37

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

CAMPUS ARAGON

"FIABILIDAD DE SOFTWARE"

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

ISIDRO LOPEZ LOPEZ

ASESOR:
MAT. LUIS RAMIREZ FLORES

SAN JUAN DE ARAGON ESTADO DE MEXICO 2001

Agradecimientos

A DJYJIEAEOHVSSOEUVSA:

Aum mani padme hum
Aum Sri Hari Aum
Aum santi santi santi

A mis padres:

Por darme todo lo que hasta ahora me dan, por la educación y el amor, su ejemplo, sus defectos, y por aceptar mi forma de ser. (¡Los amo!)

A Rubén, Ara y Jorge:

Por las revisiones hechas a este trabajo, por su apoyo y comprensión. Por no quererme cambiar.

A Griselda (La octava):

Por salvaguardarme de la burocracia administrativa que esto conlleva, por ser como es conmigo, por su empuje, su amor, su entusiasmo por haberme conquistado de la forma sutil como lo hizo, por ser un estandarte de la incertidumbre.

Al Mat. Luis Ramírez Flores:

Por ser un modelo a seguir, por la forma de guiarme en este trabajo, por ser más un amigo que un profesor, por provocar la creatividad más que la memoria en sus alumnos.

A Fernando Cruz Valadez:
(Documentum's System Engineer for Latin America)
Por ser un gran contacto por parte de Documentum, por ser buen amigo y estar ayudándome constantemente.

A Ubaldo, Ale, Sra. Griselda, Leticia y anexas:
Gracias por los ánimos

...GRACIAS A TODOS

CONTENIDO

Introducción

Objetivo

El propósito de esta tesis es abordar de manera general un tema poco estudiado, como lo es la fiabilidad de software y la calidad del mismo. Entender tanto las diferencias como las similitudes de ambos conceptos. Dar un bosquejo de uno de los métodos que se pueden utilizar para construir un modelo de calidad del software. Describir de forma general en que consiste la Ingeniería de Fiabilidad del Software y como interacciona con la fiabilidad y la calidad de este. De igual forma, describir el proceso de Aseguramiento de Calidad que se lleva a cabo cuando se desarrolla el software Documentum

Justificación

El motivo para todo lo anterior es que estos conceptos y procedimientos no son bien conocidos y por consecuencia lógica, son poco aplicados. Otro punto es el comprender las diferentes acepciones de calidad en un producto de software y para enumerar los procesos de aseguramiento de calidad en un software ampliamente comercializado (Documentum).

Hipótesis

El problema de garantizar la fiabilidad está vinculado con todas las etapas de la creación del artículo o producto y todo el tiempo en que se usa. La fiabilidad del artículo o del producto se prevé durante su diseño y calculo (Hardware) o codificación (Software) y se asegura en su producción mediante la elección correcta de la tecnología de la elaboración, el control de calidad de los materiales iniciales (Hardware), productos semiacabados - módulos ya programados en el software - y producto terminado, control de regímenes y condiciones de elaboración.

El Control Total de Calidad y la Fiabilidad están estrechamente ligados. El primer concepto hace referencia al concepto de desarrollo de nuevos productos (inspección) y el segundo con el funcionamiento del producto (operación). Es decir, la fiabilidad forma parte del proceso de Calidad en la elaboración del software (pruebas / predicciones.)

Lo anterior deriva en lo que se llama Ingeniería de Fiabilidad de Software y que es distinta de la fiabilidad de software. Desde un punto de vista general, la ingeniería de fiabilidad de software está enfocada a satisfacer al usuario final (Calidad), y la fiabilidad del software al proceso de pruebas realizadas para la predicción del buen funcionamiento del software.

Sin duda, un gran problema que se presentó en la elaboración de este trabajo fue la falta de información documentada en México respecto al tema que nos ocupa. Y los datos existentes en Estados Unidos y Australia es de gran contenido matemático – por la teorías plantadas para predecir el funcionamiento del software -. Sin embargo, aquí no se pretende demostrar el origen de las formulas matemáticas sino su aplicación practica en el desarrollo de software cada vez más funcional y confiable.

El trabajo está compuesto por cinco capítulos, un glosario, un apéndice y un prefacio (Antecedentes). En el primer capítulo, se definirá la fiabilidad del software y sus características, tales como: complejidad del software, una breve analogía entre la fiabilidad en hardware y en software, las leyes del software, los errores del software y sus orígenes. De igual forma se manejan los conceptos de sistemas complejos y como están formados estos. Se hace una breve descripción de varios sistemas complejos tratando de hacer analogía con la complejidad del software.

En el segundo capítulo se describen brevemente varios modelos de fiabilidad de software, desde como se clasifican hasta ejemplos de varios modelos. Se mencionan las suposiciones que plantean cada uno de estos modelos para su aplicación en la fase de pruebas. Pero también se hace hincapié sus debilidades o fallas que presentan varios de ellos.

. En el tercero se da una sinopsis general de lo que es la Ingeniería de fiabilidad del software enfocándose en las pruebas y su estrecha relación con la fiabilidad del mismo. De igual forma se resaltaran los pasos a seguir en la aplicación de la ingeniería de fiabilidad del software y los resultados que se han obtenido en el ámbito empresarial

El capítulo 4 es una introducción de como se pueden elaborar los modelos de calidad para el software, los pasos que implica esto, tales como: Identificar para cuales aplicaciones es el modelo y dirigir las necesidades de los diferentes grupos interesados que usaran los modelos en las diferentes aplicaciones, lo mismo que una arquitectura o diseño para el modelo.

En el capítulo 5 se da un ejemplo de cómo una empresa (Documentum Corp.) aplica su modelo de aseguramiento de calidad en sus productos (que también puede ser llamado Ingeniería de fiabilidad de software) En este se hace un recorrido por el proceso de calidad el la empresa en la elaboración de software, comenzando desde su definición de calidad hasta los procedimientos estándares de operación que manejan.

En el Apéndice A se muestran en forma resumida los estándares de Iso-9126, lo mismo que un breve resumen de los estándares IEEE 982.1-1988 & IEEE 982.2:1988

Antecedentes

Fiabilidad

Es primario entender el cuadro de trabajo de la fiabilidad en forma general, y después abordar el tema particular de la fiabilidad del software.

El Problema de la fiabilidad y su importancia para la técnica moderna

La fiabilidad es uno de los problemas fundamentales de la ingeniería. Desde la aparición de la técnica los investigadores se han preocupado de la fiabilidad. Sin embargo en los últimos 30 años el problema de la fiabilidad de los sistemas técnicos y sus componentes se ha incrementado: Esto se debe a las siguientes causas:

- El aumento en la complejidad de los sistemas modernos que incluyen hasta 10^6 elementos individuales.
- La intensidad de los regímenes de trabajo o funcionamiento del sistema o sus partes individuales: a altas temperaturas, altas presiones, altas velocidades.
- La complejidad de las condiciones en las que se explota el sistema técnico, por ejemplo: alta o baja temperatura, alta humedad, vibración, aceleración y radiación, etc.
- Las exigencias a la alta calidad del trabajo del sistema: alta precisión, efectividad, etc.
- El aumento de la responsabilidad de la responsabilidad de las funciones cumplidas por el sistema; el alto valor técnico y económico de la interrupción.
- La automatización total o parcial de y la exclusión de la participación directa del hombre, cuando sus funciones las cumple el sistema técnico, la exclusión de la observación continua y el control por parte del hombre.

Una de las razones por las cuales se agudizo el enfoque en la fiabilidad es el crecimiento de la complejidad de los sistemas técnicos, como ejemplo: el sistema balístico de cohetes intercontinentales <Atlas> contiene cerca de 300,000 elementos mientras el sistema de comando del cohete <Nike> tiene más de 1.5×10^6 elementos individuales.

La intensidad de los regímenes de funcionamiento de sistemas técnicos se caracteriza por el empleo de vapor a altas temperaturas y presiones, por el uso de altas temperaturas en las cámaras de los motores a reacción, por la utilización de altas velocidades en los motores, y otros elementos y aparatos.

La complejidad de las condiciones en al que se utilizan los sistemas técnicos modernos se caracteriza por el trabajo en una amplia gama de temperaturas variables –70 a +70° C, la existencia del vacío, alta humedad (98-100%) vibraciones de gran amplitud y espectro de frecuencias ancho, la existencia de una alta radiación solar y cósmica.

Esto da lugar a que las probabilidades de aparición de fallos e pueda aumentar de 500 a 1000 veces en comparación con la probabilidad de fallos al trabajar los sistemas en condiciones de laboratorio.

La complejidad de los equipos y las difíciles condiciones de explotación dificultan el control del buen estado de los aparatos que componen el sistema técnico moderno, lo que no permite descubrir a tiempo los procesos, que dan lugar a fallas y prever su aparición.

La responsabilidad de las funciones a cumplir por los sistemas técnicos modernos está relacionado con que su fallo, da lugar a grandes pérdidas técnicas y económicas.

Aseguramiento de la fiabilidad

El problema de garantizar la fiabilidad está vinculado con todas las etapas de la creación del artículo y todo el periodo de su empleo. La fiabilidad del artículo o del producto se prevé durante su diseño y calculo y se asegura en su producción mediante la elección correcta de la tecnología de la elaboración, el control de calidad de los materiales iniciales, productos semiacabados y producto terminado, control de regímenes y condiciones de elaboración.

La fiabilidad se conserva utilizando métodos correctos de almacenamiento de artículos y se mantiene con su correcta explotación, el entrenamiento sistemático, el control profiláctico y la reparación. Al diseñar el articulo debe de tenerse en cuenta los siguientes factores:

1) La calidad de los componentes y elementos a utilizar. La elección de los componentes, y elementos debe de realizarse teniendo en cuenta las condiciones trabajo del artículo (climáticas y de producción) Los elementos deben de satisfacer los requisitos según sus propiedades funcionales y características, poseer las resistencias mecánicas y térmicas y la rigidez eléctrica necesaria, la precisión exigida y la fiabilidad en las condiciones dadas de explotación. Hay que tratar de utilizar aquellos componentes y elementos que entran en el esquema además la construcción de artículos que han demostrado en los casos análogos, al producto que se construye, mejores resultados.

La elaboración de artículos y sistemas complejos ha demostrado que al utilizar componentes, piezas, unidades y elementos unificados aumenta de forma notable la fiabilidad del producto (sistema). Esto se debe a que los elementos unificados están mejor trabajados con respecto al esquema y la construcción y tienen una tecnología de elaboración estable y bien controlada.

Actualmente se difunde ampliamente el principio de construcción de modulo-bloques (en conjunto) de circuitos y construcciones de artículos complejos. Un articulo (sistema) complejo se compone de elementos funcionales, constructivamente presentados en forma de módulos o bloques tipos, normalizados. La estandarización de señales de entrada y salida, de parámetros de fuentes de alimentación, de las dimensiones exteriores y de unión garantiza su trabajo coordinado en el artículo.

2) Los regímenes de trabajo de componentes y elementos. Esto debe de corresponder a sus posibilidades físicas. El empleo de componentes y piezas en regímenes imprevistos para su uso es una de las fuentes fundamentales de los fallos.

No hay que permitir regímenes más pesados que los indicados en la documentación técnica oficial de los componentes, piezas o elementos y aparatos elegidos al construir o diseñar el artículo prefijado. También es importante la solución esquemática y el diseño del artículo en conjunto. La existencia de procesos transitorios en el esquema, en distintos momentos de su funcionamiento, puede dar lugar a la aparición de factores suplementarios que conducen a fallas. A las distintas variantes de distribución de los componentes piezas y elementos dentro del artículo les corresponderán diferente microclima y distintas acciones de la vibración y la radiación, con respecto a la magnitud.

Por lo tanto, la elección y el uso correcto de componentes y elementos de esquemas y detalles de construcción, la elaboración minuciosa del esquema y su composición, así como la construcción del artículo son condiciones importantes para el logro de su alta fiabilidad

3) La accesibilidad de todas las partes del artículo y a los componentes, piezas, unidades, bloques y elementos que lo integran para el examen, el control y la reparación o la sustitución. Esto es una condición importante para el mantenimiento de la fiabilidad durante la explotación. Actualmente el principio de construcción en modulo-bloques ampliamente difundido permite sustituir fácilmente elementos individuales, conservando la capacidad de trabajo general del artículo (sistema). El fácil acceso a los aparatos, elementos, unidades piezas de construcción y componentes de esquemas (circuitos) para el examen alivia la

explotación del sistema en total y asegura la rápida restitución de su capacidad de trabajo después de aparecer el fallo.

En el caso de artículos y sistemas complejos utilizan los dispositivos de control automático del buen estado del sistema. Tales dispositivos pueden utilizarse para verificar el buen estado del sistema antes de iniciar su trabajo, o para el control automático continuo y la indicación del buen estado de los aparatos del artículo durante su trabajo. La existencia de estos dispositivos, que permiten, al personal juzgar de manera objetiva la capacidad de trabajo del artículo, tienen un gran valor para su empleo efectivo;

4) Los dispositivos de protección. Al diseñar los sistemas para la regulación y el mando automático es necesaria tal formación de circuitos y construcciones de manera que el fallo en el trabajo de un elemento, unidad aparato no de lugar al estado de avería de toda la instalación. Si esto no se logra al formar el esquema (circuito) fundamental o la construcción del artículo, es necesario incluir elementos especiales o dispositivos de protección que permitan evitar el desarrollo de una situación de avería (por ejemplo el paso a un régimen de trabajo más ordinario, la conexión del sistema de mando de reserva, etc.).

Durante la producción de artículos (sistemas) debe de observarse una serie de condiciones, vinculadas con el mantenimiento de la disciplina tecnológica y la observación de la constancia de los procesos tecnológicos en la elaboración de artículos:

- El debido control de la calidad, es decir, las propiedades físico-químicas, las características y los parámetros de los materiales y artículos complementarios(Productos semiacabados, componentes de circuitos, piezas, etc.) recibidos de empresas contiguas
- La no-admisión de la alteración de la calidad de los materiales o la sustitución de los artículos complementarios de baja calidad.
- La no-admisión del empleo de artículos complementarios que han sido almacenados o transportados en condiciones desfavorables.
- El cuidado de la limpieza de la instalación, del lugar de trabajo, de las normas sanitarias del trabajo necesarias.
- La prohibición de la alteración de los regímenes en los procesos tecnológicos complejos.
- La prohibición de la alteración del ensamblaje tecnológico y las reglas del montaje eléctrico.
- El debido control por operaciones y la salida del producto acabado.
- La verificación periódica de la calidad y la fiabilidad de la producción terminada.

Los factores principales que influyen en la fiabilidad de los artículos durante la explotación son:

- Las condiciones de explotación: climatológicas y de producción. La acción de las altas o bajas temperaturas del medio ambiente; las grandes oscilaciones temporarias y diarias de temperatura y humedad; la alta humedad, la niebla, la lluvia, etc. ejercen una gran influencia en la fiabilidad de aparatos que trabajan fuera de locales. No menor es la acción de las altas temperaturas, su brusca variación, la existencia de humedad y diferentes impurezas agresivas en el aire, al utilizarlas en los locales de los talleres de fábricas metalúrgicas y químicas. La ubicación de aparatos cerca de grandes equipos y fuentes de potencia o cerca de grandes máquinas está vinculada con la acción sobre ellos de oscilaciones mecánicas y, frecuentemente, también acústicas. Esto da lugar a envejecimiento acelerado de los materiales y la aparición de fallas. Si los aparatos se instalan en sistemas móviles: barcos, trenes, automóviles, aviones, cohetes, a la acción de los factores climáticos se agrega el efecto de las vibraciones y aceleraciones.
- Un sistema de servicio, minuciosamente concebido tiene gran importancia para conservar la fiabilidad de los artículos (aparatos). El entretenimiento organizado de los aparatos, el examen y control preventivo periódico, la limpieza y el reglaje establecido conforma al reglamento, la reparación y la sustitución de las piezas y elementos desgastados, cuyas características hayan indicado desviaciones de la norma durante el control de turno, permiten evitar los fallos y prolongar el tiempo de servicio del artículo. Conviene hacer notar que la creación de un sistema de servicio correcto de sistemas técnicos complejos modernos frecuentemente requiere grandes investigaciones preliminares, dando lugar a la aparición de una nueva orientación científica vinculada con la elaboración de fundamentos teóricos y métodos técnicos de organización de un servicio óptimo.
- La calificación y la responsabilidad del personal de servicio tiene un gran valor para alcanzar la fiabilidad, la duración y la efectividad del trabajo del artículo. La fiabilidad del funcionamiento de aparatos de un mismo tipo se diferenciará sensiblemente si el personal de servicio no tiene igual preparación, o bien distinto grado de responsabilidad por el buen estado de los aparatos y el cumplimiento de las funciones prefijadas a él. La experiencia demuestra que el cambio frecuente del personal disminuye la responsabilidad y, por lo lado, obstaculiza el completo dominio de los aparatos. Para un estudio profundo y el dominio de artículos complejos modernos requiere un tiempo considerable de trabajo práctico, durante el cual se adquieren las prácticas necesarias para la realización cualitativa de los trabajos preventivos, y el reglaje y la regulación rápida y correcta de los aparatos, en la búsqueda y eliminación de fallos y defectos sencillos, la sustitución de piezas de rápido desgaste.

Es interesante valorar, aunque sea sólo una aproximación, el papel de los distintos factores en la fiabilidad de los artículos. El estudio de las causas de los fallos y defectos de aparatos radio electrónicos demuestran que aproximadamente del 40- 45% de la cantidad total de fallos provienen de los errores cometidos al diseñar, el 20% de los errores cometidos en la producción, el 30%, de las condiciones de explotación y regímenes incorrectos de utilización o el servicio incorrecto y cerca del 5-7% del desgaste natural y el envejecimiento.

Medios de elevación de la fiabilidad

El desarrollo ulterior de la técnica exige un aumento considerable de la fiabilidad de los artículos, lo que puede lograrse mediante el desarrollo de los fundamentos teóricos del diseño de los artículos a fin de asegurar los requisitos prefijados a la fiabilidad y duración y adoptar una serie de medidas de perfeccionamiento de los métodos de diseño, producción y utilización de los artículos. En la esfera del diseño es necesario:

- Conocer la física del trabajo (funcionamiento) del artículo.
- Conocer la física de los fallos.
- Utilizar materiales, productos semiacabados, elementos complementarios (productos) de alta calidad, estimular la elaboración de nuevos materiales y productos semiacabados de calidad aún más alta.
- Tener todos los datos necesarios sobre las propiedades, características y parámetros de los materiales, producto semiacabado y elementos complementarios (artículos) a fin de elegir correctamente sus regímenes y condiciones de utilizaciones.
- Crear nuevos circuitos en bloque, aparatos y sistemas de alta fiabilidad teniendo en cuenta los regímenes y las condiciones de trabajo.
- Crear construcciones seguras de los artículos teniendo en cuenta las condicio0nes de utilización, el lugar de instalación en el sistema, la organización del servicio.
- Utilizar ampliamente piezas y unidades unificadas de alta calidad.
- Emplear los principios de construcción en módulo-bloque.
- Realizar el análisis y el cálculo de las características funcionales, los cálculos de fiabilidad con respecto a tipos fundamentales de fallos de circuitos y construcciones de todos los elementos, aparatos y dispositivos fundamentales, así como de todo artículo.

En el campo de la producción es necesario:

- ✓ El estricto control de calidad de los materiales, productos semiacabados y artículos complementarios de entrada.
- ✓ El uso de los métodos tecnológicos modernos y de una instalación tecnológica perfeccionada.
- ✓ El aseguramiento de la limpieza y el confort en los locales de producción, el estricto control de las operaciones tecnológicas, el control de la calidad de funcionamiento de la instalación tecnológica.
- ✓ El control de la calidad del artículo que se elabora después de cada etapa fundamental de producción.
- ✓ El control total de las propiedades., características y parámetros de todo artículo después de su producción.
- ✓ La utilización de métodos modernos de equipamiento para el almacenaje y el transporte del artículo.

En la esfera de la explotación es necesario:

- ❖ Utilizar instrucciones y métodos de explotación minuciosamente elaborados y fundamentados, así como la profiláctica y reparación del artículo.
- ❖ Emplear sólo personal de servicio especializado, completamente instruido(después de un período de practica y rendir los exámenes) para el servicio del tipo dado en el artículo.
- ❖ Establecer correctamente los derechos, obligaciones y responsabilidades del personal de servicio.
- ❖ Organizar en las instalaciones la reunión de datos estadísticos completos y fehacientes sobre los fallos y paradas de los aparatos.
- ❖ Analizar periódicamente los datos, elaborar recomendaciones para mejorar la explotación y perfeccionar la construcción y la tecnología de elaboración de los artículos

Organizar para los tipos de artículos completamente nuevos la explotación experimental en las condiciones de la instalación o del establecimiento con participación de los diseñadores y productores de los artículos.

Capítulo I
Fiabilidad de Software

1.1 Definición de Fiabilidad del Software

A un operador de sistemas le importa si su sistema falla ya sea por motivos de software o hardware, ya que ambos tipos de fallas reducen el valor operacional del sistema. Su interés principal es incrementar el valor del mismo. Para proporcionar al operador con un sistema capaz de reunir los requerimientos de eficacia operacional, el programador de sistemas, por ejemplo, el de una oficina de programación de sistemas, necesita procedimientos prácticos para cuantificar, especificar, predecir, medir, la fiabilidad y mantenimiento del software(F/M). (Reliability and Maintainability R/M)

El estado actual de la F/M del software puede resumirse de la siguiente manera:

- Hay desacuerdo en las definiciones básicas
- Los métodos para la especificación cuantitativa no son usados o no están disponibles
- Un gran número de modelos de predicción de fiabilidad han sido propuestos; Ninguno ha sido adecuadamente comprobado
- Los procedimientos de comprobación no están disponibles
- Algunos procedimientos básicos de diseño están disponibles, por ejemplo: el diseño "Top Down", programación estructurada, etc.

Hay varios puntos de vista encontrados al respecto de qué es y como se debe cuantificar la fiabilidad de software. El conflicto aparece por el desacuerdo en la definición de lo que la "fiabilidad en el software". La fiabilidad de software como es visto por varias personas, en especial los puristas de la computación, puede ser estrictamente ligado a lo correcto del software. Ellos argumentan que un software incorrecto (que siga conteniendo errores) esta destinado a fallar tarde o temprano y por lo tanto su fiabilidad debería de ser cero (0). Una vez que el software ha sido liberado de todo error, su fiabilidad es uno (1).

Por otro lado, la fiabilidad de software como es vista por muchos ingenieros, estadísticos y practicantes, esta estrechamente ligada a la "probabilidad de fiabilidad". Estos grupos de personas argumentan que se sabe que muchos de los programas que son usados en el mundo real siguen conteniendo errores y son ejecutados día tras día sin que aparezca ninguna falla. Según ellos la fiabilidad de software debe de ser vista como la probabilidad de que un software operará sin fallas durante un tiempo determinado - Misión -. (Mision)

Una forma de resolver este conflicto es mirar hacia atrás, al problema del mundo real, y preguntarse: ¿Por qué necesitamos saber acerca de la fiabilidad en software?

El problema que se origina en el mundo real es, en términos muy simples, como sigue:

Desarrollar software que satisfaga los requerimientos del usuario del modo más eficiente posible (en el sentido de tiempo y dinero).

La solución de este problema se hace debido a que:

1. El software para el mundo real es caro y complejo
2. Los usuarios no siempre están 100% seguros de sus requerimientos
3. Los recursos (tiempo y dinero) para el desarrollo de software están siempre limitados

Aún cuando sepamos que sólo necesitamos ejecutar 2000 pruebas para sacar o exponer todos los posibles errores contenidos en un software, esas oportunidades son las que tenemos, pero en el mundo real no tendríamos ni el tiempo ni el dinero suficiente para realizar esas pruebas. Como más y más errores son descubiertos por las pruebas o por el proceso de verificación, los costos adicionales de eliminar los errores restantes crece muy rápido. Por lo tanto hay un punto donde resulta prácticamente inútil seguir con las pruebas para obtener un software 100% libre de errores. Esto explica por que el software que ha sido liberado para su uso privado y publico sigue conteniendo errores.

Si adoptamos el punto de vista de los puristas de la computación entonces casi todo el software liberado hasta la fecha (incluyendo los programas que han sido aceptados por los usuarios como muy confiables y de gran utilidad) tienen una fiabilidad cero. Y como ahora cualquier software tiene fiabilidad cero, el valor o la utilidad del concepto de fiabilidad de software está perdido.

La razón por la cual la gente invento el concepto de fiabilidad de software (o de hardware según sea el caso) es para tener una medida útil que nos ayude a tratar con el problema original de software (hardware) en el mundo real. Esta medida de fiabilidad es útil en la planeación de recursos adicionales (tiempo y dinero) para maximizar la fiabilidad del software (hardware) dentro de los recursos asignados. También es una medida útil para proporcionar al usuario seguridad acerca de la calidad del software del producto liberado.

Todas las herramientas y técnicas desarrolladas en la Ingeniería de Fiabilidad de Hardware están realmente dirigidas a resolver los problemas básicos de:

- Falta de atención a los detalles (disciplina)
- Manejar incertidumbres

Los mismos problemas básicos existen en el área del software, entonces se puede esperar una fuerte conexión entre las técnicas probadas en hardware y las nacientes técnicas en el software. Admitiendo, por supuesto, que hay diferencias entre el hardware y el software.

Pero en lugar de tratar con las diferencias nos enfocaremos en las similitudes, algunas de las cuales son:

a) La fiabilidad de hardware es una función de equipo complejo; intuitivamente uno esperaría lo mismo en el software, aunque una medida de complejidad esta por ser encontrada.

b) Los dispositivos electrónicos de estado sólido por ejemplo: transistores, microcircuitos, si son fabricados correctamente, sin tener mecanismos desgastados, se les puede ver funcionando durante largos periodos de tiempo. Los defectos que son causa de fallas (otros aparte del mal uso del dispositivo) son creados durante la fase inicial de la fabricación del dispositivo; lo mismo es cierto para el software.

c) La fiabilidad del hardware puede mejorare por el crecimiento en las pruebas de fiabilidad, por ejemplo el programa: "Prueba - Análisis - Reparación", para describir, identificar y corregir los mecanismos y vías de posibles errores , lo que causaría una falla temprana del equipo. Esto es similar a encontrar y eliminar errores (bugs) en el software, esto aumenta su fiabilidad.

Por lo tanto nos encontramos en la dualidad que existe entre la propuesta exitosa de la fiabilidad en el hardware y la naciente propuesta de fiabilidad en software. Una vez que esto se acepta, el problema se simplifica, porque los problemas del hardware y del software son planteados en forma conjunta en un contexto de sistema total.

La dualidad entre la fiabilidad del hardware y el software es descrita gráficamente en la figura 1.1, la cual ilustra los elementos claves de los programas de hardware y software durante las fases del ciclo de vida del desarrollo del sistema. Las diferencias básicas ocurren durante el desarrollo de ingeniería a gran escala, cuando el hardware es fabricado y probado, mientras el software es codificado (programado) y depurado.

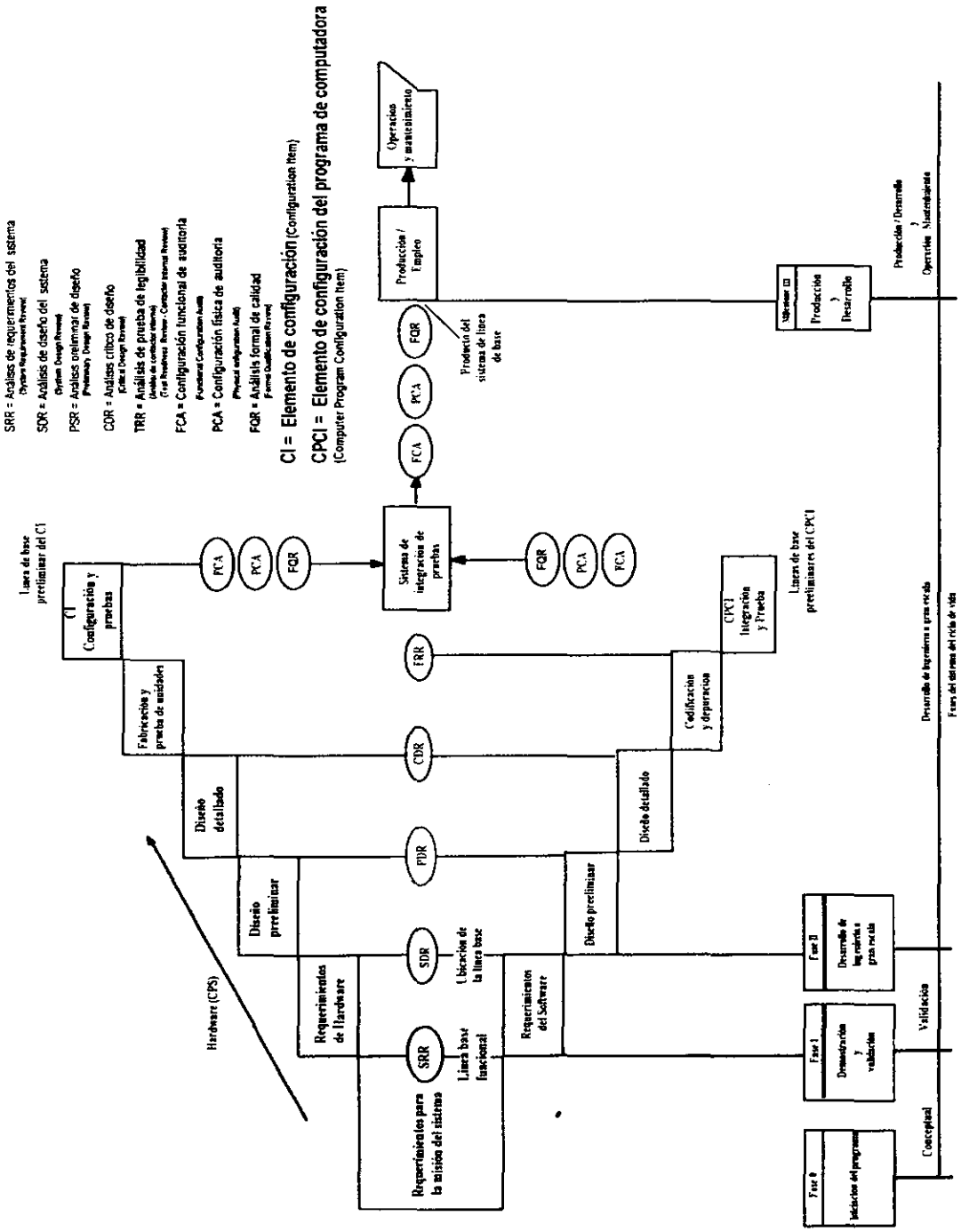


Figura 1.1 Relación del ciclo de vida del hardware y el software

1.2 La complejidad del software

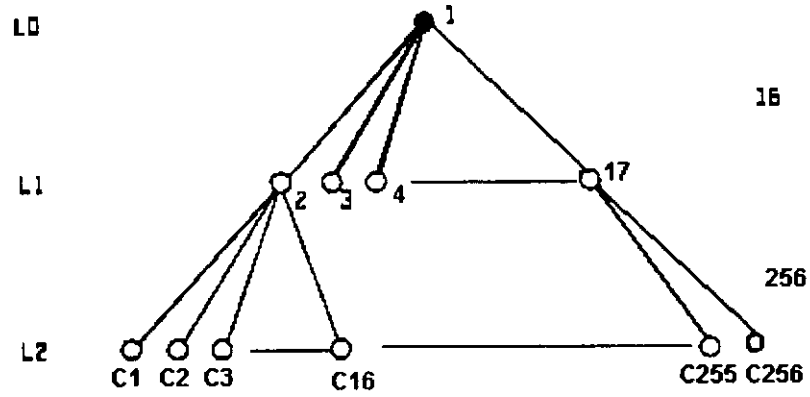
El problema básico el software, es el manejo de la complejidad, esto es bien descrito en el siguiente párrafo:

"Hemos aprendido de nuestra experiencia que: construyendo y administrando organizaciones complicadas, donde la complejidad de cualquier nivel crece demasiado en cierto rango, las funciones se deterioran, las operaciones se vuelven poco eficientes y la fiabilidad decae. Sabemos que las correcciones convenientes y las mejoras locales en eficiencia sólo pueden alargar las correcciones de los problemas y tarde o temprano tendremos que encarar una reorganización total del sistema que esencialmente debe alterar el control jerárquico y los niveles de estructura."[1]

La idea de la jerarquía es más desarrollada en la siguiente cita y es aplicada directamente al software:

" Ahora el efecto de esta relación, por ejemplo la jerarquía, es profunda cuando consideramos el diseño y la fiabilidad del sistema. Se dice que el tiempo de desarrollo del sistema es proporcional al número de niveles usados en la jerarquía para estructurar el diseño. Por ejemplo, si un diseñador tenía que construir un sistema el cual requiere 256 elementos, puede construir subconjuntos de dieciséis componentes y usando dos niveles en la jerarquía; ó puede usar cuatro componentes por subconjunto y usar cuatro niveles. El encontrará que la segunda estructura tomará solo la mitad del tiempo en diseñarla en comparación con la primera estructura, pero hay que notar el número de especificaciones que requerirá. En la figura 1.2, la primera estructura usa cuatro componentes por subconjunto y solo requiere de 17 representaciones (o trazos) de la relación de los 16 componentes para formar un subconjunto, mientras que la segunda estructura requiere 85 representaciones o trazos, describiendo una amplia relación de 4 componentes requerida para formar un subconjunto. Esto puede explicar porque nuestra intuición nos falla y escogemos el escribir las 17 representaciones; y entonces lo escribimos una y otra vez, en lugar de realizar los 85 trazos los cuales son 4 veces más pequeños. Si asumimos que el tiempo proporcional al diseño es válido, entonces el nivel de esfuerzo para cada uno de las 17 trazos sería 10 veces más grande que cualquiera de los 85 trazos usados en la primera estructura.

De forma similar la prueba y revisión de cada subconjunto es 10 veces más compleja en nuestra suposición intuitiva de 4. Esta es la razón por la que las pruebas son menospreciadas para sistemas que no están estructurados. El problema es que nosotros no estructuramos nuestros sistemas en modelos lo suficientemente pequeños."[2]



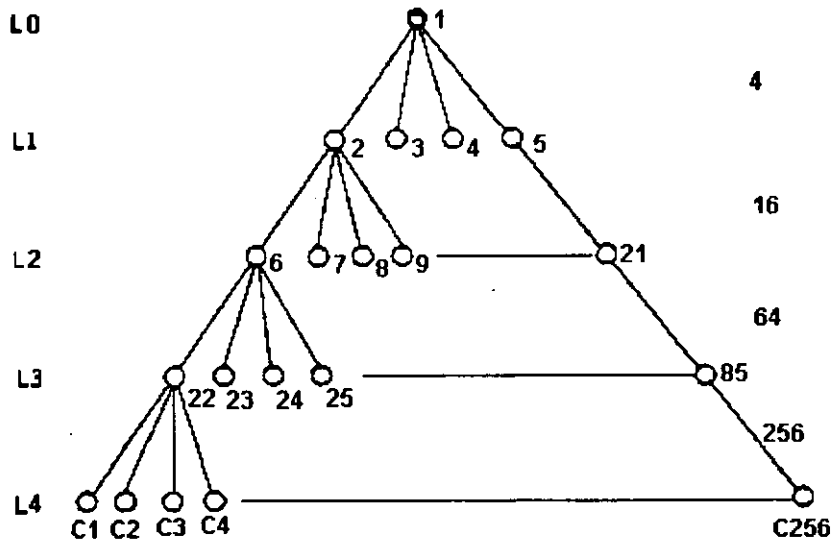


Figura 1.2 Niveles usados en la jerarquía del modelo

1.2.1 La complejidad inherente al software

Propiedades del software

Ya se sabe que algunos sistemas de software no son complejos. Son las especificaciones que terminan en aplicaciones intrascendentes, que son construidas mantenidas y utilizadas por la misma persona. Esto no significa que estos sistemas sean burdos o poco elegantes, ni se pretende minimizar el esfuerzo de sus creadores; pero tales sistemas tienen un propósito un limitado y un ciclo de vida muy corto. En estos casos es recomendable hacerse de un sistema nuevo que tratar de actualizarlo y mantenerlo para extender su funcionalidad.

La tendencia apunta hacia los desafíos que nos plantea la construcción de software de *magnitud industrial.* Aquí se encuentran aplicaciones que exhiben un conjunto muy rico de comportamientos, como el de sistemas reactivos que dirigen o son dirigidos por evento del mundo físico. Y para los cuales el tiempo y el espacio son recursos limitados; aplicaciones que sostienen cientos de miles de registros de información mientras permiten actualizaciones y consultas concurrentes; y sistemas para la gestión y controles de las entidades del mundo real tales como los controladores aéreos. Los sistemas de software de este tipo tienden a tener un largo ciclo de vida y al paso del tiempo los usuarios llegan a depender de su buen funcionamiento. En el mundo del software de magnitud industrial se encuentran también marcos estructurales que simplifican la creación de aplicaciones orientadas a un dominio específico y programas que representan un aspecto de la inteligencia humana. Aunque tales aplicaciones son generalmente para investigaciones no son menos complejas por que sirven como medio para y artefacto para un desarrollo exploratorio.

La característica distintiva del software de dimensión industrial es que resulta sumamente difícil, casi imposible, entender para el programador individual. La complejidad de los sistemas excede la capacidad humana y al parecer la complejidad de que se habla forma parte esencial de los software de gran tamaño. En esencia, se quiere dar a entender que se puede llegar a dominar esa complejidad pero nunca eliminarla.

El por qué el software es complejo de forma innata

"Se dice que la complejidad del software es una propiedad esencial no accidental."[3], Se observa que la complejidad inherente se deriva de cuatro elementos: la complejidad del dominio del problema, la dificultad de gestionar el proceso de desarrollo, la flexibilidad que se puede alcanzar a través del software y

los problemas que plantea la caracterización del comportamiento de sistemas discretos.

La complejidad del dominio del problema.

Los problemas que se intentan resolver con el software conllevan elementos de complejidad ineludible, en los cuales se encuentra una cantidad ingente de requisitos que compiten entre si, que en algunas ocasiones se contradicen. Se puede tomar por ejemplo los requerimientos para el sistema electrónico de un avión de varios motores, o un sistema para conmutación de sistemas celulares o un robot autónomo. La funcionalidad pura de tales sistemas es incluso difícil de entender y si se le agrega a esto los requerimientos no funcionales como la facilidad de uso, rendimiento, costo, capacidad de supervivencia y fiabilidad, que a menudo están implícitos.

Esta ilimitada complejidad externa es lo que hace que los errores sean una propiedad esencial y no accidental.

Ésta surge habitualmente de los problemas de comunicación que hay entre los usuarios y los desarrolladores de sistemas; los usuarios suelen encontrar grandes dificultades para expresar con precisión sus necesidades en una forma que los desarrolladores puedan comprender (Como se verá posteriormente forma parte de los orígenes de errores del software). En ciertas ocasiones ni el mismo usuario tiene la certeza de lo que realmente necesita en su sistema de software. Esto no es total responsabilidad del usuario ni de los desarrolladores del sistema, esto es debido a que ninguno de los dos grupos tiene un conocimiento profundo del dominio del otro grupo. Los usuarios y desarrolladores tiene puntos de vista diferente respecto a la naturaleza del problema y realizan distintas suposiciones sobre la posible solución. Aún cuando los usuarios tengan perfecto conocimiento de las necesidades de su sistema se disponen de pocos elementos para plantearlas con exactitud. La manera más habitual de expresar los requerimientos es mediante montañas de texto, acompañadas con una ilustraciones; y por consecuencia están abiertos a distintas interpretaciones y frecuentemente invaden el campo del diseño en lugar de limitarse a ser recursos esenciales.

Otro problema adicional es que las necesidades del sistema cambian durante su desarrollo, esencialmente porque el simple hecho de que exista un desarrollo de software altera las reglas del problema. La observación de productos en las primeras fases, como documentos de prototipos y diseño, y la posterior utilización de un sistema cuando ya este instalado y en operación ayudan al usuario a articular de una mejor forma sus necesidades reales. De igual forma que ayuda a los desarolladores a comprender el dominio del problema y a formar mejores elementos que llevaran a un mejor comportamiento del sistema. Ya que un sistema grande de software es una inversión considerable, no es admisible el desechar un sistema existente cada vez que cambien los requerimientos. Los sistemas grandes tienden a evolucionar con el tiempo, situación a la cual generalmente "se le llama erróneamente *mantenimiento del sistema* o del software. Siendo precisos, es mantenimiento cuando se componen errores; es *evolución* cuando se responden a

nuevos requerimientos del sistema; es *conservación* cuando se siguen empleando medios extraordinarios para mantener en operación un elemento de software anticuado y decadente."[4]

La dificultad de gestionar el proceso de desarrollo

La principal meta de los desarrolladores es dar la ilusión de simplicidad. Ciertamente un tamaño grande del software no es una de las mejores cualidades de un sistema. Se hace lo posible por escribir el menos código mediante mecanismos tan ingeniosos como potentes que nos dan la impresión de simplicidad. Así como la reutilización de marcos estructurales de diseño y códigos ya existentes. Sin embargo a veces es imposible evitar todos los nuevos requerimientos del sistema y se plantea la obligación o bien de escribir una gran cantidad de nuevo software o de utilizar de diferente forma el que ya está hecho. Hoy en día se suelen encontrar sistemas de cientos de miles de líneas e incluso de millones (en lenguajes de programación de alto nivel). Un hecho seguro es que individualmente nadie puede jactarse de comprenderlo completamente, aún si se subdivide, este nos resultará en cientos de miles de módulos separados. Este tipo de trabajos implica un número mayor de personas en el equipo de desarrolladores; pero lo ideal del equipo de desarrolladores es que sean los menos posibles, dado que a mayor número de elementos crecen los problemas de comunicación y una coordinación más difícil. Cuando se integra un equipo de desarrollares el punto clave es mantener una unidad e integridad en el diseño.

La flexibilidad que se puede alcanzar a través del software

El software ofrece una flexibilidad máxima por lo que un desarrollador puede manifestar casi cualquier clase de abstracción. Esta abstracción suele ser de las cosas que atraen al desarrollador a construir casi por si mismo todos los bloques en los cuales se basan sus abstracciones de más alto nivel. Como resultado de esto el software en uno de los productos más laboriosos de realizar.

Los problemas de caracterizar los sistemas discretos

Para comenzar a explicar esto, viene bien una alegoría: Si se lanza una pelota al aire, se puede predecir, de manera fiable su trayectoria por que se sabe, que en condiciones normales, hay ciertas leyes físicas aplicables. Sería extraordinario si, por haber lanzado la pelota más fuerte esta se detuviera a medio vuelo y saliera disparada un lado. En un sistema de software que no este depurado esto puede suceder con frecuencia. En una aplicación grande es común que haya cientos y miles de variables así como más de un flujo de control. El conjunto de esas variables, sus valores actuales, la dirección de ejecución y la pila actual de cada uno de los procesos del sistema indican el estado actual de la aplicación. Al ejecutarse un software en computadoras digitales se obtiene un sistema con estados discretos. Por su propia naturaleza los sistemas discretos tienen un número finito de estados posibles; en sistemas grandes hay una explosión combinatoria que hace este número gigante. Se intenta diseñar los sistemas de forma tal que el

comportamiento de una parte del sistema tenga un mínimo de influencia en otra parte del mismo sistema. Sin embargo sigue dándose el hecho de que las transacciones de fase entre estados discretos no pueden moldearse a funciones continuas. Todos lo eventos externos a un sistema de software tiene la posibilidad de llevar a ese sistema un nuevo estado, y aún más, la transacción de estado a estado no siempre es determinista. En las peores circunstancia un evento externo puede afectar el estado del sistema porque sus diseñadores olvidaron tener en cuenta ciertas interacciones entre eventos. En sistemas continuos este comportamiento sería imposible, pero en los sistemas discretos todos los eventos externos pueden afectar cualquier parte del estado interno del sistema.

Esta se podría (y generalmente se toma) como una de las principales causas por la cual se debe de probar un sistema. Para cualquier sistema que no sea trivial, es imposible hacer una prueba exhaustiva - más adelante se profundiza un poco más en este aspecto -. Ya que no se dispone de modelos matemáticos ni la capacidad intelectual para moldear el comportamiento de grandes sistemas discretos.

Las consecuencias de la complejidad ilimitada

Los usuarios del sistema no lo dudan cuando solicitan cambios en el software, dado que ellos argumentan que es simplemente cosa de programar. El fracaso en dominar la complejidad del software, lleva a proyectos retrasados, que exceden el presupuesto y que son deficientes frente a los requerimientos fijados. A menudo se le llama a esta situación Crisis de software - que por el hecho de haber existido tanto tiempo ya se le considera normal. Tristemente esta crisis se traduce en perdidas de Recursos Humanos y de oportunidades. Los desarrolladores de las empresas deben de estar, la mayoría de las veces, en el mantenimiento o la conservación del software viejo.

Dado que bajo el problema del software subyace su complejidad, vale la pena analizar varios de los sistemas complejos que nos rodean y poder comprender un poco más su organización, para que de esta forma su puedan o intenten moldear los sistemas del software.

1.2.2 Organización de los sistemas complejos

La organización de los sistemas complejos se puede comparar con varias estructuras:

La estructura de un ordenador: Una computadora es un elemento de complejidad media, la mayoría de estas se componen de elementos básicos similares: CPU, dispositivos de entrada y salida (monitor, teclado, etc.) y un dispositivo de almacenamiento secundario (generalmente la unidad de disco flexible y disco duro). Uno puede tomar cualquiera de estas partes y descomponerla más. Por ejemplo, el CPU, a grandes rasgos consta de la memoria principal, el ALU, y un BUS que se comunica con el exterior. Cada uno de estos todavía puede descomponerse en elementos más pequeños, el ALU pude dividirse en registros y lógica de control aleatorio, los cuales están constituidos por Compuertas lógicas NAND, etc.

Con este ejemplo se puede apreciar la naturaleza jerárquica de un sistema complejo. Una computadora funciona bien solo si cada uno de los elementos que la conforman trabajan en forma conjunta y adecuada. Todas estas partes juntas forman un todo lógico. Se puede entender en forma general el comportamiento de una computadora gracias a que se pueden dividir sus partes en entidades más pequeñas - se puede estudiar el funcionamiento del monitor por un lado y el del disco duro por otro. Los sistemas complejos no solo son jerárquicos, (mencionado y representado en un diagrama en la sección anterior La complejidad inherente al software) sino que los diferentes niveles jerárquicos representan diferente nivel de abstracción, cada uno de los cuales se construye sobre el otro y cada uno de lo cuales es comprensible por si mismo. A cada nivel de abstracción se encuentran dispositivos que realizan labores para ofrecerle servicio a las capas superiores. Se elige un determinado nivel de abstracción para satisfacer necesidades particulares, por ejemplo si se quiere resolver un problema de temporización de la memoria lo indicado sería examinar la computadora al nivel de compuertas lógicas, y si se quiere arreglar problemas en una hoja de cálculo este no es el nivel, de abstracción adecuado.

La organización compleja de plantas y animales: En botánica, los científicos buscan encontrar las similitudes y diferencias entre las plantas estudiando su morfología (su forma y estructura). Las plantas son complejos multicelulares y las actividades complejas que realizan, como la fotosíntesis y la transpiración son debido al trabajo conjunto de todos sus organismos.

La planta consta de tres organismos principales - raíces, tallos y hojas - y cada una de estas tiene su propia estructura. Las raíces cuentan con las raíces principales, pelos radicales, el ápice y la cofia. De forma análoga la sección

transversal de la hoja nos deja ver su epidermis, mesofilo y tejido vascular. Cada una de estas estructuras se compone a su vez de organizaciones más complejas: las células, las cuales a su vez contiene otro sistema complejo de organización. Similar al ejemplo del ordenador las partes que forman una planta forman una jerarquía y cada nivel de jerarquía conlleva su complejidad.

Todas las partes al mismo nivel de abstracción interactúan de una forma perfectamente definida. La abstracción de más alto nivel, las raíces, es responsable de absorber agua y minerales del suelo. Las raíces interactúan con los tallos, que transportan esas materia primas hasta las hojas. La hojas ocupan el agua y lo minerales que les proporciona el tallo para producir alimentos mediante la fotosíntesis. Siempre hay fronteras claras entre el exterior e interior de determinado nivel; Se puede afirmar que las partes de una hoja trabajan juntas para proporcionar el resultado como un todo. Son claros los diferentes intereses entre las partes a diferentes niveles de abstracción.

En el estudio de morfología de una planta no se encuentran partes individuales que sean responsables de cada una y de un único y pequeño paso en un solo proceso más grande, como el de la fotosíntesis. No hay partes centralizadas que coordinen directamente las actividades de las partes en niveles inferiores. En su lugar se encuentran partes separadas que actúan como agentes independientes, cada uno de los cuales desempeñan una actividad bastante compleja y contribuyen a muchas funciones de nivel superior. Solo a través de colecciones significativas de estos agentes se ve la funcionalidad a escala superior de la planta. La ciencia de la complejidad llama a esto comportamiento emergente: El comportamiento de todo es superior al comportamiento de sus partes

La estructura de las instituciones sociales

Los grupos de personas se reúnen para hacer cosas que un solo individuo no podría hacer. Algunas organizaciones son transitorias y otras perduran mucho tiempo. A medida que la compañía crece se va formando un jerarquía diferente. Las transnacionales contienen compañías que a su vez están divididas en regiones, estas en divisiones, y estas en delegaciones y oficinas locales. Si la compañía perdura las fronteras entre estas partes puede cambiar, a lo largo del tiempo puede emerger una nueva y más estable jerarquía. El grado de interactuación de los empleados dentro de una sola oficina es mayor que entre empleados de oficinas diferentes. Un funcionario encargado del correo no suele interactuar con el gerente de la empresa, pero lo hace a menudo con las personas que trabajan en recepción. Estos niveles están unificados por mecanismos comunes. Tanto el gerente como el encargado de correspondencia reciben su salario de la misma compañía financiera y ambos comparten una infraestructura común, como el sistema telefónico de la compañía para realizar sus cometidos.

1.2.3 Los cinco elementos de un sistema complejo

Se puede decir que hay cinco atributos comunes a un sistema complejo:

1-. Frecuentemente la complejidad toma forma jerárquica, por lo cual un sistema complejo se compone de susbsistemas relacionados que contienen a su vez sus propios subsistemas y así sucesivamente hasta alcanzar un nivel ínfimo de sus componente elementales.

El hecho de que muchos subsistemas complejos tengan una estructura jerárquica y que se pueda descomponer casi en su totalidad es un factor de ayuda importante, el cual nos ayuda para comprender, describir e incluso "ver" estos sistemas y sus partes. Prácticamente solo se pueden comprender aquellos sistemas que están estructurados en forma jerárquica. Es importante notar que la arquitectura de un sistema complejo, está en función de sus componentes tanto como la relación jerárquica de estos componentes. Haciendo hincapié que lo más importante no es agregar componentes de valor al sistema sino la relación entre esos componentes.

2.-La elección de qué componentes de un sistema son primitivos es relativamente arbitraria y queda en gran medida a juicio del observador

Lo que puede ser inútil para un programador puede ser de gran utilidad para el criterio de otro. Los sistemas que pueden dividirse en partes identificables se les conoce como *descomponibles* y los que sus partes no son totalmente independientes se les conoce como *casi descomponibles* y esto conduce a otro atributo común en todos los sistemas complejos:

3-. Los enlaces internos de los componentes suelen ser más fuertes que los enlaces entre componentes. Este hecho tiene el efecto de separar la dinámica de alta frecuencia de los componentes - que involucra la estructura interna de los mismos - de la dinámica de baja frecuencia - que involucra la interacción entre los componentes -.

Esta diferencia entre las interacciones intracomponentes y extracomponentes, proporciona una diferencia clara de intereses entre las diferentes partes del sistema, facilitando el estudio de cada parte relativamente aislada.

4-. Los sistemas jerárquicos generalmente están compuestos de unas cuantas clases diferentes de subsistemas en varias combinaciones y disposiciones.

Los sistemas complejos tienen patrones comunes. Estos patrones pueden llevar a la reutilización de componentes pequeños.

5-. Se encontrará invariablemente que un sistema complejo que funciona partió de un sistema simple que funcionaba... Un sistema complejo diseñado desde cero nunca funciona y no puede "parcharse" para lograra su objetivo, hay que volver a empezar desde cero, desde un sistema sencillo que funcione.

A medida que evoluciona el sistema, los objetos que en su tiempo se consideraron complejos se convierten en objetos primitivos sobre los cuales se construyen sistemas más complejos. Nunca se pueden crear objetos primitivos de forma correcta la primera vez: hay que evaluarlos en un contexto, y con el tiempo ir mejorándolos, cuando se aprende más sobre el comportamiento real del sistema.

1.2.4 Ejemplo de pruebas exhaustivas en el software

El manejo de la complejidad no es exclusivo del software. Este también es el corazón del problema de la fiabilidad del hardware. Por años ha sido de común conocimiento que la fiabilidad de hardware es una función de complejidad como se muestra en la expresión:

$$R = e^{-\sum_{i=L}^{n} \lambda i\, t}$$

donde

R = Es la probabilidad de operación sin falla en el tiempo t
λ = Rango de falla para cada componente individual

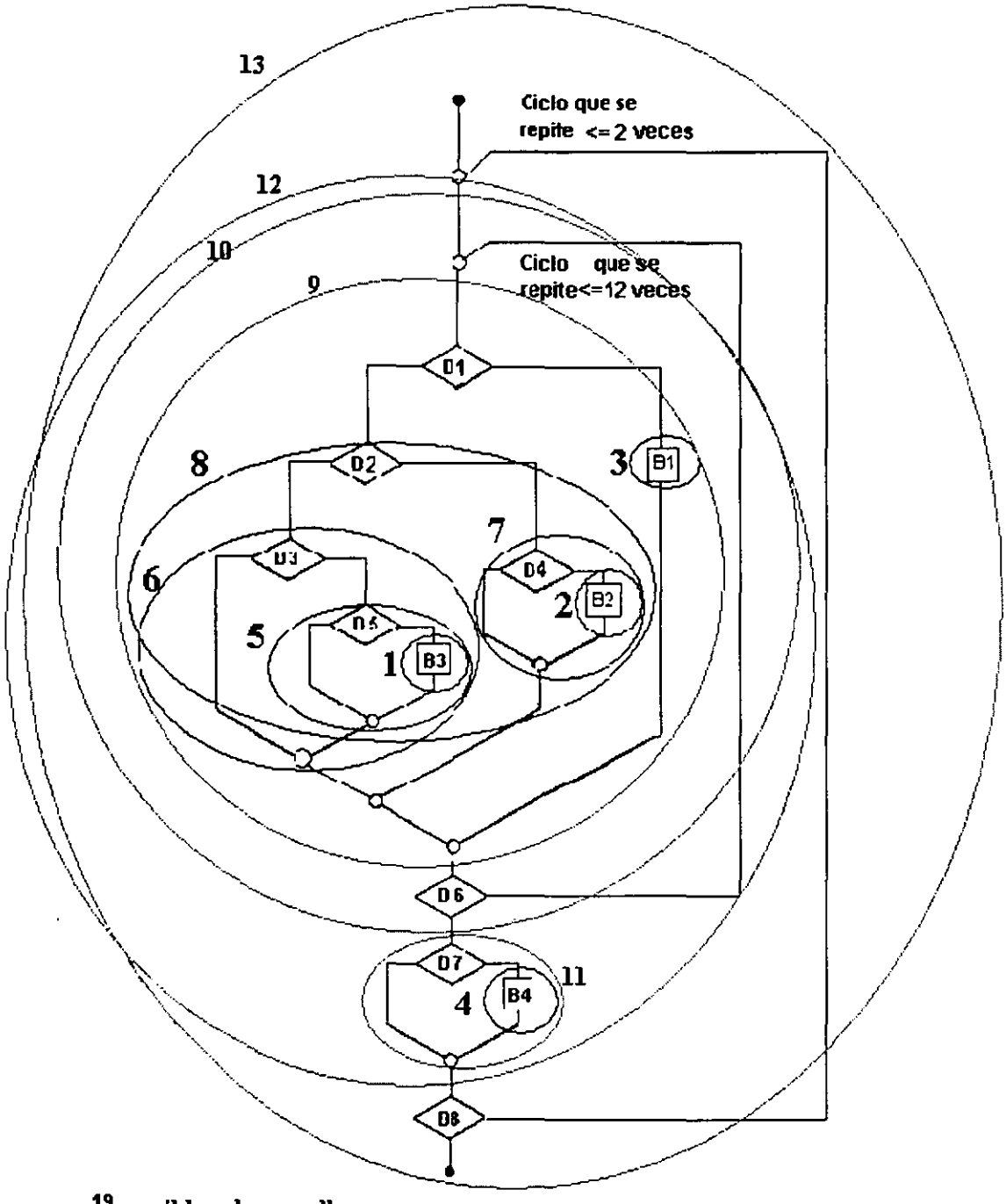
Por lo tanto , mientras mas componentes, más probabilidad de fallas.

El problema de la fiabilidad es que no tenemos la capacidad de derivar o sacar el "número de partes que lo componen". Otros aspectos del problema de la fiabilidad del software ha sido nuestra incapacidad para representar fácilmente el funcionamiento dinámico de un programa. Considere el diagrama de flujo de un programa (ver figura 1.3). Contiene cuatro bloques de código secuencial y cinco bloques de decisiones. También hay dos ciclos anidados, donde el ciclo interior puede ser ejecutado doce veces y el ciclo exterior dos veces.

Este módulo tiene 1.6 x 10 posibles maneras de cruzarse a través del diagrama de flujo, si se prueba una trayectoria cada nanosegundo nos tomaría más de quinientos años. Claramente probándolas solas no se prueba lo correcto de este módulo. La única manera en que se puede ganar seguridad en cuanto al comportamiento de este módulo es comprobar lo correcto de cada subestructura anidada. Como se puede ver en la figura hay trece bloques anidados, unos dentro de otros. En cada bloque se debe satisfacer que por todas las posibles entradas el bloque genere las salidas correctas.

Como cada bloque tiene una única entrada y una única salida, la salida de los bloques interiores sirve como un subconjunto de las salidas o entradas de los bloques externos. Por lo tanto solo trece conjuntos de pruebas deben de realizarse. Usando las técnicas de programación estructurada, el número y la dificultad de las pruebas disminuye drásticamente. Solo de esta forma se puede obtener la fiabilidad del software de una manera manejable.

Por otro lado este problema de las pruebas no es exclusivo del software, es también verdadero para el hardware. Si se tiene que probar cada trayectoria lógica posible para un estado de un microprocesador, con la finalidad descubrir lo posibles elementos defectuosos activos, asumiendo una taza de muestra de 10^{-6} segundos, se han hecho cálculos, nos tomaría 2^{17} años como se muestra en la figura 1.3



1.6×10^{19} posibles desarrollos

Probando una combinación cada 10^{-9} segundos, nos tomaría 500 años.

Figura 1.3: Diagrama de flujo de un programa

1.3 Otras características del software

El software se caracteriza por el hecho de que el producto completo es documentación (de hecho pudiera ser un solo texto). Parte de esta documentación es puramente explicativa, algo más es para beneficio del usuario, otro tanto para el apoyo a la persona que lo instala y ciertos aspectos son para el encargado de mantenimiento. La parte que se encuentra trabajando de cualquier producto basado en software (código) es también texto, su diferencia radica en el hecho de que está escrito de tal guisa que pueda ser utilizado para dar instrucciones a la máquina y que esta realice funciones especificas.

La aplicación para la cual se produce el software a menudo tiene un efecto significativo, sobre la estructura y organización del producto. Dos áreas de aplicación donde se caracteriza lo anterior son los sistemas de "Tiempo Real" y los sistemas de procesamiento de datos (como ejemplo del primer caso podremos utilizar los sistemas de Red y del segundo las bases de datos). A primera vista estas diferencias no tienen mucha importancia pero sucede que son unas de las características del software que a menudo se pasan por alto y que comprenden los datos y al programa, pero ninguno de las dos debe de ser ignorada.

En algunos casos puede ser difícil la distinción, cabe mencionar como ejemplo lo siguiente: En una base de datos, de grandes dimensiones ¿Los datos son el software? Por lo general la base de datos misma se considera como fuera del terreno del software, pero dentro del terreno computacional y de las formas sistemáticas para construir una relación de datos, mientras que los datos que definen una base de datos y su acceso son parte del software. El ignorar esto puede acarrear grandes conflictos ulteriores.

Una de las medidas que se pueden utilizar para el software son las líneas de código, dado que esta es de uso común, pero debe de considerarse esto con mucha cautela. Hay muchas formas de medir el software - tamaño, complejidad, utilidad - pero la relación entre estas y la fiabilidad del software es sutil. La información realmente efectiva se puede generar solo mediante el uso de fuentes de datos variables, por ejemplo: para establecer una medida significativa de mantenimiento, sería sensato registrar los atributos del software, como la complejidad, junto con procesos de medición, tales como informes críticos de datos. Dado que una relación de cualquiera de los dos últimos puede indicar donde se encuentra los módulos del programa en el sistema. Pero si se toman de forma independiente la información transmite cosas poco útiles.

Este es al caso de uno de los conceptos erróneos y más común en al área de software: que la productividad puede medirse por la cantidad de líneas de código que se producen por unidad de tiempo. Este enfoque puede condicionar el ambiente completo para concentrar la producción de código. El resultado, una y otra vez, es la producción de montañas de código que no pueden integrarse para trabajar en un desempeño eficiente, o la construcción de un sistema que no satisface las necesidades del cliente aún cuando pueda funcionar muy bien en el aspecto técnico.

Aunque continúe en etapa de inmadurez, la medición de software y el proceso de su producción es viable. En la practica son las medidas simples, como la medición del mantenimiento, las que proporcionan los datos necesarios para entender y mejorar la calidad.

1.4 Leyes del Software

No son leyes en toda la extensión de la palabra, son un conjunto de máximas reunidas en los últimos 30 años de experiencia que se aplican a todos los grandes proyectos del software[5]:

1) El software no se gasta. Aunque puede ser cierto en principio, de hecho no concuerda con la realidad. Muy pocos de los sistemas de software creados hace 20 o más años siguen en operación. Las líneas de código son muy fáciles de cambiar, pero los sistemas difícilmente se pueden alterar con éxito. Esto es por el hecho fundamental de que el software es un sistema con miles de dependencias entremezcladas y conectadas e interconectadas. Los requerimientos del cliente cambian cada momento, y como ocurre con los carros viejos, mantener el software viejo puede ser más caro.

2) El cambio es inevitable. El software debe de ser diseñado para darle mantenimiento. A fin de sobrevivir, debe ser construido pensando que continuará más allá del retiro de sus creadores, preparado para cuidar los casos especiales y, por lo general durar más que aquellos que lo crearon. Hay ejemplos contrarios obvios - situaciones en las cuales las aplicaciones son para equipo que pronto será obsoleto - pero es mejor equivocarse en una excesiva preparación para el futuro que muy poca.
   Una buena parte de la información acerca de los aspectos estructurales de un producto de software se pierde durante su desarrollo. Al inicio, esto no debe de suponer un problema, pero en algunas ocasiones es usual que sea precisamente esa información la que se necesita para conservar la integridad del

software durante la evolución (por lo común llamada adaptación y mejoramiento). Esto es similar a una organización que se divide en departamentos y después pierde toda la información sobre los empleados pertenecientes a cada departamento; la reorganización será muy difícil.

3) El mantenimiento mejora el software Cada vez que se hace un cambio en un producto de software, existe una posibilidad infinita de que se introduzcan nuevos errores. El efecto neto del mantenimiento debe ser mejorar el software, pero puede empeorarlo. Es más probable que lo segundo sea cierto si, por ejemplo se mantiene una documentación inadecuada o si no se ha implantado una administración de modificaciones. Se ha establecido que la información estructural (definición de subunidades semejantes a módulos y a las interfaces entre ellas) es esencial para el mantenimiento técnico y el control administrativo del de un sistema de software conforme cambia a través del tiempo. A pesar de una oleadas de técnicas que incorporan estructura a sus nombres (programación, diseño, análisis, prueba) siguen produciéndose software producido con poca atención a sus estructura física, y aún más encontrar que luego del desarrollo inicial no se mantiene un registro coherente de la estructura interna del sistema.

4) El costo real de la posesión del software es bajo. Si el costo de mantenimiento de un software no es explícito, entonces el costo por la posesión es bajo. Tomando el mantenimiento como todo el trabajo correctivo después de la entrega, el costo actual puede ser alto.
En primer lugar, crear software de cualquier magnitud cuesta bastante. Si se ha usado bastante, es muy probable que se haya gastado una considerable suma de dinero a lo largo de los años para repararlo, adaptarlo y agregarle nuevas funciones; no es extraño encontrar que ha gastado mas de tres veces el costo original del software.

5) El software nunca muere. Si nos basamos en una definición más apropiada de calidad, tal como satisfacer las necesidades percibidas de los clientes, entonces el software al que económicamente no se le puede dar mantenimiento para satisfacer esas necesidades se muere. Esta es una realidad que muchos individuos y organizaciones aprendieron hace muchos años. Aun por el largo promedio de experiencia de la gente del área parecen existir caídas (dado la vertiginosa expansión del uso de la computadora y de ahí que las personas se involucren en la creación del software).
El software vive para siempre por un cierto número de razones, buenas y malas. La razón más obvia y persuasiva es que una vez construido y en uso productivo, es por inercia que va a ser vencido si se modifica o se reemplaza. Esto a todos les agrada (jefes y usuarios). El software se ha convertido en parte de su vida y lo que ellos quieren es que sea arreglado un poco, pero no reemplazado o modificado en forma mayor.

6) El tamaño no importa. Así como se incrementa la complejidad de un proyecto de software, crece también la dificultad para lograr un producto satisfactorio. Existen límites (aunque arbitrarios) más allá de los cuales las técnicas convencionales

de desarrollo y administración tienen que llegar. Detrás de estos limites hay distintos puntos de ruptura en el tamaño del proyecto del software; a menudo en la práctica se da el caso de que duplicar el tamaño de un equipo de proyecto tiene poco efecto en la escala de tiempo. La relación entre el software y las personas que lo rodean (usuarios, operadores, etc.) tiene poca orientación con su estado presente, y lleva una gran cantidad de esfuerzo cambiar la situación y romper la relación con el software, por tanto el software vive y sigue viviendo.

1.5 Errores del software y sus orígenes

El software (también llamado programa) es, esencialmente, un instrumento para transformar un conjunto discreto de entradas en un conjunto discreto de salidas.

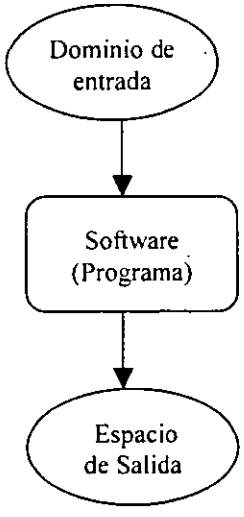


Figura 2.4 **Vista funcional del Software**

Consta de un conjunto de enunciados en código, de los cuales su función podría ser, en forma básica, una de las siguientes:

1. Evaluar una expresión y almacenar el valor en una locación ya sea permanente o temporalmente.
2. Decidir cuál es el próximo enunciado a ejecutarse.
3. Desarrollar operaciones de entradas y salidas.

Ya que la mayoría del software es producido por humanos, el producto final del software es a menudo imperfecto. Este es imperfecto en el sentido de la discrepancia que existe entre lo que el software puede hacer y contra lo que el usuario o ambiente computacional quiere que haga. El ambiente computacional se refiere a la máquina física, sistema operativo, compiladores y traductores, utilerías, etc. Las discrepancias son lo que llamamos **errores de software.**

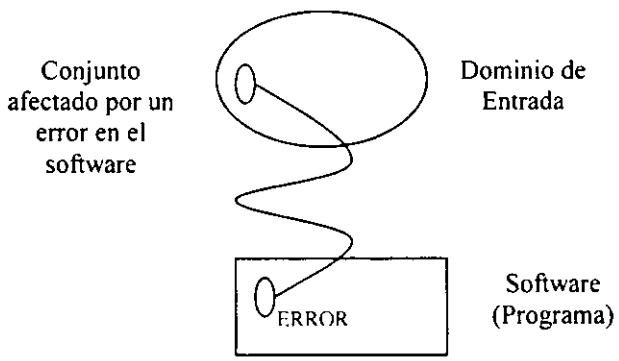


Figura 2.5 **Error de Software**

Los errores de software pueden ser atribuidos, básicamente a:

- Ignorancia de los requerimientos del usuario.
- Ignorancia de las reglas del ambiente computacional.
- Una pobre comunicación de los requerimientos del software entre el usuario y el programador o una pobre documentación por parte del programador.

Cabe destacar que aún sabiendo que el software contiene errores, no podemos saber con certeza la identidad exacta de esos errores. Hay dos formas que se usan generalmente para exponer los errores del software:

1) Comprobación del programa
2) Pruebas del programa

La comprobación del programa es más formal y matemática, mientras las pruebas del programa son más practicas y ayuda a descubrir (errores) en su método. El método de Comprobación del Programa es la construcción de una secuencia finita de declaraciones lógicas terminando en la declaración (usualmente la declaración que especifica la salida) a ser probada. Cada una de las declaraciones lógicas es un axioma o declaración derivada de declaraciones anteriores por la aplicación de la regla de inferencia. La utilización de las reglas de

inferencia en la comprobación del programa es conocida como *Método Inductivo de Inserción.* Otro trabajo en la comprobación del programa es el trabajo sobre el *Método Simbólico de Ejecución*. Este método es la base de algunos verificadores de programas automáticos. No obstante, del formalismo y la rigurosidad matemática de la comprobación del programa, sigue siendo una herramienta imperfecta, para la verificación de lo correcto del programa. Se ha demostrado que muchos programas. los cuales se comprobó que están correctos, seguían conteniendo fallas de software. Los errores fueron ocasionados por fallas en la definición de que se tenía que comprobar exactamente y donde, no fueron los mecanismos de las pruebas los que fallaron.

Las pruebas del programa es la ejecución simbólica o física de un conjunto de casos de prueba con la intención de exponer los errores contenidos (si estos existen) en el programa. Como la comprobación del programa, las pruebas de programa continúan como una herramienta imperfecta para verificar lo correcto del programa. Una estrategia de prueba dada es buena para exponer ciertos tipos de errores, pero no todos los posibles errores en un programa. Una ventaja de las pruebas de programa, es que proporcionan información exacta a cerca del comportamiento actual del programa en el medio ambiente computacional actual (en el cual se esta ejecutando); los datos que proporciona son limitados para proporcionar conclusiones acerca del comportamiento de un programa en un determinado (supuesto) medio ambiente.

Ni la comprobación ni las pruebas de programa pueden, en la práctica, garantizar completamente que un programa sea correcto. Cada uno tiene sus pros y sus contras. No pueden ser vistos como herramientas que compitan entre si. Son, de hecho son métodos complementarios para reducir la probabilidad de una falla en el programa.

1.5.1 Tipos de errores de software

Un estudio sistemático de los errores en un programa, requiere conocer que, específicamente, esos errores están y que herramientas usar para exponer ciertos tipos de errores de software. Los errores de software pueden ser agrupados en Errores de Sintaxis, Semántica, de Tiempo de Ejecución, y de Desempeño.

Errores de Sintaxis

Estos son ocasionados por diferencias entre el código del programa y las reglas de sintaxis que rigen al analizador o al analizador de léxico del programa traductor. Estos son lo errores más fáciles de detectar. Pueden ser detectados por un inspección visual del código o pueden ser detectados mecánicamente durante el proceso de compilación del programa. Los programadores expertos raramente comenten errores de sintaxis.

Errores de Semántica

Estos son debido a la discrepancia entre el código del programa y el analizador de semántica del medio ambiente computacional acepta. Entre los errores los tipos más comunes de errores de semántica están los "errores de dedo" (typechecking) y los de restricción de implantación. De nueva cuenta pueden ser detectados por medio de un inspección visual o por el analizador de semántica de un programa traductor.

Los errores de semántica y de sintaxis son detectados desde el periodo de compilación del programa. Un programa que contiene errores de semántica y/o sintaxis no puede ser ejecutado. Lo errores de semántica son debido principalmente a la ignorancia/negligencia por parte del programador acerca de las restricciones y limitaciones del lenguaje que use.

Errores de tiempo de ejecución

Como lo implica su nombre, los errores de tiempo de ejecución ocurren durante la actual ejecución del programa. Pueden ser más detalladamente descritos si se dividen en tres categorías:

Errores de Dominio

Un error de dominio ocurre siempre que el valor de una variable de un programa excede su rango declarado o excede el límite físico del hardware que representa y/o almacena la variable. El rango declarado de la variable puede ser hecho *implícita o explícitamente.*

Los errores de dominio son materia seria por:

1. La ejecución del programa es abortada
2. Los resultados del programa son inciertos

El aborto en la ejecución puede ser fatal, especialmente en un sistema de tiempo real. A pesar de su seriedad los errores de dominio nunca han sido formal y ampliamente estudiados en la literatura del software. Esto es porque el detectar lo errores de dominio puede ser muy difícil. Requiere de una exacta especificación de los rangos de las variables de entrada. Asimismo las pruebas de valor requeridas para exponer esos errores pueden ocurrir en el límite del dominio de la entrada o dentro del mismo dominio de la entrada.

Errores de "No-terminación" (Errores Infinitos)

Los errores de No-terminación simplemente es la falla de que un programa termine en un tiempo determinado sin intervención externa. La causa más común de los errores de No-terminación es cuando el programa entra en un ciclo infinito. Los errores de No-terminación también pueden ocurrir si un conjunto de programas concurrentes en un tiempo muerto.

Los ciclos infinitos son detectados por la simple ejecución de cada uno de los ciclos del programa. De cualquier manera, esta estrategia no puede garantizar la ausencia total de ciclos infinitos. Algunos ciclos infinitos solo pueden ocurrir cuando ciertas variables del programa obtienen ciertos valores. La comprobación del programa puede ser usada en ciertos programas para exponer ciclos infinitos. El problema de No-terminación de un programa, en general, sigue siendo un problema sin resolver.

Errores de Especificación

Se detectan los errores de especificación como:

- Especificaciones incompletas
- Especificaciones inconsistentes
- Especificaciones ambiguas

Permanece como un proceso informal. Esto debido principalmente a la inexistencia de un lenguaje de especificación lo suficientemente poderoso para trasladar los requerimientos del usuario en términos claros, completos y consistentes.

Una herramienta de prueba para detectar errores de especificación todavía esta por definirse.

Errores de desempeño

Los errores de desempeño existen siempre que exista una discrepancia entre el desarrollo actual (eficiencia) del programa y su desarrollo deseado o especificado. El desarrollo del programa puede ser medido en varias formas:

1. Tiempo de respuesta
2. Tiempo transcurrido
3. Espacio de memoria usado
4. Conjunto de requerimientos de trabajos

La forma de medir los anteriores parámetros de desempeño de programas puede ser muy difícil. Teorías complejas de programación tratan de estimar límites en el tiempo de ejecución de ciertos programas de algoritmos. Simulación y análisis estadísticos pueden también ser empleados para estimar el desarrollo de las variables antes dichas. De cualquier modo el uso de esas herramientas puede ser muy cara y consumir mucho tiempo.

Una herramienta de prueba de desempeño que sea económicamente (tiempo y dinero) para usarse esta por desarrollarse.

La clase de errores más cara de eliminar son aquellos que no son descubiertos hasta las ultimas partes del desarrollo de software, como cuando el software se vuelve operacional. Estos son conocidos como errores persistentes de software. Se dice que los errores persistentes son ocasionados en su mayor parte por la falla en la solución del problema (el programa) para aproximarse a resolver la complejidad del problema (requerimientos del usuario). Ejemplos de dichos errores son los computacionales ocasionados por la pérdida o insuficiencia de fundamentos y fallas para reiniciar una variable a un valor de referencia después que se usa en un segmento lógico funcional. La solución de este problema está lejos del mejor desarrollo o técnica que exista actualmente; de algún modo, la mente del programador, debe de extenderse para abarcar la complejidad lejos de su actual capacidad.

---

[1] Pattee, H. "Hierarchy Theory, The Challenge of complex system" (1973).
[2] Ronbeck, J. A. "SR-How it afects system Reliability" Canadian Reliability Symposium" (1975)
[3] Brooks, F. "No silver bullet: Essence and Accidents of Software engineering" (1987)
[4] Booch, G. "Análisis y Diseño Orientado a Objetos" (1998)
[5] Lehman M. "Programs, lifecycles and laws of software evolution." (1980)

Capítulo II
Modelos de Fiabilidad de Software

2.1 Modelos de fiabilidad de software

Se han emprendido muchos estudios en los últimos años para analizar y estudiar los datos de fallas de software con el objetivo de encontrar formas que guiarán a mejorar el desarrollo del mismo. Dichos estudios pueden ser calificados en una o dos categorías. En la primera categoría es el énfasis en el análisis de datos de fallas de software recolectados de proyectos largos o cortos durante la fase de desarrollo y operación.

Los estudios en la segunda categoría están principalmente dirigidos al desarrollo de modelos analíticos, los cuales son usados para obtener la fiabilidad y otras medidas cuantitativas en el desarrollo de software.

En los modelos de fiabilidad de software uno debe de tomar en cuenta los principales factores que inciden sobre ellos: la introducción de fallas, la remoción de fallas y el uso del software. La introducción de fallas depende principalmente de las características del producto y el proceso de desarrollo. La remoción de fallas depende del tiempo, del perfil operacional usado en las pruebas y la calidad de la actividad de eliminar los errores. El uso es caracterizado por el perfil operacional. Debido a que algunos de los factores antes mencionados son probabilísticos por naturaleza y operan en el tiempo, generalmente los modelos de fiabilidad de software se modelan sobre procesos aleatorios. La distinción de los modelos de fiabilidad se da, en términos generales, por la distribución probabilística de los tiempos de fallas o por el número de fallas experimentadas y por la naturaleza de la variación de los procesos aleatorios en el tiempo.

Un modelo de fiabilidad de software especifica la forma general de la dependencia de los procesos de fallas sobre los factores mencionados; Hemos asumido por definición que se basan en el tiempo. De cualquier forma las unidades naturales son usadas ampliamente en la práctica en lugar del tiempo, las dos son proporcionales si el uso del producto es constante. Por lo tanto, comúnmente se desarrollan en términos del tiempo, con el entendido de que las unidades naturales pueden ser sustituidas. Se puede determinar la forma del modelo estableciendo los valores de los parámetros del modelo a través de:

- **Estimación**: Aplicando procedimiento de inferencia estadística a los datos de fallas del sistema.
- **Predicción**: Determinando el valor desde las propiedades del producto y del proceso del desarrollo (Esto puede ser realizado antes de cualquier ejecución del programa)

Cabe aclarar que siempre hay cierta incertidumbre en la determinación de la forma. Una vez que se ha determinado la forma, se pueden determinar diferentes características del proceso de fallas. En muchos modelos encontramos que hay expresiones analíticas para:

- Número promedio de las fallas experimentadas en cierto punto en el tiempo
- Número promedio de fallas en un intervalo de tiempo
- La distribución probabilística de los intervalos de fallas

Un buen modelo de fiabilidad de software tiene importantes características.

- Ofrece una buena proyección del comportamiento de las fallas en el futuro.
- Calcula cantidades útiles
- Es ampliamente aplicable
- Esta basado en suposiciones

La proyección a futuro del comportamiento de las fallas asume que los valores de los parámetros del modelo no van cambiar para el periodo de proyección. Si el efecto neto de la introducción de fallas y la remoción de fallas debe cambiar sustancialmente, se debe entonces compensar los cambios o esperar hasta que las suficientes fallas hayan ocurrido para volver a estimar los parámetros del modelo.

La mayoría de los modelos de fiabilidad de software están basados - aunque no se diga explícitamente - en el uso estable del programa en un ambiente estable. Esto significa que ni el código, ni el perfil operacional esta cambiando. Si el perfil operacional cambia, el primero también. Por lo tanto los modelos se enfocan principalmente en la remoción de fallas. Unos modelos toman en cuenta la lenta introducción de errores, sin embargo algunos asumen que el promedio neto de efecto a largo plazo de todos los factores debe de ser un decremento en la intensidad de fallas. Si ni la introducción de fallas, la remoción de fallas o los cambios de los perfiles operacionales ocurren, la intensidad de fallas sería constante y el modelo se reduciría a acomodar este hecho. Esto asumiendo que se compara el comportamiento del programa con los requerimientos con tal exactitud que se encuentran todos los errores.

Para un programa que ha sido liberado y está en operación es común el volver a planear la instalación tanto de las nuevas características y de las reparaciones para el nuevo producto. Asumiendo un perfil operacional constante el programa mostrará una intensidad de fallas constante.

En términos generales, un buen modelo aumenta la comunicación en el proyecto y provee un cuadro común de entendimiento del proceso de desarrollo del software. También aumenta la visión para la administración y otras partes

interesantes. Estas ventajas son de valor aún si las proyecciones hechas con el modelo no son tan precisas como nos gustaría.

El desarrollo de un modelo de software que sea útil en la práctica conlleva una gran cantidad de trabajo teórico, herramientas de trabajo, y la acumulación de un gran conocimiento derivado de la experiencia. Este esfuerzo generalmente requiere un considerable trabajo de personas durante años. En contraste la aplicación del modelo que está bien establecido en la practica requiere una mínima parte de los recursos del proyecto.

Algunos sugieren la aplicación de varios modelos de fiabilidad de software a un mismo proyecto; el que se desarrolle de mejor manera - o alguna gran combinación de ellos – será la que se use. Este método puede ser aplicable con fines de investigación. De cualquier manera, el uso de más de uno o dos modelos es conceptual y económicamente poco practico en la vida real. Es necesario que los miembros del proyecto entienda lo que los parámetros de los modelos significan físicamente y relacionarlos con el software para que se puedan hacer juicios acertados. Ellos no podrán entender y usar varios modelos en forma simultánea. De igual forma el costo relacionado aumenta rápidamente a medida que se incrementa el número de modelos que se aplican.

2.2 Característica Generales

Un modelo de fiabilidad de software, como se comento antes, generalmente tienen la apariencia de procesos aleatorios que describen el comportamiento de las fallas con respecto del tiempo. La especificación del modelo generalmente incluye la especificación de la función de tiempo tanto como el valor principal de la función (el número esperado de fallas) o la intensidad de fallas. Los parámetros de la función son principalmente dependientes de la actividad de remoción de fallas y de las propiedades del producto del software y el proceso de desarrollo (hay que recordar que la adición sustancial de características comenzando una nueva pieza en la función de tiempo, y si son pocas las nuevas características que se han agregado, solo provoca que se ajusten los valores de los parámetros) las propiedades del producto incluyen tamaño, complejidad y estructura. La característica más importante del producto es el tamaño del código desarrollado - ya sea creado o modificado para esta aplicación -. Las propiedades del proceso de desarrollo incluyen, entre otras, tecnología en la ingeniería del software, herramientas usadas, y el nivel de la experiencia del personal. El tiempo

que se maneja en la caracterización de los modelos es acumulable, el origen puede ser establecido arbitrariamente, aunque generalmente es el comienzo de las pruebas del sistema.

Los modelos de fiabilidad de software generalmente asumen que las fallas son independientes unas de otras, esto suponiendo que, los tiempos de fallas son independientes unos de otros o asumiendo el proceso de Poisson para incrementos independientes. Este último proceso parecería, que satisface la mayoría de los casos. Las fallas son resultados de dos procesos: la introducción de los errores y su activación a través de ciertos estados de entrada; debido a que esos procesos son aleatorios la posibilidad de que una falla influya sobre la otra es pequeña. Se tendrían que dar dos factores para que esto pudiera ocurrir: Un error debería de afectar la entrada de otro durante el desarrollo, aún más, un estado de entrada que resulte en una falla para el primer error tendría que causar la selección de un estado de entrada que resulte en falla para el segundo error. Es muy improbable que ambas condiciones ocurran conjuntamente.

Algunos teóricos refutan esta postura dado que no aceptan la situación de los "dos procesos". Por ejemplo ellos argumentan que los programadores tienden a tener patrones de errores (esto es, un error puede influenciar al otro) La posibilidad de que esto pueda crear errores relacionados no implica por si mismo la existencia de fallas relacionadas. De igual forma ellos dicen que una falla puede prevenir otra falla porque esta previene el acceso a cierto código o "esconde" el mismo. La prevención de entrar a cierto código puede suceder ocasionalmente cuando se desarrollan las pruebas del sistema, pero en fases posteriores cuando ya se esta aplicando la fiabilidad de software es poco común. Las fallas que encubren a otras tienden a estar eliminados para este punto.

2.3 Proceso Aleatorios

Los errores humanos en los procesos que introducen defectos en el código y el proceso de selección que define que parte del código se va a ejecutar en cualquier momento, están sujetas de un gran número de variables dependientes del tiempo. El proceso aleatorio es ideal para esta situación. Hay dos formas equivalentes de describir el proceso aleatorio de las fallas: los tiempos de fallas o el numero de fallas en un periodo determinado.

Se denotará con T_i y $T_i^{'}$ como las variables que representan los tiempos de la i-ésima falla y la (i-1) ésima falla respectivamente. Las realizaciones (instancias especificas) de T_i y $T_i^{'}$ serán representadas por t_1 y $t_i^{'}$ respectivamente. Supóngase que $M(t)$ es un proceso aleatorio que representa el número de fallas experimentadas en el tiempo t. La realización de este proceso aleatorio será denotado por $m(t)$. El principal valor de la función

$$\mu(t) = E[M(t)]$$

la cual representa el número de fallas esperadas en el tiempo t. Se asume que la función $\mu(t)$ no es decreciente, pero si es continua y diferenciable en el tiempo t. La función de intensidad de fallas del proceso $M(t)$ es la tasa instantánea de cambio del número esperado de fallas con respecto al tiempo. Se define por:

$$\lambda(t) = \frac{\partial \mu(t)}{dt}$$

Es posible planear o manipular la sección de código a ejecutar durante la prueba. La persona o personas que prueban el sistema, tienen el control, al menos de forma parcial, del medio ambiente donde se van a ejecutar las pruebas y esto puede crear un modelo pobre, aunque las entradas se hagan de forma aleatoria. De cualquier forma un proceso aleatorio sigue siendo un modelo aceptable del comportamiento de fallas. La introducción de errores en el código y la relación entre el estado de entrada y el código ejecutado, son procesos suficientemente complejos que hacen de la predicción deterministica algo impractico. En otras palabras, no se puede predecir cuales estados de entrada son más probables de producir fallas.

Por ende, una selección determinística de los estados de entrada no va a tener efectos determinísticos en la fiabilidad. Obviamente, si las frecuencias relativas de la selección han cambiado, el perfil operacional ha cambiado y esto si afecta la fiabilidad.

Hay un caso en el cual la manipulación de las características del proceso aleatorio puede ocurrir. Se requiere de:

1. Los segmentos de programa ejecutable para diferentes estados de entrada están "separados", uno con respecto del otro.
2. Hay claras diferencias en la intensidad de errores entre diferentes segmentos del programa.

"Separados" significa que cada estado de entrada apunta a diferentes conjuntos del programa ejecutados y no hay segmentos de programas ejecutados en común por diferentes estados de entrada. Hay una clara diferencia en la densidad de errores, esto cuando algunos segmentos pueden ser código reutlizado de programas previos y algunos pueden estar recién escritos. En esta situación es posible el manipular formas de fiabilidad a una forma más alta o más baja, basándose en la selección de entradas. Se seleccionan estados de entrada que ejecuta el código que tenga alta o baja densidad de errores nótese que el carácter esencial del proceso de fallas como un proceso aleatorio no ha cambiado. No se puede predecir cuando se va a ocurrir la próxima falla, aún si se puede manipular el promedio del comportamiento. Un ejemplo de manipulación podría ser la selección de estados de entrada de forma deterministica en la cual no haya forma de volver a ejecutar segmentos de código. Si la densidad de errores es el mismo número para todos los segmentos, la intensidad de errores observada tiende a ser constante. Se supone que la reparación de errores no mostraría efectos en la intensidad de fallas. En realidad, la intensidad de fallas basada en selección de entradas aleatorias sería decreciente.

Con y sin remoción de fallas

Los modelos de fiabilidad de software deben de cubrir dos situaciones, una en la cual los errores de los programas son eliminados cuando ocurren las fallas y programas en las que no. La situación segunda ocurre cuando el producto se ha liberado y se encuentra en el campo de trabajo, para ser precisos no se esta diciendo que los errores no se van a corregir, sino que solo se esta postergando esta actividad, dado que los arreglo se notaran hasta la siguiente versión.

Cuando no se remueven los errores, la intensidad de fallas es constante por todo el periodo en el cual el software es utilizado. Por lo tanto se puede modelar convenientemente el proceso de fallas por un proceso homogéneo de Poisson. Esto implica que los intervalos de fallas están distribuidos exponencialmente y que el número de fallas en un determinado periodo de tiempo sigue una distribución de Poisson. Si la intensidad de fallas es λ y el periodo de ejecución del programa es t el número total de fallas durante este periodo es una distribución de Poisson con parámetros.

Como se mencionó anteriormente, el principal factor que afecta o que causa variación en la fiabilidad es la remoción de errores que producen fallas. En

general el tiempo de remoción no coincide con el tiempo original de fallas. Esto nos puede llevar a complicaciones cuando se trate de representar el proceso de fallas. De cualquier forma esto se puede manejar asumiendo la remoción instantánea y no contando la reincidencia de la misma falla. Sin embargo, se tendrá que contar si la reincidencia es debido a la incapacidad de localizar y remover el error. Aunque el resultado no es precisamente equivalente, es una muy buena aproximación. La mayoría de los modelos utilizan este método.

2.4 Particularización

El modelo especifica la forma general de la dependencia de los procesos de fallas sobre las variables antes mencionadas (inserción de errores, remoción de errores, y uso del software). Se puede determinar una forma especifica a partir de la forma general, al menos en teoría determinando los parámetros necesarios. Esto puede ocurrir a través de dos formas:

1. **Predicción**. Se usan propiedades del producto (software) y del proceso de desarrollo para particularizar el modelo mediante la determinación de sus parámetros - esto puede ser realizado antes que se realice una ejecución del programa.
2. **Estimación**. Se aplica procedimientos de inferencia (por ejemplo, la estimación de parámetros) a los datos de fallas.

Un modelo y un procedimiento de inferencia están generalmente asociados. Juntos estos proveen una proyección en el tiempo. Sin un modelo no se podrían hacer inferencias acerca de la fiabilidad fuera del periodo de tiempo para los cuales se tomaron los datos de fallas. De hecho no se podría hacer ninguna clase de inferencia porque el tamaño de la muestra de fallas sería 1. El modelo provee la estructura que relaciona el comportamiento en diferentes periodos de tiempo.

La inferencia incluye generalmente la determinación del rango de incertidumbre. Ya sea que se establezca intervalos confiables para los parámetros dados o se determinen distribuciones probabilísticas posteriores para cantidades significativas en caso de que se trate de inferencias bayesianas. Un intervalo confiable representa un rango de valores dentro de los cuales se espera un parámetro en el cual apoyarse con cierto grado de confianza estadística. Por ejemplo, el intervalo de 0.75 de confianza del número total de fallas que se

experimentaran en un tiempo infinito pueden ser de 150 a 175. Generalmente se extiende la determinación de rangos de incertidumbre a cantidades que son derivados de estos modelos.

2.5 Clasificación

El trabajo del modelado analítico puede ser calificado en dos grandes categorías. La primera la primera se enfoca en la naturaleza estócastica de las fallas de software, mientras que la segunda propone el uso de análisis combinatorio para proporcionar modelos de fiabilidad en software.

1. Modelos basados en la tasa de fallas (azar)
2. Modelos basados en la tasa de No-fallas

Los modelos basados en la tasa de fallas, pueden clasificarse aún más, como se muestra en la tabla siguiente (A). Esta tabla no es extensa, pero contiene los modelos basados en la tasa de fallas más comúnmente usados para la fiabilidad de software.

Los modelos basados en la tasa de no fallas pueden ser clasificados en:

- Modelos combinatorios
- Modelos basados en el dominio de entrada

(A) Tabla de modelos de fiabilidad basados en la tasa de fallas

	Clásicos	Bayesianos
	Proceso de Eutrofización Modelo de Jelsinki-Moranda	Modelo de Littlewood
	Modelo de depuración imperfecta	
	Pruebas de función lineal Modelo de tiempo de	

	Shick y Wolverton	
Conteo de errores de modelos basados en tasa de fallas	Prueba de función parabólica de Wolverton	
	Modelo de Shooman	
	Modelo de tiempo de ejecución de Musa	
	Modelo de proceso geométrico de Eutrofización de Moranda	Modelo de Littlewood y Verrall
	Modelo de proceso geométrico Poisson de Moranda	

La lista siguiente representa algunos de los modelos más populares pertenecientes a los grupos anteriormente mencionados:

Modelos combinatorios

- Modelo Hipergeométrico de Mills
- Modelo Binomial

Modelos basados en el domino de entrada

- Modelo de Brown y Lipow

Modelos basados en tasas de fallas: Suposiciones

La tasa de fallas (también conocida como tasa al azar) la función **z(t)** es definida como la probabilidad condicional que un error sea expuesto en un intervalo t para **t + dt**, suponiendo que el error no sucedió antes del tiempo **t**.

La función de fiabilidad **R(t)** es la probabilidad de que no ocurra ningún error del tiempo cero al tiempo t, de forma más amplia **z(t)** y **R(t)** son descritos de la siguiente forma:

$$z(t) = \frac{\frac{\partial R(t)}{\partial t}}{R(t)} \qquad \text{ó} \qquad R(t) = e^{(-\int_0^t z(x)dx)}$$

Los modelos basados en la tasa de fallas, difieren básicamente en la suposición acerca de la función de tasa de fallas **z(t)**. La tabla (B) muestra las diferencias entre las suposiciones que se hacen con respecto a las funciones de las tasas de fallas.

Tabla B

Modelo	Suposición de z(t)
Modelo de proceso de Eutrofización	La tasa de fallas de software en cualquier tiempo **t** se asume en forma proporcional al número de errores que permanecen en el software. Ejemplo: "para el intervalo de tiempo entre (i -1) y la i-ésima falla tenemos **z (xi) = ϕ [N - (i -1)]** Donde **N** es el error inicial contenido
Modelo lineal en proporción de fallas de Shick-Wolverton	La tasa de fallas se asume en forma proporcional al número de errores sobrantes en el tiempo de prueba del software. Para el intervalo (a,b,c>0) $Z(X_i) = \Phi[N-(i-1)](-ax_i^2 + bx_i + c)$
Modelo de Shooman	$Z(t) = K[\frac{E_T}{I_T} - \int_0^M p(x)\partial x]$ Donde: K = Constante de proporcionalidad ET = Número total de errores IT = Número total de instrucciones M = Tiempo de depuración P(x) = Número de errores por instrucción en tiempo de depuración $\int_0^M (x)\partial x$ = Número total de errores por IT removidas durante M unidades de tiempo en el tiempo de depuración

Modelo de tiempo de ejecución de Musa	$$z(\Gamma) = kfN_0 - Kfn(\Gamma)$$ Donde: K = Radio de exposición del error f = Frecuencia de ejecución linear del programa No = Contenido inicial de errores Γ = Tiempo utilizado por el CPU en la operación del programa n(Γ) = Número neto de errores corregidos durante Γ Si $\frac{\partial n(\Gamma)}{\partial t}$ = Tasa de exposición del error, entonces: $z(\Gamma) = kfN_0 e^{(-kf\Gamma)}$
Modelo geométrico de proceso de Eutrofización de Moranda	Asume que los pasos que representa el decremento en la tasa de errores entre los tiempos de falla adyacentes son variables geométricas. $Z(x_i) = DK^{i-1}$ Donde: D = Tasa de detección de errores DK = Tasa de detección de errores después de la ocurrencia del primer error . . . Dk^{i-1} = Tasa de ocurrencia de error después de la ocurrencia del i-ésimo error.
Modelo geométrico del proceso de Poisson de Moranda	Una superposición del proceso geométrico de Eutrofización y del proceso de Poisson con parámetro: $Z(x_i) = Dk^{i-1} + \theta$
Modelo de Littlewood y Verrall	$Z(t) = \lambda$ pero λ es tratada como una variable aleatoria distribuido como gamma en un parámetro formado α y parámetros de escala ψ (i), una función de incremento (i)

Modelo de Littlewood	$Z(x_i) = \lambda_i$ y λ_i es distribuido como gamma: $[(N - i)\alpha\beta + \sum_{1}^{i-1} t_j]$ Donde: N+i+1 = Número de errores restantes cuando (i -1) fallas han ocurrido t_j= tiempo de ejecución de (j-1) falla hasta la j-ésima falla α, β = parámetros de la distribución de gamma.
Modelo no homogéneo del proceso de Poisson de Goel	$P_r\{N(t) = y\} = \frac{[m(t)]^y e^{-m(t)}}{y!}$ $y = 0,1,2,...$ Donde : N(t) = Número acumulado de errores de software en el tiempo t m(t) = $a(1-e^{(-bt)})$ Número esperado de fallas de software por tiempo t $\lambda(t) = abe^{-bt}$ función de intensidad (tasa de detección de errores) $R_{X_K}\ S_{K-1}{}^{(x/s)} = e^{-\left\{e^{-bs_n} - e^{-b(s_n+x)}\right\}}$ = fiabilidad en el tiempo x donde Sn representa el tiempo acumulado en el cual "n" fallas de software han ocurrido. Y a y b pueden ser resueltas de: $\frac{n}{a} = 1 - e^{-bs_n}$ $\frac{n}{b} = as_n e^{-bs_n + \sum_{i=1}^{n} S_i}$

Los modelos basados en la tasa de fallas contienen suposiciones que son cuestionables e irreales:

1. Todos los modelos descritos con anterioridad asumen que cualquier error detectado es inmediatamente corregido. El proceso de corrección no afecta el programa. Todas las correcciones reparan los errores detectados y no traen como resultado la introducción de nuevos errores. No es muy difícil aceptar que la corrección de errores detectados en un programa puede resultar en la introducción de nuevos errores Goel y Okomoto[6] tratan con habilidad la segunda restricción antes mencionada, formulando lo que ellos denominan Método Imperfecto de Depuración *IDM*. El IDM asume que el número de errores en un sistema en un tiempo t es regido por el proceso de Markov. El tiempo de las transiciones es distribuido exponencialmente con tasas dependientes del actual error del programa. El estado de transición es regido por la probabilidad de la depuración imperfecta. Todavía nadie ha señalado el problema en el cual el proceso de depuración introduce nuevos errores en el software

2. Modelos como los de Jelsinki y Moranda, Musa y Shooman asumen que la tasa de fallas del software es un múltiplo constante de los errores sobrantes, esto es lo mismo que si se dijera: que cada error en un intervalo de tiempo dado (entre fallas) tiene la mismo oportunidad de ser detectado. Esto obviamente, no es cierto dado que los errores que se pueden detectar y que residen en una parte del código que es frecuentemente ejecutada por el usuario tienen más posibilidades de ser detectadas. Los errores que residen el la parte inalcanzable (o nunca ejecutada) del código tendrán una menor (o nula) posibilidad de ser detectados. Moranda trata hábilmente este problema mediante la reformación de del modelo de Eutrofización en el modelo geométrico de Eutrofización y más tarde en el modelo geométrico de Poisson. En esas variaciones la tasa de fallas entre intervalos de fallas adyacentes son variantes geométricas.

3. Los modelos de SchicK-Wolverton pasan a moldear donde hay un incremento en la tasa de fallas entre los errores. Esta puede ser una suposición ridícula si argumentamos que el software no se gasta. Pero puede haber casos donde la tasa de fallas del software puede incrementarse y puede ser atribuido al incremento en el número de pruebas. Este fenómeno es observado usualmente en las primeras etapas del ciclo de desarrollo del software.

4. Basando el tiempo entre fallas en el tiempo de ejecución (CPU) como lo asumen Musa y Littlewood algunas veces puede ser irreal un incremento en el tiempo entre fallas adyacentes no necesariamente indica que el software tiene menos y menos números de errores o que la fiabilidad del software se está incrementando. Un ejemplo muy simple ilustrará este punto. Consideremos un programa que solo contiene un error la misma copia del programa se le da a dos depuradores. Uno gasta mucho tiempo

ejecutando y volviendo a ejecutar el programa (el cual puede estar tentado a hacerlo en sistemas "en línea" y en tiempo compartido) tratando de descubrir el error. Por otro lado el segundo gasta mucho tiempo analizando el programa antes de ejecutar una prueba. Supongamos que ambos tienen éxito en descubrir el error. ¿Cuál es la fiabilidad resultante del software? La teoría del tiempo de ejecución dice que como el tiempo entre fallas del primer software es más largo que del segundo, el primero tiene mayor fiabilidad. Obviamente nosotros sabemos que esto no es cierto ya que ambas versiones del software tienen la misma fiabilidad. Sigue habiendo controversia acerca de cual unidad de tiempo es la más apropiada para usar en los periodos entre fallas.

5. Consideremos el supuesto de la independencia del tiempo entre fallas. La cuestión es que no es una suposición real. El proceso de pruebas que es usado para descubrir errores generalmente no es un proceso aleatorio. El tiempo para la próxima falla puede ser muy dependiente de la naturaleza y el tiempo del error anterior. Si el error previo fue crítico, entonces podemos decidir intensificar los procesos de pruebas y descubrir más errores críticos. Esta intensificación en lo procesos de prueba pueden significar un menor tiempo para que aparezca la siguiente falla, que si se hubiesen seguidos los niveles normales de pruebas.

6. Muchos de los modelos requieren del tiempo entre los datos de fallas para calcular la fiabilidad. Puede haber casos donde el tiempo medio entre las fallas sea infinito; siendo así esos modelos pasan a ser inservibles. El tiempo medio entre fallas puede ser infinito si el usuario del software tiene requerimiento que no atraviesan los caminos donde se encuentran errores.

7. Basar la fiabilidad en el número de errores restantes puede ponerse en tela de juicio. El usuario no se preocupa si el software tiene cierto número de errores sobrantes. En la medida en que el software satisfaga sus requerimientos es cuando, al menos para el usuario es 100% fiable. Littlewood[7] argumenta que un programa con dos errores en una pequeña - y de poca ejecución - parte del código puede ser más fiable que un programa que solo contenga un error, pero que se encuentre frecuentemente.

8. Todos los modelos asumen implícitamente que el proceso de pruebas, el cual genera el calculo para la tasa de fallas, será que el mismo que el medio ambiente operativo. Esto otra vez no es cierto, por que en la medida en que de fiabilidad esta condicionada sobre los requerimientos del usuario, en lugar de una simple medida incondicional de la fiabilidad del software suena y parece más realista.

Modelos basados en la tasa de "No-fallas": Suposiciones

Modelo Hipergeométrico (siembra de errores) de Mill

Este modelo requiere que un cierto número de errores sean insertados aleatoriamente (sembrados) en el programa que se va a probar. El programa es entonces probado por cierta cantidad de tiempo. El número original de errores innatos puede ser calculado del número de errores innatos y sembrados descubiertos durante la prueba.

Si

n = No. de errores sembrados.
k = No. de errores sembrados encontrados en las pruebas.
N = No. total de errores innatos
r = No. de errores innatos detectados durante las pruebas.

Entonces:

P (k - errores sembrados / errores innatos detectados) $= \dfrac{\left(\dfrac{n}{k}\right)\left(\dfrac{N-n}{r-k}\right)}{\dfrac{N}{r}}$

MLE (Máxima probabilidad calculada) para $N = \dfrac{(n)(r)}{k}$

La suposición de más peso en este modelo es que los errores innatos tienen la misma probabilidad de ser detectados que los errores sembrados.

Modelo Binomial

Si

$q_i = \Pr(errores)$ en cada ejecución de I

Entonces

Pr ("x" errores en "y" intentos) = $(x^y)q_i^x(1-q_i)^y$

Otra vez el punto fuerte de es que todos lo errores tiene la misma oportunidad de ser expuestos.

Modelo de Brown y Lipow

Si

n_e = Número de entradas para las cuales las fallas de ejecución ocurren.
n = Número de casos de pruebas.
R = Fiabilidad

Entonces:

$$R = 1 - \frac{n_e}{n}$$

De nueva cuenta la suposición de peso es la misma probabilidad de escoger n_e de n.

Modelo de Goel NHPP

Este supone que los errores pueden existir aleatoriamente en la estructura del código y su aparición es una función de tiempo de ejecución del programa. El número de errores ocurridos en el tiempo t es N(t), si la siguiente condición existe:

1. N(0)=0
2. No más de un error puede ocurrir en el intervalo de tiempo (t, t+dt)
3. La ocurrencia de un error es independiente de los errores previos.

Entonces la ocurrencia de errores es descrita por la distribución homogénea de Poisson:

$$P[N(t) = n] = \frac{[m(t)]^n}{n!} e^{[-m(t)]} \quad n >= 0$$

donde

$$m(t) = \int_0^t \lambda(s) ds$$

m(t) es el número medios de errores (s - esperados) ocurridos en el intervalo (0, t)

$$m(t) = a[1 - e^{(-bt)}]$$

donde ***a*** es el número total de errores y ***b*** es una constante, ambas pueden ser calculadas de las expresiones:

$$\frac{n}{a} = 1 - e^{-bS_n}$$

$$\frac{n}{b} = aS_n e^{-bS_n} + \sum_{i=1}^{n} S_i .....$$

Donde ***Sn*** representa el tiempo acumulado en el cual han ocurrido las fallas de software. El número de errores sobrantes en el tiempo t, suponiendo que cada error que ocurre es corregido y no se introducen nuevos errores es:

$$\tilde{N}(t) = ae^{-bt}$$

La función de fiabilidad, después de que sucedió y se corrigió el más reciente error en el tiempo S, es:

$$R(t) = e^{[[-a\{e^{(-bs)} - e^{[-b(s+t)]}\}]]}$$

2.5.1 *Ejemplos de cálculos usando modelos de fiabilidad de software*

El modelo de Musa

El modelo de Musa usa el tiempo de ejecución del programa como una variable independiente, una versión simplificada del modelo de Musa es la siguiente:

$$n = N_0\left[1 - e^{\left(\frac{-ct}{N_0 T_0}\right)}\right]$$

donde *No* es el número inherente de errores, *To* el MTTF al principio de las pruebas (MTTF = es el tiempo medio por falla) y *C* es el factor de compresión de la prueba, que es igual a la razón de tiempo de operación equivalente del tiempo de pruebas.
El presente MMTF:

$$T = T_0 e^{\left(\frac{ct}{N_0 T_0}\right)}$$

Nos da

$$R(t) = e^{\left(\frac{-t}{T}\right)}$$

De esas relaciones se puede derivar el número de fallas las cuales deben de ser encontradas y corregidas, o el tiempo de ejecución necesario para mejorar de T1 a T2:

$$\Delta_n = N_0 T_0 \left(\frac{1}{T_1} - \frac{1}{T_2}\right)$$

$$\Delta_t = (\frac{N_0 T_0}{c}) \ln(\frac{T_2}{T_1})$$

Ejemplo primero.

Se cree que un programa largo contiene cerca de 300 errores y el registro MTTF al inicio de la prueba es 1.5 horas. El factor de compresión de prueba se asume que es de 4 ¿Cuántas horas de pruebas se requieren para reducir el número sobrante de errores a 10?
¿Cuál será la fiabilidad en 50 horas de ejecución?

De

$$\Delta_n = (300-10) = (300)(1.5)(\frac{1}{1.5} - \frac{1}{T_2})$$

$$T_2 = 45 \text{ horas}$$

$$\Delta_t = (\frac{(300)(1.5)}{4}) \ln(\frac{T_2}{1.5})$$

$$\Delta_t = 382.6 \text{ horas}$$

dando

$$R_{50} = e^{\left(\frac{-50}{45}\right)} = 0.33$$

El modelo de Mills

Un modelo diferente para la predicción de la fiabilidad del software ha sido propuesto por Mills. Este es un método más pragmático, en vez de un intento por concluir un modelo matemático basado en el tiempo. Un número conocido de errores son introducidos deliberadamente en el programa. Como la depuración (inspección o prueba) es llevada a cabo, el registro muestra cuales de los errores "sembrados" son encontrados. El número de errores restantes desconocidos son entonces tomados para ser una función de la razón de los errores restantes sembrados y descubiertos.

$$n = N_0[1 - e^{(\frac{-ns}{N_s n_s})}]$$

donde *Ns* es el número total de errores sembrados y *ns* es el número de errores sembrados restantes.

Ejemplo segundo

La experiencia indica que contenga cerca de 100 errores al inicio de la validación. Diez errores son sembrados de forma deliberada, tanto en forma ordenada como en forma aleatoria,. Nueve de ellos son descubiertos al final de la validación. ¿Cuántos de los errores originales tienen probabilidades de permanecer?

$$100[1-e^{(-\frac{1}{10-1})}] = 10.16068$$

El método de Mills supera el tiempo. De cualquier forma las desventajas prácticas de para la prueba para corregir los errores sembrados son obvias. Para hacer una predicción valida de los números de errores restantes, los errores sembrados deben de ser del mismo tipo de error y en la misma proporción como el original. Probablemente este método es más adecuado para los errores de código, ya que la siembra de errores de especificación y diseño no es integro. De cualquier forma el método de Mills no ha sido aceptado de forma practica para los programas típicos, a pesar de su similitud con métodos que son usados para verificar la fiabilidad de las rutinas de pruebas para diagnosticar las fallas introducidas deliberadamente en el hardware.

El modelo de Littlewood

Littlewood intenta tomar en cuenta que diferentes errores de programa tienen diferentes posibilidades de hacer fallar al software. Si $\Phi_1, \Phi_2 ... \Phi_n$ son la tasa de ocurrencia de errores 1,2,...n, la pdf (función de densidad de probabilidad) para el tiempo de fallas del programa después de que el i-ésimo error ha sido corregido es :

$$f(t) = \lambda e^{(-\lambda t)}$$

Donde λ es la tasa de fallas del programa

$$\lambda = \Phi_1 + \Phi_2 + ... \Phi_N$$

Φ es supuesto como una distribución gamma, por ejemplo, los errores no tienen tasas constantes de ocurrencia, pero las tasas son dependientes del mismo uso del programa.

Si los parámetros de distribución gamma son (α y β) entonces se puede utilizar el método de Bayes para demostrar que:

$$f(t) = \frac{(N-1)\alpha(\beta + t')^{(N-i)\alpha}}{(\beta + t' + t)^{(N-1)\alpha+1}}$$

Donde t' es el tiempo tomado para detectar y corregir errores; de los cuales

$$R(t) = \left(\frac{\beta + t'}{\beta + t' + t}\right)^{(N-i)\alpha}$$

y

$$\lambda(t) = \frac{(N-i)\alpha}{\beta + t' + t}$$

Para cada suceso y corrección de error, λ (t) desciende por la cantidad $\frac{\alpha}{\beta + t'}$.

Se supone que todos los errores se introducen sin que más errores sean introducidos.

Ejemplo tercero

Se supone que un programa extenso que incluye un total de 300 errores, de los cuales 250 han sido detectados y corregidos en 20 horas de tiempo de ejecución. Suponiendo que se aplica el modelo de Littlewood y los parámetros de distribución son α=00.5 β=4
¿Cuál es la fiabilidad esperada en las siguiente 20 horas?

$$R(20) = \left(\frac{4+20}{4+20+20}\right)^{(300-250)0.005} = 0.859389$$

[6] Goel A., K. Okumoto "A markovian model for reliability and other performance measures of software system" (1979)
[7] Littlewood , B. "Theories of software reliability: How good are they and How can they be improved?" (1979)

Capítulo III
Introducción a la Ingeniería de Fiabilidad de Software

3.1 La Ingeniería de Fiabilidad de Software

No es de dudarse que la remoción de errores en el código, la predicción del comportamiento del producto en un tiempo determinado con condiciones determinadas, es de gran ayuda para lograr un producto de calidad. Pero tan importante es eso como la forma que se desarrolla el producto (ciclo de vida del software: Análisis, Diseño, Desarrollo, Pruebas, Implantación, Mantenimiento). Y esto aunado con técnicas complementarias en el desarrollo (La ingeniería de fiabilidad de software) puede hacer más fácil la elaboración de productos de software, con menos costos y menor esfuerzo.

En el desarrollo de software se pueden presentar muchos riesgos:

1. Poca fiabilidad del producto liberado.
2. Perdidas de planes.
3. Elevación de costos.

En respuesta a estos problemas, se ha prestado mucha atención a los mecanismos de desarrollo y prueba, y se han construido muchas piezas para soportar le proceso de desarrollo. La Ingeniería de Fiabilidad de Software significa desarrollar un producto de forma tal que alcance el "mercado" en el tiempo preciso, a un costo aceptable, y con una fiabilidad satisfactoria; Nótese que el significado de mercado en usado en su sentido más amplio, no solo refiriéndose a los productos mas conocidos y comerciales, sino también a los productos desarrollados especialmente para el ámbito militar y de gobierno los cuales también tienen tanto mercado como competidores y los usuarios pueden escoger entre varias opciones.

El análisis tradicional de desarrollo y pruebas no logra las metas especificas de la Ingeniería de Fiabilidad del Software (IFS), la IFS toma un campo y un análisis más amplio, esta ha demostrado que las pruebas mas eficientes involucran actividades que ocurren a través del ciclo de vida del producto - con la ingeniería del sistema y el diseño de tareas -. La IFS permite a la gente que realiza las pruebas que tomen el liderazgo en conocer las necesidades del cliente. Las características del producto – fiabilidad, tiempo de desarrollo y costo - definen su calidad, la cual es producto del balance de esas características, obtener un buen balance significa que se deben tomar objetivos cuantitativos para las tres características y medir las mismas durante el desarrollo del sistema.

¿Qué es la Ingeniería de Fiabilidad de Software y como ayuda al desarrollo y prueba del sistema?

LA IFS es una forma que permite al desarrollador que prueba el sistema hacer simultáneamente:

- Asegurar que la fiabilidad del producto satisface las necesidades del usuario.
- Menor tiempo para llegar al mercado.
- Reducir el costo de la producción.
- Aumentar la satisfacción del cliente y reducir el riesgo de usuarios molestos.
- Aumentar su productividad.

Se puede usar la IFS para cualquier producto *basado en software*, iniciándose desde el ciclo de vida del producto. Cabe aclarar que se usa el termino basado en software dado que no hay sistemas puramente de software, por lo tanto el hardware siempre tiene que ser tomado en cuenta en el análisis.

La IFS trabaja aplicando dos ideas básicas. La primera, es que entrega la funcionalidad deseada para el producto bajo un desarrollo mucho más eficiente por medio de la caracterización cuantitativa de lo que espera el usuario y usa esta información para:

- Concentrar recursos en las funciones críticas o más usadas.
- Hacer las pruebas lo más reales posibles, representando las condiciones en la cual se van a trabajar.

Funciones críticas se refiere a que pueden tener un valor extra si el sistema tiene éxito o más impacto si el sistema falla, este impacto puede reflejarse con respecto a vida humanas, costos o la capacidad del sistema.

La segunda, la fiabilidad del software media las necesidades de fiabilidad del cliente, el tiempo de desarrollo y la estimación del costo, por lo tanto de la efectividad. Para lograrlo, se unen las medidas de fiabilidad como una lista o planes de costo de los objetivos, que deben de satisfacer los ingenieros de estrategias. Finalmente la ingeniería de software rastrea la fiabilidad en las pruebas y las utiliza como un criterio en la liberación del producto. Con la IFS la liberación de su producto satisface las necesidades de fiabilidad y supera tanto el costo excesivo como el tiempo de desarrollo, evitando el resultado de usuarios molestos debido a un producto de poca calidad y fiabilidad.

La IFS está basada en teorías muy sólidas, que incluyen perfiles operacionales, modelos de procesos de fiabilidad aleatorios, estimaciones estadísticas y la teoría de muestreo secuencial. Personas que trabajan en el software han practicado la fiabilidad del software en forma extensa desde fechas de 1973. AT&T ha tenido y tiene la mejor práctica actual (BCP) desde

mayo de 1991. La selección de AT&T como BCP fue significativa dado que se impusieron estándares muy altos. Primero, se tiene que usar el método propuesto en varios proyectos (de 8 a 10) y obtener logros significativos, beneficios en la documentación referente al radio de costos, y medidas en términos financieros. Entonces de desarrolla una descripción detallada del método y como usarla en los proyectos junto con casos de negocios para adoptarla. Comités de gente experimentada - directores de proyectos de tercer y cuarto nivel revisan el método en un exhaustivo análisis. Generalmente el análisis dura muchos meses, el análisis detallado es delegado a directores de proyecto de primer nivel y desarrolladores senior, el análisis de la propuesta de ingeniería de fiabilidad (BCP) envuelve más de 70 personas. En 1991 solo se aceptaba una de cada cinco propuestas de ingeniería de fiabilidad BCP, en ese año se presentaron 30. En 1993 El Instituto Americano de Astronomía y Aeronáutica (AIAA) aprobó la ingeniería de fiabilidad como un estándar teniendo un gran impacto en la industria aerospacial. EL IEEE ha estado activo en la elaboración para los estándares de la ingeniería de fiabilidad.

El Centro de Operaciones Tecnológicas AT&T en la división de Redes y Servicios de Computo ha usado la IFS como estándar de sus procesos de desarrollo de software por muchos años. Este mismo centro fue el primero que ganó el Premio Nacional de Calidad Malcom Baldrige en 1994. En ese tiempo tenía el más alto porcentaje de proyectos aplicando la IFS en AT&T, Otra observación curiosa es que las cinco primeras áreas de AT&T Bell Laboratories que ganaron el premio President's Quality, usaron IFS.

3.2 *El proceso de la Ingeniería de Fiabilidad de Software*

El proceso para la aplicación de la IFS requiere, primero que se determine los sistemas asociados con el producto que se va a probar. Haciendo esto se tiene que entender los tipos de prueba de la IFS, por lo tanto primero nos enfocaremos al punto ultimo, los tipos de prueba en la IFS, y después nos dirigiremos al punto de los sistemas relacionados con la prueba.

La ingeniería de fiabilidad de software consiste básicamente de cinco actividades, son: La definición de la fiabilidad "necesaria", desarrollo de perfiles operacionales, preparación para la prueba, la prueba y la aplicación de los datos de fallas apara la toma de decisiones, esto se muestra en la figura 3.1

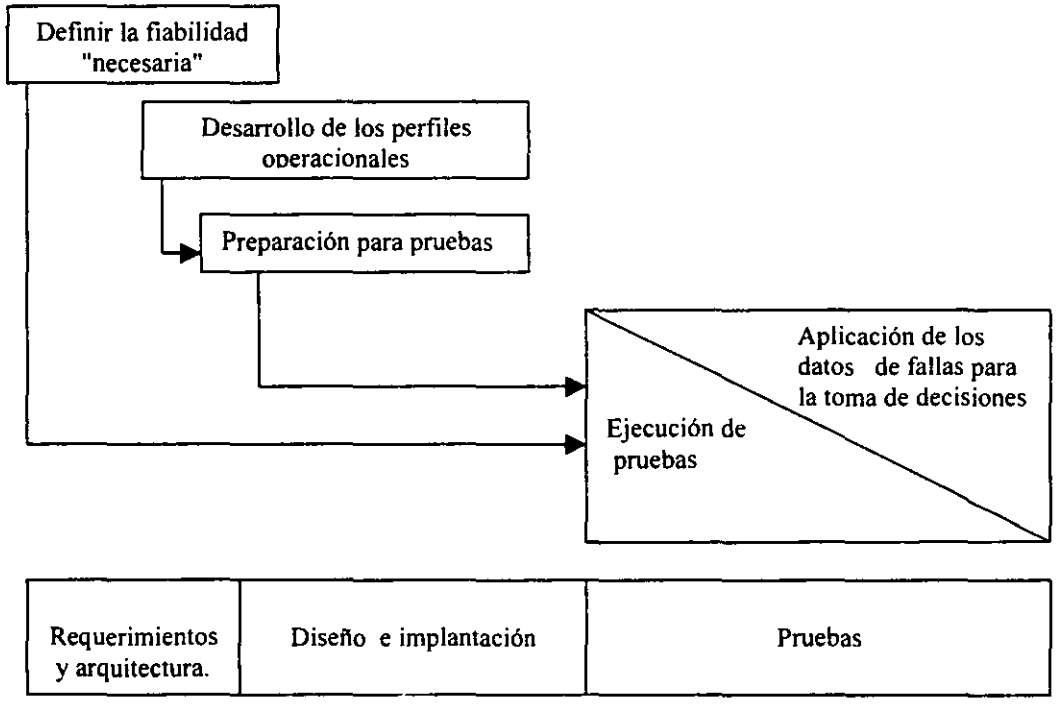


Figura 3.1 Diagrama de procesos de la ingeniería de fiabilidad de software.

Definir la fiabilidad necesaria

Definir lo que significa la fiabilidad necesaria de un producto en términos cuantitativos es uno de los pasos principales para lograr los beneficios de la IFS. La definición cuantitativa de la fiabilidad hace posible el balanceo de las necesidades de fiabilidad del cliente, la fecha de liberación, el cálculo del costo y el desarrollo y prueba de los productos de una forma más efectiva.

Desarrollo de perfiles operacionales

El desarrollo de perfiles operacionales provee información acerca de cómo los usuarios van a emplear le producto que se está desarrollando, por lo tanto se puede poner más atención a los recursos de desarrollo y prueba. Con esta información se puede mejorar sustancialmente la eficiencia del desarrollo y de las pruebas. Como una ventaja más, se pueden hacer pruebas más realistas.

Preparación para pruebas

En la fase de preparación para pruebas se aplica la información que se tiene del perfil operacional que se ha desarrollado para hacer más eficientes las pruebas. Desde la perspectiva de la IFS, la preparación para pruebas incluye los casos de pruebas y los procedimientos de prueba. Se debe preparar para cada sistema del producto que se va a probar. Sin embargo, se puede tomar ventaja de las cosas en común que pueden existir. Las prueba Beta no requiere de mucha preparación, excepto por el trabajo que implica la recolección de resultados, dado que se expone el producto directamente con al uso del o de los posibles clientes.

La IFS ayuda en la elaboración de las pruebas de característica, de carga y de regresión. La prueba de características se realiza primero. Esta consiste en ejecuciones simples de operaciones, con interacciones entre las operaciones minimizadas. El punto consiste en que si las operaciones se ejecutan adecuadamente. Es seguida por la prueba de carga, la cual intenta representar el uso y el medio ambiente en el campo de trabajo tanto como sea posible, con operaciones ejecutándose simultáneamente y con interacciones. Las interacciones pueden ocurrir directamente, a través de la lenta corrupción de base de datos o como resultado de conflicto por los recursos. La prueba de regresión consiste en la prueba de característica que es controlada después de cada modificación que implique cambios significativos. Esta prueba es periódica; generalmente una semana es el periodo estándar para realizarla, sin embargo los periodos pueden ser tan cortos como un día o tan largos como un mes. El periodo a escoger generalmente depende de factores tales como el tamaño del sistema, la volatilidad

y el grado en el cual el sistema debe de estar listo para una liberación rápida cuando las condiciones del mercado cambien. El objetivo es encontrar fallas que podrían haber sido introducidas en el proceso de cambio.

Algunas de las personas que prueban los sistemas (testers) dicen que la prueba de regresión debería enfocarse en las operaciones las cuales fueron participe del cambio de código. Este punto de vista tiene sentido si se esta seguro que los posibles efectos del cambio de código están aislados. Para esas operaciones o si los requerimientos de fiabilidad del sistema a son bajos, por lo tanto, los efectos que tenga en las otras operaciones no importan. Sin embargo, en la mayoría de los casos no se puede depender del aislamiento y de los potenciales efectos que puedan causar un deterioro inaceptable en la fiabilidad del sistema. Por ende, se deben de considerar todas las operaciones cuando se realiza la prueba de regresión. Sin embargo, un cambio siempre tiene menos probabilidades de falla que un programa totalmente nuevo, por eso no es tan necesario el probar cada operación después de cada cambio en el programa. Es poco eficiente el realizar pruebas en operaciones con pocas probabilidades de fallas con la prueba de regresión. Por eso las operaciones que se realicen en la prueba de regresión deben de ser seleccionadas de acuerdo con el perfil operacional.

En la prueba de carga es necesario probar cada perfil operacional de forma separada, de esta forma se estará tomando muestras de las posibles interacciones entre las operaciones que pueden ocurrir e ignorando las que no. En consecuencia, se deben de tomar muestras de y solo de las fallas que pueden ocurrir debido a las interacciones. Es más eficiente el planear la prueba de desempeño durante la prueba de carga. La prueba de desempeño es más realista si se basa en el perfil operacional; la única tarea extra que requiere es el hecho de planear la recolección y registro de los datos de desempeño como los tiempos de respuesta y el ordenamiento de grandes cantidades de datos.

Ejecución de pruebas

Cuando se ejecutan las pruebas se utilizan los casos de prueba y los procedimientos de prueba desarrollados durante la fase de preparación para pruebas, en la cual se plantea el nivel de eficiencia que se desea en las pruebas. La ejecución de pruebas implica tres actividades principales: asignación del tiempo de prueba, aplicación de pruebas e identificación las fallas que ocurren.

Aplicación de los datos de fallas para la toma de decisiones

Se aplicarán los datos de fallas recolectados para cada sistema en forma separada. Estos ayudaran a tomar muchas decisiones, listados de forma cronológica:

1. Aceptación o rechazo de un componente obtenido.
2. Dirigir el proceso de desarrollo de software para el producto y sus variaciones
3. Aceptación o rechazo de un supersistema
4. Liberación del producto

Las fases del proyecto en las que se puede personalizar se muestran en la parte inferior de la figura (3.1). Nótese que las fases de Ejecución de pruebas y La aplicación de datos de fallas para la toma de decisiones se producen en forma simultánea y están estrechamente ligadas.

El diagrama del proceso, por simplicidad, solo muestra el orden predominante del flujo de trabajo, aunque en la actualidad los procesos iteran y se retroalimentan, - formando un modelo de espiral, en contraste con el modelo de cascada -; por ejemplo cuando se modifican algunos requerimientos y algo de la arquitectura puede seguir, inmediatamente de esta fase, las pruebas en el proceso de desarrollo del software, cambios en la definición de la fiabilidad "necesaria" pueden ser seguidos por la ejecución de pruebas y la aplicación de datos de fallas para la toma de decisiones.

La fase de mantenimiento que sigue después de las pruebas no es mostrada en la figura anterior (3.1); Durante esta fase se puede determinar la fiabilidad alcanzada y el verdadero perfil operacional experimentado. Esta información afecta la fase de la definición de fiabilidad "necesaria" y la definición del perfil operacional para la siguiente liberación del producto.

La gente encargada de probar el sistema (tester) conduce las primeras dos actividades (Definir la fiabilidad "necesaria" y desarrollo del perfil de operación). Originalmente se pensaba que estas actividades se tenían que dejar exclusivamente a los ingenieros de sistema y a los arquitectos de sistema, cosa que no funcionó bien en la práctica; la gente que prueba el sistema (tester), quienes dependen de estas actividades está, por lo tanto, más motivados que los ingenieros de sistemas y los arquitectos del sistema para que las cosa salgan bien. Este problema se puede resolver fácilmente haciendo a estas personas parte del equipo de ingenieros y arquitectos de sistemas.

Este método ha traído beneficios inesperados, las personas que prueban los sistemas tiene mucho más contacto con los usuarios del producto, que es muy valioso para saber que comportamiento esperan del sistema, que comportamiento del mismo sería inaceptable y para entender como van a utilizar el producto los usuarios finales. Los ingenieros y arquitectos de sistema obtienen una mejor apreciación de las pruebas del sistema y de donde los requerimientos y el diseño deben de ser menos ambiguos y más precisos, entonces pueden desarrollarse la planeación de las pruebas, los casos de prueba y los procedimientos de prueba. Las personas que prueban los sistemas hacen importantes contribuciones al análisis de la arquitectura señalando importantes funciones que fueron omitidas.

La fase de la arquitectura del sistema incluye la actividad de la ingeniería de fiabilidad de software de seleccionar la combinación de estrategias para la prevenir y remover las fallas, así como el margen de tolerancia de las mismas. Por lo tanto esto afecta al diseño del producto y del proceso. La gente que prueba los sistemas no realizará esta tarea, pero necesitan entenderla pues les afecta en el desarrollo de su trabajo.

3.3 Tipos de pruebas

Hay dos tipos de pruebas en la Ingeniería de Fiabilidad de Software: la prueba de crecimiento de fiabilidad y la prueba de certificación. Estas pruebas no están relacionas con las fases de pruebas como: la unidad de prueba, la prueba del subsistema, la prueba del sistema, o la prueba beta. El objetivo principal de la prueba de crecimiento de fiabilidad es encontrar y remover fallas, durante esta prueba se utiliza la ingeniería de fiabilidad de software para calcular y rastrear la fiabilidad. Los gerentes de desarrollo y la gente que prueba los sistemas utilizan la información de la fiabilidad para guiar el desarrollo y la liberación del producto. Generalmente se utiliza la prueba de crecimiento de fiabilidad para la fase de prueba del software que se esta desarrollando en la empresa. También se puede utilizar en las pruebas beta si se está resolviendo las fallas (removiendo los errores que las producen). Para obtener una buena estimación - con los rangos moderados de incertidumbre - de la intensidad de fallas se necesita un número mínimo de fallas en el muestreo, generalmente de 10 a 20.

La prueba de crecimiento de fiabilidad incluye las pruebas de características, de carga y de regresión. La prueba de características es aquella en la cual las operaciones son ejecutadas de forma separada con las interacciones y efectos del medio ambiente de campo minimizados. Algunas veces

la iteraciones son minimizadas reinicializando el sistema entre las operaciones. La Prueba de carga involucra la ejecución de las operaciones en forma simultanea con las mismas condiciones del medio ambiente de campo donde se utilizará el software, por lo tanto se tendrán las mismas interacciones e impacto del medio ambiente que se tendrán en el campo. Las pruebas de desarrollo y aceptación son tipos de pruebas de carga. Las pruebas de Regresión son la ejecución de alguna parte - usualmente seleccionada de forma aleatoria - o todas de la prueba de características después de que el sistema construido ha tenido ha sufrido cambios significativos, se deben de incluir todas las operaciones críticas en el conjunto de pruebas de regresión.

La Prueba de Carga generalmente involucra la competencia por los recursos del sistema los problemas de tiempo y espera que puedan surgir, además de que hay una degradación de los datos con el tiempo. Los datos precedentes pueden descubrir fallas potenciales resultando de la interacción que no pueden ser simuladas por las pruebas de características y de regresión. De cualquier forma. Las interacciones son algo importante para los sistemas multiusuarios, también son importantes para los sistemas de un solo usuario, como los programas que se ejecutan en las computadoras personales, pues pueden tener diferentes interacciones entre las operaciones, dependiendo del orden en el que se ejecuten estas.

La Pruebas de Certificación no incluye el depuramiento, lo que significa que no hay intento por resolver las fallas que se detecten, determinando los errores que las provocaron y removiendo los errores. El sistema debe de estar estable, no debe de haber cambios debido a nuevas características o por remover las fallas. En la prueba de certificación se tiene que hacer una decisión puramente binaria: se acepta o no el software y si se regresa para que se vuelva a trabajar en el. En la prueba de certificación se requiere muy pocos ejemplos de fallas, de hecho se pueden tomar decisiones sin que se presente una sola falla en el periodo de ejecución del software. Generalmente solo se usa la prueba de Certificación para las pruebas de carga.

3.4 Sistemas a probar

Se puede definir sistema de cualquier forma que sea conveniente para el propósito de estudio o análisis. Este puede consistir en una combinación de elemento de hardware, software y personal. En general, se definirá como sistema a cualquier entidad que se va a probar de forma separada.

Obviamente se desea probar el sistema que se está desarrollando actualmente. Se usa la prueba de crecimiento de fiabilidad mientras se este desarrollando cada parte del producto. Esta puede ser seguida por la prueba de certificación para verificar si el cliente acreditase una prueba de aceptación. Si simplemente se esta integrando el producto solo se realizara la prueba de certificación del producto integrado. También se desea identificar las mayores variaciones del producto posibles como sistemas a probar. Una configuración de hardware diferente del producto (que realiza las mismas funciones) es prácticamente un sistema diferente. Se pueden tener sustancialmente diferentes versiones por que se tiene la capacidad de ejecutarlo en diferentes plataformas o en diferentes sistemas operativos. Los sistemas internacionales pueden tener diferentes interfaces en diferentes países.

Si se tiene supersistemas o sistemas donde el producto se comporta como componente, se pueden considerar como sistemas potenciales para ser probados de forma separada. Si los usuarios analizan el producto tomando la fiabilidad de un supersistema y la interacción de los componentes con los otros sistemas que comprende el supersistema es complicado y por lo tanto difícil de definir, entonces se querrá probar el supersistema de forma separada. Esto es por que la pruebas independientes del producto con un controlador sería aleatorio, dado que no hay la probabilidad de representar de forma adecuada la interfaz. Generalmente se prueban supersistemas en paquetes de software como sistemas operativos completos, paquetes de oficia, computadoras personales o impresoras. Si se prueban uno o más supersistemas que involucra el producto, probablemente se quiera probar los supersistemas para las variaciones de ese producto. Muchos supersistemas no puedes ser - por fines prácticos o económicos – probados como sistemas separados, pero si en la prueba Beta y por lo tanto deben de ser organizados para esa prueba. La prueba Beta, por supuesto, implica que se pruebe directamente en un ambiente de campo, identificar las fallas y aplicar los datos de las fallas. Las pruebas Beta no necesitan que se desarrollen perfiles operacionales, preparar casos y procedimientos de pruebas o realizar los casos de pruebas con los procedimientos de los mismos.

Probablemente se querrá probar componentes de un *software adquirido* (el que no es desarrollado por cuenta propia) como un sistema si su fiabilidad es cuestionable o desconocida y si una detección temprana puede prevenir costos extras o retraso en la entrega. Generalmente las pruebas son necesarias para la primera versión de un producto, aunque podrían ser necesarias para subsecuentes liberaciones si esa nueva versión contiene nuevos elementos o sufre cambios sustanciales. Nótese que el tamaño es un factor determinante en la decisión de probar o no los componentes. Si los componentes son más de uno esto tendrá costo económico considerable. Si se tiene un software adquirido de componentes pequeños, valdrá la pena probarlo si se va a usar en varios sistemas.

Parece haber un gran potencial en la aplicación de la IFS para probar librerías de objetos, necesarias para el desarrollo orientado a objetos, dado que se desea utilizar estos objetos en varios sistemas. De hecho el crecimiento en un futuro de la reutilización de los objetos depende de la unión de estas dos tecnologías. Los conceptos de objetos nos permiten una mejor modularización, pero actualmente existe mucha desconfianza en reutilizar objetos de fiabilidad desconocida.

Por supuesto que hay un límite para seleccionar el número de sistemas a probar. Cada prueba extra tiene un costo. Aunque se puedan conducir múltiples pruebas en paralelo, en un punto los recursos humanos y técnicos entrarán a una fase de saturación, lo que ocasionará retrasos en los planes. El costo de una prueba incluye el desarrollo de uno o más perfiles operacionales, desarrollo de casos y procedimiento de pruebas. Se debe de seleccionar un sistema para su prueba por separado solo cuando los beneficios de probarlo en forma separada sean mayores que el tiempo y el dinero invertido. Los beneficios son la reducción de la probabilidad de insatisfacción por parte del cliente, o retraso en los planes (ambos conllevan a la perdida de mercado) y la reducción de riesgo de costo por desarrollo extra.

El costo de probar de forma separada un sistema puede ser reducida si el sistema tiene una arquitectura operacional (sus componentes corresponden a operaciones o grupos de operaciones) porque es relativamente simple el obtener los perfiles operacionales para los componentes y por lo tanto es fácil separar los componentes para probar.

Aunque no podría ser eficiente en términos de costos el probar más de unos pocos sistemas de forma separada, no es tan caro y por lo tanto puede ser practico determinar la intensidad de fallas para un número largo de componentes que son probados en forma conjunta. Esto es porque solo se tiene que clasificar las fallas por componente y medir los procesos realizados por esos componentes (ya sea en unidades naturales o de tiempo que dura la ejecución).

De cualquier forma, debido a los errores de estimación de fallas en las muestras pequeñas, hacer cálculos de la intensidad de estas no es del todo práctico.

3.5 La relación de la Ingeniería de Fiabilidad del Software con la Calidad del Software

La IFS está muy ligada a la calidad del software (La calidad del software se tratará con mas detalle en el capitulo siguiente). La fiabilidad el software es una propiedad específica y medible dentro de las amplias definiciones de la calidad del software, la cual es ampliamente confundida. Es probablemente una de las propiedades que más involucra y es una de las más importantes por que trata con libertad las diferencias de comportamiento de las especificaciones del usuario que se presenten durante la ejecución. En otras palabras, las diferencias que se presentan de como quiere el usuario que se ejecute el producto.

La fiabilidad del software es la parte más importante de la calidad para el usuario porque cuantifica que tan bien va a funcionar le producto de software con respecto a sus necesidades. Otras cuantas medidas de calidad tienen cierta relación con la fiabilidad el software, pero esta es de forma indirecta. Por ejemplo el número de errores restantes en el producto tiene cierta conexión con la fiabilidad, pero esta totalmente orientada al desarrollador y no indican impacto en la operación. Aún más no se pueden medir el número de errores restantes sino tan solo inferir, y esto con un grado de precisión muy pobre. El número de fallas encontradas no tiene relación alguna con la fiabilidad. Si sólo son encontradas unas cuantas fallas, esto puede indicar un software confiable o un software al que se le aplicaron pruebas muy deficientes, por lo tanto es un software poco fiable. El número de errores descubiertos durante el diseño o durante la fase de inspección de código tiene propiedades similares a as fallas encontradas en términos de medidas de calidad. Otras medidas como la complejidad del programa están aún más lejos del concepto de calidad del usuario.

La IFS esta totalmente integrada, al grado que se podría decir que es la piedra angular de la Administración Total de Calidad (TQM), dado que esta provee una medición orientada al usuario que esta altamente relacionada con la satisfacción del cliente. Es la piedra angular en el sentido de que no se puede manejar la calidad sin un concepto de fiabilidad del sistema que este enfocada al usuario, y no se puede definir la fiabilidad de un sistema basado en software si no se tiene una medida de fiabilidad de software.

Capítulo IV
Modelos de Calidad del Software

4.1 *Introducción*

En los capítulos anteriores se trataron los temas de fiabilidad de software, modelos de fiabilidad de software, y la ingeniería de fiabilidad de software. En este último tema se tocó la relación que tiene la IFS con la calidad de software. Se puede decir que el fin último de la interacción entre estas disciplinas es hacer un software con calidad. El objetivo de todo esto es satisfacer las necesidades del cliente, y esto se lleva a cabo solo proporcionándole un producto de calidad.

Las propuestas para los modelos de calidad de los productos de software han tenido un éxito muy limitado, unas de las causas son las siguientes:

- El factor principal es que los modelos propuestos no han sido soportados por una teoría de tipo empírico ni por los modelos que son encontrados más comúnmente en el ambiente científico.
- El segundo factor es que estos modelos no tienen ni el conocimiento necesario o no han explotado el hecho que el software tiene un conjunto de *comportamientos* y *usos* que corresponden directamente a sus atributos de calidad de alto nivel. Lo que es interesante es que estos comportamientos y usos apuntan directamente a las necesidades de los grupos interesados que están relacionados con el software.
- El tercero es que las estrategias definitorias y de construcción empleadas no han sido disciplinadas para satisfacer los cambios de las tareas.
- Finalmente, hay un gran número de propiedades tangibles del software que son conocidas por influir positivamente en su calidad pero esas propiedades nunca han sido organizadas en un cuadro sistemático de trabajo que pueda maximizar el impacto de su peso acumulado.

El punto aquí es que cuando esas materias sean adecuadamente atendidas será posible construir un modelo práctico de calidad de un producto de software. A lo largo de este capitulo se empleará para desarrollar un modelo de calidad de un producto software una estrategia de "meta – requerimientos directos – diseño – implantación" que se concentrará en esos puntos. Se ha decidido tomar esta estrategia por dos razones: primeramente hay un número diferente de grupos interesados que tienen requerimientos de software muy distintos. El modelo se tiene que adaptar adecuadamente a esos requerimientos; como segundo punto, esta estrategia va a ser relativamente fácil para la gente que está involucrada con el software dada su amplia aplicación en el desarrollo del software.

La primera tarea para construir un modelo de calidad de un producto de software es identificar para cuales aplicaciones es el modelo y dirigir las necesidades de los diferentes grupos interesados que usarán los modelos en las diferentes aplicaciones.

La segunda etapa en la construcción de un modelo de calidad para un producto de software es identificar una arquitectura / diseño para el modelo.

La figura 4.1 encierra la alta arquitectura del modelo:

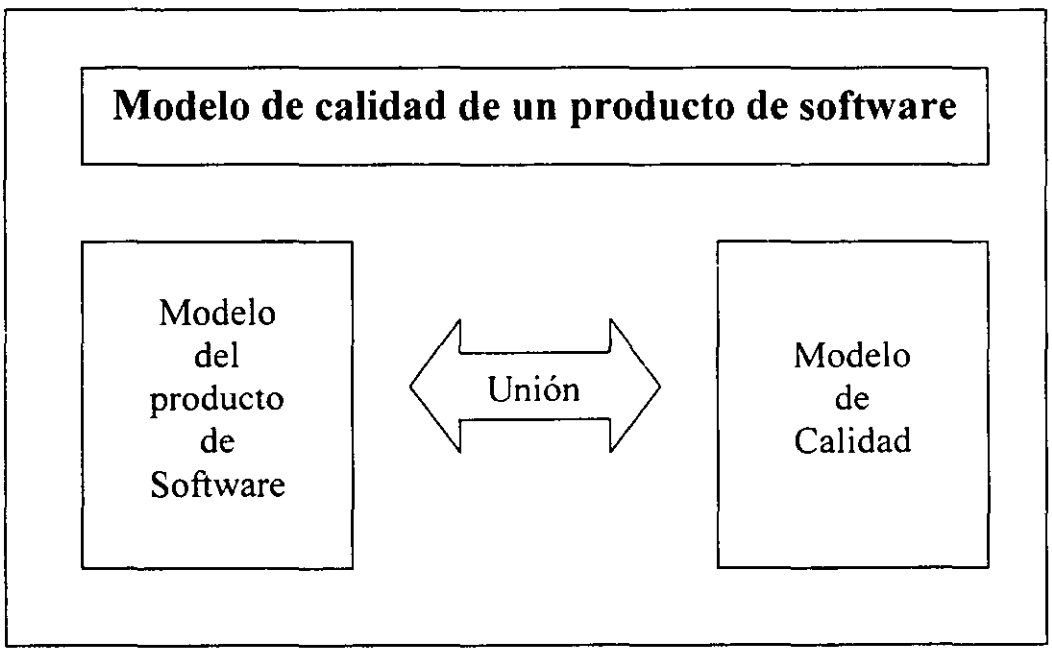


Figura 4.1 Modelo de calidad para el software

Entonces los pasos a seguir son tres:

- Construir el modelo del producto de software.
- Construir el modelo de calidad.
- Unir los modelos del el software y de calidad para construir el modelo de calidad de un producto de software.

Una vez que se ha construido el cuadro de trabajo que define y soporta un modelo de calidad de un producto de software, el paso final es definir la estrategia de parar la implantación de este modelo.

4.2 *Diseño de un modelo de calidad para el software*

El diseño para un modelo de calidad del software se realiza construyendo uno de calidad, uno de software y después uniendo estos dos para producir el modelo de calidad del software. A continuación se describirá la forma de llegar a este último.

Modelo de Calidad

Cabe aclarar las acepciones diferentes que se le dará aquí a los términos de características determinable y determinante. Las características tienen diferentes niveles de determinación. Por ejemplo, la característica de ser colorido es determinable y la característica de ser rojo es un determinante de la primera. El rojo es aún determinable, tiene bajo él determinantes, rojo escarlata, rojo ladrillo, etc. Ser un mamífero es una característica compleja, ser gato o perro son dos determinantes de esta característica determinable. Cuando dos objetos se asimilan uno al otro en muchos aspectos se puede decir que las características determinables determinan a los dos, pero no con los mismos determinantes, como se vio dos objetos pueden tener dos clases distintas de rojo.

Se tiene que ampliar el concepto de determinables y determinantes para incluir la noción de incremento de determinables a satisfacer. Esta extensión nos permite sostener que: A mayor determinantes de fiabilidad que sean satisfechos mayor será la fiabilidad del sistema.

Se sugiere que se puede usar una estrategia constructiva que caracterice los *comportamientos* y *usos* del software que contribuyen a su calidad. En la construcción de un modelo de calidad de software es bueno utilizar tanto la estrategia bottom – up como la de top - down. Se buscará

enumerar propiedades concretas y calificarlas como si fuesen características del software, y de regreso en el siguiente nivel hacia arriba se buscará enumerara las características del software que caractericen cada *comportamiento* y cada *uso* – esto corresponde a la estrategia bottom – up. También se utiliza la derivación y/o descomposición para que representen o definan propiedades abstractas (*comportamientos y usos)* en términos de comportamiento, usos y características secundarias del software – esto corresponde a la metodología top – down. Las dos metodologías antes mencionadas tienen un papel importante en el desarrollo del modelo de calidad de software.

Se empezará considerando los dos principios que guían la descomposición y la derivación.

Principio 1

Un *comportamiento* puede ser descompuesto y por lo tanto definido en términos de propiedades secundarias las cuales pueden ser descritas ya sea como *comportamientos* o como características del software.

La figura 4.2 ilustra lo anterior de forma esquemática.

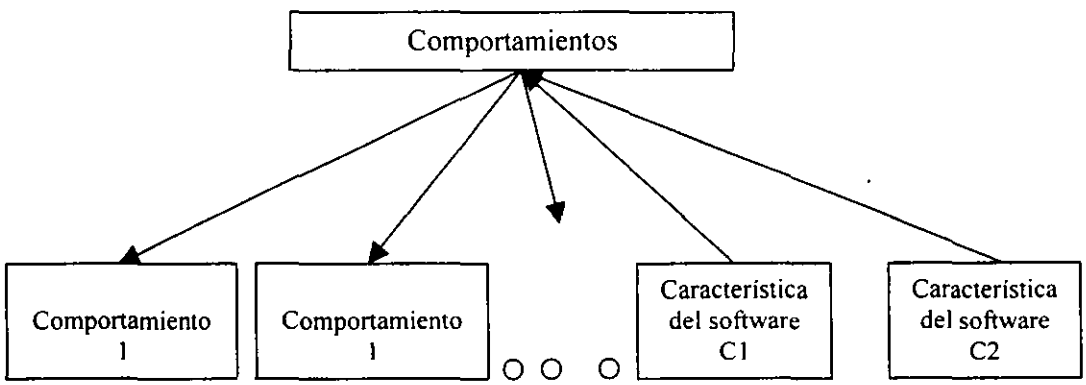


Figura 4.2 Esquema del comportamiento

Si se sigue este principio el reto que se afronta es el de dónde dibujar la línea entre los comportamientos secundarios definidos y las contribuciones de las características del software (en el diagrama anterior se nota la diferencia de la dirección de las flechas, lo cual intenta mostrar las diferencias entre las relaciones casuales de definición y de contribución. La fiabilidad, que es un atributo de calidad, es muy útil para entender. Las propiedades abstractas como la tolerancia a las fallas y la capacidad de recuperación son

determinantes de la fiabilidad. El punto es que, tanto la tolerancia a las fallas como la capacidad de recuperación define los comportamientos de la fiabilidad o las características del software que contribuyen a la fiabilidad. Lo que se esta enfrentando aquí es la tensión entre la estrategia top – down en la definición de los atributos de calidad y la caracterización bottom – up (descripciones abstractas) o la clasificación de las propiedades del software (propiedades funcionales y no funcionales) que contribuyen a los atributos de calidad de alto nivel. Ya sea que se refiera a la tolerancia de errores como un comportamiento o como una característica del software realmente no importa demasiado. Lo que es importante reconocer es que hay dos formas de llegar la misma meta. Algunas propiedades como la modularidad o la exactitud son claramente características del software más que los *comportamientos* o los *usos*. Con otros como la tolerancia a las fallas no es tan claro hacer la distinción. La intuición que se usa para hacer tales distinciones es que los *comportamientos* tienden a ser cualidades en toda la extensión del sistema mientras las características del software se asocian con componentes particulares tanto como a todo el sistema. Se puede cerrar este punto usando descomposiciones de determinables y determinantes.

Como ejemplo se puede poner el ISO – 9126 la fiabilidad determinable puede ser descompuesta en las determinantes de: tolerancia a fallas, madurez y capacidad de recuperación, la figura 4.3 lo ilustra:

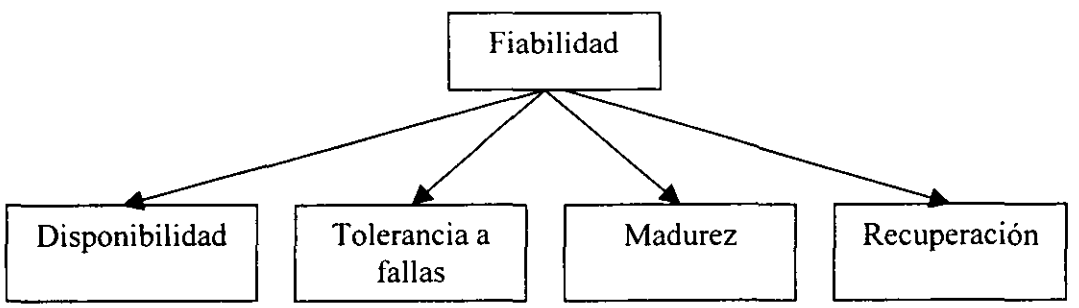


Figura 4.3 Partes de la fiabilidad determinable

El *uso* del software puede ser tratado en forma similar como el principio siguiente sugiere:

Principio 2

El uso puede ser descompuesto y por lo tanto definido en términos de propiedades secundarias las cuales se puedes describir tanto como *usos* o características del software.

La figura 4.4 ilustra este concepto en forma gráfica, los *usos* secundarios se pueden definir en forma jerárquica.

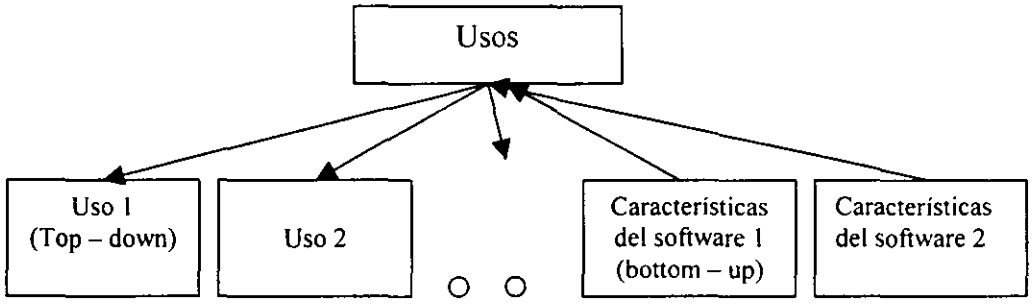


Figura 4.4 Partes del *uso*

Como ejemplo la capacidad de *uso* puede ser descompuesta en *usos* secundarios como capacidad de aprendizaje y operatibilidad.

Prosiguiendo en la construcción del modelo de calidad se aplicarán los siguientes principios de diseño, guías y suposiciones:

- Se escoge asociar propiedades abstractas, llamadas atributos de calidad, con el software
- La calidad del software puede ser representada por un conjunto de atributos de calidad de alto nivel
- Los atributos de calidad del software corresponden tanto a un conjunto de dominios de *comportamientos* independientes como a un conjunto de dominios de *usos* independientes
- Los atributos de calidad de un modelo deben de ser suficientes para satisfacer las necesidades de todos los grupos interesados asociados con el software

- Cada atributo de calidad de alto nivel del software es representado por un conjunto de propiedades secundarias las cuales son tanto *comportamientos*, *usos* o características del software
- Cada característica del software es determinada o ayuda a determinar mediante un conjunto de propiedades tangibles que se llamaran propiedades implícitas de calidad
- Las propiedades implícitas del software puede implicar tanto funcionalidad (verificar que el todas las entradas estén dentro de los rangos esperados, ayudan a cumplir el principio de diseño acerca de la protección modular)
- O propiedades no funcionales (los identificadores deben de ser auto descriptivos)

La figura 4.5 describe el modelo de fiabilidad que se propone:

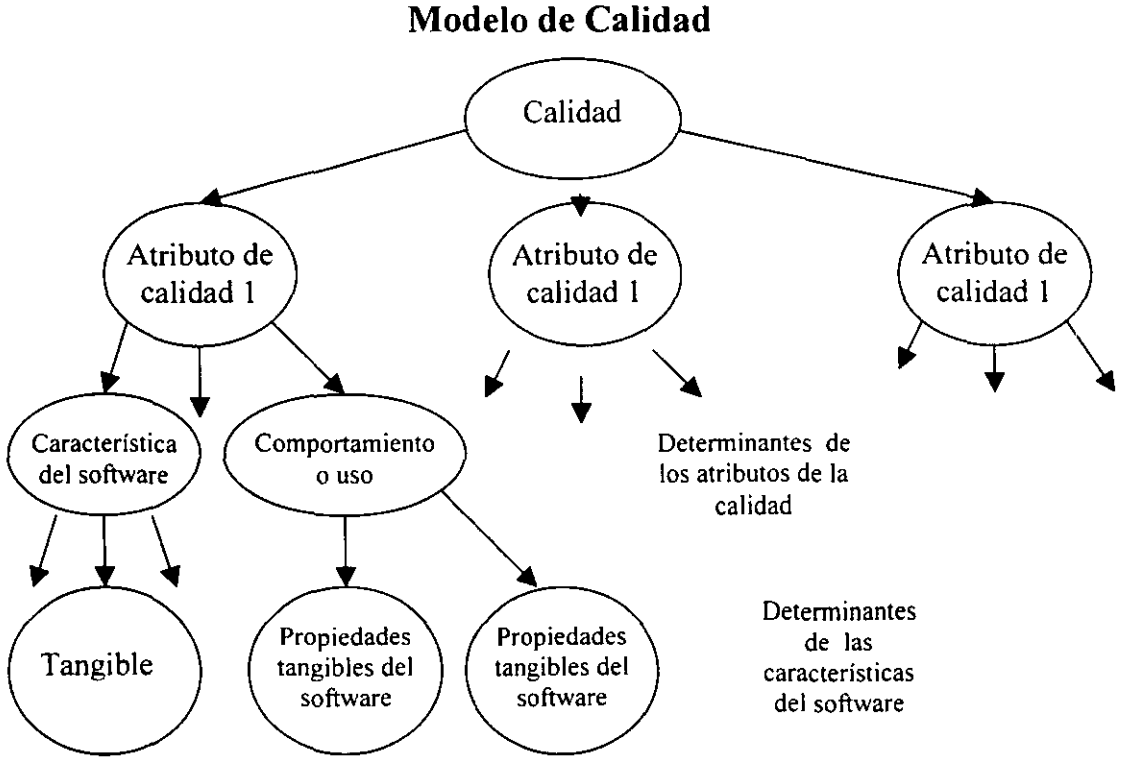
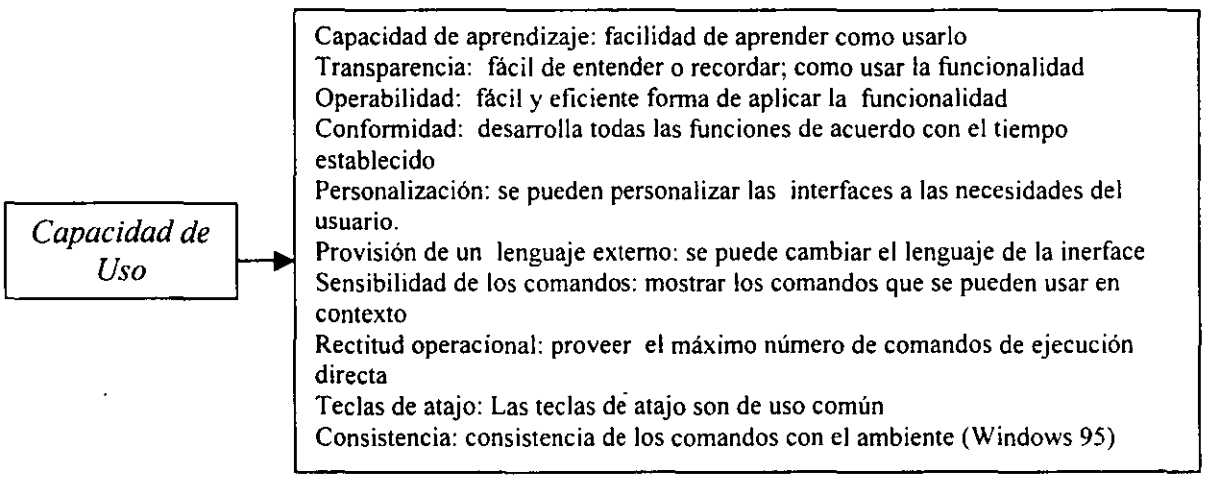


Figura 4.5 Modelo de Calidad

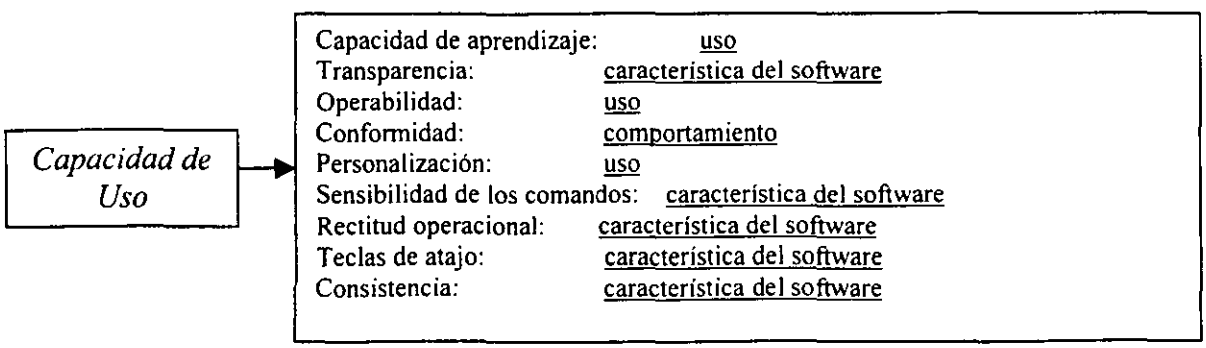
4.3 Ejemplos de Procesos

Capacidad de uso (Usability)

Para ejemplificar el proceso de definición y de representación se explorará de forma muy ligera un atributo de calidad: La capacidad de uso. Esto es sólo un bosquejo para ilustrar el proceso antes mencionado, no da una profunda y comprensiva metodología de la definición de las características de calidad. Nuestro punto de partida (se muestra abajo) es un conjunto que caracteriza / define o son manifestaciones de la capacidad de uso.



Para acomodar las propiedades anteriores en orden jerárquico usando la estrategia de definición / representación que se ha descrito primero se clasifican las propiedades ya sea como *comportamiento*, *uso* o *característica del software* de acuerdo a las definiciones dadas para esos términos (cuadro siguiente).



Los *comportamientos* y los *usos* son candidatos a ser los determinantes de alto nivel de la capacidad de uso. Se pueden tener unos comportamientos como subordinados del otro, etc. Para acomodar esto, es necesario preguntarse por cada propiedad, si esta contribuye a o determina cada comportamiento o uso. Por ejemplo, si la personalización contribuye a la facilidad de aprendizaje, a la operabilidad, y cosas por el estilo. Las características del software, en cambio representan comportamientos y usos. Una definición jerárquica que es resultado de todo este proceso es mostrada en la figura 4.6

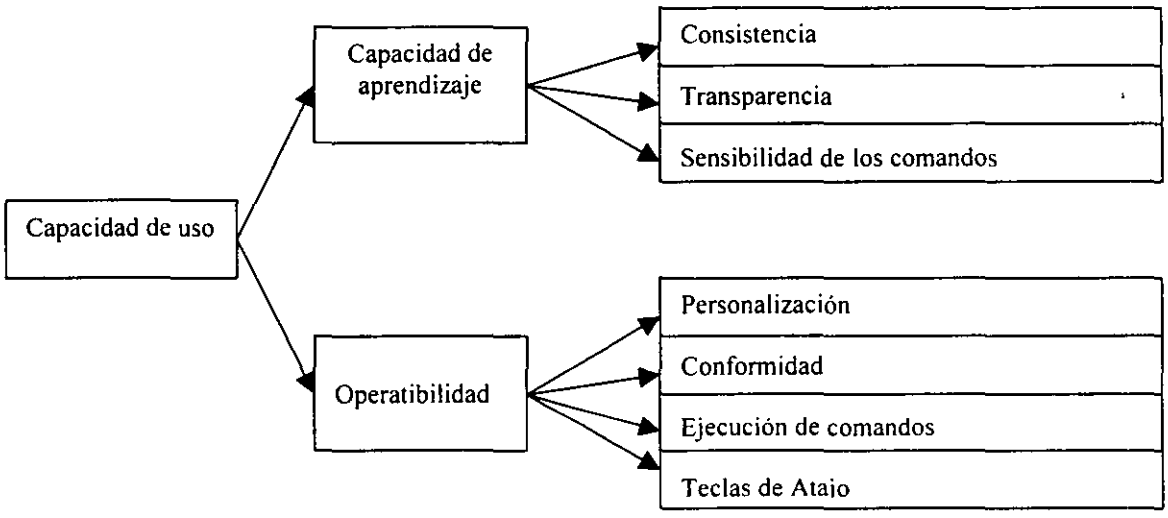


Figura 4.6 Capacidad de uso

Lo que se ha demostrado aquí es un largo método con pocas especificaciones compresivas de la capacidad de uso. De cualquier manera se espera, haber dado los suficientes detalles de lo que está vinculado en llevar a cabo como un proceso. Nótese que la propiedad de la provisión de un lenguaje externo que estaba en la lista original terminaría siendo un determinante de la personalización. Es por eso que no se muestra en la especificación jerárquica mostrada en la última ilustración.

Fiabilidad

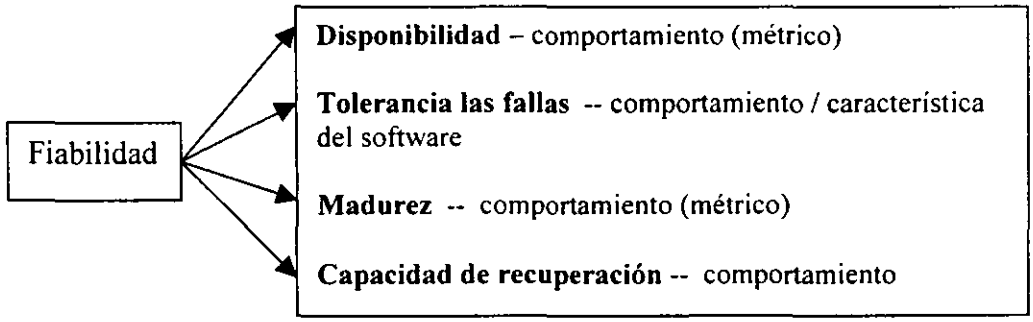
La representación del atributo de fiabilidad en la calidad, abarca muchos puntos importantes relacionados al diseño y la implantación de los modelos de calidad. La representación de cualquier atributo de calidad de alto nivel ayuda a formular y soportar un modelo funcional o modelo de uso (dependiendo si el atributo es un *comportamiento* o un *uso*) y un modelo métrico. Dichos modelos ayudan a minimizar el traslapo de comportamientos y direcciona los resultados completos.

Por su parte la fiabilidad se caracteriza como un sistema de software que se comporta en respuesta a cuatro tipos de errores:

(a) Una falla dirige a condiciones anormales que son generadas externamente y / o se muestran como entradas para el sistema de software que violan su capacidad para satisfacer el comportamiento funcional especificado
(b) Una falla que resulta en errores que ocurren durante el curso de la ejecución de un sistema de software que causan el desvío de su funcionalidad especificada y por lo tanto entrará en un estado anormal
(c) Errores que ocurren en las salidas de un sistema de software que son causadas por errores en la implantación de la funcionalidad
(d) Fallas que guían a errores del sistema de software

Un modelo válido de funcionalidad para la fiabilidad en este contexto, es decir que en los mas altos niveles hay tres clases de funcionalidad que no se traslapan que un sistema debe de incluir para mostrar un comportamiento fiable: detección de errores, reportes de errores (a través la visibilidad, que es un atributo de calidad secundario de la funcionalidad) resolución de errores. Lo antes mencionado trata sólo con situaciones donde el sistema es capaz de superar fallas (esto es, el sistema muestra el comportamiento de *tolerancia de fallas*) Hay otro tipo de funcionalidad necesaria para recuperar y reiniciar después de una falla del sistema del software (recuperabilidad). Dentro de cada uno de estos tipos de funcionalidad podría haber grados de variación en la sofisticación y estrategias alternativas para lidiar con esa situación.

Las medidas de fácil ubicación, que caracterizan a la fiabilidad de un sistema de software son los *comportamientos* maduros y su disponibilidad. La madurez puede ser interpretada como el promedio actual de los tiempos enlazados entre fallas sobre el tiempo de vida del sistema, mientras la disponibilidad es el promedio actual del tiempo en el que el sistema no está funcionando después de cada falla.



En este modelo funcional de tolerancia de fallas, por ejemplo, los abstractos superan en número a otros comportamientos y características del software como el diagnóstico de fallas, la capacidad de reconfiguración y la capacidad de supervivencia (el comportamiento donde un sistema necesita seguir operando cuando uno o más de sus subsistemas ha cesado su operación). En algunas aplicaciones un comportamiento como la capacidad de supervivencia puede ser como un requerimiento de calidad critico, por lo que hay una justificación para que crezca al mismo nivel que la fiabilidad aunque técnicamente es una forma muy especializada de fiabilidad.

A continuación se provee una posible implantación del modelo de calidad. Este contiene muchas similitudes con el de ISO-9126 (este último mostrado en el apéndice A) desde capacidad de volverse a usar en el más alto nivel hasta la inclusión de propiedades secundarias adicionales. Desde la funcionalidad hasta la capacidad de mantenimiento se habla de un ambiente simple en un sistema de hardware / software. Además se incluyen atributos de calidad que muestran consideraciones para su uso en otros ambientes.

Funcionalidad

Corrección
Seguridad
Interoperatibilidad
Visibilidad

Fiabilidad

Disponibilidad
Tolerancia a las fallas
Madurez
Capacidad de recuperación

Eficiencia

Procesador de economía
Recursos económicos
Comunicaciones económicas
Comportamientos en orden
Salidas

Utilidad

Capacidad de aprendizaje
Operatibilidad

Capacidad de Mantenimiento

Analizable
Modificable
Capacidad de probarse

Portabilidad

Independencia de la máquina
Independencia del sistema
Reemplazable
Instalable
Datos comunes

Reutilización

Independencia de representación
Independencia de la aplicación
Encapsular datos
Encapsular función
Interfaces

En resumen: Se tiene que empezar localizando a los grupos interesados y buscar sus requerimientos de calidad primarios. Sentando las necesidades de calidad de los diferentes grupos interesados es el mejor proceso. Una posibilidad de las especificaciones de las necesidades de alto nivel, aunque cuestionable, es: *Conveniencia* (o ajuste – por – propósito el cual es caracterizado por un conjunto de comportamientos del software) utilidad y adaptabilidad (la cual es caracterizada por un conjunto de usos del software por los diferentes grupos interesados. Abajo se muestra un mapa modificado / extendido de la estructura del ISO-9126. El principal interés de los clientes / patrocinadores está en la conveniencia de un sistema de software para sus propósitos. Los comportamientos *funcionalidad*, *fiabilidad* y *eficiencia* proveen una certera caracterización de alto nivel de la conveniencia.

En contraste los usuarios y los desarrolladores (gente encargada de mantenimiento) están interesados en la utilidad y adaptabilidad de un sistema de software, lo cual refleja los diferentes usos del software. Los usos de alto nivel, utilidad, portabilidad, y la capacidad de mantenimiento y la capacidad de volverse a utilizar son, generalmente, necesidades de calidad que tienen mayor prioridad en estos grupos interesados. Los patrocinadores también pueden estar interesados en lo que concierne a la capacidad de mantenimiento, etc. En algunos casos, esta será una será una característica secundaria para el grupo de los patrocinadores.

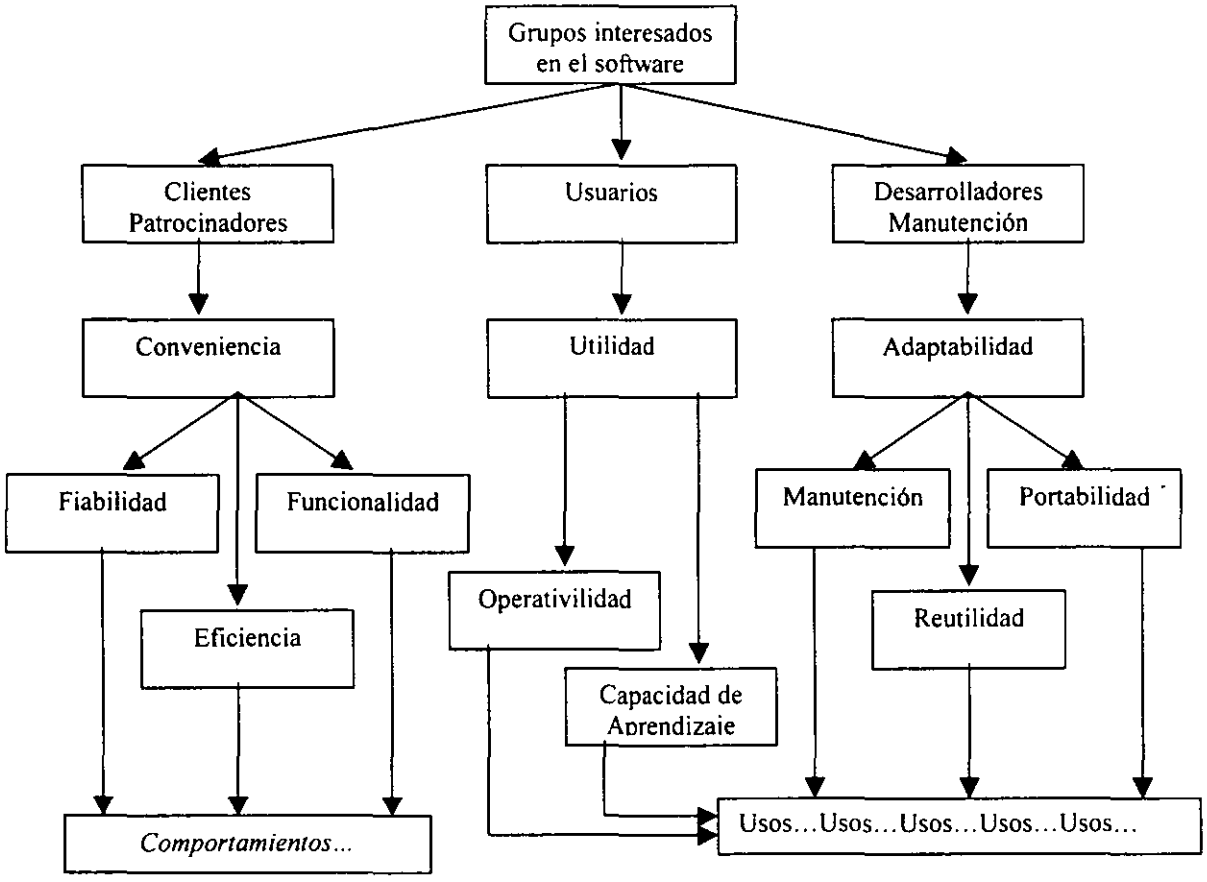


Figura 4.7 Modelo de calidad de software

Los modelos de calidad como el mostrado en la figura 4.7 no son ni absolutos ni fijos tanto en términos de escoger primero a los grupos interesados o en términos del conjunto de atributos de alto nivel de calidad que se establecieron. Si se hubiese escogido incluir a los auditores como un grupo de interés se hubiese tenido que agregar el atributo de alto nivel *verificable* que es un *uso* de alto nivel del software. El punto que se quiere demostrara es que la calidad necesita variar en diferentes contextos. De cualquier forma el cuadro de trabajo que se propuso es robusto y suficientemente flexible para acomodarse a variaciones, cambios y refinamientos.
Ahora se pasará a la segunda parte para construir el modelo de calidad del software.

4.4 *Modelo del software.*

En el centro de la construcción de modelo para el software hay dos suposiciones fundamentales:

1. El software está compuesto por dos cosas: *componentes simples o atómicos* y *componentes combinados*.
2. Lo componentes del software de varios tipos muestran propiedades tangibles que tienen impacto en la calidad del software.

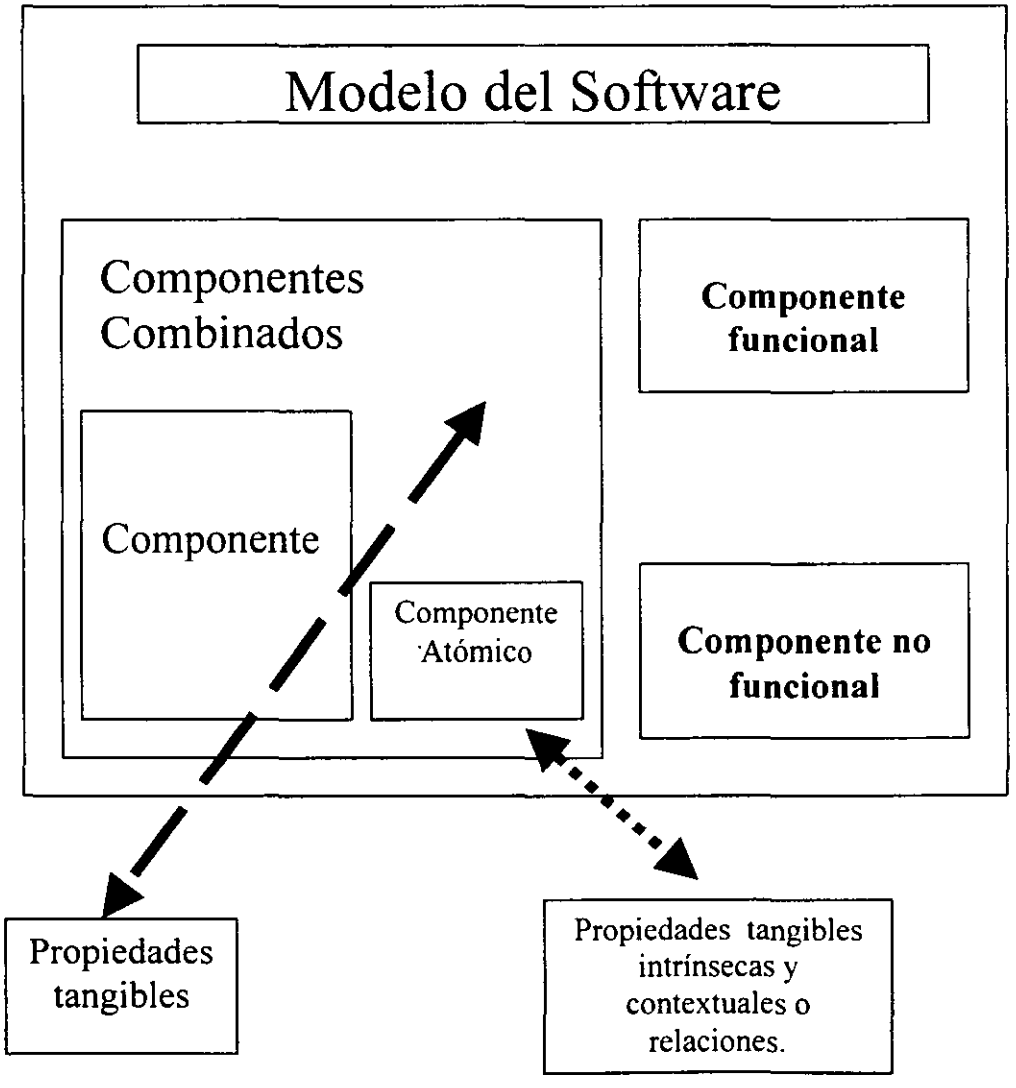


Figura 4.8 Modelo del software

El siguiente conjunto de enunciados representa un auxiliar en la construcción de un modelo del software:

- Los componentes incluyen varios tipos de enunciados los cuales se transforman en componentes combinados de bajo nivel como variables, expresiones, etc.
- Los módulos son componentes. Es conveniente tratar diferentes tipos de módulos como diferentes componentes.
- Los diferentes tipos de datos ya sean simples o combinados que fluyen entre módulos y subsistemas, y a través de interfaces son tratados como componentes distintos.
- Los componentes del software muestran propiedades concretas (incluyendo las métricas) que caracterizan la naturaleza del software, su forma y su estructura.
- El software muestra *comportamientos* y esta abierto a *usos* (ambos conjunto de propiedades son abstractas)
- Se escoge asociar las propiedades abstractas con el software, llamándolas características del software, para poder proporcionar descripciones de su naturaleza de alto nivel, sus *comportamientos*, sus *usos*, su soporte para los cambios, su forma y su estructura. Como ejemplo tenemos: Independencia de la máquina, modularidad y corrección algunas de las características del software pueden ser asociadas con los componentes del software.
- Los componentes del software muestran propiedades concretas que contribuyen a, o determinan las características del software.
- Las características del software pueden ser tanto genéricas, especificas del producto, especificas del lenguaje, o especificas del dominio.
- Los propiedades de calidad intrínsecas del software pueden ser genéricas, especificas del producto, especificas del lenguaje, especificas del dominio, funcionales o no funcionales. Las propiedades tangibles pueden ser intrínsecas/internas o contextuales/relacionales.
- Se relacionan las propiedades abstractas, llamadas atributos de calidad, con el software para representar su *comportamiento* y *usos* deseados.

En la figura 4.9 se plantea un posible esquema de clasificación para las características del software. En el interés por construir un modelo de calidad del software que sea fácil de comprender, es importante tener un cuadro de trabajo sobre el cual las características del software y sus propiedades tangibles definidas puedan ser adecuadas. Un cuadro de trabajo nos puede ayudar a sistematizar y estructurar nuestro conocimiento referente al software. También nos puede dirigir a áreas donde hay brechas, malas clasificaciones y otros problemas en nuestro conocimiento y entendimiento de las características del software. Durante la marcha se puede corregir, refinar y mejorar el cuadro de trabajo que mas se ajuste a nuestras necesidades.

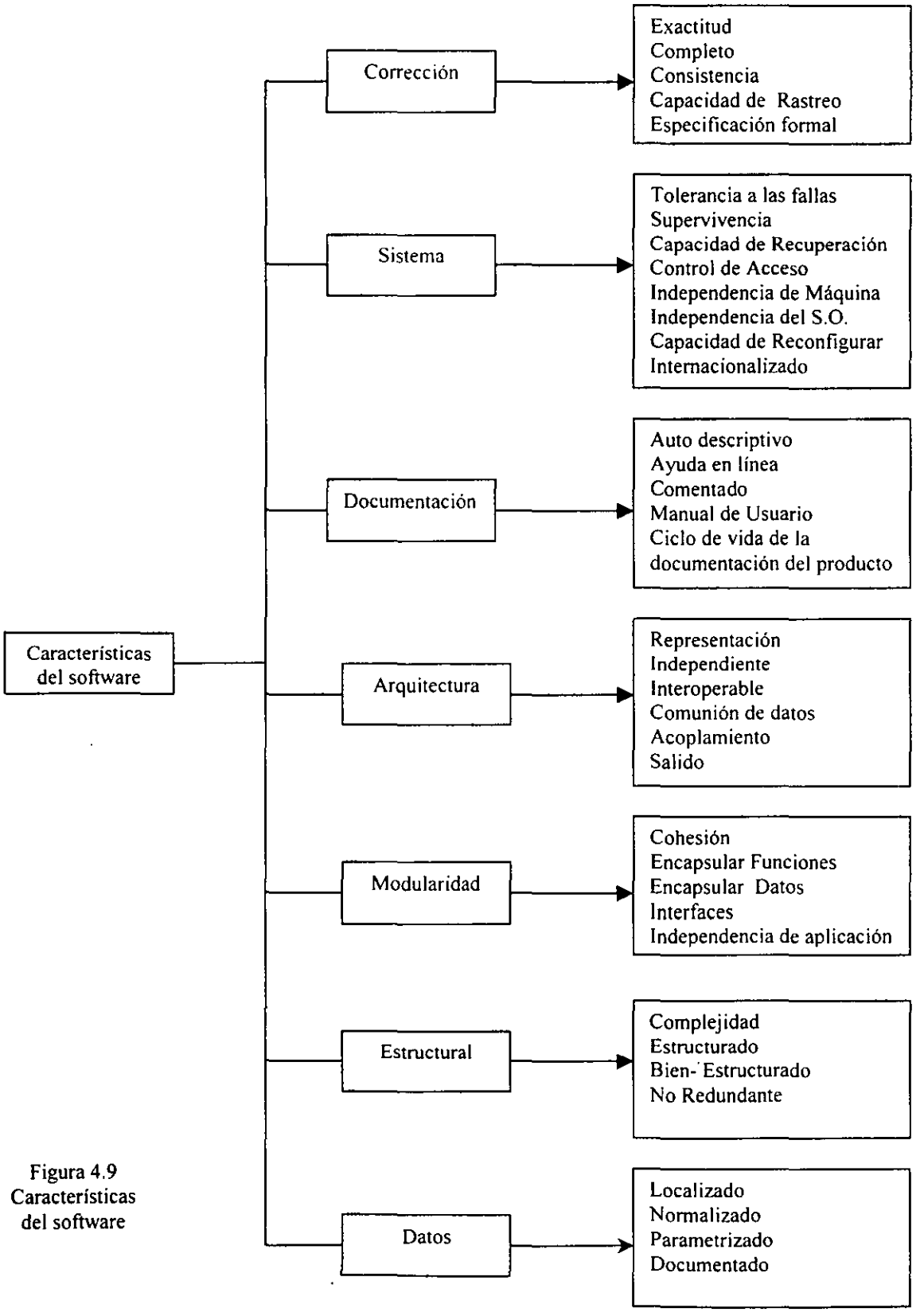


Figura 4.9
Características del software

4.5 Modelo de Unión – Axiomas para la calidad del software

Ahora que se tienen modelos de calidad y de software la tarea que falta es unir estos dos para crear el modelo de calidad del software. Para enfrentar esta tarea se necesita contestar la siguiente pregunta fundamental:

¿Que es aquello que determina la calidad del software?

La respuesta está incluida en el siguiente grupo de axiomas los cuales construyen los bloques del modelo de unión y la teoría empírica del modelo de calidad del software.

Primer Axioma de la calidad del software:

Si se admite que el software esta constituido de componentes, entonces la elección de esos componentes, sus propiedades tangibles, intrínsecas y contextuales y la forma en que esos componentes estén formados, determina la calidad del software.

El axioma puede ser representado gráficamente como lo muestra la figura 4.10:

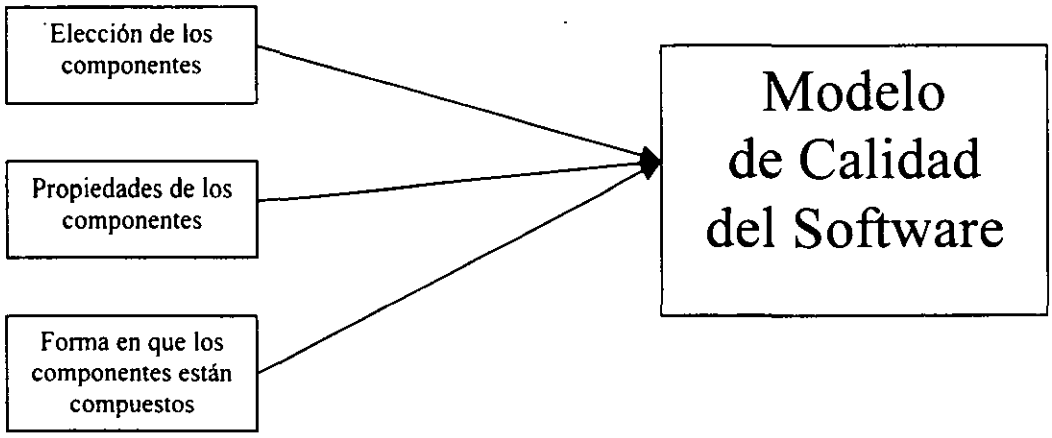


Figura 4.10 Representación del primer axioma

Ejemplo:

En el bajo nivel, variables, tipos, expresiones, ciclos, etcétera, son ejemplo de componentes. En el alto nivel el acceso a los módulos, los módulos de interfaces del hardware y del software y la comunicación de lo datos son ejemplos de componentes combinados.

Segundo Axioma de la calidad del software

El software muestra un conjunto de atributos de calidad y muestra ciertos comportamientos y usos observables que corresponden directamente a sus atributos.

Nuestro conocimiento y experiencia con el software nos permite definirlo y/o marcarlo como un cuadro de trabajo.

Ejemplo:

Una posible implantación, una argumentación del estándar ISO-9126, fue mostrada antes ya ahora se muestra nuevamente en la figura 4.11:

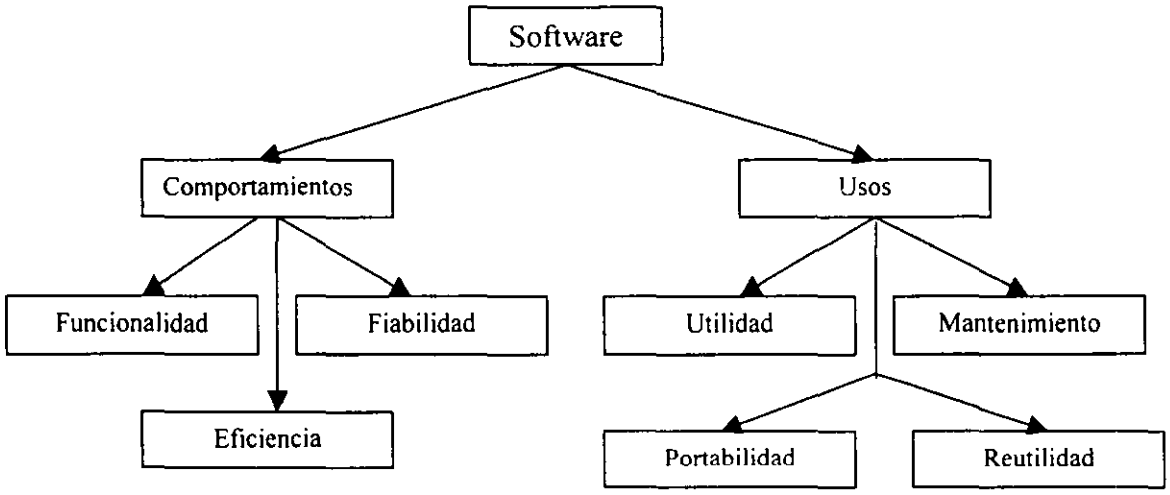


Figura 4.11 Implantación del estándar

Tercer axioma de la calidad del software

Las propiedades intrínsecas tangibles de calidad de los componentes del software, contribuyen a uno o más atributos intangibles de alta calidad del software.

Ejemplo:

La funcionalidad que verifica si un parámetro, que es usado como divisor, es diferente de cero, es un ejemplo de una propiedad intrínseca tangible la cual contribuye a la tolerancia de fallas y por lo tanto la fiabilidad de un sistema de software dado.

Cuarto Axioma de la calidad del software

Asociado con cada propiedad intrínseca tangible de un componente, está un enunciado empírico que une tanto a una característica del software, como a un comportamiento o a un uso, y por lo tanto a un atributo de calidad de alto nivel.

Ejemplo:

Identificadores auto descriptivos contribuyen a la capacidad de análisis y por lo tanto a la mantenimiento del software.

La experiencia de muchos ingenieros en sistemas, les hace aceptar este enunciado como verdad. De cualquier forma se pueden diseñar ciertos experimentos para probar su validez. Se puede hacer esto, por ejemplo, realizando dos implantaciones de un módulo auto – contenido que desarrolle una función bien definida. Una implantación podría usar exactitud, significado, nombres auto – descriptivos para todos los identificadores usados en el módulo; la segunda usaría identificadores como los de x, y, etcétera y no se dan indicadores, como su propósito. Se puede poner dos grupos con la misma información, los dos módulos y casos para probar. Por ejemplo, cuánto tiempo tardaron para deducir lo que hace el módulo, cual es el papel de cada variable, etc.

Lo que es importante de este cuarto axioma es que se asegura que la proposición para el modelo de calidad del software está construida sobre un conjunto de enunciados empíricos capaces de comprobarse. Tampoco va más allá como para asegurar que estos enunciados ya han sido comprobados, pero siempre existe la posibilidad de hacerlo. Por lo tanto hay una oportunidad tanto de validarlo como de refutarlo y corregir uno o todos los enunciados participantes en el modelo.

Como ya se ha visto la elección de los componentes, las propiedades de los componentes y la forma en que estos están compuestos determina la calidad del software. Cualquier propuesta para un modelo de calidad de software, siempre será solo una aproximación a la calidad del software como se ve en la figura 4.12.

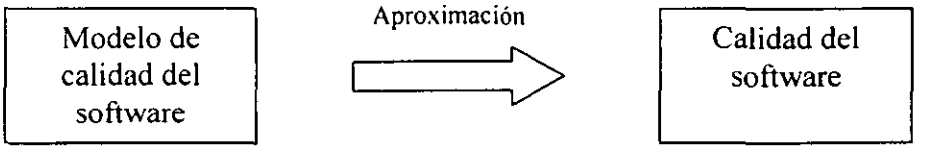


Figura 4.12 Aproximación a la calidad del software

La oportunidad en la construcción de un modelo de calidad del software es igualando / adecuando las propiedades con la naturaleza del software (las características del software) y con las propiedades que describen su comportamiento deseable, de fácil uso y respuesta al cambio (sus atributos de calidad). Para crear un modelo de unión se necesitan satisfacer los siguientes requerimientos.

- Se necesita satisfacer el principio de correspondencia para el modelo de calidad mediante el direccionamiento de las propiedades de la calidad hacia los componentes.
- Se necesitan hacer tangibles las propiedades intrínsecas - basadas en los componentes - y acumulativas y sus efectos a través del sistema del software.
- No hay un direccionamiento uno a uno entre los determinables de la calidad y los determinantes del software. Por ejemplo, las características del software, la modularidad puede contribuir al atributo de calidad de mantenibilidad y a la fiabilidad (proveyendo una protección modular.)
- Un reto es identificar un conjunto comprensivo y genérico de propiedades de calidad intrínsecas para cada característica del software y unir estos con los componentes.
- Otro reto es identificar un conjunto comprensivo de propiedades de calidad intrínsecas para cada componente. Tales como las propiedades que contribuirán a diferentes atributos de calidad.
- Se pude espera que para algunos dominios de aplicación habrá propiedades de calidad intrínsecas de un dominio determinado que tienen un impacto significante en la calidad del software.
- Hay propiedades funcionales tangibles y propiedades tangibles no funcionales del software que contribuyen a su calidad.

Capítulo V
Aseguramiento de calidad en Documentum©

5.1 Documentum

Acerca de Documentum

Documentum es una empresa estadounidense que se dedica a la elaboración de software para la administración de contenido en forma electrónica (Content and knowledge Management).

Documentum es el proveedor líder de soluciones de contenido, basado en una plataforma escalable que permite conexiones e-business. Con más de 940 clientes alrededor del mundo, tiene experiencia comprobada, tecnología de punta, arquitectura abierta, y uniones para crear lo mejor de las integraciones que los califica como los expertos en la administración del contenido de e-business.

¿Qué es lo que hace diferente a Documentum de otro software? Que permite a sus clientes crear, entregar, administrar y personalizar todo el contenido de las contribuciones para todos los objetivos de los procesos de negocios. Todo el contenido significa que debe de ser fresco, confiable e inteligente y en cualquier lugar. Los contribuidores incluyendo los equipos de colaboración y los componentes del sistema de la "cadena de suministro de contenido". Los procesos de automatización del modelo de negocio electrónico, en el cuál están incluidos los socios de negocios, clientes, empleados, y demás miembros de una comunidad virtual. Todos los objetivos que implican las personalizaciones inteligentes de contenido para cualquier audiencia a través de cualquier medio. Esto significa que el contenido no sólo se va a liberar en Internet, sino también puede ser en teléfonos celulares, en un localizador, hacia un fax, una impresora, un CD, o un PDA.

La administración de contenido de Documentum trae visibilidad a su contenido, referente al tipo o formato, a la velocidad de Internet. Con la infraestructura de colaboración, los equipos de proyectos globales trabajan juntos en un objetivo común, aún cuando esos equipos se encuentren en lugares remotos o dentro de múltiples organizaciones.

Dentro de su software contiene varios módulos dirigidos a diferentes industrias, pero todos enfocados a la administración de contenido. De igual forma su foco actual es proveer en un mismo servidor de contenido un repositorio para el web en el sentido de hacer más fácil la actualización de contenido de un sitio web, así como enlace para el desarrollo de comercio electrónico (BTB, BTC, BTE, BTG) con información actualizada y que provenga de sus creadores originales (todos los creadores pueden contribuir a actualizar lo que se maneja en los sitios web.)

La crisis en contenido

No hay duda. El e-business llegó para quedarse. Al reconocer este este hecho, las compañías entran en la carrera de llevar todo su negocio en línea manejando eficientemente la pieza fundamental en lo referente a la atracción en la economía de Internet: El contenido. Publicaciones en Internet, en las Intranets de las compañías, a través de los portales B2B, el contenido es el factor que rige los negocios, facilita la productividad de los empleados, crea nuevas oportunidades, acelera el intercambio de bienes, junta compradores y vendedores, y determina la satisfacción del cliente. El contenido es la nueva moneda. El contenido es la base para el nuevo modelo de negocios dirigido por la amplia proliferación Internet. En este nuevo modelo de negocios basado en el contenido, la preparación, aprobación y liberación de contenido debe de se un proceso tan predecible y controlable como un proceso de manufactura.

Por lo tanto, la pregunta que se hacen las compañías no es si el adoptar este nuevo modelo de negocios que coloca el contenido en el centro de todas las transacciones de negocios. En lugar de eso, la pregunta es ¿Cómo ponerlo? ¿Cómo transformarían sus organizaciones en un negocio electrónico que sea efectivo, competitivo y que no sólo sobreviva, sino que se fortalezca en la economía de Internet? Se debe de empezar por la base, y la base para esta transformación es el contenido.

Las crisis venidera

Aunque varias compañías se han transformado para satisfacer las demandas de los negocios en línea, enfrentan una crisis de crecimiento. Si el contenido es la base de este nuevo modelo de negocio, la administración de ese contenido se vuelve vital para el éxito.

Los análisis indican que el número de páginas Web que las compañías necesitarán administrar, para los próximos años, crece cerca del 200 por ciento anualmente. Aunque las compañías ya esperan manejar entre 2 y 20 sitios cada uno con 18,000-20,000 páginas. Las compañías globales del 2000 se proyectan a si mismas administrando no cientos o varios cientos de páginas, sino miles de cientos de páginas —y lidiando para asegurar que el contenido publicado es actualizado, fiable y verdadero. De hecho, los cálculos indican que el 75 por ciento del costo total de mantener un sitio Web está dirigido a la creación y administración de contenido.

Los usuarios de negocios dentro de la organización —los expertos en contenido los cuales poseen el conocimiento critico para el éxito del negocio— batallan para colocar el contenido a través de los cuellos de botellas del equipo de Internet. Departamentos que tradicionalmente trabajan de forma separada son incapaces de comunicarse efectivamente, resultando esto en perdidas de oportunidades, perdida de tiempo y una baja productividad. Las aplicaciones y el almacenamiento de datos hacen que la proliferación de información sea difícil de acceder, al menos que se tenga un software que pueda integrar aplicaciones y dejara abiertas su datos almacenados.

La respuesta a estos problemas ha sido el añadir más gente al equipo que maneja el sitio de Internet, pero esta solución no puede durar mucho. Las demandas de trabajo que se presentan al equipo de Internet para poner la información en línea de forma rápida, lo mismo que mantener el contenido fresco y dinámico son excesivas.

Transformando el negocio con Documentum

Para resolver esta crisis del contenido de las organizaciones grandes, se requiere una plataforma industrial y robusta de administración de contenido. Sólo Documentum puede proveer esa plataforma. Con la plataforma de administración de contenido Documentum 4i, se puede adoptar de forma segura un nuevo modelo de negocios que puede asegurar competitividad hoy y en el futuro.

Tiene la experiencia y el liderazgo

Con más de diez años de experiencia en administrar contenido en empresas globales, Documentum está sólidamente colocado como el líder en asistencias a compañías que desean transformarse en empresas e-business. Documentum comenzó a administrar información creada y consumida de forma interna por los empleados dentro de grandes organizaciones farmacéuticas, transfiriendo esas capacidades a otras compañías globales. Con el crecimiento de Internet, la plataforma creció para ayudar a los negocios a colocar la información fuera de los firewall y hacerla disponible para los clientes.

Actualmente Documentum provee la plataforma de administración de contenido más poderosa del mercado, permite a las organizaciones construir nuevos modelos de negocios basados en el contenido que aseguran el flujo de contenido entre las compañías, a través de los límites departamentales y organizacionales y del negocio al cliente. Con el contenido como la base, las compañías globales pueden participar en intercambio de colaboración e

información. Pueden abrir portales corporativos para acelerar el desarrollo y el tiempo para el mercado. Pueden construir avanzadas tiendas en líneas a la medida del cliente para asegurar su satisfacción.

Tiene lo que se necesita

Se ha anunciado recientemente nuevos paquetes de productos cuyo objetivo es satisfacer las necesidades de administración de contenido de las iniciativas del e-business. Diseñadas para una fácil instalación esta edición hace sencillo el tener una aplicación e-business ejecutandose con exactitud y contenido confiable:

- Documentum 4i Web Content Management Edition permite crear y publicar rápidamente, es una solución fácil de usar para desarrollar, administrar, y publicar contenido en el Web.

- Documentum 4i Compliance Edition permite desarrollar sistemas que cumplan con los requerimientos normativos o los estándares de calidad.

- Documentum 4i Portal Edition provee un portal como solución inter-empresarial, para permitir la colaboración entre equipos o áreas de toda la empresa, por la Intranet de la misma.

- Documentum 4i B2B Edition permite desarrollar aplicaciones que establecen y mantienen relaciones a través de la cadena de valores para obtener la máxima ganancia en las transacciones y mercados electrónicos.

Tiene los clientes y socios

Más de 940 compañías globales basan su administración de contenido en soluciones de Documentum para desarrollar y mantener una presencia eficiente en línea. Como líder en plataforma para la administración de contenido a escala de Internet, Documentum 4i marca el estándar para la administración de contenido que conlleva a las aplicaciones de comercio electrónico.

Sus socios son vendedores líderes que permiten personalizaciones avanzadas, lo mismo que integraciones de tecnologías y plataformas. Reconociendo la importancia y el tamaño de sus clientes base y la robustez del producto, proveedores de tecnología e integración como PricewaterhouseCoopers, BEA e iXL se han aliado con Documentum para proveer soluciones finales que satisfagan las necesidades de los clientes.

Tiene el reconocimiento de la industria

No son sólo los clientes son los que sostienen la dirección de Documentum. Documentum ha recibido un reconocimiento significante durante los últimos años de publicaciones prestigiadas como InfoWorld, Imaging and Document Solutions, Computerworld y Software Magazine. Deloitte & Touche han incluido a Documentum, por tercer año consecutivo, en su lista de Technology Fast 500 y Silicon Valley Fast 50, las cuales son creadas para reconocer las valiosas contribuciones de las compañías tecnologías de rápido crecimiento.

Tienen la visión

En Documentum, entienden la importancia del contenido como la base para nuevos modelos de negocios que están siendo adoptados por las compañías que tendrán éxito en la economía de Internet. Han construido una plataforma que emplea una tecnología abierta y escalable que puede crecer para satisfacer las necesidades de los clientes, y que ayuda a resolver la crisis en contenido. Con Documentum 4i, se tienen las herramientas que permitirán transformar la empresa en un poderoso e-business, el cual crecerá en la nueva economía satisfaciendo los nuevos requerimientos.

Soluciones para la industria

Manufactura, farmacéuticas, telecomunicaciones, servicios financieros, ingeniería y construcción, automotriz, y alta tecnología son sólo unos cuantos sectores donde el éxito del negocio gira alrededor del manejo del contenido dinámico. Esas industrias están yendo a un profundo cambio. Los intercambios en línea, obtención electrónica, comodidad de intercambios virtuales, la administración de canales se convierten en prioridades estratégicas, por lo tanto, la administración de contenido es esencial para realizar el nuevo modelo de negocios que se requiere. Con la capacidad probada para el manejo de un volumen ilimitado de contenido dinámico, Documentum se esta convirtiendo en una tecnología operacional para las organizaciones en aquellas industrias que están buscando explotar las oportunidades que emergen del e-business.

5.2 Definición de Calidad en Documentum

"Las cuatro máximas de Documentum para la administración de la calidad", que son la base para su sistema de calidad se enunciarán a continuación:

1. Definición de la calidad: Cumplir con los requerimientos.
2. Sistema de calidad: Se utiliza como prevención.
3. Desarrollo estándar: Cero defectos.
4. Medida de Calidad: Costo por cumplir y por no cumplir con los requisitos.

Con esto se establece que cualquier producto, proceso o servicio que satisfaga los requerimientos, es, un producto de calidad.

Las características de un buen *conjunto de requerimientos*, que aseguren la calidad, son las siguientes: precisión, que no sean ambiguos, completos, consistentes, medibles, con prioridad, modificables (los requerimientos pueden cambiar, pero estos cambios deben tenerse de mutuo acuerdo con el equipo del proyecto) y que se puedan rastrear. Para esto hay personal asignado cien por ciento a entender los requerimientos, a desarrollarlo bien la primera vez, y sin defecto alguno (*Cero defectos*).

La prevención involucra la comunicación, planeación, pruebas y trabajo durante el proceso para eliminar las oportunidades de disconformidad. Algunos de los elementos que se pueden incluir en este ramo son: El entrenamiento adecuado al personal, aseguramiento de un conjunto de requerimientos claros, proporcionar suficientes recursos al proyecto, análisis de base, análisis métricos, producir una tarjeta de reporte.

La figura 5.1 muestra como se incrementa los costos de encontrar errores conforme avanzan las fases del proyecto (Escalón de costos).

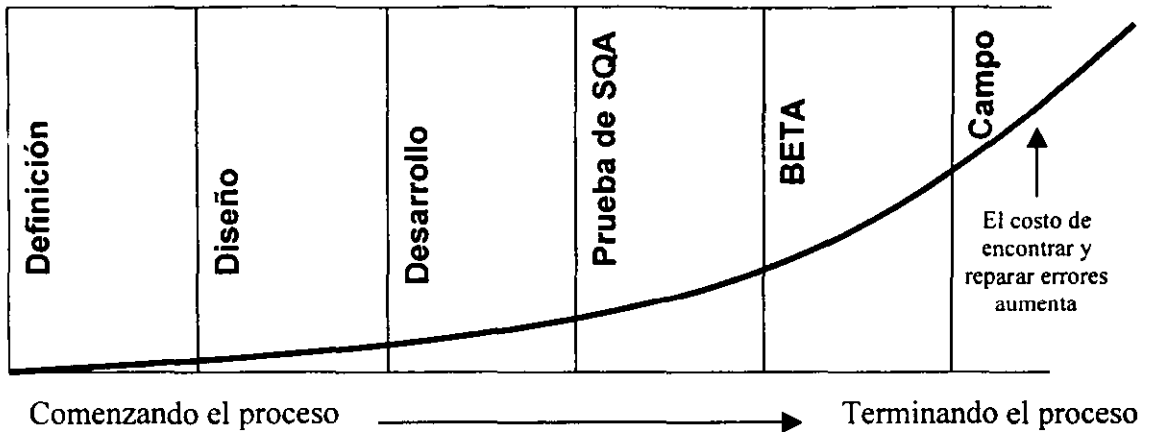


Figura 5.1 Escalón de costos

La medida de calidad se da al calcular el costo del gasto - gasto de tiempo, dinero y material – produciendo un bosquejo económico para dirigir esfuerzos a mejorar y medir las optimizaciones.

El costo total de la calidad viene dado por la suma de lo que cuesta el cumplir con los requerimientos mas el costo por no satisfacer los mismos.

Satisfacción de los requerimientos	+	Insatisfacción de los requerimientos
Entrenamiento Análisis / Inspecciones Mejoramiento de procesos		Reparación de errores Rehacer el trabajo Garantía

Componentes del Sistema de calidad

- La visión y la misión
- Estructura de la organización
- Procedimientos estándar de operación(SOPs)
- Procesos estándar de desarrollo del producto (PDP)
  (los procesos incluyen los requerimientos del producto, planes del proyecto, planes de prueba, etc)
- Personal entrenado y calificado

Concepto del equipo del proyecto

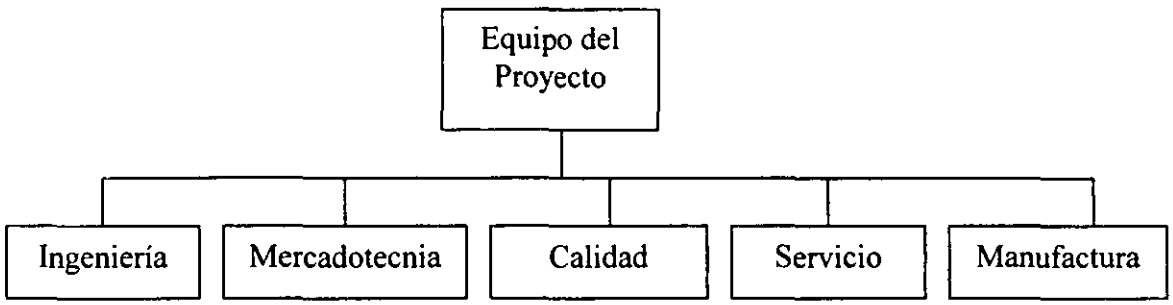


Figura 5.2 Concepto de equipo

Este concepto debe de estar en el centro del proceso de desarrollo, donde los miembros del equipo representan todas las funciones. El líder del proyecto es designado por el programa de administración. Se programan juntas semanales para revisar los estado de cada proceso y los asuntos claves (se guarda la minuta). Los integrantes del equipo de proyecto son los siguientes: Programa de Administración, Ingeniero líder de desarrollo, gerente de mercadotecnia del producto, ingeniero de aseguramiento de calidad del software (SQA), ingeniero de soporte, escritor técnico, un representante de manufactura, gerente de versión BETA.

5.3 Proceso de desarrollo del producto.

Documentum utiliza el modelo clásico del ciclo de vida del desarrollo del producto, para el desarrollo de nuevos productos así como de nuevas versiones. A este ciclo de vida ellos le llaman El Proceso de Desarrollo del Producto (PDP), del cual se muestra el diagrama de flujo, en la figura 5.3

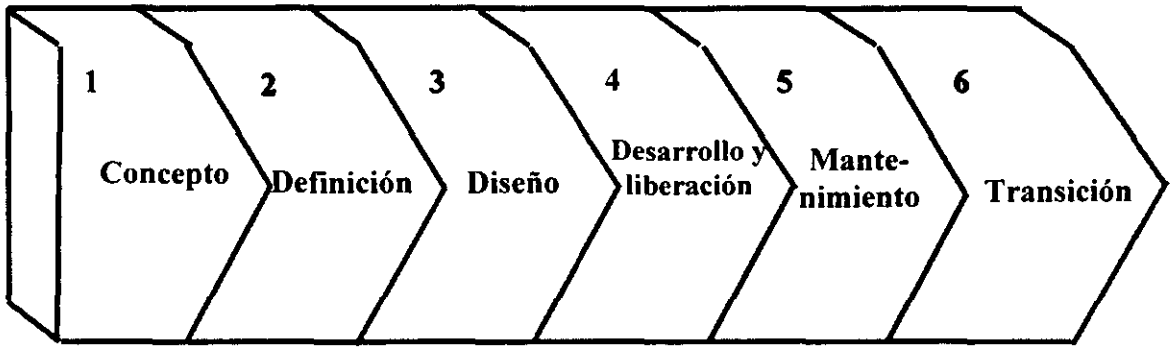


Figura 5.3 Proceso de desarrollo de un producto

El Proceso de Desarrollo del Producto consta de las seis fases mostradas en el diagrama anterior: Concepto, Definición, Diseño, Desarrollo y liberación, Mantenimiento, Transición.

En la figura 5.4 se muestra el acoplamiento de las fases que a continuación se mencionan.

Concepto:

- Se presenta la propuesta al consejero del equipo de desarrollo del producto
- Establecer su posibilidad y su atractivo
- Establecer las prioridades (sí se aprueba)
- Entregable
  - Presentación del concepto del proyecto

Definición:

- Determinar los requerimientos del mercado
- Identificar el equipo del proyecto
- Entregables:
  - Documento de los requerimientos del mercado
  - Plan del proyecto
  - Lista de requerimientos del PDP
  - Agenda maestra

Diseño:

- Escribir, analizar y aprobar planes y especificaciones
- Entregables:
  - Especificaciones (Funcionales / Diseño)
  - Plan de soporte
  - Publicaciones técnicas del plan
  - Planes de prueba, plan de prueba beta
  - Plan de servicios educacionales
  - Plan del producto, plan de lanzamiento
  - Plan de manufactura

Desarrollo y liberación:

- Realizar los códigos y las pruebas
- Escribir los manuales de usuario y el material de entrenamiento
- Entrenar al personal / equipo interno.
- Pruebas Alfa y / o Beta
- Entregables:
  - Producto probado y certificado
  - Documentación del producto

Mantenimiento:

- Soporte y mantenimiento al producto liberado
- Entregables:
  - Evaluación del producto en el mercado.
  - Muerte del software

Transición:

- Definir un plan de transición para los clientes
- Establecer el fin de vida del producto (EOL)
- Entregables:
  - Plan de fin de vida

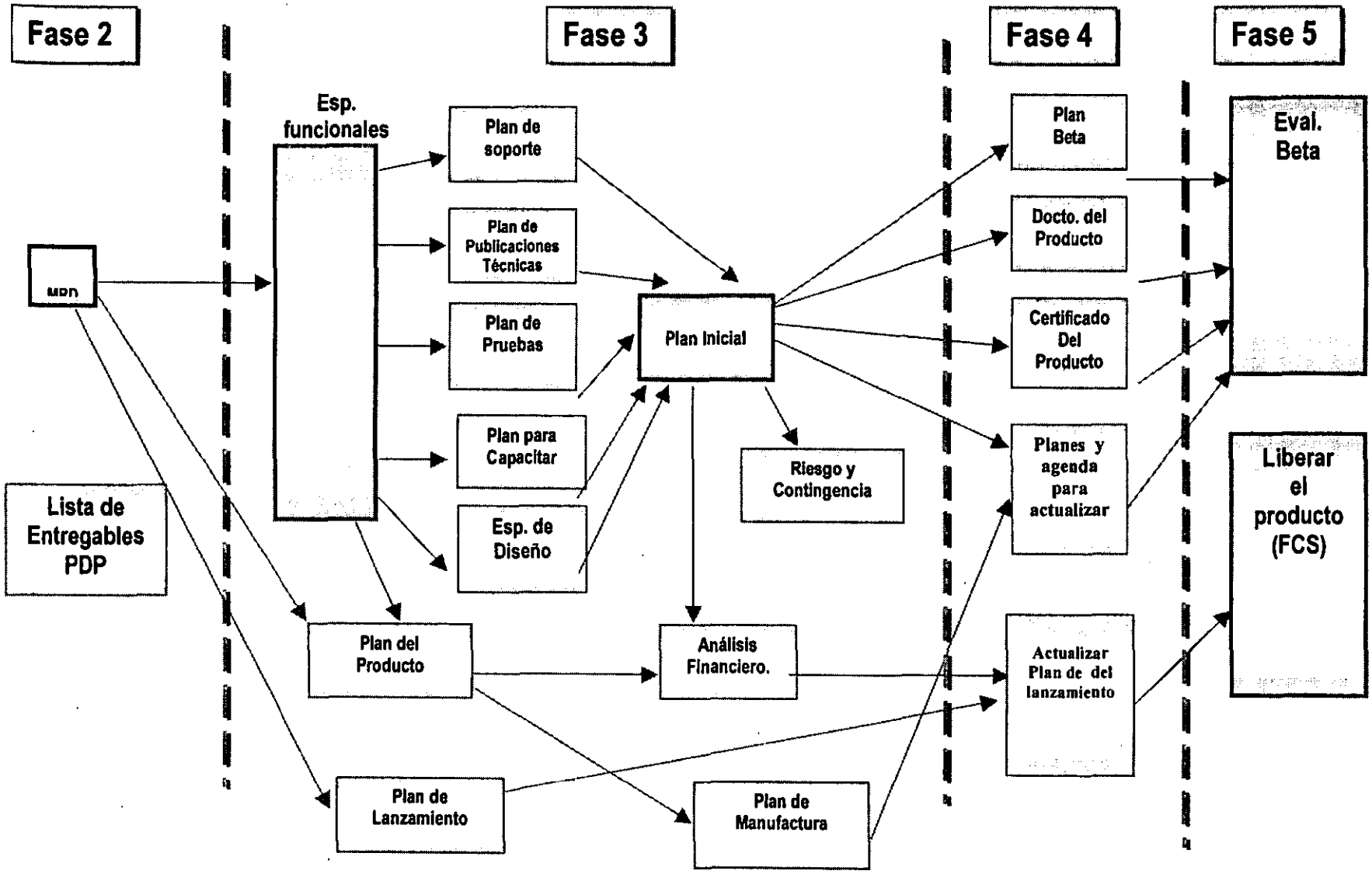


Figura 5.4 Proceso de Desarrollo

5.4 Procedimientos estándares de operación

Varias ventajas que proporciona el tener procedimientos estándar de operación son:

- Métodos uniformes
- Los buenos procedimientos reducen errores
- Son una guía de entrenamiento excelente
- Definen la interfase departamental
- Es requerida en las industrias reguladas

Algunos de los procedimientos estándar de operación que se utilizan en Documentum son los siguientes:

SOP – 001 SOP para realizar un SOP
SOP – 002 Proceso de desarrollo de un producto
SOP – 003 Cambio de Control
SOP – 004 Códigos estándar
SOP – 004 Seguridad
SOP – 006 Políticas de versión del producto
SOP – 007 Reporte y rastreo de problemas
SOP – 008 Liberación del producto
SOP – 009 Liberación de Parches
SOP – 010 Aceptación del los procesos de pruebas por SQA
SOP – 011 Documentación de las pruebas
SOP – 012 Educación de los empleados y archivos de entrenamiento
SOP – 013 Plan de recuperación en caso de desastre
SOP – 014 Calcular el valor del software
SOP – 015 Generación de casos de clientes
SOP – 016 Envío y seguimiento de errores
SOP – 017 Procedimientos de respaldo
SOP – 032 Auditorias internas de calidad

El aseguramiento de calidad, es responsabilidad del equipo de calidad, el cual forma parte del equipo del proyecto. Estos se encargan de revisar y validar toda la documentación del proyecto, preparan el plan para de pruebas para el sistema, lo mismo que desarrollan procedimientos de prueba basados en las especificaciones funcionales y el plan de prueba. De igual forma establecen las metas de calidad, prueban los errores reparados, desarrollan pruebas

automatizadas y desarrollan la prueba final de aceptación del aseguramiento de calidad en el software.

Todo lo anterior nos da como resultado el poder verificar que el producto esta trabajando correctamente, y nos arroja la seguridad que el proceso esta siguiendo los requisitos regulatorios. La evidencia documentada de como se hizo el producto, es y seguirá siendo fiable concerniente al objetivo del producto.

Una vez que se entra en la etapa de validación, lo que se comprueba y / o verifica es que el producto se haya desarrollado conforme el ciclo de vida antes mencionado, que tenga una buena documentación y además que los ingenieros usen buenas técnicas de ingeniería, que los resultados de las pruebas estén documentados y que se tenga un documento de certificación.

5.5 Aseguramiento de calidad en el software Documentum

La carta del Equipo de Aseguramiento de calidad en Documentum tiene por objetivo el construir, probar y certificar todo el software que se realiza en Documentum y asegurarse de que hay conformidad con los requisitos formales (explícitos, como ejemplo podríamos poner los objetivos del software) e implícitos (estos se refieren a los que se basan dependiendo la plataforma para la cual se desarrolla, para Windows el programa se debe de comportar como una aplicación Windows, si es para una Mac la aplicación de debe de comportar como una aplicación Mac y para Unix como una de Unix.) El equipo u organización del aseguramiento de calidad en el software esta conformado por una entidad más dentro del grupo de toda la compañía y tiene, como cualquier otra entidad, su propio organigrama.

A continuación se muestra en la figura 5.5 el diagrama de flujo del proceso de aseguramiento de calidad en Documentum, así como los documentos relacionados con el proceso de creación y la responsabilidad del grupo de aseguramiento de calidad para con estos.

Documento del Proyecto	**Responsabilidad del SQA**
Documento de los requerimientos de mercado	Analizar
Especificaiones Funcionales, Documentos de la planeación de l proyecto	Aprobar
Unidad de planes de prueba	Analizar
Especificaciones del diseño	Analizar
Objetivos de calidad	Preparar
Planes de prueba	Preparar
Reporte de defectos	Preparar
Documentación del usuario	Analizar
Noticias de cambio	Analizar/ Aprobar

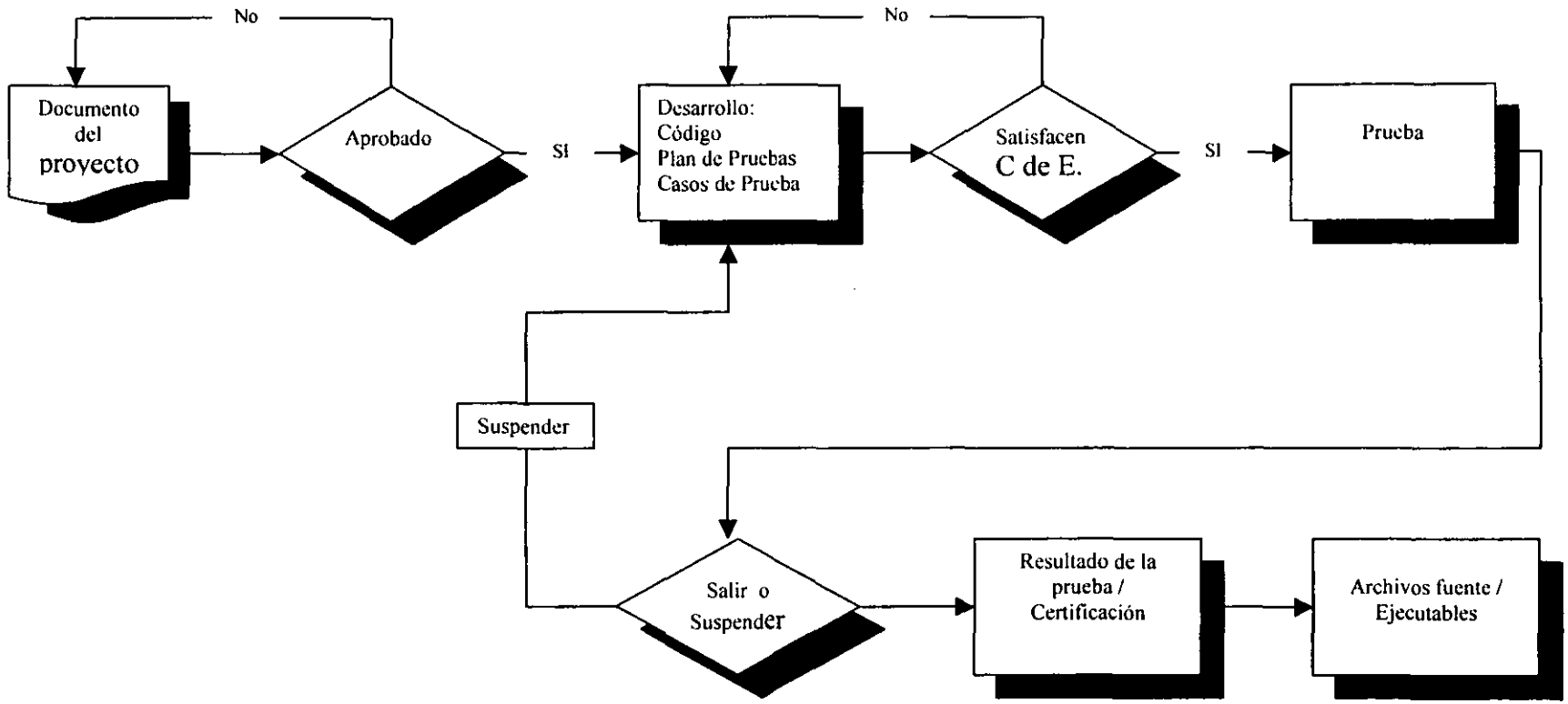


Figura 5.5 Diagrama de flujo del proceso de aseguramiento de calidad en Documentum

Las funciones del equipo de aseguramiento de calidad son las siguientes:

Revisar y firmar de conformidad las especificaciones de funcionalidad:

- ¿Está completo y correcto?
- ¿Se pueden probar los requerimientos?
- ¿Se adecuan a la forma del diseño?

Desarrollar planes y casos de prueba:

- Capacidad de rastreo y de poder completarse
- Pruebas funcionales y blanca
- Pruebas de caja negra

Desarrollo de sinopsis de las pruebas y casos:

- Revisar la sinopsis con el área de ingeniería
- Pruebas manuales y automáticas

Ejecutar las pruebas y guardar los resultados:

- Mantener e inspeccionar las pruebas grabadas
- Publicación de certificación
- Archivar el software

Una actividad relacionada de forma muy obvia con las pruebas es el cambio del código fuente de la aplicación. Este esta definido en el SOP-0003, Control de Cambios. Todos los cambios en el código fuente deben de ser documentados, revisados y aprobados antes de ser dados de alta en el repositorio del código. Todos los cambios hechos durante el ciclo de pruebas deben de ser revisados y aprobados por el grupo de aseguramiento de calidad antes de darse de alta en el código fuente. El mismo software Documentum es utilizado para crear, dirigir, y almacenar noticias de cambio.

Seguimientos de errores

Todos los reportes de errores son registrados en DM_Bugs, un sistema de rastreo de errores construido usando el software de Documentum. Todos los errores son dirigidos electrónicamente asignados por el líder del proyecto, direccionados y finalmente se regresan al equipo de aseguramiento de calidad para su verificación. El estatus de los errores en rastreado en los ciclos de desarrollo y prueba.

Las metas de calidad

Las metas de calidad especifican las tareas y las medidas de calidad que deben de ser satisfechas antes de que se considere que el proyecto está listo para liberarse. Deben de estar las firmas los grupos de ingeniería, mercadotecnia del producto, soporte técnico, administración del programa y aseguramiento de calidad. Esto se crea para cada proyecto y está contenido en el plan de prueba.

Unos ejemplos de las metas de calidad serían las siguientes:

- No introducir cambios que no estén probados al código fuente
- Todos los requerimientos deben de estar probados

El plan de pruebas

Por su parte el plan de pruebas puede contener el plan maestro de pruebas, el cual para proyectos largos este define toda la estrategia de pruebas, recursos que se necesitan, etc.

El plan de pruebas de funcionalidad está relacionado fuertemente hacia una funcionalidad en particular de las especificaciones, este es un documento detallado de plan de prueba con una gran capacidad de rastreo.

Se envía el plan de pruebas para ser aprobado por el área de ingeniería, administración del proyecto y soporte técnico. Se controlan las versiones del plan de prueba usando el software Documentum (la cobertura de las pruebas es medida por revisiones manuales de las casos de prueba, ya sea por un ingeniero del grupo de aseguramiento de calidad o por un ingeniero de desarrollo).

El plan maestro de pruebas sigue el formato dado en ANSI/IEEE Std 829-1983:

1. Identificador
2. Introducción
3. Elementos de prueba
4. Características a ser probadas
5. Características que no van a ser probadas
6. Método de prueba
7. Metas de calidad (En IEEE lo llaman criterio de salida)
8. Criterios para entrada, suspensión y reanudar la prueba
9. Entregables de la prueba
10. Tareas de la prueba
11. Necesidades ambientales (para el desarrollo)
12. Responsabilidades
13. Entrenamiento al equipo de pruebas
14. Calendar izar
15. Riesgos, dependencias, restricciones y contingencias
16. Aprobación

El plan de pruebas funcional

Está fuertemente ligado con una especificación funcional. Define una sinopsis para cada categoría de prueba. Define toda la estrategia de pruebas (orden de las pruebas, pruebas regresivas), facilita el rastreo y permite que se completen las pruebas. Para proyectos largos el plan maestro de pruebas incluye una tabla de relación de las especificaciones funcionales con el plan de pruebas funcional:

Título de la especificación funcional	**Título del plan de prueba**
Especificación de la instalación	Plan de prueba funcional de la instalación de WorkSpace
Especificaciones de las extensiones API	Plan de prueba funcional para el API
Especificación de los lazos funcionales de Documentum	Plan de prueba funcional de los lazos de Documentum

Para los proyectos pequeños o en los planes de pruebas funcionales, se incluye una tabla donde se relacionan las secciones de especificaciones funcionales con las categorías de prueba:

Especificaciones de Extensión API	Categoría de prueba
2.1 Método de checkin	Api_checkin_*
2.2 Método de checkout	Api_checkout_*
2.3 Método de Getdocbasemap	Api_getdocmap_*

Las categorías de prueba están contenidas en una o más sinopsis de prueba:

Categoría de prueba	Sinopsis de prueba
Api_check_in_*	Poner la bandera save_lock a T(verdadero). Después del checkin, verificar que el lock está puesto para la nueva versión del objeto.
	Para el argumento version_label, tanto una etiqueta implícita y una simbólica. Después del checkin, verificar que ambas etiquetas están definidas para el objeto.

En resumen, el plan maestro de prueba, relaciona las especificaciones funcionales con los planes de pruebas funcional. El plan de prueba relaciona secciones de las especificaciones funcionales a las categorías de pruebas. La categorías de prueba contiene sinopsis de pruebas. Las sinopsis de las pruebas son resúmenes de los actuales casos de prueba.

Mejoramiento de las pruebas (en los productos de Documentum):

Desarrollar pruebas automáticas

- Productos al cliente
  - Automatización de la pruebas de las interfaces de usuario están en progreso
  - Pruebas automáticas en 4I usando DFC
- Servidor

- Pruebas basadas en scripts
- 90% automáticas
- Las pruebas de presión y desarrollo se han realizado

- RightSite
  - Administrado de forma extrema para la prueba de presión
  - Interfaces de usuaro desarrolladas en web para pruebas manuales

Y sobre todo: Mejorar la eficiencia a través de la automatización de las pruebas.

Liberación del producto

Cuando se libera cualquier aplicación del producto o el producto entero, se cerifica que:

1. Verificar que el producto haya cubierto sus metas de calidad
2. Crear una plantilla o patrón del producto
3. Juntar y guardar los archivos de pruebas
4. Firma de aprobación del equipo de proyecto
5. Respaldar el código fuente, los archivos de la documentación del usuario y crear instrucciones para el almacenamiento fuera de línea y su valor
6. Publicación de la certificación del producto

Para Documentum, el mejorar en parte de su trabajo diario, tiene el objetivo de eliminar los problemas y sus fuentes, el mejoramiento los conduce a ser mejores al mismo tiempo que los problemas son corregidos. El aumento del valor a través de proporcionar un mejor servicio y productos. Mejorar el tiempo de respuesta y el ciclo de vida. Mejorar la productividad y la efectividad para sus recursos.

Los resultados del análisis de los clientes de Documentum durante el año de 1999 fueron:
¿Quedaron muy satisfechos con el desarrollo de Documentum comparado con la competencia? 74.4 % SI ;
¿Les gusto mucho como para recomendar Documentum? 82.2% Si;
¿Les gustó mucho como para volver a escoger Documentum? 80.4 % Si.

Conclusiones

A través de este trabajo se dio un bosquejo de todos los pasos que se tiene que seguir al aplicar la fiabilidad y alcanzar el fin último de esta, que es la calidad del producto de software.

Se puede concluir que la fiabilidad es un punto crucial para lograr un software que satisfaga las necesidades del cliente, esta parte esta encargada de predecir el buen funcionamiento del software a través de la eliminación del mayor número de errores posibles; Dejar el software con cero errores es prácticamente imposible, por lo tanto, mientras más fallas se eliminen de más calidad será el producto de software.

Hay diferentes métodos que se pueden aplicar para medir la fiabilidad de software, la mayoría de ellos son la probabilidad de posibles fallas en determinado lapso de tiempo basándose en los muestreos de fallas en periodos anteriores. Se sabe que hay errores en el software que pueden ser errores de sintaxis, de semántica, los de tiempo de ejecución (Errores de Domino, Infinitos, de Especificación), y los de desarrollo, pero no se sabe exactamente su ubicación hasta que se realizan las pruebas del sistema (las pruebas de crecimiento de fiabilidad, de certificación, la prueba beta).

En lo relacionado a la Ingeniería de Fiabilidad de Software, este en una disciplina auxiliar que nos ayuda en la elaboración y desarrollo de un software de alta calidad, asegura que la fiabilidad del producto satisface las necesidades del usuario, que el tiempo de llegada al mercado sea menor, reducir el costo de producción, aumentar la satisfacción del cliente e incrementar la productividad. Cabe recordar que la Ingeniería de Fiabilidad de Software trabaja bajo dos principios:

1. La entrega de funcionalidad deseada para el producto bajo el desarrollo mucho más eficiente del producto por medio de la representación cuantitativa de lo que espera el usuario.

2.La segunda es que media la fiabilidad del software, la fiabilidad del cliente, el tiempo de desarrollo y la estimación del costo del desarrollo, por ende la efectividad.

La Ingeniería de fiabilidad de software está basada en teorías muy sólidas. Dentro de las grandes empresas que han utilizado la Ingeniería de Fiabilidad del Software están Hewlett-Packard, Microsoft, AT&T, etc.

Una vez que se han seguido estos pasos de elaboración de pruebas y estimaciones de fiabilidad se puede llegar a construir o aplicar un modelo de calidad para el software. Si se desea construir un modelo de calidad para el

software lo primero que se debe de hacer identificar para que aplicaciones es el modelo y dirigir los diferentes grupos interesados que usaran los modelos en las diferentes aplicaciones. La segunda etapa es identificar una arquitectura o diseño el modelo. Por ende los pasos a seguir son tres:

1. Construir el modelo del producto del software
2. Construir el modelo de calidad
3. Unir los modelos de calidad y de software para construir el modelo de calidad para un producto de software.

Todos los pasos y métodos que se han mencionados son laboriosos y consumen tiempo y recursos, pero estos son menores a los que consumirian los posibles clientes a disgusto (perdida del cliente), mal desempeño del producto, fallas en los sistemas por causa del producto, etc. Por eso, en todo buen desarrollo de un producto de software es bueno tener en mente todas estas técnicas y poderlas aplicar. El aplicar estos procedimientos y métodos trae como consecuencia un beneficio extra a la compañía que lo aplique, sea su foco el desarrollo de software o no.

Glosario

BTB (Business to Business): Módulo del comercio electrónico que está dirigido de un negocio a otro.

BTC (Business to Consumer): Módulo del comercio electrónico que está dirigido de un negocio a sus consumidores.

BTE (Business to Employees): Módulo del comercio electrónico que está dirigido de un negocio a sus empleados.

BTG (Business to Goverment): Módulo del comercio eletrónico que esta dirigido de un negocio al gobierno.

Check in: Registro de un documento en la Docbase.

Check out: Bloqueo de un archivo en la Docbase cuando este está siendo modificado.

Docbase: Repositorio en el cual el software Documentum guarda los objetos.

Documentum: Software que permite la administración electrónica de contenido, desde simples documentos hasta el más complejo sitio Web.

DFC (Documentum Foundation Classes): Conjunto de clases de Documentum que permite el desarrollo de interfaces por terceros para que interactúen con los componentes de este.

Fiabilidad: Probabilidad de que un dispositivo funcionará sin errores en un tiempo determinado bajo condiciones determinadas.

IFS (Ingeniería de fiabilidad de software): Métodos alternativos y paralelos al ciclo de vida del software que ayudan a hacer más eficiente el proceso de desarrollo del software y a alcanzar el mercado en menos tiempo y a menor costo.

MTTF (Middle time to failure): Tiempo medio para la siguiente falla.

NHPP(Non Homogenous Poisson Process): Proceso de Poisson no homogéneo.

PDP(Product Development Process): Proceso de desarrrollo del producto.

RightSite: Parte de los productos de Documentum que se encargan de manejar las peticiones que se realizan al software vía http (puede ser por medio de la Intranet o de Internet) y dirige la petición a Documentum, al recibir la respuesta de este transforma los resultados en HTML para presentárselos al usuario.

SOP (Standar Operation Procedure): Procedimiento estándar de operación.

Propiedades del software: Para entender de la manera más amplia este concepto es necesario comprender que:

El software está constituido de componentes tanto de alto como de bajo nivel. A través de estos componentes y de la forma en que están constituidos, el software muestra propiedades que lo caracterizan y lo distinguen de otros artefactos. Algunas veces las palabras: *características*, *factores* y *atributos* son formas diferentes de llamar a lo que aquí se denomina como propiedades. Se distinguirán varios tipos de propiedades que pueden ser tanto tangibles (concretas) como abstractas (intangibles) y tanto funcionales como no funcionales. Las propiedades tangibles pueden ser medible, calculables o detectable, ya sea por medios manuales o automáticos.

Algunas de las propiedades del software son *deseables*. A esas propiedades deseables se les llama *calidad* o *atributos de calidad*. Calidad, de forma más especifica, un conjunto de atributos de calidad es el vehículo mediante el cual los diversos grupos interesados en el software expresan sus necesidades. El método de "meta-directa" para construir un modelo de calidad del software es efectivo para acomodar y balancear las necesidades de los diferentes grupos. El conjunto de *propiedades deseables* o *atributos de calidad* del software proveen una especificación abstracta o de alto nivel de lo que se llamará calidad del producto de software.

Se hace la observación de que los atributos de calidad que se asocian con el software corresponden ya sea a *comportamiento* o a *usos*. Un comportamiento es algo que el software en si *exhibe* cuando se ejecuta bajo la influencia de un conjunto de entradas (la fiabilidad y eficiencia son un comportamiento.) Desde una perspectiva funcional, el comportamiento de un software puede ser representado por un conjunto de respuestas (incluyendo las salidas y los métricos) que el sistema los exhibe a través de las interacciones de su conjunto de funciones en tiempo de ejecución, en respuesta a uno o más conjunto de entradas. Un uso es algo que los diferentes grupos interesados hacen con o para el software (portabilidad y el mantenimiento son usos.) La interpretación funcional del *uso*, es que es función "desarrollada del usuario", donde el software o el sistema de software es la entrada:

salida ⟵ función desarrollada del usuario (software)

La salida varia dependiendo del uso. Para el uso de *mantenimiento* la salida puede ser una modificación de la entrada, mientras que el uso de "capacidad de aprendizaje" puede dar como salida un conjunto de enunciados variables que reflejan el entendimiento del sistema por parte del usuario o un conjunto de tareas (más tiempo de aprendizaje) que los usuario saben como desarrollar usando el sistema de software.

Una característica del software es una propiedad abstracta (determinable) del software que clasifica un conjunto de propiedades de calidad intrínsecas del producto. No es ni un comportamiento ni un uso. La modularidad es un buen ejemplo de las características de un producto de software. Las características del software pueden corresponder tanto como a un conjunto de entidades funcionales o a un conjunto de propiedades tangibles no funcionales. Las características del software ayudan al software a satisfacer los atributos de calidad. Por ejemplo, la propiedad tangible de independencia de máquina puede de los componentes de software puede contribuir a la portabilidad del sistema.

Apéndice A

ISO 9126

¿Qué es la ISO 9126?

Es el estándar de evaluación para los productos de software de la Organización Internacional para la Estandarización (ISO).

Definir la calidad del software para un producto de software que todavía se encuentra en desarrollo es difícil tanto para el consumidor como el proveedor. El consumidor debe de entender claramente para poder dar a entender sus requerimientos. El proveedor debe de entender bien los requerimientos del usuario para poder proveer un producto con la calidad adecuada.

La ISO 9126 sirve para eliminar cualquier malentendido entre el consumidor y el proveedor. Este mejoramiento e la comunicación hará a un lado el trabajo doble que se requería por el hecho de que el producto no satisface los requerimientos del consumidor. El tiempo que toma para llegar al mercado el producto y los costos se reducen al momento que se sigue el estándar ISO 9126.

El objetivo de este estándar es proporcionar un cuadro de trabajo para la evaluación de la calidad del software. La ISO 9126 no provee los requerimientos para el software, pero provee un modelo de calidad de software que es aplicable a cualquier clase de software. Define seis características de calidad del producto y sugiere las subcaracterísticas para cada una de estas.

Las subcaracterísticas adoptadas por la ISO 9126/IEC 1991 son las siguientes:

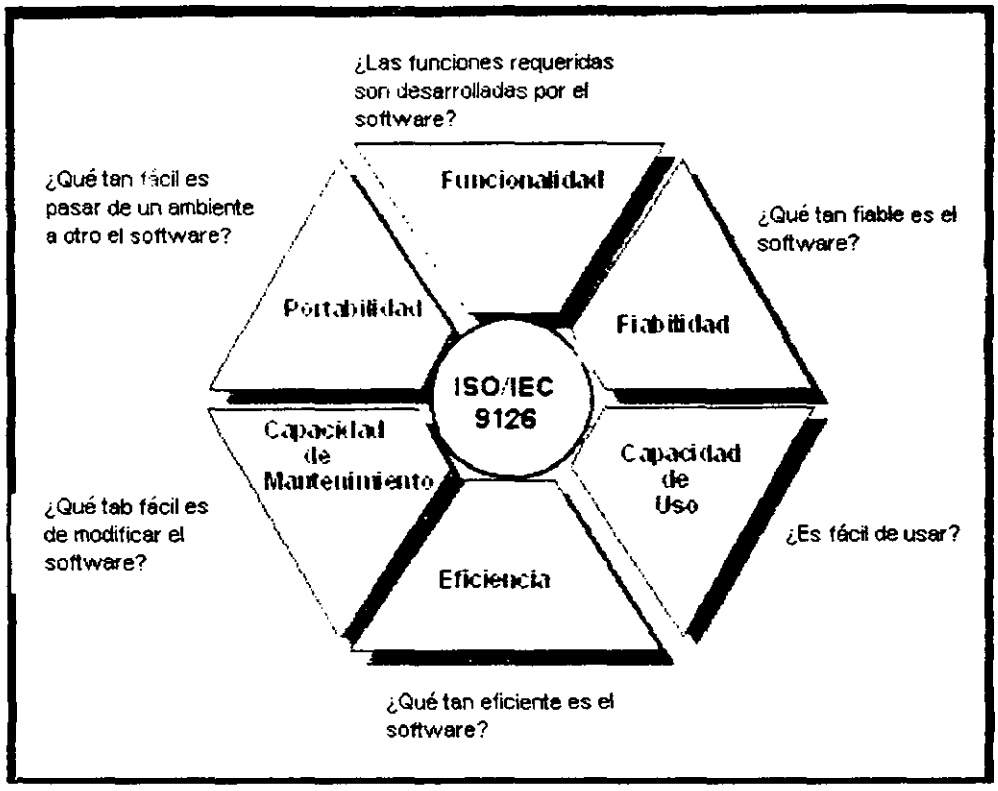


Figura A.1. Características del software

Características	Subcaracterísticas	Definiciones
	Capacidad de Adecuación	Atributos del software que se basan en la presencia y adecuación de un conjunto de funciones para tareas especificas.
	Precisión	Atributos del software que se basan en la provisión de resultados correctos o efectos.
Funcionalidad	Interoperabilidad	Atributos del software que se basan en su habilidad para interactuar con sistemas especificos.
	Conformidad	Atributos del software que hacen que el software se apegue a los estándares de la aplicación, convenciones o regulaciones en las leyes y prescripciones similares.
	Seguridad	Atributos del software que se basan en la capacidad para prevenir accesos nos autorizados, ya sean accidentales o deliberados, a los programas o datos.

	Madurez	Atributos del software que se basan en la frecuencia de fallas por errores en el software.
Fiabilidad	Tolerancia de fallas	Atributos del software que se basan en la capacidad para mantener un nivel especifica de desempeño en caso de fallas en el software.
	Capacidad de Recuperación	Atributos del software que se basan en la capacidad para reestablecer su nivel de desempeño y recuperar los datos directamente afectados en caso de falla en los tiempos y esfuerzos necesarios para ello.
	Capacidad de entendimiento	Atributos del software que se basa en el esfuerzo de los usuarios para reconocer el concepto lógico y su aplicación.
Capacidad de uso	Capacidad de aprendizaje	Atributos del software que se basa en el esfuerzo del usuario para aprender el funcionamiento de la aplicación.
	Operatibilidad	Atributos del software que se basan en el esfuerzo del usuario para la operación y el control de la operación.
Eficiencia	Tiempo de comportamiento	Atributos del software que se basan en timepo de respuesta y procesamiento y a través de tasas de salida de desempeño de su función.
	Comportamiento de recursos	Atributos del software que se basan en la cantidad de recursos usados y la duración de dicho uso durante el desarrollo de su función.
	Capacidad de análisis	Atributos del software que se basan en el esfuerzo requerido para el diagnostico de deficiencias o causas de fallas, o para la identificación de parates que van a ser modificadas.
Capacidad de Mantenimiento	Capacidad de cambio	Tributos del software que se basan en el esfuerzo requerido para la modificación, remoción de fallas o para cambios de medio ambiente.
	Estabilidad	Atributos del software que se basan en el riesgo de efectos in esperados de las modificaciones.

	Capacidad de pruebas	Atributos del software que se basan en el esfuerzo requerido para la validación de las modificaciones del software.
Portabilidad	Adaptabilidad	Atributos del software que se basa en la oportunidad para su adaptación a diferentes ambientes determinados sina plicar otras acciones o medios que aquellos provistos para este propósito en el software.
	Capacidad de instalación	Atributos del software que se basan en el esfuerzo requerido para instalar el software en un ambiente determinado.
	Conformidad	Atributos del software que hacen que el software de apegue a los estándares o convenciones relcacionadas con la portabilidad.
	Capacidad de Reemplazo	Atributos del software que se basan en la oportunidad y esfuerzo usado en lugar de definir otro software en el ambiente de ese software

IEEE 982.1-1988
&
IEEE 982.2:1988

Título:

IEEE 982.1: IEEE Standard Dictionary of Measures to Produce Reliable Software

Diccionario de medidas estándar para la producción de software fiable

IEEE 982.2: IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software

Guía para la utilización del Diccionario de medidas estándar para la producción de software fiable

Naturaleza:

Especificación técnica internacional de un consorcio industrial

Ámbito:

DESARROLLO DE SISTEMAS DE INFORMACIÓN

Procesos del ciclo de vida del software

Indicadores y métricas para la gestión del desarrollo

Origen:

IEEE

Computer/Software Engineering

Fecha(s) de publicación:

IEEE 982.1 y IEEE 982.2: 1988

Proyectos de revisión de IEEE 982.1 y IEEE 982.2: 12/1995

lo que se puede hacer para disminuir los efectos de la falta de fiabilidad de productos casi terminados. Se quiere apoyar así la producción de software fiable, más que estimar los fallos de un producto cuando ya está prácticamente finalizado. No obstante, los dos enfoques, el tradicional de "medida de la fiabilidad" y el de "construir la fiabilidad" se ponen en contexto en este documento.

Estructura / Partes de la norma:

Ámbito y referencias
Definiciones
Medidas para producir software fiable
Medidas: Organización y clasificación
Marco de realización de las medidas
Análisis de errores, defectos y fallos. Mejora de la fiabilidad.

Conexión con otras normas:

IEEE 729-1983, Glosario estándar de terminología de ingeniería de software.

Bibliografía

Musa, John D. Software Rreliability Engineering. McGraw-Hill. 1998.

Norris Mark, Rigby Peter. Ingeniería de Software explicada. Noriega Editores. 1995.

Pattee, H. Hierarchy Theory, The Challenge of complex system. Ed. George Braziller Inc. (1973).

Pham, Hoang. Software Reliability. Springer. Verlag Singapore. 2000.

Ronbeck, J. A. Software Reliability - How it affects system Reliability. Pergamon Press. Canadian Reliability Symposium (1975)

United States Departament of Defense. MIL-HDBK-338/1A (Military Handbook, Electronic Reliability Design -Software Reliability-)1988.

Villalobos Marveya, Olarte C. Marcela. Orientación para la elaboración y presentación de tesis. Ed. Trillas (1993)