la jerarquía y, por tanto, se encuentran antes.

Si existen problemas más generales de control, es esencial reconocerlos cuanto antes. Si se selecciona la integración primero en profundidad, se puede ir implementando y demostrando las funciones completas del software.

La estrategia descendente suena relativamente fácil, pero, en la practica, pueden surgir algunos problemas logísticos el más común de estos problemas se da cuando se requiere un procesamiento de los niveles más bajos de la jerarquía para poder probar adecuadamente los niveles superiores. Al principio de la prueba descendente, los módulos de bajo nivel se reemplazan por resguardos, por tanto, no pueden fluir datos significativos hacia arriba por la estructura del programa. El encargado de la prueba tiene tres opciones: 1) Retrasar muchas de las pruebas hasta que los resguardos sean reemplazados por los módulos reales, 2) Desarrollar resguardos que realicen funciones limitadas que simulen los módulos reales, 3) Integrar el software desde el fondo de la jerarquía hacia arriba. La figura 2.3., ilustra varias clases de resguardos típicos, desde la más simple (resguardo A) hasta la más compleja (resguardo D).

El primer planteamiento hace que perdamos cierto control sobre la correspondencia de ciertas pruebas específicas con la incorporación de determinados módulos. Esto puede llevar a la dificultad en la determinación de las causas de error y tiende a violar la naturaleza altamente restrictiva del enfoque descendente. El segundo planteamiento es factible pero puede llevar un significativo incremento del esfuerzo a medida que los resguardos se hagan más complejos. El tercer planteamiento denominado prueba ascendente, se discute a continuación.

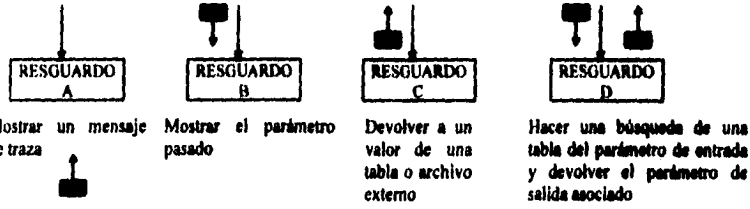


Figura 2.3. Resguardos

Integración ascendente:

La prueba de integración ascendente, empieza la construcción y la prueba con los módulos atómicos (o sea, módulos de los niveles más bajos de la estructura del programa). Dado que los módulos se integran de abajo hacia arriba, el procesamiento requerido de los módulos subordinados siempre está disponible y se elimina la necesidad de resguardo.

Una estrategia de integración ascendente puede ser implementada mediante los siguientes pasos:

1. Se combinan los módulos de bajo nivel en grupo (a veces denominados construcciones) que realicen subfunción específica del software.
2. Se escribe un conductor (un programa de control de la prueba) para coordinar la entrada y la salida de los casos de prueba
3. Se prueba el grupo.
4. Se eliminan los conductores y se combinan los grupos moviéndose hacia arriba por la estructura del programa.

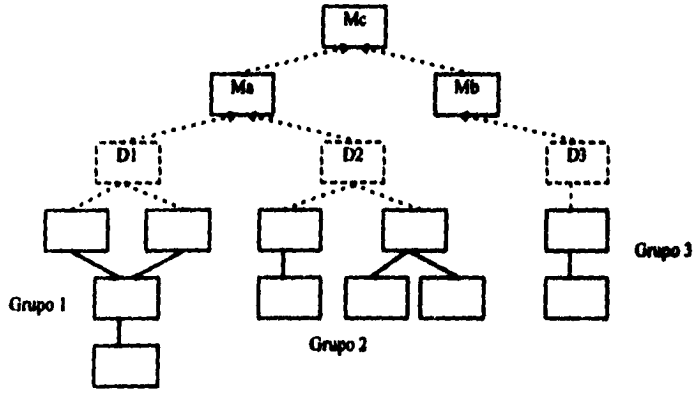


Figura 2.4. Integración ascendente

La integración sigue el esquema ilustrado en la figura 2.4. Se combinan los módulos para formar los grupos 1, 2 y 3. cada uno de los grupos se somete a prueba mediante un conductor (ilustrado como un bloque punteado). Los módulos de los grupos 1 y 2 son subordinados de Ma. Se eliminan los conductores D1 y D2 y se conectan los grupos directamente a Ma. De forma similar, se elimina el conductor D3 del grupo 3 antes de integrarlo con el módulo Mb. Finalmente, tanto Ma como Mb se integran en el módulo Mc y así sucesivamente.

Ha habido muchas discusiones sobre las ventajas y desventajas de la prueba de integración ascendente frente a la descendente. En general, las ventajas de una estrategia tiende a convertirse en una desventaja para la otra estrategia. La principal desventaja del enfoque descendente es la necesidad de resguardos y las dificultades de prueba que pueden estar asociadas con ellos. Los problemas asociados con los resguardos pueden quedar desplazados por la ventaja de poder probar de antemano las principales funciones de control. La principal desventaja de la integración ascendente es que "el programa como entidad" no existe hasta que se ha añadido el último módulo.

La selección de una estrategia de integración depende de las características del software y, a veces, del plan del proyecto. En general, el mejor compromiso puede ser un planteamiento combinado denominado prueba sandwich que use la descendente para los niveles superiores de la estructura del programa junto con la ascendente para los niveles subordinados.

3.5.1. DOCUMENTACION DE LA PRUEBA DE INTEGRACION

Un plan general para la integración del software y una descripción de las pruebas específicas deben quedar documentados en una especificación de prueba. El siguiente esquema presenta el perfil de una especificación de prueba que se puede usar como plantilla de trabajo para dicho documento.

1. Alcance de la prueba
2. Plan de prueba
    - A. Fases de prueba
    - B. Agenda
    - C. Software adicional
    - D. Entorno y recursos
3. Procedimiento de prueba n (descripción de la prueba para la fase n)
    - A. Orden de integración
        1. Propósito
        2. Módulos a ser probados
    - B. Pruebas de unidad para los módulos de la subfase
        1. Descripción de pruebas para el módulo m
        2. Descripción del software adicional
        3. Resultados esperados
    - C. Entorno de prueba
        1. Herramientas técnicas especiales
        2. Descripción del software adicional
    - D. Datos de los casos de prueba
    - E. Resultados esperados para la subfase n
4. Resultados de prueba obtenidos

5. Referencias
6. Apéndices

3.6. PRUEBA DE VALIDACION

Tras la culminación de la prueba de integración, el software está completamente ensamblado como un paquete, se han encontrado y corregido los errores de interfaz y puede comenzar una serie final de pruebas del software "la prueba de validación" . La validación puede definirse de muchas formas, pero una simple indicación es que la validación se logra cuando el software funciona de acuerdo con las expectativas razonables del cliente.

Las expectativas razonables están definidas en la especificación de requisitos del software.

3.6.1. CRITERIOS PARA LA PRUEBA DE VALIDACION

La validación del software se consigue mediante una serie de pruebas de la caja negra que demuestran la conformidad con los requisitos. Una plan de prueba traza las pruebas que se han de llevar a cabo y un procedimiento de prueba define los casos de prueba específicos que se usaran para demostrar la conformidad con los requisitos. Tanto el plan como el procedimiento estarán diseñados para asegurar que se satisfacen todos los requisitos funcionales, que se alcanzan todos los requisitos de rendimiento, que la documentación es correcta e inteligible y que se alcanzan otros requisitos (p. ej.: portabilidad, compatibilidad, recuperación de errores, facilidad de mantenimiento).

Una vez que se procede con cada caso de prueba de validación puede darse una de dos condiciones: 1) Las características de funcionamiento o de rendimiento están de acuerdo con las especificaciones y son aceptadas o 2) Se descubre una desviación de las especificaciones y se crea una lista de deficiencias. Las desviaciones o errores descubiertos en esta fase del proyecto raramente se pueden corregir antes de terminar el plan a menudo es necesario negociar con el cliente un método para resolver las deficiencias.

3.7. PRUEBAS ALFA Y BETA

Es virtualmente imposible que un encargado del desarrollo del software pueda prever como un cliente usará realmente un programa. Se pueden interpretar mal las instrucciones de uso, se pueden usar regularmente extrañas combinaciones de

datos y una salida que puede estar clara para el que realiza la prueba puede resultar ininteligible para un usuario normal.

Cuando se construye software a medida para un cliente, se lleva a cabo una serie de pruebas de aceptación para permitir que el cliente valide todos los requisitos llevado a cabo por el usuario final en lugar del equipo de desarrollo, una prueba de aceptación puede ir desde un informal "paso de prueba" hasta la ejecución sistemática de una serie de pruebas bien planificadas. De hecho la prueba de aceptación puede tener lugar a lo largo de semanas o meses descubriendo así errores acumulados que pueden ir degradando el sistema.

Si el software se desarrolla como un producto que se va a usar por muchos clientes, no es práctico realizar pruebas de aceptación formales para cada uno de ellos.

La mayoría de los constructores de productos de software llevan a cabo un proceso denominado prueba alfa y beta para descubrir errores que parezca que sólo el usuario final puede descubrir.

La prueba alfa es conducida por un cliente en el lugar de desarrollo. Se usa el software de forma natural, con el encargado del desarrollo "mirando por encima del hombro" del usuario y registrando errores y problemas de uso las pruebas alfa se llevan a cabo en un entorno controlado.

La prueba beta se lleva a cabo en uno o más lugares del cliente por los usuarios finales del software. A diferencia de la prueba alfa, el encargado de desarrollo, normalmente, no esta presente. Así, la prueba beta es una aplicación "en vivo" del software en un entorno que no puede ser controlado por el equipo de desarrollo.

El cliente registra todos los problema (reales o imaginarios) que encuentra durante la prueba beta e informa a intervalos regulares al equipo de desarrollo. Como resultado de los problemas anotados durante la prueba beta, el equipo de desarrollo del software lleva a cabo modificaciones y así prepara una versión del producto del software para toda la base de clientes.

3.8. PRUEBA DEL SISTEMA

La prueba del sistema, realmente, está constituida por una serie de prueba diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Aunque cada prueba tiene un propósito distinto, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas.

Un clásico problema de la prueba del sistema es la "delegación de culpabilidad". Esto ocurre cuando se descubre un error y cada uno de los creadores de cada elemento del sistema echa la culpa del problema a los otros. Más que verse envuelto en tal disparate, el ingeniero de software debe anticiparse a los posibles problemas de integración 1) Diseñar caminos de manejo de errores que prueben toda la información procedente de otros elementos del sistema; 2) Llevar acabo una serie de pruebas que simulen la presencia de datos en mal estado o de otros posibles errores en la interfaz del software; 3) Registrar los resultados de las pruebas como "evidencia" en el caso de que se le señale con el dedo; 4) Participar en la planificación y el diseño de pruebas del sistema para asegurarse de que el software es probado de forma adecuada.

3.9. PRUEBA DE COMPARACION

Hay situaciones (p. ej.: control de vuelo, de centrales nucleares) en las que la fiabilidad del software es algo absolutamente crítico. En ese tipo de aplicaciones, a menudo se utiliza hardware y software redundante, para minimizar la posibilidad de error. Cuando se desarrolla software redundante, varios equipos de ingeniería de software separados desarrollan versiones independientes de una aplicación, usando las mismas especificaciones. En esas situaciones, se debe probar cada versión con los mismos datos de prueba, para asegurar que todas proporcionan una salida idéntica. Luego, se ejecutan todas las versiones en paralelo y se hace una comparación en tiempo real de los resultados, para garantizar la consistencia.

Con las lecciones aprendidas de los sistemas redundantes, los investigadores han sugerido que, para las aplicaciones críticas, se deben desarrollar versiones del software independientes, incluso aunque sólo se vaya a distribuir una de las versiones en el sistema basado en computadora final. Esas versiones independientes son la base de una técnica de prueba de la caja negra denominada prueba de comparación o prueba mano-a-mano.

Cuando se han producido múltiples implementaciones de la misma especificación, a cada versión del software se le proporciona como entrada los casos de prueba diseñados mediante alguna otra técnica de la caja negra. Si las salidas producidas por las distintas versiones son idénticas, se asume que todas las implementaciones son correctas. Si la salida es diferente, se investiga cada una de las aplicaciones para determinar el defecto que es responsable de la diferencia en una o más versiones. En la mayoría de los casos, la comparación de las salidas se puede llevar a cabo mediante una herramienta automática.

La prueba de comparación no es infalible. Si el error se encuentra en la especificación a partir de la cual se han desarrollado todas las versiones, lo más

probable es que todas las versiones reflejen ese error. Además, si todas las versiones independientes producen unos resultados idénticos, pero erróneos, la prueba de comparación no detectará el error.

3.10. PRUEBA DE RECUPERACION

La prueba de recuperación es una prueba del sistema que 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 de estado del sistema, de la recuperación de datos y del rearranque. Si la recuperación requiere de la intervención humana, hay que evaluar los tiempos medios de reparación para determinar si están dentro de unos límites aceptables.

Muchos sistemas basados en computadoras deben recuperarse de los fallos y resumir el procesamiento en un tiempo previamente especificado. En algunos casos un sistema debe ser tolerante a fallos, o sea, los fallos de procesamiento o deben hacer que cese el funcionamiento de todo el sistema. En otros casos, se debe corregir una falla del sistema en un determinado periodo de tiempo a riesgo de que se produzca un serio daño económico.

3.11. PRUEBA DE SEGURIDAD

Cualquier sistema basado en computadora que maneje información sensible o lleve a cabo acciones que puedan perjudicar (o beneficiar) a los individuos es objeto de penetraciones impropias o ilegales. La penetración incluye un amplio rango de actividades: "piratas" que intenten penetrar sistemas por deporte, empleados disgustados que intentan penetrar por venganza e individuos deshonesto que intentan penetrar para obtener ganancias personales ilícitas.

La prueba de seguridad intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán.

Durante la prueba de seguridad, el encargado de la prueba desempeña el papel de un individuo que desea penetrar en el sistema, ¡todo vale! debe intentar hacerse con las claves de acceso por cualquier medio externo del oficio, puede atacar al sistema con software a medida, diseñado para romper cualquier defensa que se haya construido, debe bloquear el sistema, negando así el acceso a otras personas; debe producir a propósitos errores del sistema, intentando penetrar durante la recuperación o debe curiosear en los datos públicos intentando encontrar la clave de acceso al sistema.

Con el suficiente tiempo y los suficientes recursos, una buena prueba de seguridad terminará por penetrar en el sistema. El papel del diseñador del sistema es hacer que el costo de penetración sea mayor que el valor de la información obtenida mediante la penetración.

3.12. PRUEBA DE RESISTENCIA

Las pruebas de resistencia están diseñadas para enfrentar a los programas con situaciones anormales en esencia. El sujeto que realiza la prueba de resistencia se pregunta: "¿a que potencia puedo ponerlo a funcionar antes de que falle?".

La prueba de resistencia 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 uno o dos; 2) Incrementar la frecuencia de datos de entrada en un orden de magnitud con el fin de comprobar como responden las funciones de entrada 3) Ejecutar casos de prueba que requieran el máximo de memoria o de otros recursos; 4) Diseñar casos de prueba que puedan dar problemas con el esquema de gestión de memoria virtual; 5) Diseñar casos de prueba que produzcan excesivas búsquedas de datos residentes en disco. Esencialmente, el encargado de la prueba intenta tirar abajo el programa.

3.13. PRUEBA DE RENDIMIENTO

La prueba de rendimiento esta diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado. La prueba de rendimiento se da durante todos los pasos del proceso de prueba. Incluso a nivel de unidad, se debe asegurar el rendimiento de los módulos individuales a medida que se llevan a cabo las pruebas de la caja blanca. Sin embargo, hasta que no están completamente integrados todos los elementos del sistema no se puede asegurar realmente el rendimiento del sistema

3.14. SEGURIDAD DEL SOFTWARE

La seguridad del software es una actividad de garantía de calidad del software que se centra en la identificación y evaluación de los riesgos potenciales que pueden producir un impacto negativo en el software y hacer que falle el sistema completo. Si se pueden identificar pronto los riesgos en el proceso de ingeniería del software, se pueden especificar las características de diseño del software que podrán eliminar o controlar los riesgos potenciales.

Como parte de la seguridad del software, se puede dirigir un proceso de análisis y modelización. Inicialmente, se identifican los riesgos y se clasifican por su importancia y su grado de riesgo.

Cuando se han identificado estos riesgos del sistema se utilizan técnicas de análisis para asignar su gravedad y su probabilidad de ocurrencia. Para que sea efectivo, se tiene que analizar el software en el contexto del sistema completo.

Se pueden usar técnicas de análisis como el análisis del árbol de fallos, la lógica de tiempo real o los modelos de redes de Petri, para predecir la cadena de sucesos que pueden producir los riesgos y la probabilidad de que se de cada uno de los sucesos que componen la cadena.

El análisis del árbol de fallos construye un modelo gráfico de las combinaciones secuenciales y concurrentes de los sucesos que pueden conducir a un suceso o estado del sistema peligroso. Mediante un árbol de fallos bien desarrollado, es posible observar las consecuencias de una secuencia de fallos interrelacionados que ocurren en diferentes componentes del sistema. La lógica de tiempo real (LTR) construye un modelo del sistema mediante la especificación de los sucesos y las acciones correspondientes. El modelo suceso-acción se puede analizar mediante operaciones lógicas para probar las valoraciones de seguridad de los componentes del sistema y su temporización.

Aunque la fiabilidad y la seguridad del software están bastante relacionadas, es importante entender la sutil diferencia que existe entre ellas. La fiabilidad del software utiliza el análisis estadístico para determinar la probabilidad de que pueda ocurrir un fallo del software. Sin embargo, la ocurrencia de un fallo no lleva necesariamente a un riesgo o a un accidente. La seguridad del software examina los modos según los cuales los fallos producen condiciones que pueden llevar a accidentes. Es decir, los fallos no se consideran en vacío, sino que se evalúan en el contexto de un completo sistema basado en computadora.

3.15. REVISIONES DEL SOFTWARE

Las revisiones del software son un "filtro" para el proceso de ingeniería del software. Las revisiones se aplican en varios momentos del desarrollo del saftware y sirven para detectar defectos que puedan así ser eliminados. Las revisiones del software sirven para "purificar" las actividades de ingeniería del software que hemos denominado análisis, diseño y codificación.

Existen muchos tipos diferentes de revisiones que se pueden llevar adelante como parte de la ingeniería del software. Cada una tiene su lugar. Una reunión informal alrededor de la máquina de café es una forma de revisión, si se discuten

Como parte de la seguridad del software, se puede dirigir un proceso de análisis y modelización. Inicialmente, se identifican los riesgos y se clasifican por su importancia y su grado de riesgo.

Cuando se han identificado estos riesgos del sistema se utilizan técnicas de análisis para asignar su gravedad y su probabilidad de ocurrencia. Para que sea efectivo, se tiene que analizar el software en el contexto del sistema completo.

Se pueden usar técnicas de análisis como el análisis del árbol de fallos, la lógica de tiempo real o los modelos de redes de Petri, para predecir la cadena de sucesos que pueden producir los riesgos y la probabilidad de que se de cada uno de los sucesos que componen la cadena.

El análisis del árbol de fallos construye un modelo gráfico de las combinaciones secuenciales y concurrentes de los sucesos que pueden conducir a un suceso o estado del sistema peligroso. Mediante un árbol de fallos bien desarrollado, es posible observar las consecuencias de una secuencia de fallos interrelacionados que ocurren en diferentes componentes del sistema. La lógica de tiempo real (LTR) construye un modelo del sistema mediante la especificación de los sucesos y las acciones correspondientes. El modelo suceso-acción se puede analizar mediante operaciones lógicas para probar las valoraciones de seguridad de los componentes del sistema y su temporización.

Aunque la fiabilidad y la seguridad del software están bastante relacionadas, es importante entender la sutil diferencia que existe entre ellas. La fiabilidad del software utiliza el análisis estadístico para determinar la probabilidad de que pueda ocurrir un fallo del software. Sin embargo, la ocurrencia de un fallo no lleva necesariamente a un riesgo o a un accidente. La seguridad del software examina los modos según los cuales los fallos producen condiciones que pueden llevar a accidentes. Es decir, los fallos no se consideran en vacío, sino que se evalúan en el contexto de un completo sistema basado en computadora.

3.15. REVISIONES DEL SOFTWARE

Las revisiones del software son un "filtro" para el proceso de ingeniería del software. Las revisiones se aplican en varios momentos del desarrollo del saftware y sirven para detectar defectos que puedan así ser eliminados. Las revisiones del software sirven para "purificar" las actividades de ingeniería del software que hemos denominado análisis, diseño y codificación.

Existen muchos tipos diferentes de revisiones que se pueden llevar adelante como parte de la ingeniería del software. Cada una tiene su lugar. Una reunión informal alrrededor de la máquina de café es una forma de revisión, si se discuten

problemas técnicos. Una presentación formal de un diseño de software a una audiencia de clientes, ejecutivos y personal técnico es una forma de revisión.

3.15.1. REVISIONES TECNICAS FORMALES

Una revisión técnica formal (RTF) es una actividad de garantía de calidad del software que es llevada a cabo por lo profesionales de la ingeniería del software. Los objetivos de la RTF son: (1) Descubrir errores en la función, la lógica o la implementación de cualquier representación del software; (2) Verificar que el software bajo revisión alcanza sus requisitos; (3) Garantizar que el software ha sido representado de acuerdo con ciertos estándares predefinidos; (4) Conseguir un software desarrollado de forma uniforme y (5) Hacer que los proyectos sean más manejables. Además, la RTF sirve como campo de entrenamiento permitiendo que los ingenieros más jóvenes puedan observar los diferentes enfoques al análisis, diseño e implementación del software. La RTF también sirve para promover la seguridad y la continuidad, ya que varias personas se familiarizarán con partes del software que, de otro modo, no hubiera visto nunca.

La RTF es realmente una clase de revisión que incluye recorridos, inspecciones, torneos de revisiones y otras tareas de revisión técnica formal del software. Cada RTF se lleva a cabo mediante una reunión y sólo tendrá éxito si es bien planificada, controlada y atendida.

3.15.2. LA REUNION DE REVISION

Independientemente del formato que se elija para la RTF, cualquier reunión de revisión debe acogerse a las siguientes restricciones:

- Deben convocarse a la revisión (normalmente) entre tres y cinco personas.
- Se debe preparar por adelantado, pero sin que requiera más de dos horas de trabajo a cada persona.
- La duración de la reunión de revisión debe ser menor de dos horas.

Con las anteriores limitaciones, debe resultar obvio que cada RTF se centra en una parte específica (y pequeña) del software total. Por ejemplo, en lugar de intentar revisar un diseño completo, se hacen inspecciones para cada módulo o pequeño grupo de módulos. Al limitar el centro de atención de la RTF la probabilidad de descubrir errores es mayor.

El centro de atención de la RTF es un producto un componente de software (p. ej.: una porción de una especificación de requisitos, un módulo de diseño detallado, un listado en código fuente de un módulo). El individuo que ha desarrollado el producto -productor- informa al jefe de proyecto de que el producto está terminado y que se requiere una revisión. El jefe del proyecto contacta con un jefe de revisión, que es el que evalúa la disponibilidad del producto, genera copias del material del producto y las distribuye a dos o tres revisores para que se preparen por adelantado. Cada revisor estará entre una y dos horas revisando el producto, tomando notas y también familiarizándose con el trabajo. De forma concurrente, también el jefe de revisión revisa el producto y establece una agenda para la reunión de revisión que, normalmente, queda convocada para el día siguiente.

La reunión de revisión es llevada a cabo por el jefe de revisión, los revisores y el productor. Uno de los revisores toma el papel de registrador, o sea, de persona que registra (de forma escrita) todos los sucesos importantes que se produzcan durante la revisión. La RTF comienza con una explicación de la agenda y una breve introducción a cargo del productor. Entonces el productor procede con el "recorrido de inspección" del producto (explica el material), mientras que los revisores exponen sus láminas basándose en su preparación previa. Cuando se descubren problemas o errores válidos, el registrador los va anotando.

Al final de la revisión, todos los participantes en la RTF deben decidir si (1) aceptan el producto sin posteriores modificaciones; (2) rechazan el producto debido a los serios errores encontrados (una vez corregidos, ha de hacerse otra revisión) o (3) aceptan el producto provisionalmente (se han encontrado errores menores que deben ser corregidos, pero sin necesidad de otra posterior revisión). Una vez tomada la decisión, todos los participantes terminan firmando, indicando así que han participado en la revisión y que están de acuerdo con las conclusiones del equipo de revisión.

3.15.3. REGISTRO E INFORME DE LA REVISION

Durante la RTF, uno de los revisores (el registrador) registra dinámicamente todas las láminas que vayan surgiendo. Al final de la reunión de revisión, resume todas las láminas y genera una lista de sucesos de revisión. Además, prepara un informe sumario de revisión. Un informe sumario que debe responder a las siguientes tres preguntas:

1.- ¿Qué fue revisado?
2.- ¿Quién lo revisó?
3.- ¿Qué se descubrió y cuáles son las conclusiones?

El informe sumario de revisión toma la forma ilustrada a continuación:

Informe sumario de la revisión técnica

Identificación de la revisión:
Proyecto: Número de revisión:
Fecha: Lugar: Hora:

Identificación del producto:
Material revisado:
Productor:
Breve descripción:

Material revisado: (anótese cada elemento por separado)

Equipo de revisión: (indíquese el jefe y el registrador)

	Nombre	Firma
1.		
2.		
3.		
4.		
5.		

Aprobación del producto:
Aceptado: como está ( ) con modificaciones menores ( )
No aceptado: revisión principal ( ) revisión secundaria ( )
Revisión no terminada: (explicación a continuación)

Material adicional adjuntado:
Lista de sucesos ( ) Materiales de producción anotados ( )
Otros (especifíquese)

En general, esa simple página (con posibles suplementos) se adjunta al registro histórico del proyecto y puede ser enviada al jefe del proyecto y a otras partes interesadas.

La lista de sucesos de revisión sirve para dos propósitos: (1) Identificar áreas problemáticas dentro del producto y (2) Servir como lista de puntos de acción que guíe al productor para hacer las correcciones.

A continuación aparece un ejemplo de una lista de sucesos.

Número de revisión:
Fecha de revisión:

Lista de Sucesos

1. Los prólogos de los módulos x, y no son consistentes con los estándares de diseño. Se debe establecer explícitamente el propósito (la referencia no es aceptable) y se debe especificar la declaración de los elementos de datos.
2. El contador de bucle para la interpolación entre los ejes x, y, z se incrementa demasiado una vez para el control de paso del motor. El equipo de revisión recomienda otra comprobación de las especificaciones de paso del motor y la corrección (como sea preciso) del contador de bucle.
3. El equipo de revisión recomienda una modificación del algoritmo de comparación de posición para mejorar el rendimiento en tiempo de ejecución.

3.15.4. DIRECTRICES PARA LA REVISION

A continuación se muestra un conjunto mínimo de directrices para las revisiones técnicas formales:

1. Revisar el producto, no al productor.

Una RTF involucra gente y egos. Llevada a cabo adecuadamente, La RTF debe llevar a todos los participantes a un sentimiento agradable de estar cumpliendo su deber. Si se lleva a cabo incorrectamente, la RTF puede tomar el aura de inquisición. Se deben señalar los errores adecuadamente; el tono de la reunión debe ser distendido y constructivo; no debe intentarse dificultar o batallar. El jefe de revisión debe moderar la reunión para garantizar que se mantiene un tono y una actitud adecuados y debe inmediatamente cortar cualquier revisión que haya escapado al control.

2. Fijar una agenda y mantenerla.

Un mal de las reuniones de todo tipo es la deriva, la RTF debe seguir un plan de trabajo concreto. El jefe de revisión es el que carga con la responsabilidad de mantener el plan de la reunión y no debe tener miedo de cortar a la gente cuando se empiece a divagar.

3. Limitar el debate y las impugnaciones.

Cuando un revisor ponga de manifiesto una lámina, podrá no haber unanimidad sobre su impacto. En lugar de perder el tiempo debatiendo la cuestión, debe registrarse el hecho y dejar que la discusión se lleve a cabo en otro momento.

4. Enunciar áreas de problemas, pero no intentar resolver cualquier problema que se ponga de manifiesto.

Una revisión no es una sesión de resolución de problemas. A menudo, la resolución de los problemas puede ser encargada al productor por sí solo o con la ayuda de otra persona. La resolución de los problemas debe ser pospuesta para después de la reunión de revisión.

5. Tomar notas escritas.

A veces es buena idea que el registrador tome las notas en una pizarra, de forma que las declaraciones o la asignación de prioridades pueda ser comprobada por el resto de los revisores, a medida que se va registrando la información.

6. Limitar el número de participantes e insistir en la preparación anticipada.

Se ha de mantener el número de participantes en el mínimo necesario. Además, todos los miembros del equipo de revisión deben prepararse por anticipado. El jefe de revisión debe solicitar comentarios escritos (que muestren que cada revisor ha revisado el material).

7. Desarrollar una lista de comprobaciones para cada producto que haya de ser revisado.

Una lista de comprobaciones ayuda al jefe de revisión a estructurar la reunión de RTF y ayuda a cada revisor a centrarse en los asuntos importantes. Se deben desarrollar listas de comprobaciones para los documentos de análisis, de diseño, de codificación e incluso de prueba.

8. Disponer recursos y una agenda para las RTF

Para que las RTF's sean efectivas, se deben planificar como una tarea del proceso de ingeniería del software. Además, se debe trazar un plan de actuación para las modificaciones inevitables que aparecen como resultado de una RTF.

9. Llevar a cabo un buen entrenamiento de todos los revisores.

Por razones de efectividad, todos los participantes en la revisión deben recibir algún entrenamiento formal. El entrenamiento se debe basar en los aspectos relacionados con el proceso, así como en las consideraciones de psicología humana que atañen a la revisión.

10. Repasar las revisiones anteriores.

Las sesiones de información pueden ser beneficiosas para descubrir problemas en el propio proceso de revisión. El primer producto que se haya revisado puede establecer las propias directivas de revisión.

CAPITULO IV.

TECNICA DE DESARROLLO DE SOFTWARE DE CALIDAD

4.1. CONCEPTO DE PROYECTO

Un proyecto es, ante todo, un proceso. Proceso es un conjunto de acciones que permite obtener un producto, un servicio o una modificación de éstos, los procesos pueden ser contables, financieros, administrativos, científicos, técnicos, etc.

Se distinguen por el fin que persiguen, los recursos y el tiempo necesario para su ejecución.

Ahora bien, no sólo son procesos los proyectos, muchos de los trabajos en la Institución se efectúan siguiendo procesos continuos, los cuales comprenden actividades que se repiten día tras día con las mismas características, o con variaciones tan pequeñas o previsibles que las convierten en actividades de rutina; en este caso se encuentran por ejemplo: La producción normal de empresas que trabajan para almacén, la actualización de la contabilidad, la prestación de servicios bancarios, la operación bancaria interna, los informes a los jefes, el control de calidad, etc.

Por otro lado, un proyecto se caracteriza por:

- Un conjunto de actividades relacionadas y dependientes entre sí.
- Un conjunto limitado de recursos.
- Un conjunto de restricciones internas y/o externas.
- Una fecha de inicio y término previamente establecidas.
- La no repetición en condiciones idénticas.
- La producción de una sola unidad o de un número muy pequeño, en el caso de procesos productivos.

En síntesis, un proyecto se define como [13]:

Un conjunto de actividades especiales y novedosas que están interrelacionadas y que tienen que realizarse en una fecha determinada, para lo cual se cuenta con un número limitado de recursos, con objeto de obtener un nuevo producto o servicio o la modificación de estos.

4.1.1. CICLO DE VIDA DE UN PROYECTO

Como todo proceso, el proyecto tiene una conclusión; el tiempo que transcurre del inicio al término, es el ciclo de vida de un proyecto.

Con frecuencia se confunde el proyecto en sí, con el producto que de él se obtiene. Esta situación quedará más clara con el siguiente ejemplo:

PROYECTO	PRODUCTO
Sistema completo de inversión que permita al Banco enfrentar la competencia.	Cuenta Universal

El proyecto y el producto tienen una duración independiente entre sí.

4.1.2. PROBLEMAS RELACIONADOS CON LOS PROYECTOS

Cuando realizamos cualquier actividad, ocasionalmente se presentan problemas, es evidente que cuando se trata de lograr un proceso de innovación u optimización, invariablemente tendrán que enfrentarse diversas clases de problemas desde aquellos que puedan producir el fracaso hasta aquellas dificultades insignificantes.

Los problemas fundamentales que posiblemente se encuentren en un proyecto son:

- Desconocimiento de aspectos que lo afectan, tales como:

1. Actividades necesarias
2. Duración de éstas
3. Materiales nuevos e inusuales
4. Tecnologías insólitas
5. Personal desconocido
6. Etc.

Todo lo anterior hace especialmente difícil la previsión

- Diferencias importantes entre los criterios y prácticas de los técnicos y administradores

- Frecuentes interferencias entre las actividades rutinarias y el proyecto

Actitud del personal ante presiones de tiempo y situaciones frustrantes. Además de éstos, existen otros problemas que se detectan sólo hasta el desarrollo.

La resolución de los problemas que se presentan no es igual en todos los casos, y esto se debe a que cada proyecto es único; sin embargo, es posible efectuar una serie de acciones que permitan evitar los problemas o por lo menos reducir su perjuicio.

4.1.3. METODOS DE ADMINISTRACION DE PROYECTOS

La técnica de evaluación y revisión de programas ("Program Evaluation and Review Technique" PERT) y el método del camino crítico ("Critical Path Method" CPM) son dos métodos de planificación de proyectos que se pueden aplicar al desarrollo de software, ambas son conocidas como Técnicas de Ruta Crítica.

Las Técnicas de Ruta Crítica son las herramientas más poderosas para planear y controlar un proyecto ya que brindan los siguientes beneficios:

- Son un marco de referencia consistente para planear, programar, implantar y controlar cualquier proyecto.
- Ilustran la interdependencia de todas las actividades, las tareas y las operaciones.
- Ayudan asegurar comunicaciones adecuadas entre departamentos, grupos y funciones.
- Definen la fecha esperada para terminar un proyecto.
- Identifican las denominadas actividades críticas que si se retrasan, también retrasan la terminación del proyecto.
- Identifican actividades con holgura que pueden ser retrasadas en tiempo determinado, sin problema, o de las que sus recursos pueden ser compartidos temporalmente sin causar daño.
- Determinan las fechas en que las actividades pueden empezarse para que el proyecto se realice de acuerdo a como fue programado.
- Ilustran aquellas actividades que necesitan coordinarse estrechamente para prevenir problemas de recursos.
- Permiten identificar aquellas tareas que pueden o deben hacerse en paralelo para cumplir el tiempo programado.
- Dependiendo del método que se utilice, pueden permitir estimar la probabilidad de terminar un proyecto en diferentes etapas.

Las Técnicas de Ruta Crítica no sustituyen el criterio de las personas que las utilizan, pero si colaboran con ellas en la obtención de resultados óptimos.

4.1.4. SEGUIMIENTO Y CONTROL DEL PROYECTO

Un antiguo dicho dice: "Los proyectos de software se salen de la agenda día a día". Un descuido de un día en la agenda raramente será fatal para el proyecto. Pero los días se van acumulando y, sobre la duración total del proyecto, los pequeños retrasos pueden producir grandes problemas.

Una de las gestiones del proyecto es seguir la pista y controlar el proyecto de software una vez que está en curso. El seguimiento se puede llevar a cabo de diferentes formas:

- Realizando reuniones periódicas sobre el estado del proyecto, en las cuales cada miembro del equipo informe de los progresos y de los problemas.
- Evaluando los resultados de todas las revisiones realizadas en todo el proceso de ingeniería del software.
- Determinando si los objetivos parciales del proyecto se están cumpliendo en la fecha programada.
- Reuniéndose informalmente con los técnicos para conocer sus valoraciones subjetivas acerca del progreso de cada momento y los problemas que acechan en el horizonte.

En realidad, todas estas técnicas de seguimiento las utilizan los gestores de proyectos con mucha experiencia.

4.2. FASES EN EL DESARROLLO DE UN PROYECTO

El proyecto, como proceso que es, está conformado por etapas que van desde su inicio hasta su conclusión, aunque bien es cierto que todos los proyectos difieren entre sí, es posible definir, en términos generales, un modelo que muestre gráficamente las etapas más representativas, este modelo puede observarse en la figura 4.1.

Identificar las diferentes etapas de un proyecto permitirá subdividirlo en partes con fines importantes para su conclusión, partes que se pueden medir y planificar como entidades, proporcionando el marco de referencia, y en el caso de proyectos grandes provee puntos intermedios de motivación al realizarse el logro de metas inmediatas.

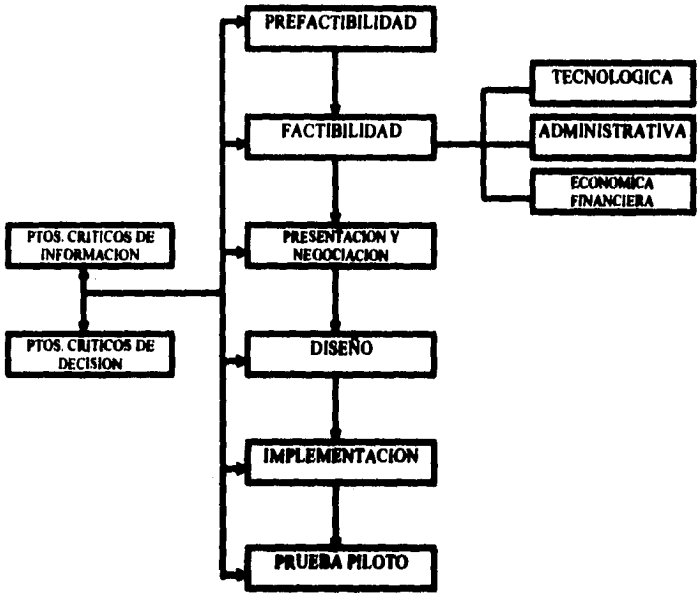


Figura 4.1. Etapas de un proyecto

4.3. PREFACTIBILIDAD

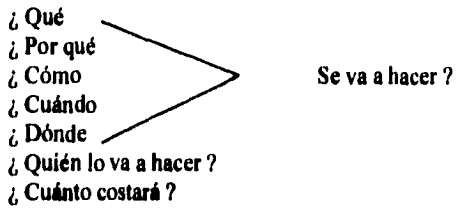
En esta etapa el proyecto es una idea que no ha tomado forma del todo, escasamente se tiene una imagen mental de un proyecto que cause innovación o resuelva un problema optimizando lo ya existente.

En una palabra, el proyecto hasta aquí, es sólo un concepto, posiblemente originado en forma espontanea por la creatividad del encargado del área de sistemas que capta una necesidad o una oportunidad o por alguna petición concreta por parte de los jefes, por tanto, sólo se concibe el detalle de los requisitos, aspectos funcionales que han de cumplirse, y la preparación de los planes generales para actuar.

4.4. FACTIBILIDAD

En este momento el proyecto es ya más tangible, más concreto por lo que es importante analizar y argumentar que tan viable es su realización, cuáles serán sus consecuencias, etc.

En síntesis, al efectuar el estudio de factibilidad deben responder las siguientes preguntas:



De los resultados del estudio de factibilidad puede depender el que se siga adelante o se detenga.

A fin de asegurar un enfoque integral, la factibilidad debe determinarse desde distintos puntos de vista.

Aún cuando los criterios para establecer la factibilidad variarán de un proyecto a otro, siempre deben considerarse los siguientes aspectos:

- Tecnológicos
- Administrativos
- Económicos-Financieros

4.4.1. FACTIBILIDAD TECNOLOGICA

Aquí es importante enfocar nuestro análisis al cómo se obtendrá el producto, servicio o modificación, para lo cual se revisan los siguientes factores:

1.- ¿Se requiere equipo especial para el proyecto ?
- Qué clase de equipo
- Existe este equipo
- Si no existe ¿ Es posible obtenerlo ?
- Es razonable el costo
- Cómo se puede tener acceso al equipo
- Cuándo se usará
- Quién dará la autorización
- Quién lo instalará
- Quién lo usará
- Habrá que entrenar personal para usarlo

2.- ¿ Cuánto personal se requiere para desarrollar el proyecto ?
- Que clase de personal se requiere ¿ Especializado ? ¿ No especializado ?
- Que clase de especialidad se requiere
- En función a la duración y complejidad del proyecto
- Se requiere contratar personal
- Qué clase de personal, planta u honorarios
- Se puede seleccionar personal que ya se encuentra laborando
- Cuáles podrían ser las fuentes de selección para reunir personal
- Quién hará la selección
- ¿ Tiene que trabajarse con equipos ya existentes ?
- Habrá que darle al personal entrenamiento especial

La factibilidad técnica es frecuentemente el área más difícil de evaluar en esta etapa del proceso de desarrollo del sistema. Debido a que los objetivos, las funciones y el rendimiento son de alguna manera confusos, cualquier cosa puede parecer posible si se hacen las consideraciones adecuadas. Es esencial que el proceso de análisis y de definición se realice en paralelo con el análisis de factibilidad técnica. De esta forma, se pueden juzgar las especificaciones concretas según se van determinando.

Las consideraciones que van asociadas normalmente a la factibilidad técnica son:

Riesgo del desarrollo. ¿ Puede el elemento del sistema ser diseñado de tal forma que las funciones y el rendimiento necesario se consigan dentro de las restricciones determinadas en el análisis ?

Disponibilidad de recursos. ¿ Hay personal calificado para desarrollar el elemento en cuestión ?¿ Están disponibles para el sistema otros recursos necesarios (de hardware y de software) ?

Tecnología. ¿ Ha progresado la tecnología relevante lo suficiente como para poder soportar el sistema ?

4.4.2. FACTIBILIDAD ADMINISTRATIVA

En este punto se deben examinar las siguientes cuestiones:

1. ¿ Es necesario modificar la estructura de organización del departamento para desarrollar el proyecto?
2. ¿ Quienes son los principales promotores del proyecto ?
3. ¿ Cuáles son las áreas que tendrán que intervenir ?

4. ¿ Cómo se obtendrá la participación de dichas áreas ?
5. ¿ Quién asignará las responsabilidades a las diferentes áreas ?
6. ¿ Cuál será el área con mayor responsabilidad ?
7. ¿ Cuáles áreas tendrán las responsabilidades críticas que podrían determinar el éxito o el fracaso del proyecto ?
8. ¿ Cómo y a quien se elaborarán los reportes de las áreas ?
9. ¿ Qué requisitos de tipo legal deberán cubrirse ?

4.4.3. FACTIBILIDAD ECONOMICA-FINANCIERA

Además de los aspectos cualitativos, los cuantitativos son necesarios para conocer la factibilidad total de un proyecto.

1. ¿ Cuál es el costo/beneficio ?
2. ¿ Cuál será el plazo para recuperar la inversión ?
3. ¿ Cuál será la rentabilidad esperada ?
4. ¿ Cuáles serán los requerimientos financieros durante el lapso del proyecto ?
5. ¿ Cuál será la inversión que se haga en la compra de equipo ?
6. ¿ Cuál será el costo en sueldos ?

Aunque no es recomendable, en ciertas ocasiones no se dispone de información cuantitativa y el proyecto queda sustentado sólo en opiniones, dejando al tiempo y a los hechos que muestren si el proyecto valía o no la pena.

En general todos los analistas se hacen las siguientes preguntas:
¿ Qué tan detallado y profundo debe elaborarse el estudio de factibilidad ?
Dependerá de muchos aspectos:

- Qué tan complejo es el proyecto
- Qué tan importante es el proyecto

Lo requieren los jefes, para autorizar y aprobar el proyecto ¿ Cuándo debe realizarse el estudio de factibilidad ?

- Si la complejidad del proyecto lo permite, en forma muy general, antes de la aceptación del mismo con efectos de presentación y negociación para su aprobación.
- Si lo requiere la complejidad del proyecto o lo solicitan para otorgar la aprobación final: Después de presentado y negociado el proyecto.

Como se observa en esta etapa del proyecto, se preparan las propuestas para la realización y el financiamiento del mismo.

4.4.4. ANALISIS ECONOMICO

Entre la información más relevante que contiene el estudio de factibilidad se encuentra el análisis de costo-beneficio, una evaluación de la justificación económica para un proyecto de sistema basado en computadora. El análisis de costo-beneficio señala los costos del desarrollo del proyecto y los contrasta con los beneficios tangibles e intangibles del sistema.

El análisis de costo-beneficio es complicado porque los criterios varían según las características del sistema a desarrollar, el tamaño relativo del proyecto y la recuperación esperada de la inversión como parte del plan estratégico del departamento. Además, muchos beneficios obtenidos de los sistemas basados en computadora son intangibles (p. ej.: una mejor calidad del diseño mediante una optimización iterativa, una mayor satisfacción del cliente debida a un control programable y unas mejores decisiones comerciales a partir de datos de ventas con formato previamente analizados). Puede ser difícil lograr comparaciones directas cuantitativas.

4.5. REVISIONES

Como vimos en el capitulo III la revisión es una de las actividades que se deben seguir en cada una de las etapas de un proyecto para lograr la calidad en el desarrollo del software. Freedman y Weinberg [2] argumentan de la siguiente forma la necesidad de revisiones:

- El trabajo técnico es realizado por humanos lo cual implica que esta sujeto a errores.
- Aunque la gente es buena cazando algunos de sus propios errores, algunas clases de errores se le pasa por alto más fácilmente al que los origina que a otras personas.

Una revisión cualquiera que esta sea es una forma de aprovechar la diversidad de un grupo de personas para:

1. Señalar la necesidad de mejoras en el producto de una sola persona o un grupo.
2. Confirmar las partes de un producto en las que no es necesario o no es deseable una mejora.
3. Conseguir un trabajo técnico de una calidad más uniforme o al menos más predecible, que la que puede ser conseguida sin revisiones, con el fin de hacer más manejable el trabajo técnico.

Efectuar reuniones para la revisión de un proyecto de desarrollo de software es una de las funciones del líder del proyecto; para evitar que éstas sean poco productivas, es necesario que el tiempo de la reunión se administre de manera efectiva, para lograrlo se recomienda considerar 3 aspectos:

- Orden del día.
- Manejo de la reunión.
- Acta y compromiso.

Orden del día

1. Establezca una agenda que contenga finalidades.
2. Sea específico, diríjase a tratar los puntos más importantes y de ser posible delimite qué decisiones deben tomarse planteando el análisis de las consecuencias de tales decisiones.
3. Si lo cree necesario publique la agenda por adelantado.

Manejo de reuniones

1. Limite el tiempo de las discusiones y las decisiones, sin embargo, sea flexible, cuando para llegar a una decisión sobre un punto en particular se requiere más tiempo, recuerde que es de mayor importancia obtener conclusiones positivas y lograr los objetivos de la reunión que cumplir un tiempo determinado.
2. Sintetice al término de una reunión o tema, asegúrese de que cada persona salga de la reunión con las mismas conclusiones.

Acta y compromiso

Solicite a alguno de los integrantes levante un acta de los puntos tratados, en la reunión, los compromisos y responsabilidades asignadas, así como de las decisiones tomadas.

Aliente la participación de todos, no deje que se pierda una buena idea por falta de colaboración, asegure el logro de resultados de su equipo.

4.6. PRESENTACION Y NEGOCIACION

La realización del proyecto no será patente hasta que no exista la autorización y el apoyo que este requiere, para lo cual es importante que el promotor o autor del mismo identifique la instancia o instancias que tienen la facultad de aprobar, sin dejar de lado el nivel jerárquico y la comunicación ascendente.

Una vez realizada esta actividad, documente su proyecto de la siguiente manera:

- Cuál es el objetivo
- Por qué surge
- A que necesidades responde
- En términos generales de qué trata
- Cuáles son los beneficios para el departamento, el usuario, etc.
- Qué factibilidad tiene
- Qué obstáculos se podrían encontrar y cuáles serían las posibles alternativas de solución
- Tenga siempre presente que si usted mismo no está convencido de que el proyecto debe ser desarrollado, no logrará convencer a los demás

Ya documentado éste, seleccione la información que le será útil para elaborar un reporte escrito en forma concreta y directa, de tal manera, que le permita negociarlo; si es necesario, elabore materiales de apoyo. Ver figura 4.2, no olvide que de la presentación puede depender la decisión favorable o no para que el proyecto inicie, continué o sea detenido.

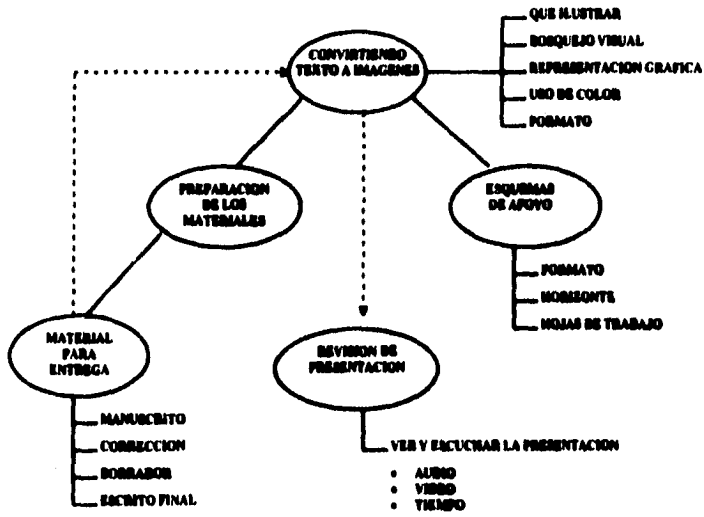


Figura 4.2. Presentación

El siguiente paso será realizar la negociación, considerando los principios que se presentan a continuación:

- Madurar la presentación
- Abordar con beneficios
- Estimular el deseo
- Tratar los hechos
- Retirar los obstáculos
- Obligar a la acción

Madurar la presentación. Revise nuevamente la información que va a presentar. Elija el momento oportuno considerando todos los aspectos, incluso la urgencia del proyecto.

Abordar con beneficios. Mencione el objetivo y los antecedentes del proyecto e inmediatamente después:
- Plantee los beneficios en la forma más atractiva que pueda
- Explique en que proporción se da el costo/beneficio

Estimular el deseo. Muestre cómo el proyecto satisface una necesidad real y concreta y de ser necesario insista en sus beneficios, y como éstos responden a una necesidad concreta.

Tratar los hechos. Si ya ha sido obtenida la decisión o para ello se requiere de mayor información, explique los hechos:
- Cómo se desarrollaría el plan
- Cuándo deberá empezarse
- Qué áreas intervienen
- Magnitud
- Y otros factores de importancia

Retirar los obstáculos. Es posible que existan objeciones de suficiente peso para detener la decisión o inclinarla en forma negativa.

Identifique los obstáculos y trate de eliminarlos, atenuarlos o proponer alternativas de solución, si las objeciones son muy importantes, la alternativa de solución debe satisfacer completamente a la audiencia.

Obligar a la acción. No intente obtener inmediatamente la decisión, solicítela y de tiempo razonable para la reflexión; ¿Cuánto? dependerá de la audiencia y de la importancia del proyecto; si la decisión necesitara de varios días, solicite fechas de decisión y negocie las mismas de acuerdo a sus planteamientos anteriores.

4.6.1. PLANEACION Y EJECUCION DE LA NEGOCIACION

La planeación constituye el factor clave, para el éxito en una entrevista de negociación.

El analista, experto en el área de sistemas, puede estar seguro de sus conocimientos técnicos, e incluso, tener en mente un plan formado con base en los datos que conoce del usuario, sin embargo, es durante la entrevista en donde podemos evaluar sus resultados a través de las acciones concretas que nos permiten afirmar que ha tenido éxito.

La planeación permite recabar información suficiente para preparar el contacto con el usuario y realizar la entrevista con objetivos previamente definidos.

La planeación y ejecución, son análogos a la causa-efecto en un fenómeno. Quien controla las causas dirigirá los efectos, quien posee mayor información controlará la entrevista de negociación. Quien obtiene información y sabe analizarla puede desarrollar un plan de acción. Sólo desarrolla un plan de acción quien posee los instrumentos adecuados de trabajo y el dominio técnico de lo que esta ofreciendo.

El usuario no desea saber de sistemas y lo experto que somos en esta área a él le interesa de como sus necesidades pueden ser satisfechas a través de nuestros productos y/o servicios. El usuario no quiere que el analista le diga que hacer, él quiere decidir. Si el experto le ofrece las mejores opciones, él tomará sus propias razones.

La calidad en el servicio reside en la atención personal profesional que recibe el cliente del personal del área de sistemas. Un analista que domina perfectamente su especialidad, es un analista preparado, pero aun no podemos afirmar que sea un experto en negociación.

4.6.2. PLANEACION ESTRATEGICA DE LA ENTREVISTA

Cuando se ha logrado establecer el día y la hora de la cita con el usuario, es porque ya se tiene fijado el objetivo de la entrevista.

Distinguiremos dos tipos de entrevista:

1. La entrevista informativa
2. La entrevista resolutiva

La entrevista informativa tiene por objeto el obtener información específica del cliente para poder elaborar un diagnostico de su situación y proceder a armar el

paquete de servicios adecuado. Desde que se establece la ambientación con el cliente, hasta que se pide la cita para la siguiente entrevista, que será la de la propuesta formal, se dirige la entrevista a base de preguntas, con una actitud de "escucha responsiba", es decir, captar lo que el usuario esta expresando.

La entrevista resolutiva tiene por objeto cerrar el compromiso de trabajo, lo cual se hace después de proponer el producto que se tiene preparado, lo cual se hace a través de la "argumentación", que es la presentación del producto en términos de beneficios.

El tener preparada la argumentación que se va a hacer en la entrevista resolutiva, en términos de beneficios, es determinante al éxito de la misma.

Características. Es el conjunto de componentes que integran el producto y/o descripción cuantitativa del mismo (tamaño de código, velocidad de procesamiento, etc.).

Beneficios. Son las razones por las que el usuario compra. Es hablar del producto, mencionando como opera en beneficio del usuario. Es describir como el producto cubre las necesidades del usuario. Es responder a la pregunta mental del usuario ¿a mi, en que me beneficia?

Ventajas. Las ventajas, constituyen el argumento que damos al cliente resaltando aquello que nuestro producto puede hacer en relación a lo que ofrece la competencia.

4.6.3. MANEJO DE OBJECIONES Y CIERRE

El manejo de objeciones y el cierre, se encuentran internamente ligados, de hecho cada objeción nos acerca al momento del cierre. El manejo de las objeciones, sigue una técnica de alianza con el cliente, para establecer emocionalmente la seguridad de que le hemos comprendido.

En la entrevista resolutiva, debemos de tener en mente la contestación a las posibles objeciones del cliente. Y los pasos que se deben seguir son los siguientes:

1. Escuche con atención la objeción
2. Haga preguntas en caso necesario, para estar seguro de que se ha comprendido al cliente
3. Haga un resumen de lo que el cliente le ha dicho
4. De una respuesta

El cierre representa el final de la entrevista resolutiva, gracias a el podemos estar seguros del éxito o fracaso.

Si se ha hecho un análisis de las características del cliente (perfil), también se ha realizado un diagnostico y se le ha ofrecido lo que requiere en materia de sistemas, habrá habido preguntas acerca del como operar la idea, pero finalmente el resultado final es el cierre.

Algunos autores hablan de distintos tipos de cierre:

El cierre indirecto, es aquel que se da haciendo alguna pregunta al cliente, que lleva implícita la idea del cierre del contrato.

El cierre de "Dado por hecho", es el que se hace suponiendo que el cliente ha aceptado las condiciones una vez resueltas las objeciones o dudas.

Si por alguna causa, el cliente habla de datos que no teníamos contemplados y que nos hacen replantear el plan propuesto, podemos proceder a los llamados "Cierres parciales", es decir, de aquello que puede ya llevarse a cabo y solicitamos el día y hora de la siguiente cita para presentar la propuesta que queda pendiente.

Errores típicos en el cierre:

- Presionar al cliente a una decisión, cuando aun tiene dudas.
- Insistir en que el plan que traemos preconcebido es el mejor plan. Esto ocurre cuando los jefes han pasado muchas horas de trabajo de escritorio pensando en todos los detalles y le parece difícil aceptar que es necesario modificar su "Plan perfecto".
- Preguntar de modo "Ceremonioso", ¿Entonces, va a aceptar nuestros servicios?, ¿Entonces, si procedemos?. Es decir, plantear al cliente que ha llegado el momento de que decida y que en nuestra mente existe la duda de si aceptará o no, esta inseguridad el cliente la detecta y posiblemente prefiera en ese momento determinante, pensarlo mejor, ver otras opciones, consultarlo, etc.

4.7. PROTOTIPOS

Normalmente un cliente define un conjunto de objetivos generales para el software, pero no identifica los requisitos detallados de entrada, proceso o salida. En otros casos, el programador puede no estar seguro de la eficiencia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma en que debe realizarse la interacción hombre-máquina. En estas y muchas otras situaciones,

puede ser mejor método de ingeniería del software la construcción de un prototipo.

La construcción de prototipos es un proceso que facilita al programador la creación de un modelo de software a construir. El modelo tomará una de las tres formas siguientes: (1) un prototipo en papel o un modelo basado en PC que describa la interacción hombre-máquina, 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, o (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.

La figura 4.3. muestra la secuencia de sucesos del paradigma de construcción de prototipos. Como en todos los métodos de desarrollo de software, la construcción de prototipos comienza con la recolección de los requisitos. El técnico y el cliente se reúnen y definen los objetivos globales para el software, identifican todos los requisitos conocidos y perfilan las áreas en donde será necesario una mayor definición. Luego se produce un "diseño rápido". El diseño rápido se enfoca sobre la representación de los aspectos del software visibles al usuario. El diseño rápido conduce a la construcción de un prototipo. El prototipo es evaluado por el cliente/usuario y se utiliza 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.

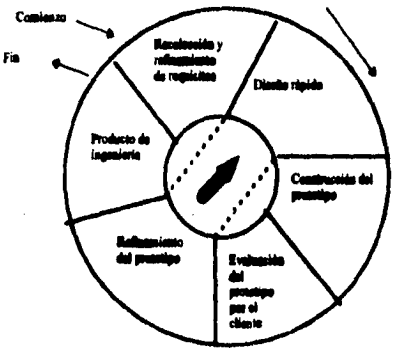


Figura 4.3. Creación de prototipos

4.8. DISEÑO DEL PROYECTO

El diseño se realiza en 2 niveles

- Preliminar
- Detallado

El diseño preliminar, también conocido como; Conceptual o Ingeniería básica, plantea el proyecto en forma general, sin pormenorizar y definiendo las partes básicas que intervendrán.

La revisión, el análisis y la crítica constructiva por distintos criterios, especialmente opuestos, permitirá enriquecer el diseño, encontrar y homogeneizar alternativas de solución.

El diseño detallado o ingeniería de detalle, en este punto los más pequeños aspectos son importantes, sin embargo, es relativamente fácil resolver los problemas por todo el trabajo que hasta este momento se ha desarrollado, al terminar el diseño se recomienda solicitar nuevamente distintas opiniones a fin de asegurar su calidad.

Es muy importante tener presente que si se involucra a la gente, podrán disminuirse los efectos desfavorables que en forma natural se producen frente a un cambio.

4.8.1. DISEÑO Y CALIDAD DEL SOFTWARE

A lo largo del proceso de diseño, la calidad del diseño resultante se evalúa mediante las revisiones técnicas formales, que se describen en el Capítulo III. Para evaluar la calidad de una representación del diseño debemos establecer unos criterios que determine cuándo es bueno, y es importante tomar en cuenta lo siguiente:

1. Un diseño debe exhibir una organización jerárquica que haga un uso inteligente del control entre los componente del software.
2. Un diseño debe ser modular; esto es, el software debe estar dividido de forma lógica en elementos que realicen funciones y subfunciones específicas.
3. Un diseño debe contener representaciones distintas y separadas de los datos y de los procedimientos.
4. Un diseño debe llevar a módulos (p.ej.: subrutinas o procedimientos) que exhiban características funcionales independientes.
5. Un diseño debe llevar a interfaces que reduzcan la complejidad de las conexiones entre los módulos y el entorno exterior.

6. Un diseño debe obtenerse mediante un método que sea reproducible y que esté conducido por la información obtenida durante el análisis de los requisitos del software.

Esas características, deseables para un buen diseño, no se consiguen fácilmente. El proceso de diseño en la ingeniería del software conduce a un buen diseño mediante la aplicación de principios fundamentales de diseño, de una metodología sistemática y de una concienzuda revisión.

4.9. IMPLEMENTACION

El proyecto se inicia, la coordinación de los recursos humanos y la obtención de los recursos materiales, tecnológicos y económicos es un asunto de vital importancia, para la consecución del objetivo; y las decisiones que se tomen, deberán ser las correctas para evitar problemas.

Puntos críticos de información. En el modelo teórico, las etapas que componen el proyecto se realizan en forma secuencial. En la práctica, habrá ocasiones en que la misma dinámica del proyecto vaya planteando situaciones que permitan que una etapa sea iniciada antes de terminar otra. Para que estos cambios no afecten negativamente el éxito de la tarea es necesario contar con los elementos que posibiliten la disminución del nivel de riesgo y la toma de decisiones.

Para distinguir estas situaciones se identifican dos momentos cruciales en cada etapa del proyecto figura 4.3. el P.C.I. y el P.C.D.

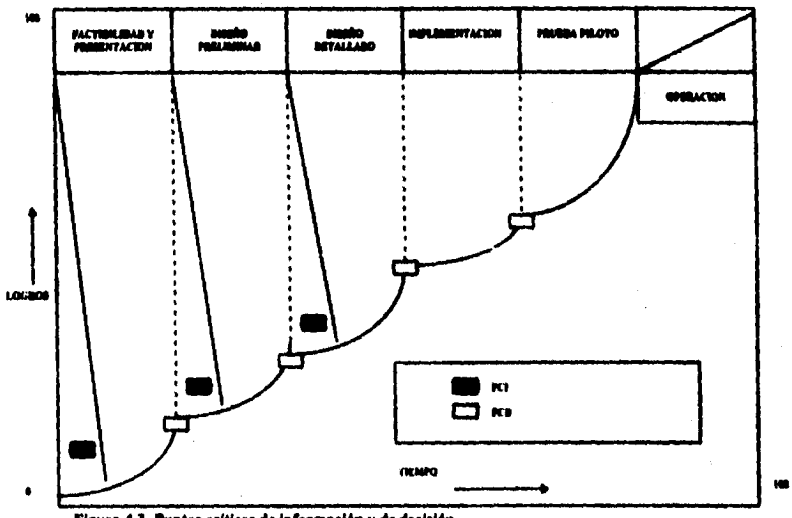


Figura 4.3. Puntos críticos de información y de decisión

El P.C.I. o "Punto crítico de información" es el lapso o período en el cual debe obtenerse la información para que el proyecto avance. En las etapas de prefactibilidad y factibilidad, por ejemplo, se requiere información suficiente para sustentar la viabilidad del proyecto y poder continuar con la siguiente etapa.

El P.C.D. o "Punto crítico de decisión" es el punto, donde debe ser tomada la decisión de iniciar o no la siguiente etapa, dependiendo del nivel de riesgo, esta decisión no elimina de ningún modo la necesidad de terminar la etapa anterior, y esta ha de concluirse con toda la documentación que amerite el caso.

Dado que, para disminuir el nivel de riesgo en cualquier decisión es indispensable basarse en información adecuada y oportuna, y que, a su vez una decisión genera nueva información, a lo largo del proyecto se presentan alternadamente puntos críticos de información y decisión.

La implicación que el P.C.I. y el P.C.D. tienen sobre el avance del proyecto es posibilitar que simultáneamente pueda realizarse la conclusión de una etapa y el inicio de otra.

4.10. PRUEBA

Cuando se ha concluido el desarrollo del proyecto es tiempo de implantar el software, o de promover e iniciar la operación del mismo, sin embargo, es aconsejable que antes de ordenar la operación normal definitiva se efectúen pruebas en las que se practiquen las actividades bajo condiciones casi normales y con las limitantes que conlleva una situación de verificación.

Si en estos momentos se considera necesario, se harán los últimos ajustes.

Es recomendable que en esta etapa participen, no sólo las personas a cuyo cargo estuvo el proyecto, si no también las que se encargarán de efectuar la operación normal del mismo, esto asegurará una comunicación eficiente entre ambos grupos y evitará fallas en la recepción de instrucciones.

Normalmente las pruebas que se desarrollan para programas de computadoras pequeños se enfocan como si se tratara de una sola unidad. Además, debido a las presiones que existen por parte de los jefes con respecto a la liberación del producto no permiten la elaboración de una estrategia de prueba bien planeada.

Un modelo del proceso de pruebas que se puede recomendar a las áreas de programación para confirmar la calidad de un software es el siguiente:

- Se genera un conjunto de pruebas (datos de entrada del programa y datos de salida esperados). Los datos de entrada pueden obtenerse usando casos límites o simplemente generándolos en forma aleatoria (Prueba de la caja negra).

- El programa se ejecuta usando el conjunto de pruebas y las salidas se comparan con las salidas esperadas. También se hace una revisión para determinar si el conjunto de pruebas es "adecuado".

- Si al menos hay un caso de prueba que detecta un error, el programa es corregido y se realizan pruebas de regresión; si no se detecta ningún error en las pruebas de regresión pero el conjunto de pruebas es "inadecuado", se incluyen pruebas adicionales en dicho conjunto.

- El proceso continúa hasta que el programa se ha ejecutado usando un conjunto de pruebas "adecuado" que no es capaz de exponer ningún error.

En este punto el programa se entrega. Aunque no se esta garantizado de ser correcto, entre "mejor" sea el criterio que determina si el conjunto de pruebas es "adecuado" más es la confianza de que el programa sea correcto.

4.11. MANTENIMIENTO

El mantenimiento de un producto, es por supuesto, mucho más que una "corrección de errores". Podemos describir el mantenimiento describiendo las cuatro actividades que se llevan a cabo tras liberar un proyecto.

La primera actividad de mantenimiento es debida a que no es razonable asumir que la prueba del software haya descubierto todos los errores latentes de un gran sistema de software. Durante el uso de cualquier gran programa, se encontrarán errores, siendo informado al desarrollador. El proceso que incluye el diagnóstico y la corrección de uno o más errores se denomina mantenimiento correctivo.

La segunda actividad que contribuye a la definición de mantenimiento se produce por el rápido cambio inherente a cualquier aspecto de la informática. Se anuncian nueva generaciones de hardware en ciclos de unos 24 meses; regularmente, aparecen nuevos sistemas operativos o nuevas versiones de los antiguos; frecuentemente, se mejoran o modifican los equipos periféricos y otros elementos de los sistemas. Por otro lado, la vida útil del software de aplicación puede fácilmente superar los diez años, sobreviviendo al entorno del sistema para el que fue desarrollado. Por tanto, el mantenimiento adaptativo es tan necesario como usual.

La tercera actividad que se puede aplicar a la definición de mantenimiento se produce cuando un paquete de software tiene éxito. A medida que se usa el software, se reciben de los usuarios recomendaciones sobre nuevas posibilidades, sobre modificaciones de funciones ya existentes y sobre mejoras en general. Para satisfacer estas peticiones, se lleva a cabo el mantenimiento perfectivo. Esta actividad contabiliza la mayor cantidad de esfuerzo empleado en el mantenimiento de software.

La cuarta actividad de mantenimiento se da cuando se cambia el software para mejorar una futura facilidad de mantenimiento o fiabilidad, o para proporcionar una base mejor para futuras mejoras. A menudo denominada mantenimiento perfectivo.

4.12. CONTROL DE CAMBIOS

Como vimos en el capitulo I, llevar un control en los cambios que se realicen durante el desarrollo y el mantenimiento del software contribuye a obtener un software con calidad..

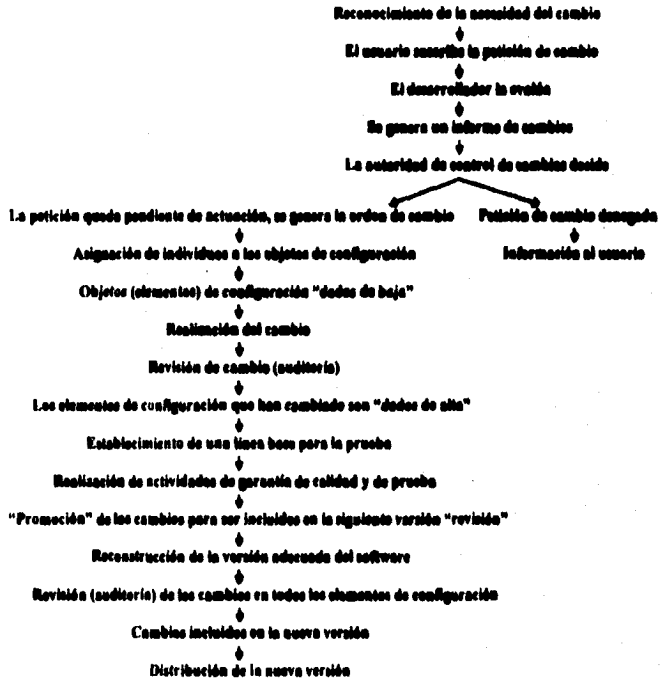


Figura 4.5. El proceso de control de cambios

El proceso de control de cambios que normalmente se sigue está ilustrado en la figura 4.5. Se hace una petición de cambio y se evalúa para calcular el esfuerzo técnico, los posibles efectos secundarios, el impacto global sobre otras funciones del sistema y los costos estimados. Los resultados de la evaluación se presentan como un informe de cambios a la autoridad de control de cambios (una persona o grupo que toma la decisión final del estado y la prioridad del cambio). Para cada cambio aprobado se genera una orden de cambio. La orden describe el cambio a realizar, las restricciones que se deben respetar y los criterios de revisión y de auditoría. El producto que se obtiene como resultado de los cambios se considera un software con otra versión.

4.13. REGISTRO Y REALIZACION DE INFORMES

Después de la planeación y programación del proyecto, el siguiente paso es la implantación, la cual se entiende como el desarrollo del proyecto, la realización concreta de lo planeado.

De ahí la importancia de la información para la toma de decisiones, con respecto a la marcha del proyecto y al logro de los objetivos.

Para ello se debe establecer un sistema eficaz que nos auxilie para obtener la información pertinente.

La recopilación de la información debe fluir de los componentes del equipo hacia el administrador del proyecto y viceversa, por tanto, es indispensable garantizar hasta donde sea posible, la oportunidad y exactitud de ésta.

En ella, el personal involucrado observa los resultados, por lo que se convierte en una importante fuente de motivación, además, siendo un auxiliar en la toma de decisiones, permite tomar las medidas correctivas correspondientes, independientemente de que posibilita la elaboración de pronósticos realistas y oportunos.

Resulta difícil pensar en examinar ocasionalmente la información, se debe reconocer su carácter dinámico, para advertir las fallas oportunamente, el objetivo es contar con una cantidad razonable, en el momento más conveniente.

El control de dicha información ha de estar definido de tal forma que constantemente propicie la difusión de elementos que motiven al personal para ser productivos y alcanzar la meta planeada.

La comprobación paso a paso del cumplimiento del diseño, es tarea del administrador del proyecto, auxiliado por un grupo que analice también, los modelos, planes de pruebas y pruebas preliminares y finales. Esta actividad debe efectuarse periódicamente y conducir a tomar la decisión crítica en cuanto a si ya se ha de proseguir con la siguiente fase.

Para fijar la frecuencia de estos informes, debe considerarse la importancia del proyecto y que el tiempo para prepararlos guarde proporción con el mismo.

Se han diseñado diversos tipos de formatos para los informes; existen desde los escritos a mano, hasta los producidos por computadora. En el apéndice A se muestran dos de estos.

Pero la elección del o los formatos que se han de utilizar en cada caso estará determinada por la utilidad que represente y la posibilidad de elaborarlo y entenderlo perfectamente.

4.14. ESTRUCTURAS ORGANIZACIONALES

No hay una forma de organización "optima" para la administración de proyectos. En un sólo departamento pueden existir simultáneamente diversas formas de organización.

En la mayoría de los casos, para administrar un proyecto dado, el método ideal debe de sacrificarse en aras del beneficio de toda la organización.

Estructura funcional. La mayoría de las estructuras de las empresas responden a este tipo de organización. En esta estructura las actividades se agrupan en funciones mayores, por ejemplo en ventas estarían: Promoción, comercialización, investigación de mercados, etc.; En producción reportarían: Manufactura, control de calidad, mantenimiento, etc.; En administración: Caja, contabilidad, nóminas, etc.

Aquí la autoridad se transmite sobre las líneas del organigrama y la comunicación formal fluye de arriba hacia abajo y viceversa sólo por esas líneas. Los proyectos difícilmente pueden existir en este tipo de organización por los conflictos que se provocan entre las distintas áreas, por esta razón se han diseñado estructuras especiales.

Estructura por proyecto. Aquí el encargado del proyecto tiene autoridad directa sobre todos los recursos del proyecto, durante el tiempo que éste dure. Este tipo de estructura presenta las siguientes ventajas y desventajas:

Ventajas	Desventajas
Autoridad.	Sensación de estar atrapado.
Canales directos de comunicación.	Ataduras cuando algunos recursos necesitan tiempo parcial.
Flexibilidad en el equipo de trabajo.	Falta contacto con otros proyectos y soluciones.
Propósitos únicos.	En el curso del desarrollo no puede verse fácilmente el término del proyecto.
Camaradería.	
Prioridades claras	
Sentido de pertenencia.	
Jefe único.	

Estructura matricial. En este tipo de organización, la autoridad sobre los recursos del proyecto está en manos del jefe del departamento, el líder del proyecto no tiene autoridad directa.

A continuación se presentan ventajas y desventajas de la estructura matricial:

Ventajas	Desventajas
Flexibilidad para el uso de recursos parciales.	Comunicación en distintos canales.
Estímulo técnico para proyectos múltiples.	Autoridad del líder del proyecto.
Capacitación para el futuro.	Interferencia por prioridades de otros proyectos.
Constancia del jefe del departamento.	Sensación de tener dos jefes.
Seguridad.	Mezclar varios proyectos a la vez.

Este tipo de estructura es el que con mayor frecuencia se observa.

4.15. DEFINICION DE RESPONSABILIDADES Y OBLIGACIONES DE LOS PARTICIPANTES

Uno de los aspectos fundamentales para la organización del equipo es definir claramente las responsabilidades y obligaciones de los miembros del mismo ya que, además de ser una fuente importante de motivación, evitará que se dé:

- Duplicidad de esfuerzos
- Actividades no realizadas
- Desorganización

Para asignar correctamente las obligaciones debe partirse en primera instancia de las actividades, tareas u operaciones que se planean realizar.

Al relacionar las operaciones con las personas y los puestos que participan en el proyecto, se obtendrá un cuadro de responsabilidades, como el presentado en la Figura 4.4, dicho formato puede ser de gran utilidad para apreciar con rapidez el papel de la función que le corresponde desarrollar a cada uno de los recursos humanos, para evitar que la responsabilidad se diluya debe haber un responsable para las actividades.

1.- LISTE LAS ACTIVIDADES EN EL LADO IZQUIERDO Y EN LA PARTE SUPERIOR LAS PERSONAS O LOS PUESTOS
2.- ASIGNE EN CADA CASO UNA LETRA : R, A, I, P, E, PARA INDICAR EL PAPEL, LA FACULTAD O LA RELACION
R = RESPONSABLE A = APROBAR I = INFORMAR P = APOYAR E = EVALUAR

ACTIVIDAD / PERSONAS O PUESTOS										OBSERVACIONES

Figura 4.4 Cuadro de responsabilidades

Otro punto importante al realizar la asignación de responsabilidades, consiste en definir y distinguir las facultades que tiene el personal asignado durante el desarrollo de la tarea; esto último estará en función de la magnitud de la actividad, el nivel jerárquico del personal y la descripción de su puesto.

Para simplificar y clarificar la definición de la tarea que una persona ha de desempeñar dentro del proyecto, puede elaborarse un documento descriptivo específico para todo el personal, partiendo del líder del proyecto; en dicho documento se podrán estipular los puntos más importantes de su función.

Tratándose de un proyecto, los elementos esenciales pueden ir cambiando con cada asignación. El documento no es una descripción de puestos, es un apoyo, una combinación de objetivos específicos de una tarea o proyecto, y una serie de descripciones del trabajo.

El proceso del documento es un proceso de negociaciones cuyo resultado es que la definición de actividades sea clara, su propósito es garantizar que ambas partes (líder y personal) se entiendan y, por consiguiente, la clave está en que la persona

que recibe la asignación repase o elabore el documento y posteriormente, lo discuta con la persona que le ha asignado el trabajo a efecto de asegurarse que el convenio/entendimiento es mutuo. El documento ofrece dos ventajas fundamentales:

- Puede eliminar muchas conjeturas, que de otra manera pasarían inadvertidas y, por consiguiente deriva en mejores decisiones.
- Reduce riesgos innecesarios pues permite mayor claridad para la planificación y el control.

Los elementos esenciales de dicho documento son:

Metas. ¿Cómo sabrá una persona que su trabajo ha terminado? ¿Cuáles son los puntos básicos, claros y mensurables del trabajo asignado? ¿Quién medirá el trabajo y cómo? ¿Cuáles son las prioridades relativas de los puntos de medición, comparándolos entre sí y con otros trabajos? ¿Cuáles son las medidas intermedias y cuál su prioridad?

Autoridad. ¿Quién tiene autoridad para contratar y revisar los recursos? ¿Existe alguna forma de retroalimentación a la organización en un proyecto matricial? ¿Qué es autoridad de compras? ¿Quién controla los fondos?

Informes. ¿Cuáles son las relaciones para presentar informes en sentido ascendente o lateral? ¿A quién dirigirse cuando hay que delegar autoridad, quitar obstáculos o hacer revisiones formales periódicas?

Riesgos. ¿Cuáles son los riesgos clave considerados inherentes?

Recursos clave. Las personas, el material o el equipo que el administrador considere importante para el éxito.

No se puede esperar que todos los puntos antes mencionados queden definidos completamente al inicio del proyecto, algunos evolucionarán con el paso del tiempo, y otros cambiarán conforme el proyecto o el entorno cambian o van aclarándose. Sin embargo, es importante que el líder o los componentes del equipo traten de resolver estos puntos lo mejor que puedan, y que continúen buscando mayor precisión conforme avance el trabajo. Esto aumentará la seguridad profesional y disminuirá los riesgos personales inherentes al trabajo.

Las revisiones periódicas de los logros y las deficiencias, mostrarán cuándo la persona llega a un punto de medición clave. No se debe esperar el punto "mágico" de los semestres o años para revisar el avance individual, esta revisión ha de efectuarse conforme se van logrando los puntos o conforme van surgiendo deficiencias. De tal forma, la información será inmediata, permitirá que el

proceso individual de administración vaya ajustándose y proporcionará información específica para realizar las revisiones semestrales o anuales.

4.16. LA TECNICA FRENTE A LA CALIDAD

La técnica propuesta se basa principalmente en analizar y proponer puntos importantes que se deben seguir en cada una de las etapas que forman el desarrollo de un proyecto. Así esta técnica es de gran utilidad para los desarrolladores de software.

Esta técnica cumple con los puntos uno, dos, tres, cinco, seis y siete que se requieren en la sección 1.3.3. para lograr un software de calidad. El punto cuatro puede ser llevado a cabo por los encargados del desarrollo del software como parte de una revisión técnica formal, debido a que los estándares en el desarrollo de un software varía de empresa a empresa.

Puntualizando, en relación a los puntos listados en la sección 2.3.1., la técnica cumple los puntos uno, cinco, siete, ocho y nueve. Los puntos dos, tres, cuatro y seis son aspectos administrativos que deben ser considerados por cada una de las empresas por lo que rebasan el alcance de la técnica propuesta.

En resumen, dentro de una empresa comprometida a satisfacer los requisitos del estándar ISO 9001, esta técnica resultaría de gran utilidad.

En relación al modelo CMM, se debe considerar que una técnica no es toda una empresa. Ahora bien, para empresas que se encuentran en el nivel inicial esta técnica sería de gran utilidad ya que se supone que no se cuenta con ninguna metodología para el desarrollo del software. En el nivel dos se habla de documentar y apegarse a estándares, aspectos que esta técnica ayuda a cumplir. En el nivel tres se habla de procesos estandarizados, que esta técnica también ayuda a reforzar. Así pues, esta técnica ayuda a una empresa a lograr el nivel dos y tres. En lo relativo al nivel cuatro, debido a sus limitaciones de tipo administrativo queda fuera de alcance.

Para terminar, debe considerarse nuevamente el enfoque de esta técnica hacia sistemas medianos y pequeños, probablemente en un ambiente de productos de software en pequeña escala. En ese tipo de empresa tal vez resulte poco realista hablar de los niveles cuatro y cinco del modelo CMM.

CONCLUSIONES

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

Sin embargo, mediante la combinación de métodos completos para todas las fases del desarrollo del software, mejores herramientas para automatizar estos métodos, bloques de construcción más potentes para la implementación del software, mejores técnicas para la garantía de calidad del software y una filosofía predominante para la coordinación, control y gestión podemos conseguir una técnica para el desarrollo de software con calidad.

Algunos puntos básicos que requieren de especial atención por parte de los administradores de un proyecto para lograr un producto con calidad se presentan a continuación:

- Evitar la rotación constante del personal que integra un equipo de desarrollo ya que esto provoca: que no se cumpla con el calendario establecido para desarrollar un proyecto, perdida de tiempo al capacitar a los nuevos integrantes, falta de continuidad en el desarrollo de los proyectos.

- Que existan sistemas de normalización y certificación en las empresas para que estas elaboren productos con calidad y puedan competir en el mercado.

- La documentación actualizada que indique la descripción de las variables, la interrelación de programas, así como de las bases de datos que son afectadas por dichos programas.

- La aplicación de un plan de prueba que ayude a asegurar una efectiva detección de errores de programación antes de entregar la aplicación al usuario final.

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

A pesar de que la calidad de un sistema puede ser bastante cuestionable, éste permanecerá en operación si cumple con lo requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente. Permanecerá en uso con mayor razón si el administrador tiene suficiente habilidad para conducirse en el terreno político del cliente, de tal forma que obtenga de éste todo el apoyo para que el sistema se siga utilizando así, mientras se corrigen los "detalles" encontrados en el sistema y a la vez se incorporan nuevos requerimientos.

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

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

APENDICE A

RESUMEN GLOBAL DEL ESTADO QUE GUARDA EL PROYECTO

AUTOR	FECHA	VERSION No.	NOMBRE DE PROYECTO
ADMINISTRADOS DEL PROYECTO	DIRIGENTE PRINCIPAL DEL PROYECTO	SUST. VERSION No.	No. DE PROYECTO

No. DE LA ACTIVIDAD	NOMBRE DE LA ACTIVIDAD	RESP. DE LA ACTIVIDAD		FECHA DE INICIO	FECHA DE TERMINO	IMP. DE LA ACTIVIDAD	COSTO DE LA ACTIVIDAD	FECHAS DE REVISION					
			PLAN.										
			REAL										
			PLAN.										
			REAL										
			PLAN.										
			REAL										
			PLAN.										
			REAL										
			PLAN.										
			REAL										
			PLAN.										
			REAL										
			PLAN.										
			REAL										
			PLAN.										
			REAL										

SEMANA			No. SEM.	RESPONSABLE	REPORTE DE AVANCE	
	DEL					NOMBRE DEL PROYECTO
	AL					CLAVE DEL PROYECTO

ACTIVIDAD		FECHA											
		PROGRAMADA				REAL				TEORICO AVANCE	REAL AVANCE	OBSERVACIONES
DESCRIPCION	DURACION	INICIO		TERMINA		INICIO		TERMINA				
		DIA	MES	DIA	MES	DIA	MES	DIA	MES			

BIBLIOGRAFIA

[1] Bersoff, E. H. (1984), Elements of Software Configuration Management, IEEE Trans. Software Eng., SE - 10(1), 79-87.

[2] Freedman, D.P. and G.M. Weinberg, Handbook of Walkthroughs, Inspections and Technical Reviews, Dorset House, 1990.

[3] Mark, N., Peter, R. and Malcolm, P., The Healthy Software Project, John Wiley & Sons.

[4] McCabe, T., A Software Complexity Measure. IEEE Trans Software Engineering, Vol. 2, No. 6.

[5] McCall, J.P. Richards and G. Walters, Factors in Software Quality, Three Volumenes, November 1977.

[6] Musa, J.D. and Ackerman, A.F., Quantifying Software Validation: The to Stop Testing, IEEE Software, May 1989.

[7] Myers, G., The Art of Software Testing, Wiley 1979.

[8] Pickard, M. M. (1993), Software Engineering Notes, Vol. 18, Número 3, Stephen F. Austin, State University.

[9] Pressman, R. S., Ingeniería del Software un Enfoque Practico, Mc Graw Hill, Tercer Edición.

[10] Rakos, J. J., Software Project Management for Small to Medium Sized Project, Prentice Hall, Englewood Cliffs, New Jersey 07632.

[11] Royce. W. W. (1970), Managing the Development of Large System, Proc. Westcon, Ca., USA.

[12] Sommerville, I., Software Engineering, Addison Wesley, Cuarta Edición.

Articulos:

[13] Administración Dinámica de Proyectos, Banco Nacional de México, (1993).

[14] Soluciones Avanzadas, Año 2, Número 8, "Una panorámica de la ingeniería de software".

[15] Soluciones Avanzadas, Año 4, Número 32, "ISO 9000 Una Visión General".