18

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

CAMPUS ACATLAN

LA CALIDAD DEL SOFTWARE

TESIS

QUE PARA OBTENER EL TITULO DE:

LICENCIADO EN MATEMATICAS APLICADAS Y COMPUTACION

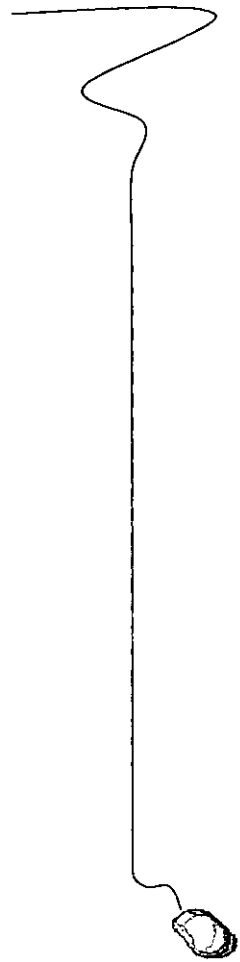
P R E S E N T A :

GERMAN GARCIA GARCIA

ASESORA: ACT. MA. DEL CARMEN GONZALEZ VIDEGARAY

JULIO DEL 2001

Para Gina y Amanda
El amor y el milagro



A mis padres: Esperanza y Rodrigo
A mis hermanos: Jacobo, Penélope y Rodrigo
con todo mi amor

AGRADECIMIENTOS

Durante el desarrollo de este trabajo tuve la suerte de encontrarme con personas que de varias formas contribuyeron a la conclusión del mismo. Gracias a todos ellos. A Eduardo Román por la lectura del manuscrito y las aportaciones al mismo que permitieron concluir la parte final. A Peny y Rodrigo por facilitarme el equipo aún más de lo que pensaba. A Esperanzo y Rodrigo, mis padres, a quienes debo el impulso durante todos los años de estudio. A Jacobo, por todo lo que me ha enseñado.

A Gina y Amanda por estar en todo momento a mi lado.

A los miembros del jurado: Fis. Jorge Luis Suárez Madariaga, Ing. Rubén Romero Ruiz, Act. Luz María Lavín Alanís, Lic. Juan Carlos Rendón Aguilar y especialmente a mi asesora Act. Ma. Del Carmen González Videgaray, quien me hizo sentir de regreso a casa. Gracias a todos por su tiempo y dedicación al revisar mi trabajo. Finalmente quiero expresar mi más sincero reconocimiento y gratitud a cada uno de mis profesores en la carrera cuya dedicación y arduo trabajo hacen vivir a la UNAM.

INDICE

LA CALIDAD DEL SOFTWARE

Introducción 1

1. Teoría de la calidad del software 11

1.1 Introducción a la calidad 11
1.1.1 ¿Qué es la calidad? 12
1.1.2 El software 18
1.1.3 El problema del software 22
1.1.4 La calidad del software 25
1.1.5 Calidad del producto, calidad del proceso. 28
1.1.6 Visiones de la calidad 32
1.1.7 El aseguramiento de la calidad del software. 37
1.1.8 La calidad como parte de la cultura 40
1.2 Modelos jerárquicos de la calidad 46
1.2.1 Definición: ¿Qué es un modelo jerárquico? 46
1.2.2 Los modelos jerárquicos de Boehm y McCall 49
1.2.3 Interrelación de los criterios de calidad 55

2. Elementos de la calidad del software 59

2.1 Medición de la calidad de software 59
2.1.1 Medición de la calidad 60
2.1.2 Métrica del software 61
2.1.3 ¿Porqué medir? 63
2.1.4 ¿Qué medir? 64
2.1.4.1 Métricas orientadas al tamaño. 66
2.1.4.2 Métricas orientadas a la función 67
2.1.4.3 Métricas para la calidad del software 71
2.1.5 Problemática de la métrica 71
2.1.6 ¿Qué hace buena a una métrica? 72
2.2 Desarrollos en la medición de la calidad 74
2.2.1 El trabajo de Gilb 74
2.2.2 La perspectiva japonesa 84

2.3 Reuso de Software 90
2.3.1 El problema y su entorno 90
2.3.2 Definiciones y conceptos 91
2.3.3 Beneficios del reuso de software 94
2.3.4 Los obstáculos para la reusabilidad 97
2.3.4.1 La reusabilidad vista por la academia 97
2.3.4.2 El hilo negro 98
2.3.4.3 Experiencias fallidas con la reusabiliadad 99
2.3.4.4 Falta de motivación para el reuso de software 102
2.3.5 Niveles de reusabilidad 104
2.3.5.1 Reuso de código 104
2.3.5.2 Reuso de datos 106
2.3.5.3 Reuso de diseños 106
2.3.5.4 Reuso de especificaciones 107
2.3.5.5 Otros tipos de reuso 108
2.3.6 ¿Cómo abordar la reusabilidad? 109
2.3.6.1 El acercamiento tecnológico a la reusabilidad 109
2.3.6.2 El acercamiento administrativo a la reusabilidad 115
2.4 Confiabilidad del Software 119
2.4.1 Términos básicos y conceptos 120
2.4.2 Perfil de operación 122
2.4.3 La confiabilidad y el entorno 124
2.4.3.1 El software 125
2.4.3.2 El hardware 127
2.4.4 Prevención de errores 128
2.4.4.1 Programación estructurada 130
2.4.4.2 Clasificación de datos 131
2.4.5 Tolerancia de errores 132
2.4.6 Manejo de excepciones 136

3. El manejo de la calidad del software 139

3.1 Metodologías de software 139
3.1.1 Definición: Ciclo de vida en cascada 141
3.1.2 Métodos y metodología 142
3.1.3 Problemática del desarrollo en cascada 144
3.1.4 Tipos de ciclos de vida 146
3.1.5 Modelos incrementales y en espiral 149
3.1.6 Transformación automática de modelos. 150
3.1.7 El ciclo de vida con prototipos 152
3.1.8 Metodología y cultura 154
3.2 Las herramientas CASE y las metodologías 154
3.2.1 El boom de las metodologías en la ingeniería de software 154
3.2.2 Herramientas CASE 160
3.2.3 Desarrollo evolutivo 163
3.2.4 Las herramientas CASE y la calidad 165

3.2.5 El cambio de cultura 167
3.3 Sistemas para el manejo de la calidad 169
3.3.1 Perspectiva histórica de los sistemas de manejo de la calidad 170
3.3.2 Aseguramiento de la calidad del software 175
3.3.3 Cultura para la calidad: clave para la administración de la calidad . 178
3.3.4 El problema de los requerimientos de usuario 186
3.4 Estándares de calidad 189
3.4.1 El propósito de los estándares 190
3.4.2 La serie ISO9000: una administración genérica de la calidad 191
3.4.3 Certificación en ISO9000 203
3.4.4 Desarrollos recientes e impacto 205

4. Futuro de la calidad del software 211

4.1 La producción de software 211
4.1.1 La fábrica de software 212
4.1.2 Procesos de software 214
4.1.3 El costo de la calidad 219
4.2 La calidad del software 222
4.2.1 Una visión estratégica de la calidad 222
4.2.2 La visión de Deming 225
4.2.3 La efectividad de la tecnología de información 231
4.2.4 El desafío del futuro 233
4.3 Recomendaciones en torno a la calidad del software 235

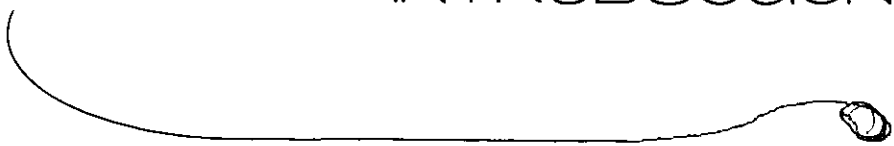
Conclusiones 239

Anexos 247

Anexo A: Instituciones dedicadas a la tecnología de software 247
Anexo B: Procesos de software: el modelo del SEI 251
Anexo C: El proyecto COQUAMO 260

Bibliografía 265

INTRODUCCION



Desde los inicios de la civilización la humanidad ha necesitado información como una ayuda en la lucha por la supervivencia, así como en los intentos por administrar las organizaciones. La creciente complejidad de la sociedad, sobre todo en la forma en que se manifiesta en las organizaciones sociales, políticas y económicas ha aumentado en gran medida la necesidad de tener información más oportuna y conveniente.

La aparición de la computadora marca un hito en el manejo de la información. Por primera vez ésta toma características nuevas: es más oportuna, su proceso es más rápido y puede llegar con mayor rapidez a un número cada vez más grande de usuarios.

Los datos procesados por las computadoras dependen del software para su uso. En un principio, el software se reducía a la colocación de cables de manera que los datos procesados tomaran un cierto flujo de proceso. Con el avance tecnológico y la aparición de computadoras de más tipos —personales, minicomputadoras, mainframes— el desarrollo de las mismas exigió la complejidad del software que las operaría para resolver y automatizar problemas y tareas complicadas.

En la actualidad, muchos de los aspectos de la vida cotidiana —incluso aquéllos que no son tan evidentes—, están controlados por computadora. De la correcta operación y funcionamiento de los sistemas de cómputo que los soportan

dependen los vuelos comerciales, las transacciones bancarias, la red de energía eléctrica que llega a nuestros hogares o el cálculo para el suministro de elementos de los cuales dependemos como el agua o la transportación de alimentos.

Así, la correcta operación y funcionamiento de computadoras se traduce en la seguridad de las personas que viajan en un avión, en la garantía de que una comunicación telefónica pueda ser realizada o en la seguridad de que importantes operaciones comerciales puedan llevarse a cabo mediante transferencias de fondos entre bancos. Si algo falla, indudablemente tendría consecuencias catastróficas.

La necesidad de calidad en la producción de software parece ser obvia. El grado de dependencia que la sociedad ha creado con las computadoras en todos los campos obliga a que éstas operen con un alto grado de confiabilidad y eficiencia. Aunado con el acelerado avance tecnológico que se ha operado en las décadas recientes en el campo de la informática y la ingeniería, ha aparecido una nueva necesidad: corresponder la complejidad del hardware con la eficiencia en la producción del software.

Desafortunadamente la calidad no es un concepto fácilmente aplicable a un ente que carece de existencia física, como lo es el software. Generalmente los errores que se corrigen, pueden aparecer nuevamente. Cuando es posible detectar los errores producidos por programas de computadora, frecuentemente ya se han generado costos (retrasos de producción, posibles reprocesos, asignación de personal para corregir las fallas, etc.) que indudablemente afectan en la calidad del servicio que se ofrece.

En un entorno donde el proceso de información se hace importante, las piezas que articulan esta maquinaria deben ser las mejores. Una de ellas es el software. Su importancia es crucial. Aún la mejor computadora, la más avanzada tecnológicamente, puede resultar inservible si el software que la utiliza es de mala calidad.

¿Qué es, entonces un software de calidad? Contestar la pregunta tiene que ver con diversos factores.

En el presente siglo el auge de la calidad en los procesos de manufactura ha llevado a la aplicación de revolucionarias teorías con resultados sorprendentes. Países como Alemania o Japón, víctimas de la guerra y de economías destrozadas han renacido como fuertes potencias económicas. Ambos son pueblos tenaces, perseverantes y trabajadores que han sabido hacer de la calidad una forma de vida. El aprovechamiento de recursos, tanto materiales como humanos, donde el despilfarro no puede permitirse, ha sido la constante del cambio.

Las importantes teorías que sobre la calidad de los procesos productivos se han implementado en tales países producto del trabajo de teóricos como Deming y Juran, pueden llevarse también al campo de la creación de software.

El software tiene características que lo hacen particular dentro del contexto de un proceso de manufactura. Se trata de un producto intelectual, no es tangible, su existencia es efímera en tanto que cobra vida sólo cuando se ejecuta en una computadora. Es el producto del análisis y la síntesis de un problema al mismo tiempo que es una solución del mismo. Como producto es único. El desarrollo de software se realiza una sola vez y su duplicado constituye un proceso simple y libre de cualquier error.

Entender al software no solamente como un conjunto de programas sino en un contexto más amplio, como el producto de un proceso de análisis, diseño y desarrollo, en donde participan muchas personas a distintos niveles constituye un paso importante en el camino hacia la creación de software de calidad.

El desarrollo de software debe entenderse como un proceso creativo, no sólo como una larga secuencia de instrucciones escritas en algún lenguaje de computadora. En este sentido su contexto es el ciclo de vida de sistemas que implica una serie de etapas que van desde la definición del problema hasta el

mantenimiento del sistema, pasando por las fases de análisis, diseño y desarrollo.

Sin importar el modelo que se utilice para el desarrollo de sistemas —cascada, modelos en espiral, prototipo, etc.— éste se relaciona con alguna metodología de trabajo que conduce de principio a fin, en un proceso *continuo*, la metamorfosis de un problema, de ser un problema nebuloso a ser una solución por computadora. El objetivo es la creación de software que cumpla con las especificaciones y satisfaga las necesidades que lo invocaron.

La calidad del software es la cobertura de ciertos criterios definidos en un modelo que garantice el cumplimiento con las especificaciones y la solución al problema. No sólo se trata de eliminar errores. No basta con crear programas sin código muerto, entendibles y de sintaxis correcta. No será suficiente liberar un producto que si bien cumpla con las especificaciones requeridas por el usuario, lo abandone en sus procesos de mejora y no le ofrezca soporte después de su instalación.

Se trata también de cumplir algún grado razonable —definido en un entorno específico— de características inherentes al producto, al software. Se debe cumplir, si así se requiere, con niveles de confiabilidad, disponibilidad, facilidad de mantenimiento, portabilidad, etc. con las restricciones de tiempo, costos y recursos disponibles propios de cada proyecto. Se habla de llevar en todo momento la calidad en la mente como objetivo, implementando mejores procedimientos durante el ciclo de vida del sistema.

Los criterios que definen un prisma de calidad no son sólo adjetivos; son el producto de un trabajo en equipo donde los participantes toman un papel crucial. La calidad no se adhiere al software, dice Deming. No es algo que pueda añadirse para darle un toque de excelencia. Es el producto del trabajo razonado, metódico y consciente de cada integrante dentro del desarrollo del sistema. Es tener un compromiso con la calidad y con la mejora continua, no sólo del producto en sí —del software—, sino de los procesos que lo crearon.

Hablar de software como producto intelectual, es hablar de personas. Es hablar de cultura, de preparación, de conocimientos. Hablar de calidad del software se traduce en actitudes hacia el trabajo, en la forma en que una organización toma el compromiso con la calidad en general, a la implementación de técnicas y herramientas que garanticen en todo momento la presencia de la calidad.

Prefiero hablar del software como un proceso creativo porque en él intervienen el talento, la imaginación y la actitud hacia la vida de las personas involucradas. En pocas palabras, la cultura es el entorno en la creación del software. Hablo no sólo de la cultura organizacional, sino de la cultura de los individuos.

Sin duda el gran desafío de la calidad es el cambio de cultura hacia la misma. Países como Estados Unidos que históricamente se consideraron como punta de lanza en el campo tecnológico, se ven actualmente preocupados por la productividad y eficiencia de los procesos de desarrollo de software frente a la amenaza de países, *de culturas*, que le han arrebatado la supremacía en el concierto mundial.

Dentro de una organización, el desarrollo de software se gesta dentro de un ambiente determinado. Sólo con el compromiso de todos y cada uno, la calidad no será vista como un ente por separado que se incorpore al sistema desarrollado, sino como un conjunto de prácticas estándares corporativas, como una forma de actuar cotidiana; en suma como una *cultura*, que siendo inherente a los procesos, a las personas, sea el ambiente donde la calidad se imprima de manera natural.

Un objetivo del presente trabajo es mostrar los temas más sobresalientes en el panorama de la calidad del software, que permitan al lector ubicar al desarrollo de software dentro del marco de la calidad.

Parece increíble que en la actualidad, cuando los sistemas de cómputo han tomado una complejidad antes insospechada, cuando la ciencia y la tecnología han llevado al hombre a conocimientos tan maravillosos sobre el Universo, el debate sobre la calidad del software en la comunidad especializada sea visto

como un tema adicional, acogido generalmente por proyectos con continuos tropiezos y difíciles patrocinios y opacado por el brillo que en la actualidad el avance tecnológico despliega.

A lo largo de estas páginas se muestra un marco teórico de la calidad, los elementos que la componen para el software abordando aspectos de la gestión de la calidad y temas relacionados como estándares y metodologías. Es una constante a lo largo de la exposición, el tema del factor humano dentro del desarrollo de sistemas, por la enorme relevancia que esto tiene en el proceso del software.

El primer capítulo expone el significado de la calidad, los obstáculos que hay al definirla y las particularidades de la calidad del software. Se expone la calidad del producto y la calidad del proceso en el contexto de una visión global del proceso creativo del software.

En este primer apartado se inicia la exposición del cambio de cultura como factor necesario en el camino hacia la calidad. Se presentan también los modelos clásicos en el diseño de software —especialmente modelos jerárquicos— y la relación que éstos tienen en la producción de software de calidad. Esto intenta reforzar la idea de que la calidad final de un producto de software es resultado del cuidado de la calidad en todas las etapas del proceso creativo, no solamente en la etapa de programación.

Aquí se define el marco teórico sobre del entorno de la calidad visto tanto desde la perspectiva técnica (análisis, diseño y desarrollo del programa) como de la administrativa (visión empresarial de la calidad) destacando la unión necesaria entre ambas para el correcto manejo de los conceptos de la calidad del software.

En el capítulo dos se exponen tres temas particularmente importantes para la calidad del software: métricas, reuso y confiabilidad.

Cuando existe calidad puede pasar inadvertida; cuando no existe, se hace evidente. Para poder mejorar la calidad del software se hace necesario medirla, clarificando las variables sujetas a medición en el desarrollo de software; si no

existen indicadores que reflejen el grado de calidad en el software como producto y como proceso, de poco servirán los esfuerzos que hagan las organizaciones por conseguir la calidad.

El problema de determinar un modelo que represente la calidad del software es motivo de discusión y debate entre especialistas. La solución apunta a definir qué medir en cada entorno específico, creando un conjunto de características importantes que deben mejorarse continuamente. En este contexto se habla de las métricas posibles del software mostrando un panorama general de su uso.

Un factor importante para crear software de calidad es la capacidad que éste pueda tener para ser utilizado en otros sistemas, evitando la multiplicación de esfuerzos en el proceso de desarrollo. El reuso del software es un concepto muy expandido actualmente y que ha probado —cuando se aplica eficientemente—, su capacidad de incrementar los niveles de productividad y por tanto, calidad en el desarrollo de sistemas.

A este respecto, se expone el reuso en un sentido amplio, que incluye la reutilización —reciclaje— de cualquier componente inserto en la maquinaria del desarrollo de software, que sea útil en nuevos proyectos. Se expone el reuso de código, de diseños, de especificaciones y de personal.

El último concepto desarrollado es el de *confiabilidad*. La confiabilidad del software tiene que ver con el manejo de las fallas que presenta un sistema durante su operación. Se diferencian los conceptos de *falla* y *error* para presentar esquemas generales que analizan la confiabilidad del software y que implementan medidas para lograr ese objetivo.

El capítulo tres tiene como objetivo mostrar un panorama amplio en el que la calidad del software se implemente. ¿Cómo se implementa? ¿Qué partes del proceso toca? ¿Cuáles son los elementos relevantes en la tarea de administración en una organización que ha adquirido el compromiso con la calidad del software?

Los aspectos metodológicos en el desarrollo de software se abordan como el marco en que la calidad tomará vida. Se exponen diferentes formas de desarrollar

software para destacar lo necesario del progreso ordenado y consciente —metodológico—, en la creación del software.

Se ahonda también en los llamados *sistemas de calidad*, los procesos que sigue la elaboración de software. Se destacan aspectos importantes en el establecimiento de sistemas de calidad: complicaciones en el levantamiento de información y concreción de requerimientos de usuario, o la problemática del factor humano en el establecimiento de los sistemas. Se destaca la importancia no sólo de la instalación sino también del impulso y desarrollo de una cultura de la calidad en todos los participantes de los procesos de creación de software.

Se discute brevemente el significado de las herramientas CASE y el papel que juegan en el desarrollo de sistemas en la actualidad en relación con la creación de software de calidad, producido en tiempo y bajo restricciones de costos.

La última parte del capítulo se refiere a algunos de los estándares más populares utilizados en la certificación de la calidad del software y de los procesos asociados. Se muestra su propósito y las ventajas e inconvenientes de utilizarlos.

El cuarto capítulo desarrolla el concepto de fábrica de software como camino a seguir en la actualidad y hacia el futuro en el desarrollo de sistemas. Se aborda la importancia de los procesos que existen en la creación de software y se relacionan con las ideas desarrolladas por Deming en el campo de la manufactura y que él mismo sugiere aplicar en el ámbito de la ingeniería de software.

Se destaca en la exposición la visión estratégica que debe imperar en el contexto de la calidad del software y la correspondencia que debe existir entre la estrategia de negocio y la estrategia de sistemas en las organizaciones.

Adicionalmente se incluyen como anexos una breve lista de Instituciones con funciones de investigación, educación o servicios relacionadas con la calidad del software; se presenta brevemente el proyecto COQUAMO, de calidad desarrollado en Reino Unido y se detalla el modelo de madurez organizacional desarrollado por el Software Engineering Institute que es actualmente

ampliamente aceptado para la ubicación de las empresas en nivel de evolución en función de sus procesos, con miras a mejorarlos en el marco de la calidad.

Finalmente, el presente trabajo intenta motivar al lector para que su visión cambie en torno a la calidad. No sólo cuando se desarrolle software hay que considerar hacer las cosas bien desde el principio. Esto redundará en procesos más eficientes y cada vez que se revisen, mejores. Aumentará asimismo la productividad y fomentará la educación e investigación. En suma, producirá mejores profesionales y mejores sistemas.

No hay fórmulas mágicas ni mucho menos inmediatas. Sabiendo que el camino hacia la calidad del software es un proceso, queda en las manos de las organizaciones elegir ese camino y en la capacidad de los individuos hacerlo posible, no sólo por hacer mejor software, sino también para ser mejores profesionistas.

1

TEORIA DE LA CALIDAD DEL SOFTWARE

"La calidad es difícil de definir, imposible de medir, fácil de reconocer"

B. Kitchenham

1.1 Introducción a la calidad

Siempre que se habla de un tema se hace necesario definir los términos que se usan. Sin embargo, un término como *Calidad* puede resultar ambiguo y por lo tanto difícil de definir.

La calidad es generalmente transparente cuando está presente pero fácilmente reconocible cuando está ausente. Este es el caso de cuando se descompone un auto o de cuando no podemos efectuar una transacción bancaria porque "no hay sistema".

La naturaleza que engloba al término *calidad* nos remite a pensar en grados de excelencia, economía, eficiencia, confiabilidad, buen servicio. En los siguientes párrafos se abordará el concepto en su aspecto multifuncional y multifacético.

1.1.1 ¿Qué es la calidad?

Definir el término *calidad* tiene ciertas dificultades. Existe un amplio rango de definiciones del término, algunas académicas y otras que la experiencia ha forjado, pero todas enmarcadas dentro de un cierto contexto. Según Deming, *la calidad sólo puede definirse en función del sujeto.*

Las siguientes son algunas de las definiciones para el término:

Calidad [1]***:*** *Manera de ser de una persona o cosa:* artículo de buena calidad; *Clase:* tejidos de muchas calidades; *carácter, genio, índole, valía, excelencia de una cosa (...)*

Calidad : *Cumplir especificaciones. Cero defectos (P. Crosby).*

Calidad: *cumplimiento de los propósitos. Adecuación para el uso satisfaciendo las necesidades del cliente (J.M. Juran).*

Calidad : *Es un grado predecible de uniformidad y fiabilidad a bajo costo, adecuado a las necesidades del mercado (E. W. Deming).*

Calidad : *Costo que un producto impone a la sociedad desde el momento de su concepción. (G. Tagushi).*

Calidad : *La totalidad de las características de un producto o servicio puede basarse en su habilidad para satisfacer especificaciones ó necesidades implícitas (ISO, 1986).*

La acepción que aquí interesa tiene que ver con la manera de ser del objeto y también con su valía, con su excelencia.

Si se tuvieran tres autos: un Rolls Royce, un Ford Fiesta y un Volkswagen Sedán responder a la pregunta ¿cuál es el de mejor calidad? no sería una cuestión trivial. Un análisis comparativo básico tendiente a responder esta cuestión se presenta en la Tabla 1-1.

El cuadro muestra algunas características del término *calidad,* dignas de ser mencionadas:

---

[1] Diccionario Larousse de la Lengua Española. México, 1979.

a) ***La calidad no es absoluta***. Puede significar diferentes cosas en diferentes situaciones. Tanto el Rolls Royce como el VW Sedán representan productos de calidad, pero en diferentes sentidos. *"La calidad no puede ser medida con base en una escala cuantificable, en el mismo sentido que una propiedad física como la temperatura o la longitud"* [2].

b) ***La calidad es multidimensional***. No puede resumirse en una simple medida cuantitativa, ya que siempre se relaciona con diferentes factores. Algunos de ellos son medibles, como el rendimiento del combustible o la máxima velocidad de ejecución. Otros, como el terminado de la pintura o la belleza del diseño son factores no medibles objetivamente pero que sí contribuyen en la asignación del nivel de calidad.

c) ***La calidad está sujeta a restricciones***. El primer criterio para preferir comprar un Ford Fiesta en lugar de un Rolls Royce es el precio.

   Este factor puede ser importante pero lo es más el hecho de que los recursos en el proceso de producción que se relacionan con ese precio accesible, tales como personal calificado o buenos materiales, pueden ser limitados. Esto lleva a tener un límite de producción fijado por la existencia de los recursos de producción. En tal caso la calidad se supedita a esta disponibilidad de recursos.

d) ***La calidad se relaciona con compromisos aceptados***. Donde la calidad está restringida y se adquieren algunos compromisos, ciertos criterios de calidad pueden ser más fácilmente aceptados que otros. Así, algunas características del producto pueden ser más importantes que otras. Por ejemplo: la confiabilidad de un automóvil puede ser más importante que el lujo en sus detalles; se puede preferir un mejor rendimiento de combustible en vez de un diseño moderno. Los criterios de calidad que no puedan ser omitidos serán atributos críticos y conformarán el marco con el que la calidad se compromete.

---

[2] Gillies, A. (1992) Software quality, theory and management. Londres: Chapman. p.5

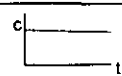
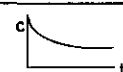
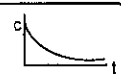
Características	Rolls Royce	Ford Fiesta	Volkswagen Sedán
Apreciación subjetiva de calidad	Se considera tradicionalmente como sinónimo de calidad. La calidad en este auto se considera sin restricción de precio.	Se considera como de buena calidad, pero es una solución de transporte orientada a las masas. El término calidad se relaciona con los niveles de especificación, calidad de manejo y buenos terminados, pero las expectativas son menores si se considera que es un producto diseñado para un cierto precio.	La calidad en este auto se relaciona con su bajo precio. Si se consideran factores como comodidad, buen manejo o rendimiento en altas velocidades, la batalla está perdida contra otros modelos. A pesar de su buen precio, si se consideran factores a largo plazo como confiabilidad y durabilidad se pierde atractivo hacia este auto. Un auto barato que cumpla cierto nivel de especificación es aceptable, pero un auto barato que incurra en compromisos a largo plazo, no representa una buena solución de calidad.
Detalles de producción	En su producción se cuidan todos los detalles, sin importar el costo. Producción sobre pedido.	Producción en serie a gran escala.	Producción masiva en serie.
Percepción de la manufactura	Se percibe una excelente manufactura resultante de cuidados y largos procesos de producción.	Bien cuidada; énfasis en el aspecto funcional del producto.	Sólo se cuida el aspecto utilitario del producto.
Especificaciones de diseño	Precisas y clásicas.	Funcionales e innovadoras.	Aceptables
Terminados	Excelentes	Buenos	Aceptables
Tiempo de vida	Largo	Medio	Corto
Rendimiento de combustible	Moderado	Bueno	Excelente
Condiciones generales de manejo	Excelentes	Buenas	Aceptables
Niveles de seguridad	Excelentes	Aceptables	Malos
Nivel de lujo	Alto	Medio	Ninguno
Precio de venta	Muy caro	Medio	Bajo
Confiabilidad a través del tiempo			

Tabla 1-1 Comparación de calidad en tres modelos de automóvil

e) ***Los criterios de calidad no son independientes***. Los criterios que determinan la calidad (precio, rendimiento, comodidad) no son independientes e interactúan unos con otros causando conflictos. Por ejemplo, el rendimiento de la gasolina se relaciona inversamente con la velocidad a la que se maneja un auto. Así, si un auto de calidad es aquél que desarrolla una gran velocidad con un uso óptimo de combustible, se presenta un conflicto entre esos dos factores deseables. El equilibrio que se logre entre ambos, determinará en buena medida el nivel de calidad general que tenga el auto.

Las características de un producto o servicio que se relacionan con su grado de calidad muestran que ésta tiene sentido siempre dentro de un contexto. Por lo tanto, varias pueden ser las percepciones del término *Calidad*. Van Genuchten [3] distingue entre cinco definiciones de calidad:

- Definición trascendental
- Definición basada en el usuario
- Definición basada en el producto
- Definición basada en la manufactura
- Definición basada en el valor

La definición trascendental dice que la calidad es sinónimo de excelencia innata y de acuerdo con esto, la calidad es absoluta y universalmente reconocible a pesar de que no se pueda definir con toda precisión.

La calidad existe, buena o mala, y puede percibirse; *"se siente"* de manera natural dentro del contexto del producto.

La definición basada en el usuario se ejemplifica con la definición de Juran: *"la calidad es ajustarse al uso"*. Considerando que diferentes usuarios tienen diferentes necesidades, aquellos productos que mejor satisfagan las necesidades del usuario son considerados como los de mayor calidad.

---

[3] Van Genuchten, M. (1992) Towards a software factory. Holanda: Kluwer, p. 51

Desde este punto de vista la calidad es subjetiva porque depende de quien la percibe. El producto que para alguien significa de buena calidad, en tanto que le es suficientemente útil, práctico o barato (si esas son sus variables de calidad), para otro puede no satisfacerle y considerarlo de calidad inferior.

La definición basada en el producto ve a la calidad como una variable precisa y medible. Las diferencias en la calidad reflejan diferencias de algunos atributos que posee el producto. Esta definición ve a la calidad como una característica inherente al producto, no sólo como algo adicional a él, afirmando por tanto que la calidad es objetiva.

Lo que no es medible no existe, es una máxima de quienes utilizan esta definición. Si los atributos del producto no se pueden medir, no se puede hablar de su calidad.

La definición basada en la manufactura considera a la calidad como la conformidad con las especificaciones. Este tipo de definición es el más aceptado por los desarrolladores de software porque un elemento vital en el proceso del software son precisamente las especificaciones, los requerimientos que le dan origen.

Si un producto o servicio cumple con lo esperado, con lo establecido de antemano, se puede hablar de cierto nivel aceptable de calidad. Ésta existe en la medida en que las características del producto sean completadas.

La definición basada en el valor considera a la calidad no aislada como en las anteriores definiciones, sino en relación con los costos. Esta definición considera un producto de calidad el que ofrece buena ejecución a un precio aceptable [4].

[4] A este respecto, la ingeniería de software debe conducir a un balance entre requerimientos de calidad por un lado, y tiempo y costos por el otro.

Los diferentes tipos de definición pueden ser clasificados para salvar el obstáculo que implica la conexión entre el producto y la calidad en su manufactura (Tabla 1-2).

Definiciones de calidad del producto	***Objetiva***	***Subjetiva***
Punto de vista del usuario		Trascendental Basada en el usuario Basada en el valor
Punto de vista del ingeniero	Basada en la manufactura Basada en el producto	Trascendental Basada en el valor

Tabla 1-2 Clasificación de las definiciones de calidad

La clasificación considera dos figuras importantes en todo proceso: el usuario, quien utiliza el producto pensado en sus necesidades, y el ingeniero quien lo diseña y produce valiéndose de procesos y técnicas.

El cuadro clasifica las definiciones trascendental, basada en el usuario y basada en el valor como *subjetivas*. Lo son tanto desde el punto de vista del usuario como desde el punto de vista del ingeniero ya que las tres dependen de un criterio intangible; la calidad en cada una de las definiciones no se está midiendo sino *percibiendo*.

Por el contrario, las definiciones basada en la manufactura y basada en el producto se consideran como objetivas ya que implican una *cuantificación* en términos aceptados ampliamente. Ambas también se relacionan entre sí ya que se considera que una objetiva definición basada en el producto es la base para una definición objetiva basada en la manufactura. Estas dos definiciones representan el punto de vista del ingeniero y se establecen en términos utilizados por ellos.

La tabla de clasificación muestra que no existe una sola definición objetiva de *calidad*. Las definiciones trascendental y basada en el usuario son compartidas por usuarios e ingenieros. El hecho de que tales definiciones sean subjetivas les

resta valor para establecer un lenguaje que exprese claramente los requerimientos de los usuarios. En la medida que uno comprenda las necesidades del otro, estableciendo un lenguaje común y claro, se podrá obtener un mejor producto.

1.1.2 El software

¿Qué es el *software*? La respuesta no es tan simple de explicar como se esperaría. Entender algunas de las características del software ayudará tanto en su definición como en las particularidades que ésta presenta cuando se le relaciona con el concepto de calidad:

- ***El software no tiene existencia física***. Al ser un producto intelectual, no se le puede tocar, medir en su dimensión real. A diferencia del hardware, es una entidad lógica en lugar de física.

  Su naturaleza efímera le ha creado complicaciones históricas. En un principio, el software se regalaba en la compra del hardware. Esto le restó importancia en sí, comparado con la construcción de computadoras. Hoy es muy difícil que una compañía pague una cantidad igual de hardware por un software de buena calidad sin pasar por largos procesos de evaluación y autorización. Esta situación es el resultado de un malentendimiento de la producción intelectual: el software se hace una vez y puede copiarse ilimitadamente.

- ***Su existencia cobra significado sólo con la intervención de una tercera figura: la computadora***. Ambos elementos se deben la existencia mutuamente; sin embargo la computadora existe, es visible; el software sólo se nota durante su ejecución y a través de los resultados que produce (reportes, archivos, etc.)

- ***El software no es un producto terminado***. Evoluciona a la par que las necesidades que lo crearon. La solución por computadora de cualquier problema cambia a lo largo del tiempo en razón de la complejidad del problema que está orientado a resolver.

- ***Su hábitat cambia constantemente***. La producción de software tiene que responder a la velocidad con que la tecnología produce computadoras, *hardware*, cada vez más sofisticado, rápido, adecuado para situaciones más complejas. El software con frecuencia no tiene un solo hogar y debe adaptarse a cada uno rápida y eficientemente.

- ***El software depende de requerimientos***. Los usuarios que demandan soluciones de informática no siempre tienen un producto final en mente que cubra sus expectativas. Al cambiar sus requerimientos cambia el software que necesita.

  Es muy común en la literatura actual sobre calidad del software encontrar relevante este punto. Para muchos, especificar los requerimientos es el problema crucial.

  La práctica cotidiana apoya esta teoría. Para una organización es muy difícil poner en términos claros y no ambiguos lo que necesita. A veces esto se debe a la complejidad técnica de lo requerido, pero frecuentemente se debe a que no está claramente definido lo que se quiere.

- ***El software no se fabrica: se crea, se desarrolla***. Al no ser un producto manufacturado, no se compone de piezas ensambladas en una línea de producción. En este sentido no debe verse como un objeto sino como una obra cuyo resultado final depende de un conjunto de factores humanos [5].

  Aunque existen similitudes entre el desarrollo de software y la fabricación de hardware, ambas actividades son fundamentalmente diferentes. En las dos, la buena calidad se logrará con un buen diseño, pero en el hardware se pueden presentar problemas de calidad que en el software no existen o son corregibles. Las dos actividades dependen de personas, pero la relación entre ellas y el producto es completamente diferente para el software, debido principalmente a su naturaleza intelectual.

---

5 Edward Yourdon en Decline and fall of the american programmer muestra el concepto complementario ***peopleware***, que se refiere a la intención de las organizaciones para focalizar sus esfuerzos en el mejoramiento del software principalmente en los recursos humanos. Más adelante se tocará este punto en el presenta trabajo.

Los costos de desarrollo de software están en la ingeniería, en el proceso que se sigue para su creación, no en los costos de materiales. Esto significa que los proyectos de software no se pueden gestionar como si fueran proyectos de fabricación [6].

- ***El software no se estropea.*** El nivel de fallas en un equipo de cómputo a lo largo del tiempo se muestra en la Figura 1-1. Esta relación conocida como *curva de bañera* indica que al principio de su vida un equipo puede mostrar relativamente muchas fallas que, una vez corregidas, disminuyen el índice de fallas a un nivel estable durante cierto tiempo. Producto del uso del equipo, este presentará nuevamente fallas justo antes del final de su vida.

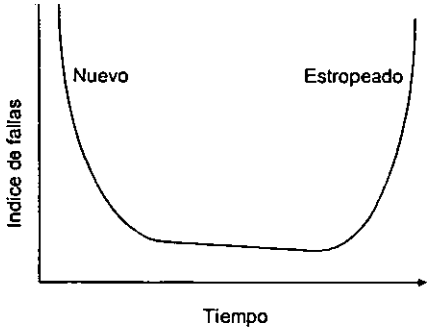


Figura 1-1 Curva de fallas del hardware [*]

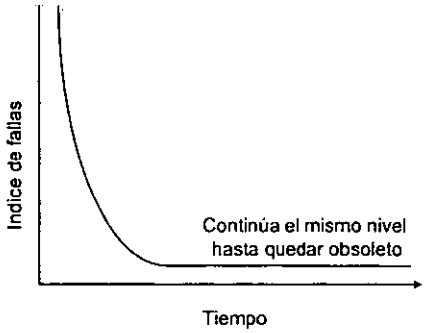


Figura 1-2 Curva idealizada de fallas del software

El software no es susceptible de los mismos males del entorno que hacen que el hardware se estropee. La Figura 1-2 muestra la curva idealizada de fallas para el software. Los defectos no detectados harán que el software falle en las primeras etapas de su vida. Suponiendo que no hay nuevos errores, la curva se aplana a un nivel estable. La implicación es clara: el software no se estropea si no se modifica, pero sí se deteriora al modificarse.

---

[6] A este respecto, Pressman aclara en Ingeniería del software que si bien se ha desarrollado el concepto de *Fábrica de software* en la pasada década, éste no implica que el desarrollo de software y la fabricación de hardware sean equivalentes. El concepto *Fábrica de software* recomienda el uso de herramientas para el desarrollo automático del software.

[*] Pressman, R. (1993) Ingeniería del software. España: McGraw-Hill. p.12

La Figura 1-3 muestra la curva de fallas de software en un ambiente más real: cuando el sistema sufre cambios. Conforme éstos tienen lugar, es posible que se introduzcan nuevos defectos, haciendo que las fallas presenten picos. Antes de que el nivel de fallas se estabilice, se presenta otro cambio. Lentamente, el nivel mínimo de fallas comienza a crecer; el software se va deteriorando debido a los cambios.

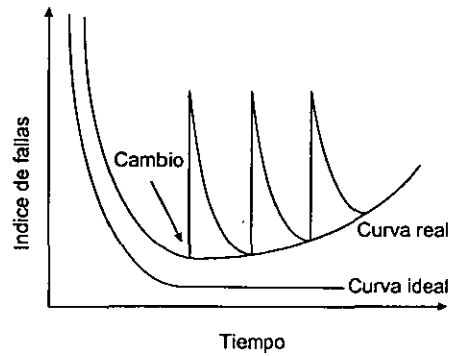


Figura 1-3 Curva real de fallas del software *

Las características del software lo hacen ver poco común para la mayoría de la gente, a pesar de que, indudablemente, muchas de sus actividades cotidianas sean resueltas por soluciones de software. ¿Es entonces un producto? ¿Es un servicio? Las siguientes definiciones aclaran el término:

Software [7] : *Datos escritos o impresos, tales como programas, rutinas o lenguajes simbólicos esenciales para la operación y mantenimiento de computadoras.*

Software [8]: *(...) aquellos programas, procedimientos, reglas y documentación posible asociada con la computación, así como los datos pertenecientes a la operación de un sistema de cómputo*

* *Ibid.*

7 The American Heritage Dictionary, EU, 1983. *Trad. del aut.*

8 Fairley, R. (1987) Ingeniería de Software. México: McGraw-Hill p. 2

> *"El término* **software** *es aplicable a cualquier programa escrito para computadora.(...) Un* programa *es una serie de instrucciones dadas a la computadora, que causan la ejecución de una tarea o secuencia de acciones. (...) Esencialmente la diferencia entre* software *y* hardware *es que el primero se relaciona con lo que es procesado en la computadora mientras que el hardware describe el equipo usado en este proceso"* [9].

Por *software* se puede entender no sólo la serie de programas que se desarrollan para resolver o automatizar un problema en particular; no se limita a la solución de problemas comunes en las oficinas: es a la vez el conjunto de programas que soporta el trabajo de las computadoras para garantizar su servicio (sistemas operativos, sistemas de monitoreo de redes, etc.). Voss (1991) señala:

> *"¿Qué hay de difícil en el software? La principal razón es que el proceso de desarrollo de software no es entendido por gran parte de la industria. Es muy difícil controlar algo que no es entendido. Tampoco su importancia es aceptada".* [10]

Entender qué es el software y sus particularidades ayuda a definir el marco en el que la calidad opera, ya que un conjunto de programas, a los que se refieren las definiciones anteriores, es a la vez intangible y su presencia puede ser notada en el momento que un usuario ve resueltas sus necesidades de automatización.

1.1.3 El problema del software

Se calcula que se gastan 40 billones de dólares anualmente en procesamiento de datos, sólo en Estados Unidos. Y de acuerdo a estimaciones, las cifras se cuadruplican aproximadamente cada diez años [11].

En un principio, el hardware representaba la mayor parte del costo en una solución integral. Hoy, el costo de las computadoras desciende cada década,

---

9 Orilia, S. (1982) Introduction to business data processing. E.U.: McGraw-Hill. Se manejan también términos relacionados como "software del sistema" (*system software*) que es el conjunto de programas que controla las actividades de un sistema de cómputo o "software aplicativo" (*applications software*) que se refiere a programas desarrollados para un fin específico (p. ej. sistemas de nómina o de inventario).

10 Voss, A. (1991), The real importance of software quality and the role of quality systems. en Ince, D. (Ed) Software Quality, reliability tools and methods. EU: Chapman.

11 Dunn, R. (1990) Software Quality, concepts and plans. EU: Prentice Hall. p. 2

abriendo un campo cada vez mayor, para el florecimiento de ambiciosos proyectos de software.

A pesar de que los requerimientos de software son cada vez más complejos, el avance tecnológico parece no respetarle el paso (Figura 1-4).

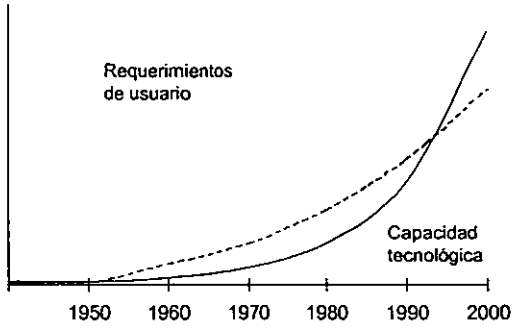


Figura 1-4 Requerimientos de usuario vs. capacidad tecnológica [*]

Mientras se abarata el hardware creando una demanda insaciable de más software, nos preocupamos más del costo de este software y de la productividad asociada en su creación.

De hecho, se estima que aun con la introducción de la programación estructurada, herramientas de diseño o lenguajes modernos, la tasa más optimista de incremento en la productividad en el campo de la programación es el 5% anual[12].

El costo no es el único problema. A quienes producen software en Estados Unidos no les preocupaba ser un segmento profesional bien pagado, en comparación con similares canadienses, franceses o ingleses. Peor aun es el caso de personal de Brasil, Singapur o India, que ganan 5 o 6 veces menos; o los Chinos, que a principios de los 90's se cotizaban en 180 dólares mensuales, con el mismo grado académico que su similar norteamericano.

[*] Shiller, L. (1990) Software Excellence. EU: Yourdon Press, p. 7

12 *Ibid.* p. 3

El panorama en México no parece ser afortunado en el concierto mundial donde el sueldo de un profesional de sistemas puede ser 3 o 4 veces más bajo a un similar de Estados Unidos.

La productividad, en relación con los costos es una relación que los norteamericanos tendían a ver, en su caso, como una suerte de posición privilegiada asumiendo que en otras partes se producía software de menor calidad. A la par, sus precios se veían beneficiados por tratados gubernamentales, tarifas arancelarias y negociaciones gremiales.

La balanza ha tendido a cambiar en los recientes años. Se calcula que cuando la productividad por programador en Japón es de 2,000 líneas de código, en E.U. es sólo de 300. Este dato es cuestionable si se considera que los programadores japoneses desarrollaron proyectos menos complejos que los norteamericanos, pero al saber que los primeros han tenido al mismo tiempo una activa participación adicional como ingenieros de calidad, la percepción cambia.

Más importante: en general, un producto de software japonés se libera aproximadamente con 100 veces menos errores que un similar producido en E.U. Aunque no existen encuestas suficientemente amplias al respecto, existe un escenario de desventaja para los norteamericanos (cf. Yourdon, 1992 y Dunn, 1990).

Aparte de los costos, hoy se considera el problema del tamaño del software. Mientras se consideraban proyectos grandes si contenían 100,000 líneas de código, hoy existen sistemas que rompen la barrera del millón de líneas. Esto podía ser ciencia ficción hace algunos años; hoy es una realidad que hay que afrontar.

Con la aparición de maravillas tecnológicas se crean estructuras conceptuales muy complejas que pueden ir más allá de nuestros procesos cognoscitivos. Antes se habían ya experimentado estructuras muy complejas, que van desde al análisis ecológico de una granja, hasta el estudio de la galaxia o del propio cuerpo humano. Pero *"una cosa es estudiar algo que existe, que ha sido diseñado o ha*

evolucionado (...) y otra es inventar una estructura de una complejidad tan grande que confunde a sus creadores" [13]. Como producto de la preocupación del problema del tamaño del software, se han desarrollado empresas y centros de investigación enfocados al problema de la tecnología de software (ver Anexo A).

La mayor parte del tiempo se trabaja dentro de límites establecidos de complejidad. Al mismo tiempo en que las computadoras proliferan incrementando su poder y reduciendo su espacio y su precio, encontramos nuevas aplicaciones para ellas que permiten expandir la creatividad en el desarrollo, pero que también crean dificultades para la administración.

Visto de lejos, la gestión de proyectos de software es la administración de la complejidad. Con el fin de mejorar la tarea de manejar la complejidad se han desarrollado cantidad de trabajos. Uno de ellos es el concepto de *programa de calidad*. Los programas de este tipo incluyen todas las etapas del desarrollo de software, desde el análisis hasta el mantenimiento posterior a la instalación, en un entorno metodológico y de cultura institucional.

Si bien los programas de calidad representan una parte de la solución del software, la noción del *programa de calidad* representa un problema en sí misma: si se utiliza la calidad como un mecanismo para reducir la angustia de la administración del software, debe quedar claro lo que se entiende por *calidad* y sus implicaciones en el software.

1.1.4 La calidad del software

Calidad del software es un término especificamente aplicado en un contexto desarrollado en las últimas décadas y propiciado por la avanzada tecnológica: la programación de computadoras.

Si consideramos a un programa (o a un conjunto de programas) de computadora como un producto en tanto que ellos reflejan un esfuerzo humano en

[13] *Ibid.* p. 3. El autor cuestiona, no sin levantar serias polémicas entre científicos e ingenieros de software, la factibilidad de llevar a cabo proyectos tan grandes como *Star Wars* pues existe la duda del grado de conocimiento humano que se puede tener de un software que rebasa todo límite de complejidad.

su creación (ejecución de análisis, escritura de código, pruebas, etc.), las definiciones presentadas anteriormente pueden ser aplicadas al software.

Sin embargo, se han acuñado algunas definiciones específicamente para referirse a la calidad del software. Gillies (1992) apunta que se debe referir a la calidad del software como *cumplimiento de las necesidades* y establece que la calidad debe emparejarse con las expectativas que se tienen para que resuelva determinado problema.

La definición anterior reconoce dos de las características fundamentales en el rompecabezas de la calidad del software:

1. La conformidad con sus especificaciones y,
2. El cumplimiento del propósito trazado.

Estos elementos se pueden resumir como:

- ¿Éste software representa una buena solución?
- ¿Soluciona el problema real?

El Departamento de Defensa de EU define a la calidad del software como *"el grado en el que los atributos del software le permiten llevar a cabo su propósito final de uso"* [14], lo cual lleva a combinar la necesidad de dar una buena solución a los requerimientos al contestar la pregunta correcta.

La calidad del software se hace complicada particularmente cuando se le compara dentro de un contexto como el de la manufactura. Si se habla de tipos de calidades en este campo, la del software se diferencia principalmente debido a sus propias características mencionadas arriba: su existencia efímera, el cambio de especificaciones a lo largo del tiempo, la evolución del hardware, las expectativas del cliente.

Considérese la comparación con un automóvil. No se espera que el fabricante del motor mantenga al cliente siempre actualizado con las últimas

14 Gillies, A. *op. cit.*, p.7

especificaciones técnicas, pues esto obligaría a reparar o cambiar el motor constantemente. Si el cliente requiere las últimas innovaciones, entonces compra un nuevo automóvil. Éste es el caso de paquetes de software como los procesadores de palabras o programas de utilería. La naturaleza de este tipo de sistemas es que cumplen las expectativas de una completa adaptabilidad y flexibilidad. El objetivo de estas soluciones es proveer un cumplimiento exacto de las *necesidades del usuario.*

El panorama puede ser frustrante si se consideran problemas tales como:

- La respuesta a ¿quién es el cliente? puede ser ¿es un departamento de sistemas? ¿es la dirección de la empresa? ¿la organización en su conjunto? ¿es el usuario final?
- La necesidad de reconciliar diferentes necesidades de usuario, p. ej. *expertos* o *novatos* quienes incluso pueden ser la misma persona en diferentes puntos del tiempo.
- La necesidad cada vez más imperiosa de hacer sistemas que integren a toda la organización.

En el proceso de software existen diferentes actores, que pueden convertirse en su momento en clientes o usuarios. En este sentido es cada vez más cercana la visión de la calidad del software como *apego a las especificaciones* [15].

Se puede considerar que un automóvil provisto de tapicería de piel es de más calidad que otro que tenga asientos terminados en vinilo. Lo mismo puede ocurrir con el software. Análogo a la tapicería de un automóvil, una base de datos relacional es de mayor calidad que otra compuesta por archivos "planos". La primera da al propietario (usuario) más utilidad, mayor conveniencia, o quizá mayor bienestar. Si se analoga el deterioro del tablero de instrumentos con el software, éste no rechinará con el tiempo pero sí se hará más difícil de modificar.

15 Del inglés "fitness for use", se puede entender como que cumple lo requerido, que se disponen de los elementos suficientes para garantizar su uso.

Ambas nociones de calidad del software caen dentro del concepto de Juran: *fitness for use.*

1.1.5 Calidad del producto, calidad del proceso.

Las nociones de calidad citadas tienen que ver más con el producto final, el software de usuario, el *paquete* o *el sistema*, que con el proceso de creación que precedió a su liberación y luego a su utilización por parte del usuario.

Si se restringe a productos de software, la idea de cumplir con los requerimientos es, ciertamente, inclusiva. Sin embargo una aproximación más completa debe necesariamente incluir al *proceso* que resulta en *productos* listos para ser utilizados, incluyendo también los problemas relacionados con el manejo de la complejidad del software.

Esto implica que para resolver los problemas que se han enfrentado en los recientes años en la creación de software, los programas de calidad en software deben incluir la buena ejecución en la programación y deben contemplar la creación de software cada vez más completo. En suma se habla de :

- Productos de software.
- Procesos de software.

1.1.5.1 La calidad del producto de software.

Regresando a las definiciones de calidad, considérese el asunto de las vestiduras de piel contra las de vinilo por un lado, y las tolerancias en la manufactura, por el otro. ¿Cómo se percibe exactamente la adecuación para el uso del producto de software?. Nuestra percepción del cumplimiento dependerá de cómo es usado.

Para ejemplificar esto, tomemos el servicio de cajeros automáticos en un banco. Para el cliente la percepción de cumplimiento del producto está en términos de *confiabilidad* y *disponibilidad*. Cuando llega a un cajero, el cliente

espera tener un tiempo de respuesta si bien no inmediato, sí rápido una vez que introduce su tarjeta. Puede haber un número de excepciones que alteren al servicio, como falta de energía, descomposturas técnicas o simplemente la tarjeta del cliente puede estar dañada. Inmediatamente el cliente siente una *dependencia* de esos factores, para obtener el servicio. Adicionalmente sabe que si se le ofrece un nuevo servicio, digamos el cambio de clave confidencial, suele no funcionar como le explicaron en los folletos de propaganda.

Del lado del banco, la percepción puede ser distinta. Para el personal involucrado también la disponibilidad del servicio es capital, pero entendida en otros términos. Para adicionar nuevos servicios, debieron instalar esa funcionalidad (actualizar el software) sin interrumpir el servicio. Quienes monitorean el buen funcionamiento de los cajeros automáticos pueden no tener (si por olvido de los desarrolladores no se consideró) toda la información a la mano para resolver situaciones que se presenten. Su percepción será tanto mejor cuanto todos los problemas que se puedan presentar puedan ser resueltos rápidamente y cuando los errores del factor humano puedan ser resueltos también por el software.

Si el cliente no encuentra el servicio de cajeros automáticos correcto, acudirá al servicio de sucursal, que a su vez es usuaria de la misma fuente de datos utilizada por los cajeros automáticos para su servicio. Para un gerente de sucursal la calidad en los cajeros automáticos puede ser medida por el número de quejas que reciba del servicio.

Finalmente otro tipo de usuarios son los programadores que dan servicio de mantenimiento al software. Ellos dependen de una buena documentación, suficiente claridad dentro del código y estructuras de software que no violen las especificaciones de los programas que se modifiquen. Por ejemplo, si un programa utiliza ciertas áreas de memoria, se debe garantizar que esas áreas sigan siendo las mismas cuando un cambio al programa se lleve a cabo.

Se puede engrosar la lista de usuarios de un sistema de cajeros, que perciben la calidad del producto, pero el punto se ha mostrado: muchos factores entran dentro de la calidad en los productos grandes de software.

1.1.5.2 La calidad del proceso de software

Quizá la percepción más acabada de la *calidad* en un proceso de programación tenga que ver con el resultado final del proceso: *"el proceso es de gran calidad si el producto resultante es percibido como de gran calidad"* [16].

La calidad de los productos es necesaria pero no suficiente para definir la calidad del proceso. Aunque los productos de buena calidad puedan resultar dependientes de un buen proceso, se deben considerar otras visiones que hay de la calidad, especialmente cuando se habla de los programas de calidad.

Aparte de la incuestionable relación entre producto y proceso, el éxito de un proyecto de software debe medirse no sólo en términos del producto mismo, sino también se deben considerar los eventos atendidos durante su desarrollo. Si el líder de proyecto se detiene a recalendarizar su plan como consecuencia del olvido de especificaciones cruciales, o si el jefe de ese líder toma medidas para que los costos de esas replaneaciones se hagan sin afectar desmesuradamente el presupuesto de la empresa o si el programador ocupara disciplinadamente parte de su tiempo en la aburrida tarea de documentación, estaremos hablando de que el proceso es de gran calidad.

Se puede comentar brevemente que en estos tiempos muchas empresas han visto en los puestos gerenciales de aseguramiento de calidad (o calidad de procesos, cualquiera que sea el nombre en particular) como los depositarios exclusivos de los asuntos de la calidad del software. Esto ha llevado a que los niveles de alta dirección no estén sensibilizados de lo que la calidad del software significa. Más aún, por ejemplo, a pesar de la experiencia y preparación que pueda tener el personal de un departamento de calidad, es común que decidan

---

16 Dunn, R. *op. cit.* p.6

atribuirse tareas tales como las pruebas del software. Aunque toda actividad de prueba es necesaria para asegurar la calidad, las pruebas representan sólo una parte del gran problema de la calidad del producto y por sí solas no contemplan el amplio rango de acción que implica la calidad del software.

Obtener un buen producto es sólo uno de los objetivos de los directivos. Otros son terminar en tiempo, cuidar los costos, y administrar. A un proceso de calidad le interesan todos esos objetivos. Aunque no conozca su nombre, cualquier administrador percibe al proceso de calidad como la extensión de la satisfacción del cliente, el incremento en las ventas o el aumento en las ganancias con costos más bajos. Quienes tienen conciencia de la calidad en el proceso tienen en mente los tiempos de desarrollo, el cuidado de los presupuestos, y el control sobre cada etapa del desarrollo

Un problema importante con los procesos es que éstos son rara vez diseñados. Simplemente suceden, o eufemísticamente, evolucionan. En un principio, una oficina bancaria podía resolver sus proceso informáticos tal vez con un programador y un supervisor. Una década más tarde cuando la oficina intente ser parte de una red de servicios más completa, la naturaleza de los procesos puede no cambiar. Evoluciona rápidamente mediante una sucesión de "parches" al sistema. Eso puede cumplir el objetivo de lograr un producto estable, pero pagando el precio de la evolución a un software cada vez más complejo y difícil de modificar, generado por procesos obsoletos.

El diseño de procesos de desarrollo y mantenimiento de software de manera consciente, es uno de los pilares en los que se apoya la llamada *ingeniería de software* [17]. Parece ser que ésta se enorgullece de serlo, llamando a los programadores ingenieros de software, independientemente del nivel de programación que tengan. En todo caso el establecimiento de procesos de software basados en metodologías razonadas y soportados por tecnología moderna, es el sello del éxito en este campo.

---

17 La frase proviene de "*software engineering*" término acuñado en la Universidad de Munich por el Prof. L. Bauer y que sustituyó, a partir de 1967, a "*software tinkering*", que se utilizó para evocar la idea de crear algo mediante la operación de instrumentos.

1.1.6 Visiones de la calidad

Lo que se ha podido ver hasta este punto es que la calidad es de construcción *multidimensional* (Figura 1-5) . Esto no excluye que se le pueda al mismo tiempo clasificar de acuerdo a un número de perspectivas o puntos de vista. La complejidad del asunto es que esos puntos de vista son muy diversos y pueden causar conflictos unos con otros.

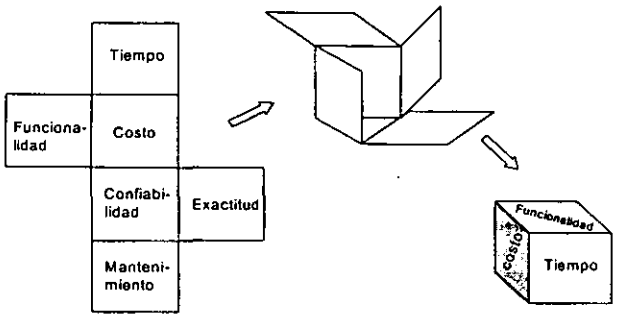


Figura 1-5 La Calidad como construcción multidimensional.[*]

Cada visión proviene de un contexto en particular y por sí solo tiende a darnos un panorama parcial. En este sentido la calidad del software depende de quien la califique. Los integrantes del proceso generalmente poseen una visión de la calidad, razón por la que no es extraño, como se ha expuesto, que la calidad en términos de *usuario o cliente*, sea diferente a la del *diseñador o proveedor.*

En la práctica, los participantes en el proceso de desarrollo de software generalmente forman un espectro mucho más ancho que simplemente usuarios y diseñadores. Dentro de un proyecto de software se pueden identificar las siguientes figuras principales, cada una con sus propias prioridades y preocupaciones.

[*] Tomado de *Gillies, op. cit.*

Líder de proyecto. Tiene la responsabilidad del proyecto del lado del proveedor. Está obligado a entregar un producto confiable y mantenible que satisfaga al cliente. Sin embargo, él tiene fechas de entrega y limitaciones de presupuestos; se ve obligado a adquirir un compromiso de cumplir su cometido tomando en cuenta tales restricciones.

Analista de sistemas. El analista es quien más contacto tiene en el terreno del cliente. Se relaciona con los usuarios y entiende sus necesidades más que cualquier otro en el proceso de desarrollo. Tiende a defender la funcionalidad y las especificaciones técnicas del sistema contra las presiones ejercidas por limitantes externas.

Programador. Es quien escribe el software. Usualmente defiende a ultranza el producto de su trabajo y difícilmente se le convence de que su código no cumple con las especificaciones técnicas. La naturaleza del trabajo que ejecuta dentro del desarrollo, lo hace ver de manera parcial la totalidad; frecuentemente esta característica interfiere al asumir en su práctica profesional la única capaz de modificar el proceso de creación de software.

Auditor de calidad. Dentro de una organización su trabajo consiste en detectar diferencias en la solución de la calidad, sea por defectos técnicos o por una pobre concepción de los requerimientos. *"El auditor no puede ganar. Cualquier determinación suya aumenta la carga de trabajo del programador, afecta negativamente los tiempos y presupuestos del líder de proyectos, y si no actúa logra la insatisfacción del usuario"* [18].

Usuario final. Esta persona tiene poco que ver en el proceso de desarrollo (estrictamente hablando) pero es quien finalmente utiliza el sistema. Las reacciones de los usuarios determinarán en el mediano y largo plazo la aceptación del sistema.

Director de negocio. Es el jefe del usuario y usualmente es quien instiga el proyecto. Debe encontrarle soluciones a los trabajadores y debe justificar el

[18] Gillies, *op. cit.* p. 9

tiempo utilizado en las labores. Esta figura puede tener cualquier nombre dentro de la organización pero es quien, sin utilizar frecuentemente de manera directa un software, impulsa su puesta en marcha; su visión estratégica lo obliga a dar soluciones que reporta a la alta dirección.

Patrocinador del proyecto. El patrocinador es quien paga el sistema. Cuida el progreso diario del proyecto deseando un término exitoso que incremente su prestigio y justifique su gasto. Un proyecto exitoso se relaciona con la entrega a tiempo dentro del presupuesto establecido con ciertos niveles de calidad. Este participante puede ser a su vez el director de negocio, si se habla de desarrollos *in-house.* Tratándose de empresas que proveen soluciones generales de negocios a terceros, esta figura es claramente diferente.

Cada integrante antes descrito tiene sus propios objetivos dentro del proyecto. Muchos de estos objetivos pueden causar conflictos unos con otros. La calidad general que incluya a cada uno de ellos puede juzgarse en términos de qué tan bien cumplen su trabajo al final del día.

Es necesario señalar que quienes llevan desventaja en la resolución de conflictos, son quienes están más cerca de del software y sus efectos: los programadores y el usuario final.

En el apartado 1.1.1. se presentaron y clasificaron cinco definiciones que de manera general pueden aplicarse al término *calidad.* Gráficamente la calidad puede verse como un poliedro conformado como un todo en la medida en que cada punto de vista (visión) le da sentido (Figura 1-6).

En relación con el software, cada una de éstas extensiones de la definición general, pueden entenderse como:

a) ***Visión trascendental.*** Como se refiere a la excelencia innata de un producto, es difícil de cuantificarla y por tanto imposible en un sentido lógico a un proyecto grande de software. Un intento de construir una excelencia innata dentro de un software puede ser limitada por restricciones, especialmente de costos.

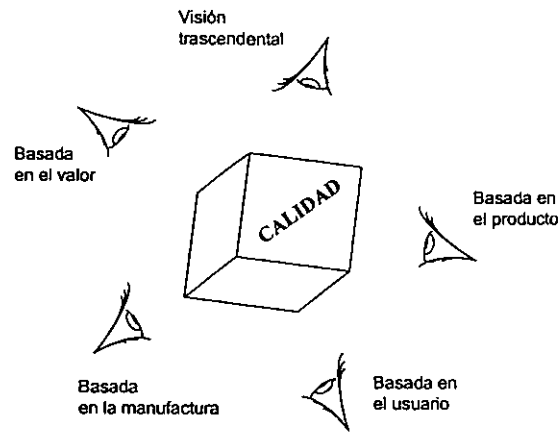


Figura 1-6 Cinco visiones de la calidad

b) ***Visión basada en el producto***. En términos económicos: a mayor calidad, mayor costo. Como la calidad de considera parte integral del software, y no una característica adicional, ésta se consigue de dos maneras principalmente:

1. Dándole al software mayor funcionalidad. Por ejemplo, integrando mecanismos de seguridad eficientes. Considerando también el proceso utilizado, se puede generar una solución de mayor calidad teniendo mayor cuidado en el desarrollo, por ejemplo, mediante la implementación de un sistema de administración de la calidad.

2. Intentando un balance en los costos. Los seguidores de Crosby se contraponen a este postulado argumentando que "la calidad es gratis", refiriéndose al hecho de que las mejoras en la calidad en cada etapa del proceso de manufactura conducen a menos defectos, a una mayor confiabilidad en el uso del producto, y por tanto a una reducción en el mantenimiento. La polémica aparece cuando los presupuestos de software apuntan que aproximadamente el equivalente al 80% de los costos de desarrollo se destinan al mantenimiento.

c) ***Visión basada en el usuario***. La idea de "cumplir su cometido", establecida por Juran ha sido, en las pasadas décadas, muy sacrificada en la ingeniería de software en favor de la precisión técnica que deben tener las soluciones automatizadas. Así, asegurarse de que la solución de software desarrollada resuelve el problema real, no parece tener relevancia, aunque en la práctica esta actividad no es nada simple.

Los aspectos de la calidad son frecuentemente difíciles de cuantiifcar satisfactoriamente. En el desarrollo de software, estos asuntos son desdeñados debido principalmente a la ambigüedad recurrente en la definición de los problemas.

Entender qué significa dar a los usuarios una solución como la que quieren se traduce en un eterno problema entre la gente involucrada en el desarrollo de sistemas. Argumentos del tipo "los usuarios no tienen claro lo que quieren", "no saben nada de calidad", "nunca será posible darles exactamente lo que piden", son el pan de cada día. Sin embargo, y aun siendo ciertas estas afirmaciones, es responsabilidad de los desarrolladores corregirlas.

El criterio actual de las organizaciones en este sentido ha cambiado radicalmente de la negación rotunda: "es imposible hacerlo", a indicar la factibilidad: "todo es posible desarrollar, pero en cierto tiempo y costo". No es simplemente la diferencia de lenguaje entre usuarios y técnicos lo que impide el acuerdo; tanto más elementos tenga el usuario para decidir sobre lo que le conviene, cuanto mejor será la solución. La información, indudablemente, debe salir del personal de sistemas.

d) ***Visión basada en la manufactura***. Esta es la visión más ampliamente aceptada por ingenieros de software y se fundamentan fuertemente en metodologías de desarrollo secuenciales, como el tradicional modelo de cascada.

Esta visión mide la calidad en términos de conformidad con los requerimientos. En relación al software, los requerimientos son las especificaciones

producidas durante el diseño, que implican los requerimientos en sí mismos y la comprensión de que son aceptables.

Actualmente esta visión está siendo promovida a otros tipos de metodologías, herramientas de ingeniaría asistida por computadora (CASE) y esquemas de calidad, introducidos por Crosby.

En la manufactura propiamente del software, el acercamiento a la calidad como *cero defectos*, ha crecido en popularidad. El transfondo es que los errores aparecerán, pero que se pueden evitar. Por supuesto un software libre de errores es muy recomendable, pero la calidad, en lo que se refiere al software, es mucho más que eso. La programación puede involucrar técnicas de calidad, pero invertir más en este punto sería incosteble y dejaría de lado etapas no menos importantes del desarrollo.

No basta hablar de *cero defectos* en software, especialmente hablando de la operación de sistemas de cómputo grandes. Las cifras suelen no tener credibilidad, en especial cuando, a pesar de un número reducido de fallas, no se note mejoría en la ejecución del software.

e) ***Visión basada en el valor.*** En relación con los proyectos de software, la restricción de los costos no se reduce sólo al aspecto financiero; el escenario se complica al incluirse recursos humanos, tiempo, recursos tecnológicos. Como se ha visto, otra restricción no menos importante son los costos asociados en la resolución del conflicto usuario-requerimientos técnicos.

1.1.7 El aseguramiento de la calidad del software.

Del inglés *Software Quality Assurance (SQA)*, es para muchos un nombre técnico para el proceso de pruebas. En el proceso del software, probar es lo que hacen los programadores para encontrar *bugs* (errores) en sus programas. Desafortunadamente existen algunos problemas con esta percepción clásica del SQA:

- *Asume que la gente prueba sus propios programas.* Mientras los programadores participan en la revisión de sus propios programas, incluso por

iniciativa propia, es bien sabido que un programador no espera encontrar sus propios errores en sus programas; su objetivo es demostrar que *funcionan*, no que *fallan*.

- *Asume que los bugs son una forma independiente de vida*. No existe un sólo tipo de *bug* para el programador común. Los clasifican en vivos, pasivos, pequeños o grandes, asesinos o inofensivos. Así cada uno es tratado en forma independiente y se omite la asociación de la aparición de un error con algo que el programador hizo.

  En contraste, las organizaciones japonesas de software normalmente nombran genéricamente a los *bugs* como *spoilage* (basura) y lo tratan como un todo. Saben que este *spoilage* tiene múltiples asociaciones que deben también resolverse, aunque no sean evidentes o no hayan todavía generado fallas. Sólo resolviendo todas estas asociaciones se resolverá el error global que se hizo evidente a partir de uno particular.

- *Asume que las pruebas se hacen una sola vez, al final de la etapa de desarrollo*. Sin embargo se sabe que en el desarrollo de software se debe procurar la prevención de errores y su detección oportuna en las fases previas a cada etapa.

- *Asume que las pruebas se hacen sobre el* ***producto****, no sobre el* ***proceso*** *que le dio origen*. La posición sostenida en este trabajo es que las mejoras en productividad y calidad provienen de las mejoras en los procesos de desarrollo de software tanto como del software mismo.

Una percepción más amplia del aseguramiento de la calidad del software implica la relación entre la calidad del producto y la del proceso, en un ambiente mostrado en la Figura 1-7.

Bajo este esquema, se adiciona una figura a las tradicionales del proceso de desarrollo: el área de SQA. Si bien su función es vigilar que los acuerdos planeados para el control de la calidad se cumplan, esto no implica una labor necesariamente de control. Su tarea es de carácter informativo.

Figura 1-7 Ámbito del aseguramiento de la calidad del software

Entre las tres figuras mostradas existe el vínculo de la información, necesaria para todo proceso de desarrollo; sin embargo, la función de control está en manos del administrador, que regula las actividades de los participantes.

Así, el departamento de SQA, como Yourdon (1992) menciona, se convierte en vigilante de que las "idades" del software se cumplan: reusabilidad, compatibilidad, confiabilidad, portabilidad, etc. La labor de un programa de SQA será útil en la medida en que:

- Minimice el número de defectos en el software que libere.
- Cree mecanismos para controlar el desarrollo y *mantenimiento*, optimizando la relación entre costos y tiempos.
- Tenga la certeza de que el producto de software sea colocado adecuadamente en el mercado y/o cubra las expectativas del usuario que lo originó.
- Induzca mejoras en la calidad de siguientes versiones del producto.

La calidad asociada en un producto de software se construye desde el principio de su proceso, con el talento de las personas que lo crean, de acuerdo con la tecnología utilizada y con la disciplina de una administración que tenga la visión general del proceso.

Un grupo de SQA es parte de éste último componente, la disciplina de administración. Un buen equipo de trabajo en aseguramiento de la calidad del software debe envolver los tres elementos: gente talentosa, buena tecnología y competente disciplina en la administración.

1.1.8 La calidad como parte de la cultura

Existe entre los teóricos de la calidad del software una marcada tendencia a enfatizar los procedimientos, las herramientas y los sistemas. Esto quizá por la relación que no ha podido disolverse entre los procesos de manufactura y la programación de sistemas.

Una posición personal es que la calidad del software es más un asunto de *gente*, que de *herramientas*. Ya se han explicado los participantes principales en el proceso de creación de software. La calidad en éste rubro tiene que ver con ellos porque:

- Son las personas, como individuos y no las organizaciones como ente, las que enfrentan problemas a ser resueltos por el software.
- Las personas definen los problemas y establecen las soluciones.
- La gente diseña e implementa el código fuente.
- La gente es además quien prueba el software.
- Finalmente, los seres humanos son quienes utilizan el software, y quienes por tanto, perciben la calidad de la solución que les ofrece.

Las herramientas, los procesos y la administración apegada a la calidad, son sólo ayudas en la consecución de la calidad; son a su vez, las respuestas que dan las organizaciones a los problemas que tienen. La complejidad del asunto es que la implementación de técnicas no garantizan *per se* un software de calidad, ya que como señala Voss:

"La conformidad con estándares de calidad de sistemas no es, en si misma, suficiente para garantizar la calidad de un producto o servicio de software. La calidad necesita crearse dentro de la organización y para lograrlo necesitamos una **cultura para la calidad**. *Esto no puede alcanzarse sólo mediante una decisión (...)"* [19]

El éxito de tareas resueltas por herramientas de cómputo (por ej. del tipo *CASE*), depende en gran medida del factor humano que las utiliza. Si bien con ellas se aprovecha la automatización, el resultado siempre depende del talento humano.

"Si el problema es que no existe alternativa cuando un buen proceso se está convirtiendo en algo difícil de administrar, (...) entonces las herramientas pueden ser una forma costosa pero efectiva de proceder. Pero si no se entiende el proceso de desarrollo, confiar en las herramientas es un desastre". [20]

Por ejemplo, si un líder de proyecto no cuenta con las bases de información para una correcta toma de decisiones, aun la mejor herramienta no podrá dar resultados. Aún peor, el focalizarse en el uso de las herramientas puede gastar tiempo y no estar atacando el problema real. [21]

Quienes se oponen al registro de los sistemas de calidad dicen que el esquema no puede garantizar la calidad de un producto final tal como el software.

Esto básicamente porque una parte importante del sistema de calidad es definir cómo se controlará la calidad para luego hacerlo. Si este control es pobre, probablemente se creará un producto con una calidad deficiente.

"Es un hecho que el simple cumplimiento de un estándar como ISO9001, no es suficiente. Es vital establecer una cultura de calidad, en la que todos reconozcan la importancia de la calidad, y la parte que juegan en su logro" [22].

El hecho de considerar importante al conjunto de factores humanos que influyen en toda actividad, sin incluir el desarrollo de software, deja abiertos algunos temas

---

19 Voss, A., *op. cit.*

20 *Ibid.* p. 45

21 Según un informe de la firma *Price waterhouse* (1988): el promedio de error por miles de líneas de código: era en EU entre 10 y 50, mientras que en Japón sólo 0.2. A raíz de esto, IBM ha reducido sus errores después de la instalación en 10%. Fuera de contexto estas cifras no dicen mucho, pero vale la pena enfatizar que las compañías se están dando cuenta del problema y que empiezan a atacarlo.

22 Yourdon, E. (1992) Decline and fall of the american programmer. EU: Prentice Hall, p. 50.

relacionados con la normatividad de las actividades dentro de un marco de calidad.

A reserva de una explicación mas detallada de los estándares de desarrollo y de calidad, baste saber que buena parte de los puntos a considerar en el fortalecimiento de un ambiente de calidad del software, provienen de la administración, del entendimiento del papel que cada integrante juega en el proceso o del nivel de involucramiento que se experimente en cada actividad.

Las siguientes son algunas reflexiones en torno a estos temas.

ACTITUDES DE LA ADMINISTRACION

Como en otras cosas, una guía debe provenir desde arriba. La gente es más perceptiva de las actitudes de administración de lo que creen los administradores. *"si la calidad no se toma seriamente desde la dirección, tampoco será tomada en cuenta por el resto de la organización"* [23]. Esta actitud debe comenzar desde arriba y bajar en toda la estructura. De hecho una parte importante de la norma ISO9001 es establecer claramente la política de calidad de la dirección y definir responsabilidades.

El cambio de cultura se refleja en niveles como:

- Documentos: deben ser adecuadamente administrados y autorizados.
- Planes: se deben preparar planes y monitorear su avance.
- Especificaciones: se deben poder probar para determinar cuando el trabajo realizado las ha cumplido.

TRAMOS DE RESPONSABILIDAD

Es esencial hacer a las personas responsables de la calidad de su trabajo y no que recaiga esta responsabilidad exclusivamente en supervisores de la calidad.

23 *Ibid.*

La responsabilidad del control de calidad debe ir a la par con la responsabilidad de hacer el trabajo.

En términos de software esto implica establecer dentro del plan del proyecto, criterios de control de calidad y responsabilizarse de que esos controles sean aplicados y los registros obtenidos sean conservados.

EL PAPEL DEL DEPARTAMENTO DE CALIDAD

Aunque tiene diferentes responsabilidades, éstas varían de acuerdo a necesidades locales:

- Es el guardián del sistema de calidad y administra los manuales de calidad.
- Provee asesoría a proyectos en materia de calidad, particularmente para elaborar planes de calidad.
- El departamento de calidad es la autoridad que aprueba los planes de calidad y los estándares.
- Maneja programas de mejora continua.
- Organiza programas de auditoria a los sistemas de calidad, verifica cumplimiento y reporta deficiencias.

El departamento de calidad se sintetiza en ésta última función. Por un lado debe ayudar al equipo del proyecto y por otro, debe avisar de problemas y deficiencias.

INVOLUCRAMIENTO

Una parte crucial del proceso de creación de una cultura para la calidad es involucrar al equipo de trabajo en asuntos de calidad, ejemplos de esto son:

a) ***Participación en la preparación de estándares***. Los estándares deben ser escritos por quienes los usarán. En este sentido, el departamento de calidad tiene más un carácter de consejero y aprobador. De esta forma se promueve el uso apropiado de estos estándares.

b) ***Práctica de retroalimentación.*** Siempre que se encuentre una falla en el sistema de calidad, el equipo de trabajo debe ser motivado para que lo corrija. Es responsabilidad de éste asegurarse de que el sistema es adecuado para él.

c) ***Participación en la revisión.*** El equipo de trabajo debe cooperar en la revisión del trabajo de la gente. Esto no sólo beneficia al trabajo que se revisa sino que mejora las habilidades del revisor y esto mejora la calidad en general.

d) ***Involucramiento en auditorías.*** Son principalmente auditorías al sistema de calidad, no al equipo de trabajo. Es importante generar un equipo de auditoria ajeno, tanto al proyecto que se audita como al departamento de calidad.

e) ***Rotación de personal.*** El departamento de calidad no debe ser visto como un equipo de trabajo aparte del proyecto, ni como un lugar donde el personal no deseado es adscrito. *"Es muy recomendable para algunos departamentos de calidad, ser formados con personal de proyectos con base en la rotación. Así puede ver la calidad en diversos proyectos y puede ver los errores que comete"* [24]. Una limitación es el tiempo disponible del personal asignado al desarrollo de proyectos. Es aconsejable siempre tener en mente que un equipo de la plantilla del personal se dedique a la calidad.

Es importante considerar que existen impedimentos para que madure una cultura de la calidad, ya que todo cambio fundamental presenta dificultades.

ACEPTAR LA NECESIDAD DE INVERSION

Establecer un sistema de calidad como la ISO9001 requiere de una inversión inicial. Puede no ser mucho pero si no se toma en serio el papel de la calidad, es un gasto muy grande. Se debe tomar conciencia de este gasto en que debe incurrir la empresa.

ACEPTAR LA NECESIDAD DE MEJORAR

Se necesita reconocer que todo lo que hacemos se puede mejorar. Se deben dejar de lado los argumentos de defensa y admitir que es más importante el

24 *Ibid.* p. 53

trabajo que nuestra estancia en un departamento, tener conciencia de que toda situación es mejorable y no acuñar la idea de que todo cuanto hacemos está bien.

LA NECESIDAD DE ESTAR ABIERTOS

Es muy usual que los programadores consideren su actividad como digna de secreto. En tal sentido ver sus programas es como descifrar los códigos inscritos en una pintura. Debemos sobreponernos al miedo de ser criticados y estar abiertos para aprender.

La actitud abierta no está exenta de problemas pero se debe pugnar por trabajar en un ambiente de confianza, donde los problemas puedan ser discutidos honestamente y donde exista el compromiso fiel del mejoramiento de la calidad.

FACTORES IRRACIONALES

Establecer un cambio cultural nunca es fácil ni mucho menos trivial. La comunicación humana es fuente de malos entendidos y de dificultades. A lo largo de nuestras vidas aprendemos de nuestro pasado, y de esto depende nuestra forma de actuar en el presente. Se deben construir patrones sólidos sobre los que se pueda construir nuestra actuación en la calidad.

BENEFICIOS

Un acercamiento inicial de la calidad puede tener beneficios como:

- Establecer claramente responsabilidades.
- Reconocer que el desarrollo es importante para el éxito del proyecto e identificar los pasos dentro del mismo que son determinantes.
- Se busca conseguir la calidad a título propio y no como resultado de la coerción.
- Aprendemos no sólo de la experiencia propia, sino de la experiencia de la gente involucrada.

- Ayudando a otros a mejorar su trabajo ayudamos a mejorar el nuestro.
- Se crea un ambiente de honestidad donde se pueden reconocer las fallas y los aciertos.
- La sabiduría de la compañía se documenta y comparte; no queda en manos de individuos.
- Se gana al saber que las cosas están bien y que fueron bien hechas.
- Se logra una ventaja comercial pues se evita el retrabajo, teniendo la certeza de que las cosas fueron correctamente hechas.

1.2 Modelos jerárquicos de la calidad

Una parte importante cuando se habla de la calidad del software, se refiere específicamente a los programas involucrados. Sin embargo, un software de calidad surge necesariamente de procesos de desarrollo de calidad.

El análisis de calidad no debe fundarse exclusivamente sobre la base del producto, los programas, sino debe incluir todo el entorno donde el desarrollo de software tiene lugar, estos es, procesos, funciones organizacionales, gente.

En los siguientes párrafos se mostrará un acercamiento al análisis de la calidad para el producto. Un análisis ulterior considerará también al entorno que lo crea.

1.2.1 Definición: ¿Qué es un modelo jerárquico?

Para comparar la calidad en diferentes situaciones, tanto cualitativa como cuantitativamente, se hace necesario establecer un *modelo de calidad*. Existe un número de estos modelos, muchos de ellos de naturaleza jerárquica.

Para explicar la aproximación jerárquica de los modelos, considérese el siguiente ejemplo. Para evaluar el desempeño de un alumno de primaria, básicamente se tiene una boleta con los siguientes datos:

Materia	Calificación	Comentarios del maestro
Español		
Matemáticas		
Ciencias Naturales		
Promedio		

En los recientes años el nivel de los estudiantes se ha vuelto más sofisticado y la evaluación se ha complicado. Las materias se han dividido en habilidades. Así por ej. para evaluar el desempeño en español, se examina la habilidad de lectura, de escritura, y gramática (Figura 1-8). En términos jerárquicos, el modelo requiere un nivel más para expresar los cambios:

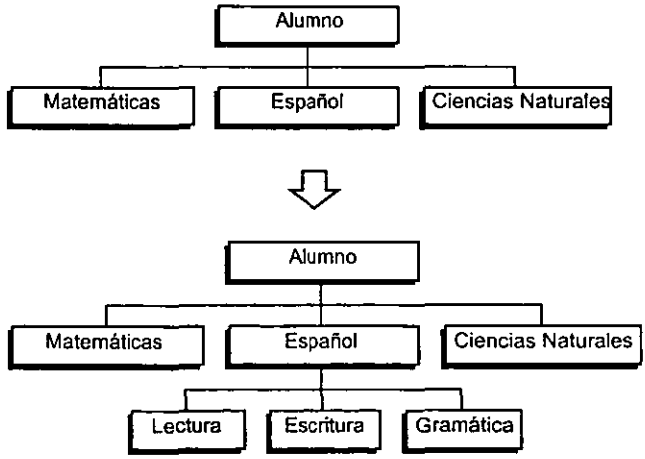


Figura 1-8 Evolución a un modelo jerárquico

Varios de los puntos expuestos tienen un paralelo al calificar la calidad del software. Muchos autores consideran un modelo similar para el software donde la

aproximación jerárquica se expone como una serie de niveles. Estos son conocidos genéricamente como *modelos jerárquicos de calidad*.

Estos modelos datan de los años 70, cuando la PC aún no existía y los departamentos de proceso de datos eran lo más actual. Dos en particular, Boehm (1978) y Mc Call (1977) son utilizados en nuestros días.

Un modelo jerárquico de calidad del software se basa en una serie de criterios de calidad, cada uno asociado con un sistema métrico. La Figura 1-9 ejemplifica este tipo de modelos.

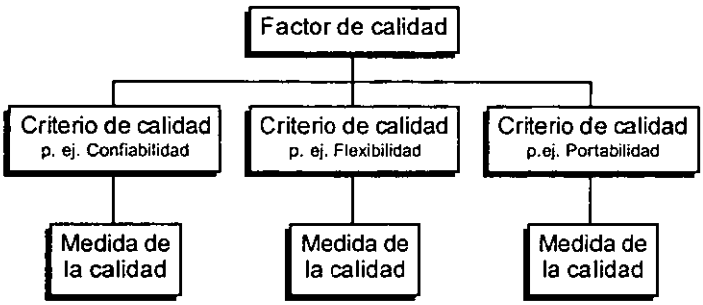


Figura 1-9 Un modelo jerárquico para la calidad del software

Los criterios de la calidad del software más frecuentemente utilizados incluyen confiabilidad, seguridad y adaptabilidad. Algunas preguntas surgen de inmediato con respecto a dichos criterios:

- ¿Qué criterios de calidad se deben utilizar?
- ¿Cómo se interrelacionan?
- ¿Cómo se deben combinar los sistemas de medición de cada uno para obtener una medida de calidad que tenga sentido?

Como un ejemplo del conjunto de medidas asociado con cada característica, la Figura 1-10 muestra que las medidas asociadas con la confiabilidad pueden incluir precisión, consistencia, tolerancia de errores y simplicidad.

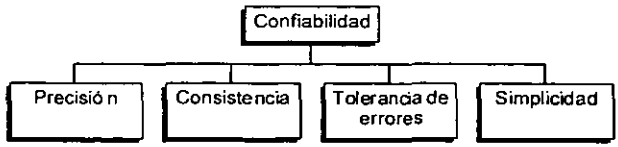


Figura 1-10 Medidas asociadas con la confiabilidad

Este es un punto crucial de la calidad del software que se complica porque en la vida real las cosas no son tan sencillas. En un capítulo posterior se verá que una misma medida se puede relacionar con más de un criterio de calidad, lo que complica alcanzar una visión general. Así, muchos autores han relacionando la complejidad del código fuente inversamente con la confiabilidad . Al mismo tiempo la complejidad se relaciona inversamente con otros criterios de calidad como la exactitud y la facilidad de mantenimiento.

1.2.2 Los modelos jerárquicos de Boehm y McCall

El modelo jerárquico inicialmente propuesto por McCall, ha sido retomado por varios autores hasta la actualidad. El modelo está dirigido a los desarrolladores de sistemas para ser utilizado durante la etapa de desarrollo.

En esta parte se mostrará los atributos de calidad que hacen a la calidad del software más operacional. En un intento inicial para hacer menor la brecha entre usuarios y desarrolladores, los criterios se escogieron de manera que reflejaran la visión de los usuarios tanto como las prioridades de los desarrolladores.

El modelo parte de la identificación de tres áreas de trabajo en los proyectos de software, resumidas en la Tabla 1-3.

Área	Descripción
Operación del producto	La operación requiere que el producto pueda aprenderse fácilmente, que se lleve a cabo eficientemente y que los resultados sean los requeridos por el usuario.
Revisión del producto	Esta área se relaciona con la corrección de errores y con la puesta a punto del sistema. Es importante porque se considera generalmente la parte más costosa del desarrollo de software.
Transición del producto	Este aspecto puede no ser relevante en todas las aplicaciones. Sin embargo, constantemente el software cambia de entorno; las tendencias a moverse a procesos distribuidos y la rápida evolución del hardware son factores que incrementan su importancia.

Tabla 1-3 Las tres áreas de software señaladas por McCall

La operación, la revisión y la transición del producto corresponden al ciclo de vida del sistema. Los atributos de la calidad del software se relacionan con cada una de estas áreas.

La Figura 1-11 resume el concepto de calidad de software considerando las tres áreas de McCall.

Figura 1-11 El triángulo de la calidad del software

La operación de un software determina si cumple con las necesidades del usuario. La función de mantenimiento empieza después de su liberación. En este punto, la revisión de los atributos de calidad se vuelve importante, especialmente la flexibilidad y la facilidad de mantenimiento. La transición de los componentes de un producto a otro nuevo cobra significado cuando el objetivo es la reutilización.

Una tipificación de cada atributo se muestra en la Tabla 1-4.

Atributo	Tipificación	Definición
Exactitud	¿Hace lo que yo quiero?	Límite en el que un programa satisface las especificaciones y cumple los objetivos del usuario.
Confiabilidad	Lo que hace ¿lo hace correctamente todo el tiempo?	Es la facilidad para no fallar. Representa la precisión con la que se espera que corra un programa.
Eficiencia	¿Correrá en mi computadora tan bien como sea posible?	Representa los recursos de cómputo y código necesarios para que un programa se ejecute. Tiene que ver con el uso de recursos como tiempo de procesador. Se divide en dos categorías: eficiencia de ejecución y eficiencia de almacenamiento.
Integridad	¿Es seguro?	Es la protección del programa contra accesos no autorizados.
Facilidad de uso	¿Lo puedo correr?	Esfuerzo requerido para aprender, operar, preparar las entradas e interpretar las salidas del programa. Representa la facilidad con la que se usa un software.
Facilidad de mantenimiento	¿Lo puedo corregir?	Esfuerzo requerido para localizar y reparar un error dentro de un programa que se utiliza cotidianamente.
Facilidad para ser probado	¿Lo puedo probar?	Es la facilidad para probar un programa, asegurarse de que no tiene errores y que cumple las especificaciones.

Atributo	Tipificación	Definición
Flexibilidad	¿Lo puedo cambiar?	Esfuerzo requerido para modificar un programa a partir de cambios ocurridos en su ambiente de ejecución.
Portabilidad	¿Lo podré utilizar en otra máquina?	Esfuerzo requerido para transferir un programa de un ambiente de hardware a otro.
Reusabilidad	¿Podré reutilizar partes del software?	Es la facilidad con que un programa puede ser utilizado en otras aplicaciones.
Interoperatividad	¿Podré tener interfaces con otros sistemas?	Esfuerzo requerido para acoplar un sistema con otro.

Tabla 1-4 Definiciones de los atributos de la calidad de software.

Los factores mostrados corresponden muy cercanamente a los atributos descritos por Boehm (1977), cuyo modelo se desarrolló para establecer características de la calidad del software bien definidas y bien diferenciadas.

En este modelo jerárquico las jerarquías se dan por la subdivisión de los criterios de calidad. La primera división se hacer de acuerdo a los usos dados al sistema.

Éstos pueden ser de utilidad "*general*" o "*como está*". La utilidad "*como está*" es un subtipo de la general, muy cercano al criterio de operación del producto de McCall. Existen otros dos niveles. Del intermedio se derivan las características básicas que son susceptibles de medición. El modelo se presenta en la Figura 1-12.

Este modelo se basa en más criterios en comparación con el modelo McCall (Tabla 1-5).

Ambos modelos comparten algunas características típicas de su naturaleza jerárquica, y también del momento histórico en que se originaron:

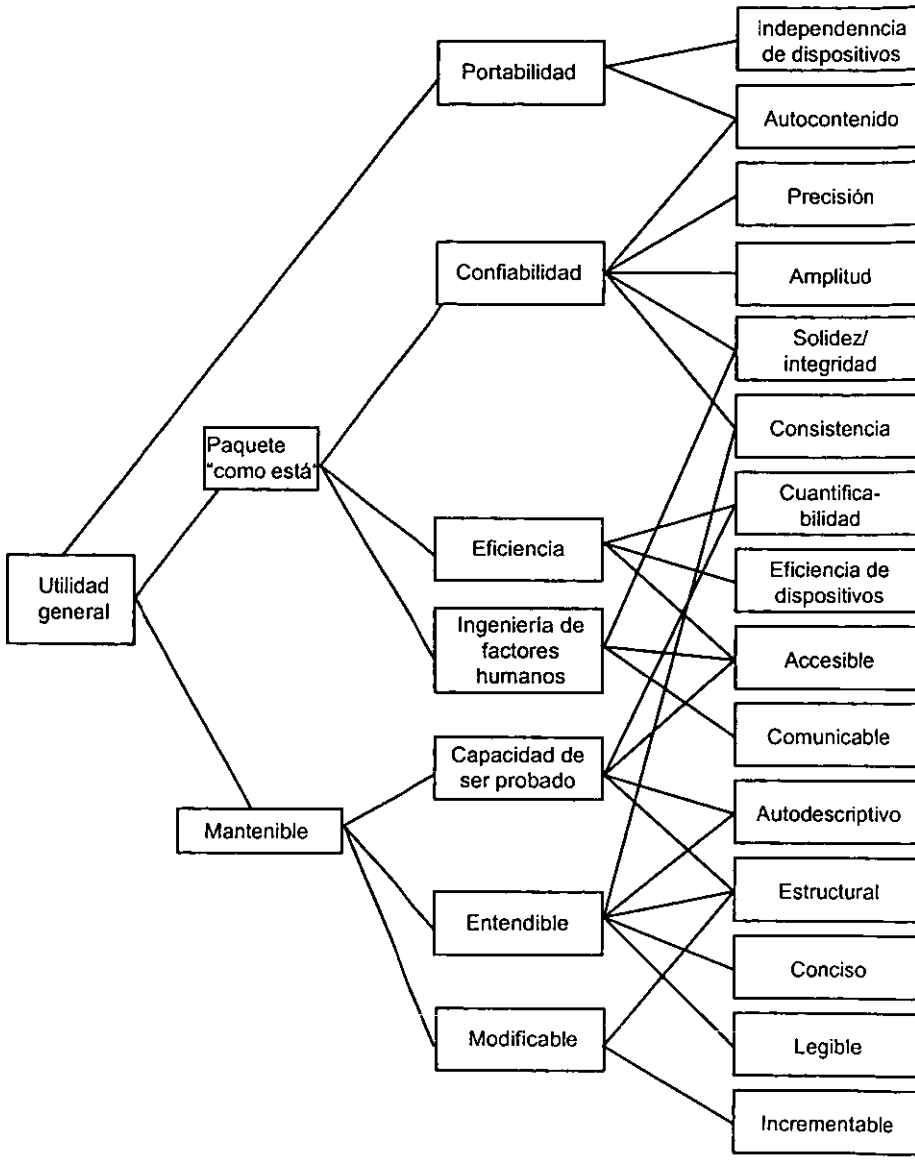


Figura 1-12 El modelo de calidad de Boehm.

Facilidad de uso	Documentación	Legibilidad
Claridad	Elasticidad	Integridad
Eficiencia	Exactitud	Validez
Interoperatividad	Portabilidad	Generalidad
Reusabilidad	Facilidad para modificar	Economía
Modularidad	Facilidad de mantenimiento	

Tabla 1-5 Criterios de calidad de Boehm

- Los criterios de calidad se basan supuestamente en el punto de vista del usuario. En realidad representan parcialmente la visión de los desarrolladores [25].
- Los modelos se focalizan en donde los desarrolladores pueden analizar rápidamente. Así, los criterios presentados están conectados con algún aspecto técnico de la calidad del software. Más aún, las mediciones de tales criterios no son homogéneas; mientras para algunas existen varias medidas válidas, para otras no hay una medida que las evalúe.
- La medida de la calidad en general se obtiene con la ponderación de todas las características. El resultado puede sin embargo, ser limitado.
- Los modelos jerárquicos no pueden ser validados. No pueden demostrar que su sistema métrico refleja fielmente cada criterio.

Ningún modelo es exclusivo. Cada aplicación determinará la importancia de cada una de las características. Los dos puntos de vista expuestos son compatibles en muchas de las definiciones de criterios de calidad; en términos generales son más parecidos que diferentes. La disparidad entre ambos es un tema sumamente discutido actualmente. De ese debate se puede sacar una conclusión: cualquiera que sea el modelo utilizado, así como la asignación y ponderación de cada uno de los criterios, lo importante es contar con el modelo, que es el marco en el que la calidad del producto opera.

25 Incluso algunos son propios del desarrollador, y se definen en sentido inverso. Así por ej. la facilidad de mantenimiento realmente implica la ausencia de esfuerzo para encontrar errores en los programas.

1.2.3 Interrelación de los criterios de calidad

Uno de los problemas que debe encarar una buena estrategia de calidad es el conflicto entre los atributos de la calidad. Los modelos jerárquicos comprenden criterios de calidad y el sistema métrico para la calidad. Las medidas tomadas sobre cada criterio no proveen por separado una visión general de la calidad. Para obtener una medida global, el conjunto de mediciones debe combinarse.

Esta combinación complica el esquema porque los criterios de calidad suelen causar conflictos unos con otros; para resolver ese conflicto debe llegarse a acuerdos que permitan un balance entre los factores, y que tiendan a una solución ideal. La Figura 1-13 muestra la relación existente entre los distintos factores.

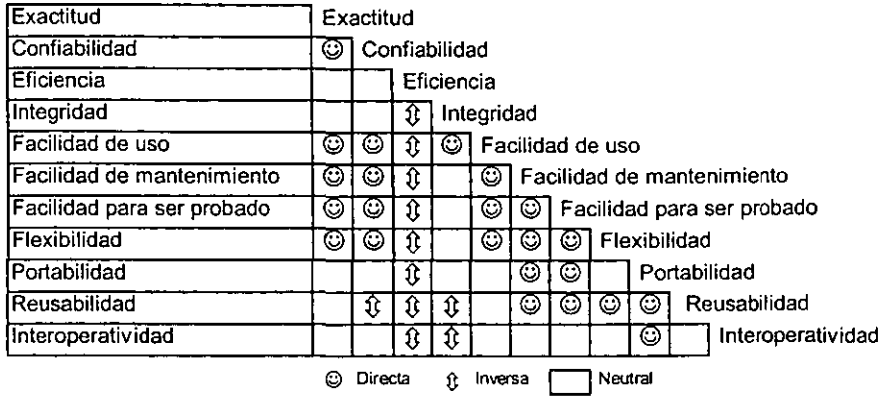


Figura 1-13 Interrelación entre los criterios de calidad (esquema de Perry)

Esta relación de criterios considerada por Perry, y de la cual existen también otras aproximaciones, desentraña algunas relaciones importantes:

Integridad vs. Eficiencia (inversa).

Todo control de acceso a programas o paquetes de software, así como programar el esquema de seguridad con el que debe operar, requiere código

adicional y por tanto tiempo de proceso extra; esto puede traducirse en un *performance*, o nivel de ejecución más bajo.

Facilidad de uso vs. Eficiencia (inversa).

Las mejoras en la interfaz computadora/humano se traducen en más necesidades de código y de recursos[26].

Facilidad de mantenimiento y pruebas vs. Eficiencia (inversa).

El código fuente compacto, conciso y optimizado es difícil de mantener. Inevitablemente el código bien estructurado, con comentarios suficientes y en forma modular es menos eficiente. Al mismo tiempo el código bien estructurado es más fácil de probar.

Facilidad de mantenimiento vs. Flexibilidad (directa).

Los programas que permiten actividades de mantenimiento con facilidad en general están bien estructurados. Esto apoya cualquier alteración requerida en el software; por tanto existe una relación complementaria entre ambos factores.

Facilidad de mantenimiento vs. Reusabilidad (directa)

De igual manera, mientras más estructurados y fáciles de mantener sean los programas, su código, todo o en parte, puede ser reutilizado dentro de una biblioteca de subrutinas, o dentro del código de otro programa.

Portabilidad vs. Reusabilidad (directa).

El software que pueda cambiar con facilidad de plataforma tecnológica, depende poco de especificaciones particulares de cada ambiente. En general tienden a ser bien estructurados. Ambas características apoyan fuertemente el reuso de los programas.

Precisión vs. Eficiencia (neutral).

26 En algunos casos, por ej. sistemas expertos aplicados a la medicina, incluso la interfaz humana es más grande que el conjunto de programas en sí.

La precisión del código, es decir, el cumplimiento de las especificaciones no tiene relación directa con la eficiencia. Un código correcto puede o no, ser eficiente en el momento de su operación.

Facilidad de mantenimiento vs. Integridad (neutral).

Los programas que sean bien estructurados y por tanto de mantenimiento sencillo, no necesariamente garantizan un buen esquema de seguridad. Incluso un sistema bien estructurado puede no tener un esquema de integridad que lo respalde.

El esquema de Perry es útil en la medida que muestra en un panorama general los atributos que conforman la calidad del software como calidad del producto. Sin embargo puede tener algunas debilidades que es necesario aclarar. Una es el establecimiento de que las relaciones son conmutativas, cosa que en la práctica no siempre ocurre. Otro es que las características pueden no ser transparentes en todas las aplicaciones. Según la aplicación en particular (con una fuerte dependencia), las relaciones entre las características pueden variar.

El establecimiento de las características de la calidad y sus relaciones, redundará en un modelo de calidad de software aplicable dentro de un entorno en particular. El modelo debe funcionar sobre las bases que se definan como útiles en cada caso específico.

2

ELEMENTOS DE LA CALIDAD DEL SOFTWARE

"Lo que no es medible, hazlo medible"

Galileo

2.1 Medición de la calidad de software

Si bien la calidad es el producto del esfuerzo de un conjunto de personas que elaboran y mantienen los sistemas de software, es imperativo que ésta sea medida primero, para revisar que las expectativas de los usuarios se hayan cubierto y segundo, para entrar en un proceso continuo de mejora de la calidad tanto en los productos como en los procesos de desarrollo de software.

En esta sección se discutirá la problemática asociada con la medición del software y de su calidad, desde la necesidad de la medición hasta la importancia en la definición de los criterios del software a medir para conformar un buen sistema de medición.

2.1.1 Medición de la calidad

En recientes años ha quedado claro que un problema que preocupa en el campo del desarrollo de software es la medición de la calidad. En el campo de las ciencias exactas la medición de las propiedades asociadas a un evento es precisa y no cae en ambigüedades. Considerando que buena parte de los profesionales en el desarrollo de software provienen de esta formación académica, las características del software se deben hacer medibles de cierta manera; en esta práctica se debe garantizar que no se distorsione la situación, y que la figura derivada refleje fielmente la propiedad que está bajo escrutinio.

Cuando se estima el nivel de calidad se hace en términos de *métricas*. Una *métrica del software* es una propiedad medible que es indicador de uno o más de los criterios de calidad que se busca medir.

Así, una métrica de la calidad debe cumplir algunas condiciones:

- Estar claramente ligada al criterio de calidad que intenta medir.
- Ser sensible a los diferentes grados que presente el criterio.
- Proveer un juicio objetivo del criterio que pueda ser traducido a una escala adecuada.

Por lo tanto, una métrica no es lo mismo que una medición directa. Considérese como ejemplo el martillo usado en las ferias para medir la fuerza (Figura 2-1). Este aparato no mide la fuerza en un sentido directo, absoluto y verificable pero si indica la fuerza en términos de la altura alcanzada en la columna. En un intento, la altura obtenida será proporcional a la fuerza utilizada en el golpe, pero se afectará por otros factores, como la técnica de golpeo o la fuerza del viento. La altura lograda por un golpe es una *métrica*, es decir, es una propiedad medible relacionada con el criterio en análisis.

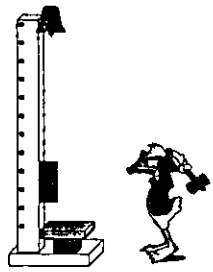


Figura 2-1 La métrica como indicador cuantitativo

En contraste, considérese el uso del termómetro basado en el empleo de la expansión lineal del mercurio para medir la temperatura. La relación entre expansión y temperatura es lineal, directa y verificable bajo experimentación. La expansión del mercurio depende sólo de la temperatura y se establece una escala absoluta considerando los puntos de congelación y ebullición del agua como extremos.

Refiriéndose a la calidad, Gillies establece la diferencia:

> *"Los criterios de calidad nunca dependen de una sola propiedad en este sentido* [hablando de un termómetro] *y no existen puntos de referencia para establecer una escala absoluta (...) la calidad del software no puede ser medida en términos de escalas simples, absolutas y libres de ambigüedad"* [27]

Las técnicas de medición aplicadas al software tienden a ser semejantes a las utilizadas en las ciencias sociales, donde las propiedades son igualmente complejas y ambiguas.

2.1.2 Métrica del software

En la mayoría de los desafíos técnicos, la medición y las métricas ayudan a entender tanto el proceso técnico que se utiliza para desarrollar un producto, como el producto mismo. El proceso se mide para intentar mejorarlo. El producto se mide para intentar aumentar su calidad.

---

[27] Gillies, A. *Op. Cit.* p. 35

En principio parece que la necesidad de medir es evidente. Después de todo, es lo que nos permite cuantificar, y por consiguiente, tener un control adecuado. Pero la realidad es diferente. Frecuentemente la medición conlleva a una gran controversia y discusión. ¿Cuáles son las métricas apropiadas para el proceso y para el producto? ¿Cómo se utilizan los datos que se recopilan?. Preguntas como estas surgen cuando se intenta mediar algo que no había sido medido en el pasado. Éste es el caso de la Ingeniaría del software (el proceso) y del software (el producto).

Una *métrica del software* es cualquier medida relacionada con un sistema de software, proceso o documentación relacionada. Algunos ejemplos son el número de líneas por programa o el número de fallas reportadas en un programa que ha sido liberado en producción.

Algunos autores (Sommerville, 1992; Gillies, 1992) dividen a las métricas en dos clases genéricas, de acuerdo a su función, que se resumen en el mismo esquema: pueden ser métricas *predictivas* o *de control* [28] (Figura 2-2).

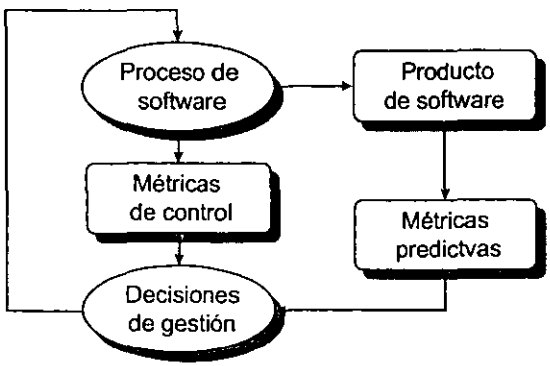


Figura 2-2 Métricas predictivas y de control

[28] La clasificación es muy parecida a las métricas físicas, divididas en *medidas directas* (la longitud de un tornillo) y *medidas indirectas* (la calidad del tornillo, medida como la cantidad de tornillos rechazados durante el proceso de producción).

a) *Las métricas de control* (o descriptivas) son aquéllas usadas en la administración para controlar el proceso del software ya que describen su estado en el momento de la medición. Ejemplos de ellas pueden ser el tiempo de proceso o el espacio de almacenamiento utilizado.

   Las estimaciones y medidas de estas métricas pueden ser usadas para refinar el proceso de planeación del proceso.

b) *Las métricas predictivas* son medidas de un atributo del producto que pueden ser usadas para predecir un producto de calidad.

2.1.3 ¿Porqué medir?

La medición es algo común en el mundo de la ingeniería. Se miden pesos, dimensiones físicas, voltajes, etc. Sin embargo, la medición no es algo tan común en el campo de la ingeniería del software. Se encuentran dificultades sobre qué medir y cómo evaluar las medidas.

De acuerdo a Pressman (1993), existen algunas razones para medir el software:

- Para indicar la calidad del producto.
- Para evaluar la productividad de la gente que desarrolla el producto.
- Para evaluar los beneficios (en términos de calidad y productividad) derivados del uso de los nuevos métodos y herramientas de la ingeniería del software.
- Para establecer una línea de base para la estimación.
- Para ayudar en la justificación del uso de nuevas herramientas o de formación adicionales.

Una clasificación general de las métricas del software se puede ver en la Figura 2-3. Las métricas de productividad se centran en el rendimiento del proceso de la ingeniería del software; las métricas de calidad indican cómo se ajusta el software a los requerimientos implícitos y explícitos del cliente, y las métricas técnicas se

centran en las características del software (por ej. la complejidad lógica o el grado de modularidad), más que en el proceso que se siguió para su desarrollo.

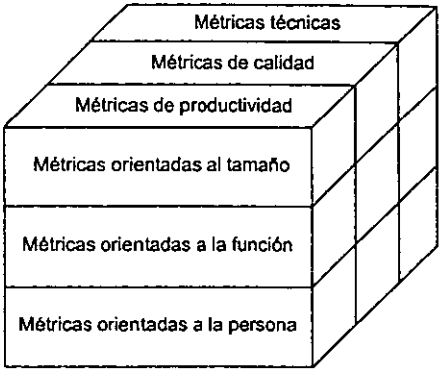


Figura 2-3 Clasificación de las métricas del software.

En una segunda dimensión se puede establecer otra clasificación: las métricas orientadas el tamaño se utilizan para obtener medidas directas del resultado y la calidad de la ingeniería del software. Las métricas orientadas a la función proporcionan medidas indirectas, mientras que las métricas orientadas a la persona proporcionan información sobre la forma en que la gente desarrolla software y sobre el punto de vista humano de la efectividad de las herramientas y métodos.

Juntas e interrelacionadas estas métricas pueden reflejar algunas características de los productos o procesos dentro de la ingeniería de software.

2.1.4 ¿Qué medir?

Toda organización actual que aspire a destacar en el desarrollo de software debe tener una firme iniciativa en el rubro de la métrica y debe decidir *qué medir.*

En principio parece atractivo medir todo; sin embargo, no es recomendable medir todo lo que parezca interesante. Hacerlo podría ser demasiado caro, podría

tomar tiempo en exceso e invariablemente inundaría a la empresa con una avalancha de números.

En el extremo contrario, no se puede fincar una base sólida de decisión sobre un par de "números mágicos" que intenten reflejar la situación. [29]

La métrica que se decida deberá representar los intereses de la empresa, y no conformarse de números que resulten vacíos. Así, por ejemplo, las mediciones que se relacionan con los costos corren el riesgo de reflejar sólo una parte de la situación; la productividad puede ser mejorada con arreglos en la contabilidad de los costos en vez de mejorías en las causas reales.

Así, es frecuente el establecimiento de métricas para el desarrollo, pero no para el mantenimiento de los sistemas. Como resultado, los desarrolladores pueden hacer pasar su trabajo bajo el rubro de mantenimiento, conservando —aparentemente— los niveles esperados de productividad.

Un esquema completo de medición, debe considerar métricas en los campos de:

- ***Producto.*** Se refiere al código de los programas: número de líneas de código, número de errores en las mismas, etc.
- ***Proceso.*** Ejemplos de métricas pueden ser el porcentaje de saturación de CPU al momento de ocurrir una falla o el tiempo de desarrollo en relación al número de errores.

Ambas son necesarias para reflejar una situación real en el entorno del software. Más aún: los ingenieros de software necesitan información sobre su proceso para aprender cómo cambiarlo, cómo mejorarlo.

En resumen, las necesidades mismas de la empresas dictarán qué métricas utilizar. Yourdon (1992), sugiere:

---

29 Según Yourdon, E. *op. cit.*, quienes se encargan de la métrica en las empresas, generalmente toman sólo algunas mediciones para comprobar, después de algún tiempo, que la Dirección no hace nada con las

> *"Para las* [organizaciones] *principiantes, esto* [la importancia de las métricas] *significa que deberán enfocarse en medir lo básico: tamaño del software, defectos del software y esfuerzo utilizado en la creación del mismo. Métricas adicionales —por ejemplo, métricas de respuesta de los usuarios— pueden ser usadas posteriormente, si se relacionan con los objetivos del negocio que sean importantes para directivos y usuarios."* [30]

En los siguientes apartados se expondrán brevemente algunas características de las principales subdivisiones de la métricas expuestas.

2.1.4.1 Métricas orientadas al tamaño.

Las métricas de este tipo son medidas directas del software y del proceso empleado en su desarrollo. Si una organización lleva un sencillo registro, puede tener una tabla orientada al tamaño como la mostrada en la Tabla 2-1.

Proyecto	Esfuerzo (persona-mes)	Gasto (miles de dólares)	Líneas de código (miles)	Pág.Doc.	Errores	Personas
ALA-1998	23	90	12.1	365	29	3
BKT-1999	62	234	27.2	1124	86	5
COL-1999	43	162	20.2	1050	64	6

Tabla 2-1 Métricas orientadas al tamaño

Con los sencillos datos mostrados en la tabla se puede desarrollar, para cada proyecto, un grupo de métricas sencillas de productividad y de calidad.

Productividad = Líneas de código (LDC) / esfuerzo

Calidad = Errores / LDC

Costo = Gasto (dólares) / LDC

medidas que meticulosamente han tomado. Destaca también que la tasa de mortalidad en las iniciativas de métrica es alta: el 80% de las empresas deja de tomar mediciones en un periodo aprox. de 18 meses.

30 *Ibid.* p. 183

Documentación = Páginas / LDC

Las métricas orientadas al tamaño son bastante polémicas y no son aceptadas universalmente como mejor modo de medir el proceso de desarrollo de software. Este tipo de métricas usan el número de líneas de código (LDC) como medida clave; sin embargo, a pesar de su fácil obtención, no es del todo aceptada su validez pues su significado puede variar al considerar factores como el lenguaje utilizado o el grado de modularidad en un programa.

2.1.4.2 Métricas orientadas a la función

Este tipo de métricas son medidas indirectas del software y del proceso por el cual se desarrolla. En lugar de calcular las LDC, las métricas orientadas a la función se centran en la funcionalidad o utilidad del programa.

Ejemplo de estas métricas son los *puntos de función* (PFs), obtenidos mediante una relación empírica entre las medidas cuantitativas del software y las valoraciones subjetivas de la complejidad del mismo.

Los PFs se calculan llenando la tabla que se presenta en la Figura 2-4. Se determinan cinco características de la información y se muestran los cálculos en la tabla. Los valores se definen de la siguiente manera:

- *Número de entradas de usuario.* Se cuenta cada campo que el usuario introduce en la aplicación.
- *Número de salidas de usuario.* Se cuentan todas las salidas (datos) que la aplicación proporciona al usuario. Las salidas se refieren a pantallas, reportes, mensajes de error, etc.
- *Número de peticiones de usuario.* Una petición es un dato ingresado al sistema, como producto de alguna acción interactiva entre el usuario y el software. Es una combinación de una entrada y una salida de usuario.
- *Número de archivos.* Es la cantidad de archivos maestros en la aplicación.

Parámetros de medida	Cuenta	Factor de peso simple	medio	complejo		Total
No. de entradas de usuario		x 3	4	6	=	
No. de salidas de usuario		x 4	5	7	=	
No. de peticiones de usuario		x 3	4	6	=	
No. de archivos		x 7	10	15	=	
No. de interfaces externas		x 5	7	10	=	
Cuenta total →						

Figura 2-4 Cálculo de los Puntos de Función

- *Número de interfaces externas.* Son todas las entidades de información enviadas para proceso a otros sistemas.

Una vez obtenidos los datos requeridos, se les asocia un valor de complejidad, que aunque obtenido con algún método, resulta ser una ponderación subjetiva.

La siguiente relación se utiliza para el cálculo de los puntos de función:

$Pfs = \text{cuenta total} \times [0.65 + 0.01 \times \Sigma f_i]$

donde,

cuenta total es la suma de todas las entradas en la tabla de la Figura 2-4.

f_i $(i = 1,14)$ son los valores de ajuste de complejidad basados en las respuestas a las preguntas de la Tabla 2-2.

Tanto los valores constantes de la ecuación como los factores de peso aplicados a los parámetros de medidas han sido determinados empíricamente.

Una vez que se calculan los PFs, se usan de forma análoga a las LDC como medida de los atributos del software:

Productividad = PF / personas-mes

Calidad = errores / PF

Costo = Gasto (dólares) / PF

Documentación = páginas de documentación / PF

	Pregunta	0	1	2	3	4	5	6
1	¿Requiere el sistema copias de seguridad y recuperación fiables?							
2	¿Se requieren comunicaciones de datos?							
3	¿Existen funciones de proceso distribuido?							
4	¿Es crítico el rendimiento?							
5	¿El sistema se ejecutará en un entorno operativo existente y utilizado?							
6	¿Requiere el sistema entrada de datos interactiva?							
7	¿Se requieren muchas pantallas en línea conectadas al sistema?							
8	¿Se actualizan los archivos maestros en línea?							
9	¿Son complejas las entradas, las salidas, los archivos o las peticiones?							
10	¿Es complejo el procesamiento interno?							
11	¿Se ha diseñado el código para ser reutilizado?							
12	¿Están incluidas en el diseño la conversión y la instalación?							
13	¿Se diseñó el sistema para poder ser instalado en varias organizaciones?							
14	¿Se diseñó el sistema para facilitar los cambios y para ser fácil de usar?							

1. Sin influencia	3. Moderado	5. Significativo
2. Incidental	4. Medio	6. Esencial

Tabla 2-2 Cálculo de Puntos de Función

Originalmente, la medida de puntos de función se utilizó en aplicaciones de sistemas de información gerenciales. Luego de adecuaciones, se llegó al esquema de *puntos de característica*, que permite que esta medida se utilice en aplicaciones de software.

La medida del punto de característica engloba a aplicaciones de complejidad algorítmica alta; los sistemas en tiempo real, de control de procesos y de sistemas operativos son ejemplos de desarrollos de complejidad algorítmica alta que pueden ser analizados con esta métrica.

Para calcular los puntos de característica se cuentan y ponderan algunos factores, además de incluir a otro componente del software: los algoritmos[31]. La Figura 2-5 muestra una tabla utilizada para calcular los puntos de característica. En ella se relaciona sólo un valor de peso en los parámetros de medida y se calcula el valor del punto de característica global usando la misma ecuación empleada para los puntos de función.

Parámetros de medida	Cuenta	Peso	Total
No. de entradas de usuario		x 4 =	
No. de salidas de usuario		x 5 =	
No. de peticiones de usuario		x 4 =	
No. de archivos		x 7 =	
No. de interfaces externas		x 7 =	
Algoritmos		x 3 =	
Cuenta total →			

Figura 2-5 Cálculo de métricas del punto de característica

La métrica de los puntos de característica y los puntos de función representa lo mismo: funcionalidad o utilidad del software al igual que la métrica LDC. Sus defensores sostienen que los PFs se calculan independientemente del lenguaje de programación empleado y que se basan en datos más probables de ser conocidos durante el desarrollo de un proyecto. Quienes se oponen lo hacen sosteniendo que las medidas se basan más en factores subjetivos que en la objetividad de los datos.

31 Los algoritmos pueden ser definidos como "un problema de complejidad computacional limitada que se incluye dentro de un programa de computadora" (Pressman, *op. cit.* 1993). Ejemplos de algoritmos son la inversión de una matriz o la decodificación de una cadena de bits.

2.1.4.3 Métricas para la calidad del software

Dos momentos en el tiempo son importantes considerar al hablar de las métricas en este rubro: las métricas de calidad durante el proceso de desarrollo de software y las métricas usadas una vez que el producto se ha liberado a los usuarios.

Las métricas usadas durante el desarrollo incluyen la *complejidad del programa*, *la modularidad* y el *tamaño del programa*. Una vez que el software se distribuye, las métricas se orientan a la medición de los defectos no descubiertos en las pruebas, y en la *facilidad de mantenimiento* del sistema.

2.1.5 Problemática de la métrica

Establecer un modelo jerárquico que incluya diferentes métricas para reflejar la calidad del software puede verse como una opción atractiva; sin embargo, las métricas más comúnmente descritas en la literatura limitan su efectividad por las siguientes razones:

- ***No pueden ser validadas.*** La relación de las métricas con lo que intentan medir es compleja. Resulta difícil establecer el grado de correlación entre la métrica, la medida sobre la que se basa y el criterio que se desea medir. Como resultado, es aún más difícil validar esta correlación.
- ***Generalmente no son objetivas.*** En un mundo ideal las medidas deberían ser objetivas, absolutas y transferibles. En la práctica las medidas son relativas, a menos de que se establezca un punto fijo que defina el principio de una escala (como el punto de ebullición y congelación del agua.) *En la calidad no hay puntos fijos.* Aunque haya criterios aparentemente certeros, es difícil mostrar que las correlación entre tales medidas y el criterio de calidad, es absoluta.
- ***La calidad es una entidad relativa, no absoluta.*** En la práctica, muchos de los aspectos de la calidad pueden ser juzgados sólo en términos relativos. Sin haber puntos fijos que delimiten escalas precisas, las medidas se aceptan en

términos de la experiencia. De la misma forma, sólo hasta que se defina el marco de la calidad (los criterios de la que está formada), se podrá establecer un conjunto de métricas que encaje en ese marco.

- ***Dependen de un pequeño número de propiedades medibles.*** Aunque existe un gran número de métricas, muchas de ellas parecen ser las variaciones de un pequeño grupo.
- ***No miden el espectro completo de los criterios de calidad.*** En el conjunto de criterios citados por McCall expuestos más adelante, existen varios que no tienen una métrica asociada, por ej. la eficiencia.
- ***Una misma métrica puede medir más de un criterio.*** La consecuencia de las relaciones tan complejas establecidas entre los criterios es que una misma métrica se utilice para medir varios criterios. Así, el número ciclomático relacionado con la complejidad de un programa, se usa para medir la confiabilidad y la facilidad de mantenimiento.

2.1.6 ¿Qué hace buena a una métrica?

En este campo se han hecho muchas contribuciones de investigadores como McCall, Boehm y Watts. Éste último publicó un trabajo resultado de las diferentes métricas utilizadas en 10 años. Él sugiere siete criterios de *bondad* para que una métrica del software pueda ser usada eficientemente (Tabla 2-3).

Desde el punto de vista de las organizaciones de software, Yourdon (1992), establece que toda métrica propuesta para su uso, debe ser:

- ***Entendible.*** La utilidad de una métrica se disminuye si quienes la utilizan no comprenden su significado o las derivaciones que implique. Quizá por esta razón la métrica *líneas de código* es ampliamente aceptada. Su significado es obvio y su aplicación uniformemente entendida.
- ***Ampliamente probada.*** Una métrica propuesta por algún investigador puede tener una base intuitiva importante y aún un sustento teórico fuerte, pero hasta no ser probada en la práctica, debe utilizarse con reserva.

Objetividad	Los resultados deben estar libres de influencias subjetivas. No debe importar la persona que efectúa la medición.
confiabilidad	Los resultados de las mediciones deben ser precisos y repetibles.
Validez	La métrica debe medir la característica correcta.
Estandarización	La métrica no debe ser ambigua y debe permitir la comparación.
Facilidad de comparación	La métrica debe ser comparable con otras medidas del mismo criterio.
Economía	Mientras más simple, y por lo tanto más barata y fácil de usar sea la medida, será mejor.
Utilidad	La medida debe cubrir una necesidad especifica; no debe simplemente medir una propiedad por sí misma. No debe medir por medir.

Tabla 2-3 Siete criterios para una buena métrica

- ***Económica.*** La métrica debe ser relativamente fácil de capturar y/o calcular. Es importante que el trabajo que implica la captura de la métrica no interfiera con el desarrollo del software en proceso. En el mejor de los casos, las tomas deben hacerse de la forma más natural.

- ***Tener un nivel alto de apalancamiento.*** Será inútil el hecho de recolectar datos con el fin de mejorar la productividad en un 0.005 %. Una buena métrica ayudará a apuntalar aquéllas partes del desarrollo de software donde el menor de los cambios produzca mejoras sustanciales.

- ***Oportuna.*** Por el beneficio de la ingeniería del software y de los líderes de proyectos, las métricas realizadas deben estar disponibles en tiempo real. Publicar las métricas seis meses después del término de un proyecto, no servirá para el proyecto mismo; las métricas se convierten en este caso en datos históricos.

2.2 Desarrollos en la medición de la calidad

2.2.1 El trabajo de Gilb

En el campo de la calidad, existen teóricos cuyas aportaciones se llegan a considerar leyes inamovibles. Deming o Crosby, por ejemplo, pueden ser considerados por muchos como *gurús* cuyas ideas deben seguirse irrestrictamente.

Tal comportamiento, sin embargo, no es siempre apropiado ni útil. Paralelamente a los teóricos de la calidad en general, Gilb aparece como un *gurú* en el terreno de la calidad del software. Su trabajo puede entenderse como el conjunto de dos grandes ramas. La primera apunta al proceso de desarrollo en sí mismo. Así, Gilb es un importante propulsor del método evolutivo mostrado en la Figura 2-6.

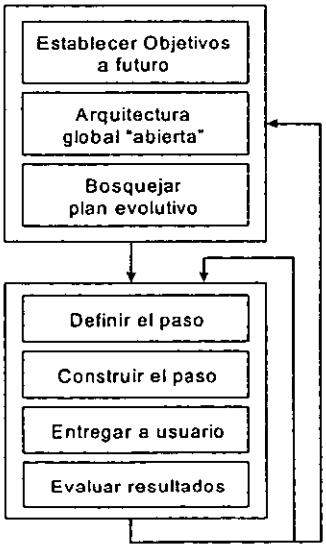


Figura 2-6 Desarrollo de software evolutivo

Como complemento a este proceso está el uso de una *plantilla de calidad*, en vez de un modelo jerárquico rígido. La principal característica de esta plantilla es

que se diseña para cumplir requerimientos locales. La filosofía de esta aproximación es que la calidad depende principalmente de un conjunto pequeño de recursos críticos, que además pueden variar de una aplicación a otra.

Dentro de tal visión, el papel de la ingeniería de software es determinar cuáles son los criterios de calidad críticos y definir hasta dónde deben estar presentes.

El método evolutivo es visto por Gilb como la satisfacción de esos criterios que son críticos. Es un proceso iterativo que pretende llegar a objetivos multidimensionales medibles. En cada etapa, el desarrollador intenta maximizar la distancia avanzada hacia el objetivo final, mientras minimiza el gasto de recursos.

Es sabido que tratar de establecer detalladamente un desarrollo de software desde el inicio puede constituir una tarea difícil y un gasto excesivo de tiempo, que pocas veces conduce a un final acertado. Si se desarrolla el sistema por etapas, el desarrollador se estará moviendo con mayor precisión a la meta final, entendiendo y mejorando las necesidades del usuario, encontrando errores oportunamente, ofreciendo valor agregado al usuario, que eventualmente pueda convertir en ganancias.

Así el proceso enfatiza el papel del usuario. Este debe conducir a un producto que cubra sus necesidades mejor, permite hacer cambios durante el proceso de desarrollo e imprime a los usuarios el sentido de pertenencia sobre el producto.

Con tales ventajas, es lógico preguntarse porqué el método no es ampliamente usado. Gilb apunta algunas áreas asociadas con la implentación del mismo.

- El simple hecho de lo novedoso del método
- La necesidad de capacitación y educación continua y sus costos asociados
- La necesidad de una administración efectiva
- La necesidad de medir los avances contra la meta final

Ciertamente, siempre existe resistencia al cambio, pero el principal problema es que *el proceso requiere personas que piensen de manera diferente en la forma de resolver los problemas.*

La práctica de dividir un proyecto en unidades más pequeñas para un desarrollo incremental, es extraña para muchos líderes de proyectos [32] .

Se acepte o no la práctica del proceso evolutivo, la liga entre la calidad del proceso de desarrollo y el producto final, está bien establecida. La calidad del producto se mide en términos de una *plantilla de calidad.* Ésta modela la calidad en términos de atributos de calidad —tales como la usabilidad— y atributos del recurso, que incluyen tiempo y costo.

Esta división refleja el mundo real, donde la calidad está sujeta a la disponibilidad de recursos. Un conjunto pequeño de recursos serán los críticos y constituirán la base de presupuesto, la programación de tiempos para las entregas o la asignación de recursos humanos con la experiencia adecuada. La *plantilla de calidad* aparece resumida en la Figura 2-7.

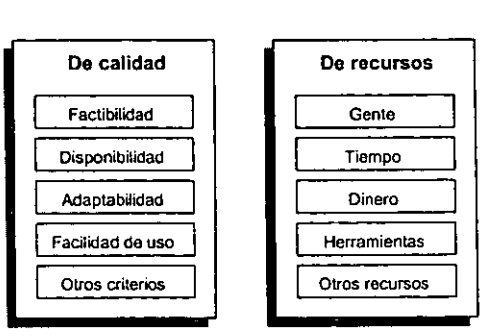


Figura 2-7 Plantilla para la calidad

32 Una analogía con este asunto puede ser el lenguaje de programación utilizado. Los lenguajes orientados a objetos o aún los de cuarta generación requieren una forma de pensar diferente al proceso de pensamiento utilizado para la escritura de programas en COBOL o FORTRAN; sin embargo, mucha gente programa en lenguajes 4G de la misma forma que lo hiciera en 3GLs, lo cual, a todas luces, resulta altamente inapropiado.

Factibilidad

La factibilidad se define como la habilidad del sistema para trabajar, por ejemplo, procesar transacciones. Tal como los atributos se dividieron en atributos de calidad y de recursos, cada atributo se puede subdividir posteriormente. La factibilidad puede considerarse en términos de capacidad de proceso, capacidad de almacenamiento o tiempo de respuesta, entre otras cosas.

Para Gilb, los términos significan:

- *Capacidad de proceso* es la habilidad para procesar transacciones dentro de una unidad de tiempo determinada.
- *Capacidad de almacenamiento* es la capacidad del sistema para guardar cosas en algún medio fisico y disponer posteriormente de ellas. Básicamente, datos.
- *Tiempo de respuesta* es la medida en que el sistema responde a un evento particular.

Disponibilidad

La disponibilidad se refiere a la proporción de tiempo real en que un sistema puede ser usado. Algunos sub-atributos son confiabilidad, facilidad de mantenimiento e integridad.

Para que un sistema pueda ser usado por las personas o pueda ofrecer atender los requerimientos de otros sistemas, sus componentes y servicios deben estar en operación el mayor tiempo posible, en el momento requerido.

Un sistema con disponibilidad baja que pueda ser usado solamente en periodos cortos de tiempo o cuando se requiera se deba esperar largo tiempo por sus servicios, no es de gran utilidad.

Confiabilidad

La confiabilidad es el grado en que el sistema hace lo que se supone debe hacer. Debido a que el propósito de cada sistema es diferente, así como es diferente el propósito de cada parte en un sistema, la forma en que este rubro se evalúa, también puede variar.

La confiabilidad, desde el punto de vista de Gilb, se establece en términos de fidelidad, veracidad y viabilidad, tanto para el código (*logicware*) como para los archivos de datos *(dataware)*. El resumen se presenta en la Tabla 2-4.

Confiabilidad	Código	Fidelidad	Se relaciona con la precisión con la que se implementa un algoritmo especificado
		Veracidad	Tiene que ver con la representación que el algoritmo especificado hace del *mundo real*
		Viabilidad	Es el alcance en que el algoritmo cumple con sus especificaciones de diseño en términos de *performance* y de requerimientos cubiertos por el sistema
	Datos	Fidelidad	Cuan exacta se representa una idea en forma de datos dentro de una aplicación
		Veracidad	Cuan bien los datos representan el *mundo real*
		Viabilidad	Cuan bien los datos requeridos cumplen las restricciones de diseño

Tabla 2-4 Clasificación de los criterios de confiabilidad

Facilidad de mantenimiento.

Este criterio se refiere al proceso para el manejo de fallas. Algunos de sus sub-atributos se presentan en la Figura 2-8 .

- *Tiempo de reconocimiento* del problema es el tiempo requerido para reconocer que la falla existe.
- *Retraso administrativo* es el tiempo que pasa entre el reconocimiento del problema y la actividad que se defina para repararlo.

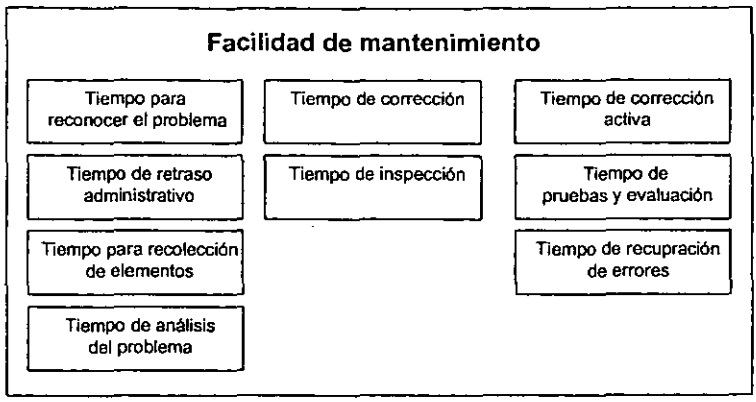


Figura 2-8 Facilidad de mantenimiento

- *Recolección de elementos* es el tiempo requerido para juntar toda la información relevante respecto a un problema, por ejemplo, análisis de programas, revisión de documentación, etc.
- *Análisis del problema* es el tiempo requerido para rastrear el origen del problema. En éste periodo se revisa el problema, sus repercusiones y sus posibles causas. Sin resolverlo, se determinan los elementos que produjeron el error.
- *Tiempo de corrección* es el tiempo requerido para llegar a una posible solución.
- *Tiempo de inspección* es el tiempo que se toma para evaluar la solución planteada.
- *Corrección activa* es el tiempo calculado para implementar la solución hipotética.
- *Prueba* representa el tiempo adecuado para probar diferentes casos y validar los cambios.
- *Evaluación de las pruebas* es el tiempo utilizado en evaluar los resultados de las pruebas.

- *Recuperación* representa el tiempo requerido para recuperar y restaurar al sistema y sus servicios.

Integridad

La integridad de un sistema es la medida de su facilidad para permanecer intacto mientras se encuentra bajo riesgo. Se puede relacionar con la capacidad de mitigar estos riesgos con funciones de seguridad definidas.

Los riesgos pueden provenir del factor humano (deliberadamente o no) o por acción del equipo de cómputo, a nivel de software o de hardware. La integridad afecta a la disponibilidad. Un sistema con integridad pobre, es probable que no esté disponible por mucho más tiempo.

Adaptabilidad

La adaptabilidad se considera en términos de improbabilidad, facilidad de extensión y portabilidad, donde:

- *Improbabilidad* es el tiempo que toma hacer cambios menores al sistema, donde el plazo total debe incluir actividades como documentar.
- *Facilidad de extensión* es la factibilidad de agregar nueva funcionalidad al sistema.
- *Portabilidad* es la facilidad de mover un sistema de un ambiente de cómputo a otro.

Facilidad de uso

Este término debe entenderse como la sencillez para usar y la efectividad al usar un sistema. Debe considerarse en términos de:

- *Requerimientos de entrada* son las capacidades humanas tales como nivel de inteligencia, manejo del lenguaje o cultura requerida para operar un sistema.
- *Requerimientos de capacitación* son los recursos, especialmente tiempo, requeridos para obtener cierto nivel de operación del sistema.
- *Gusto por el sistema* es una medida del agrado que los usuarios sienten por el sistema.

Los atributos de recursos

Estos atributos incluyen, según Gilb, tiempo, gente, dinero y herramientas.

- El recurso *tiempo* puede ser de dos tipos: el tiempo de calendario utilizado para entregar un producto final y el tiempo utilizado por el sistema una vez implementado, para completar alguna tarea.
- Los recursos de *gente*, pueden ser medidos en términos de horas-hombre. Sin embargo, este planteamiento puede no ser muy real en la medida en que el factor humano se determina por sus habilidades. En tal caso, la disponibilidad de cierta persona en particular puede ser un factor crítico. Si se requiere un programador de C, de nada nos servirán todos los programadores de COBOL disponibles.
- Los *recursos monetarios* se refieren tanto a costos de desarrollo como de mantenimiento. Ya que los presupuestos son una constante en todos los proyectos y tomando en cuenta la apreciación de muchos en el sentido de que el 80% de esos recursos se utilizan para mantener al sistema, esta área es una de las más susceptibles de ser incluida en los programas de mejoras en la calidad.
- Las *herramientas* engloban todos los recursos físicos, desde el aire acondicionado para las personas hasta equipo específico de cómputo para que cierto proyecto pueda llevarse a cabo.

La filosofía que respalda estos atributos es que el desarrollo de software no se va al vacío y la calidad no puede ser mejorada continuamente sin relación con los

costos en el sentido más amplio de la palabra. Los atributos de recursos actúan como restricciones sobre las continuas mejoras en la calidad (Figura 2-9),

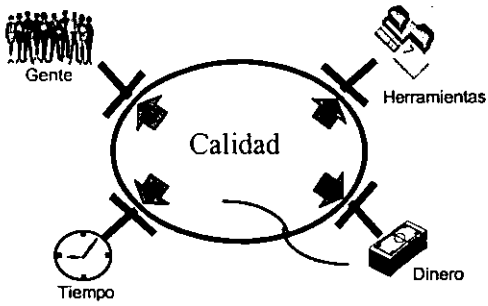


Figura 2-9 Los recursos restringen la calidad

Frecuentemente son las restricciones críticas las que determinan el nivel de calidad y no la incompetencia. La pregunta ¿lo quieren bien o lo quieren rápido? cobra sentido en el proceso de desarrollo cuando tales restricciones de tiempo o recursos se contraponen con los criterios de calidad.

Medidas para la plantilla

Gilb sugiere una serie de medidas para cuantificar los atributos presentados. No son métricas tradicionales como las presentadas con anterioridad ya que pretenden ser definidas localmente y no son necesariamente transferibles.

Así, la capacidad de proceso puede ser medida en unidades de tiempo tales como :

- Transacciones por segundo
- Registros por minuto
- Bytes por linea
- Bits transferidos por segundo

La Tabla 2-5 muestra las mediciones sugeridas por Gilb.

Atributo	Sub-atributo	Medida General	Ejemplo
Factibilidad	Capacidad de proceso	Unidades por tiempo	Transacciones por segundo
	Capacidad de respuesta	Acciones por tiempo	Tiempo de respuesta en seg.
	Capacidad de almacenamiento	Unidades almacenadas	Bytes por registro
Disponibilidad	General	Porcentaje de disponibilidad	Tiempo disponible / tiempo total
	Confiabilidad	Tiempo promedio entre fallas	Tiempo total / numero de fallas
	Facilidad de mantenimiento	Tiempo promedio para reparar	Tiempo para reparar el 90% de *bugs* de prueba
	Integridad	Totalidad	Grado en que el software está intacto
Adaptabilidad	Improbabilidad	Tiempo para cambios menores	tiempo para adicionar casos de prueba
	Facilidad de extensión	Tiempo para adicionar una función	Tiempo para adicionar el 10% de la lógica
	Portabilidad	Esfuerzo para la transferencia	Porcentaje de esfuerzo para el transporte
Facilidad de uso	General	Grado de productividad	Tiempo para adquirir un nivel básico de habilidad
	Nivel de entrada	Cualificación del nivel	Facilidad de lectura
	Aprendizaje	Tiempo para aprender	Duración del entrenamiento requerido
	Facilidad de manejo	Productividad neta	Tareas por hora
	Gusto	Grado de actitud positiva hacia el sistema	Resultados de encuestas

Tabla 2-5 Las medidas de Gilb

El modelo planteado por Gilb resulta notable no tanto por sus atributos específicos sino por algunas de las siguientes características:

- Uso de un esquema (*plantilla*) en lugar de un modelo rígido, construido con base en las características propias del entorno

- Reconocimiento explícito de las restricciones sobre la calidad
- Reconocimiento de recursos críticos
- Uso de medidas definidas localmente
- Fuertes ligas con el proceso de desarrollo

Finalmente, se hace necesario aclarar que este modelo ha sido tomado como base para trabajos posteriores, radicando en esto su importancia. Sin embargo, esta aportación es, y ha sido, blanco de severas críticas debido principalmente a que la plantilla se define para cada aplicación, imposibilitando la comparación y haciendo al proceso de medición de la calidad un gran consumidor de tiempo y recursos.

Existen esfuerzos de desarrollo tendientes a crear herramientas que representen los elementos teóricos de la calidad de software. Uno de los esfuerzos más relevantes ha sido el proyecto Inglés COQUAMO (Constructive Quality Model) que se presenta en el ANEXO C.

2.2.2 La perspectiva japonesa

El control de la calidad en Japón, en lo que se refiere al proceso de manufactura, ha sido líder mundial. Por tanto, vale la pena considerar cómo ven los japoneses la calidad en el software así como el proceso de desarrollo de software.

Yasuda[33] ha presentado un panorama general del aseguramiento de la calidad en el desarrollo de software. Se visión arranca desde la perspectiva de que la experiencia Japonesa y sus procedimientos son una "combinación singular de la experiencia de la ingeniería de software en oeste y el control de la calidad japonés".

33 Yasuda, K. (1989) Software quality assurance activities in Japan en *Japanese Perspectives in software engineering*. Inglaterra: Addison-Wesley

Yasuda establece que la calidad significa el grado de satisfacción del usuario. Para alcanzar ese grado de satisfacción es necesario tener un producto de alta calidad que cumpla estándares nacionales, *in-house* (en caso de una empresa), o definidos por el cliente en el marco de sus especificaciones. Esto se refiere a un *programa para la calidad.*

Adicionalmente, es necesario cuando se especifica el diseño, cumplir con los requerimientos del usuario. Esto se conoce como *diseño de la calidad.* Los requerimientos de programas se expresan en términos de especificaciones internas, mientras que las necesidades de los usuarios son expresadas como especificaciones externas, que juntas forman las especificaciones totales del software. Este punto de vista se resume en la Figura 2-10 .

En Japón, el aseguramiento de la calidad en software es tratado como si se produjera jabón o aparatos electrónicos. Los japoneses establecieron la primera *fábrica de software* en 1969. El hecho de *producir* software enfatiza la necesidad de construir la calidad a través del proceso de desarrollo, desde la *recepción* hasta el *empaque*, acentuando la importancia de la continuidad dentro del proceso total.

En los procesos de manufactura, la calidad se logra mediante el control de la calidad. En términos japoneses, el control de la calidad se define como

> *"Un método sistemático para proveer productos o servicios económicamente, que cumplan con los requerimientos de los usuarios"* [34]

El control de la calidad en Japón enfatiza los siguientes aspectos:

- La calidad debe ser la prioridad más alta, ya que redunda en beneficios a largo plazo
- Todo el personal debe estar involucrado
- El control de la calidad debe estar orientado hacia el consumidor

[34] JIS (Japanese Industrial Standards)

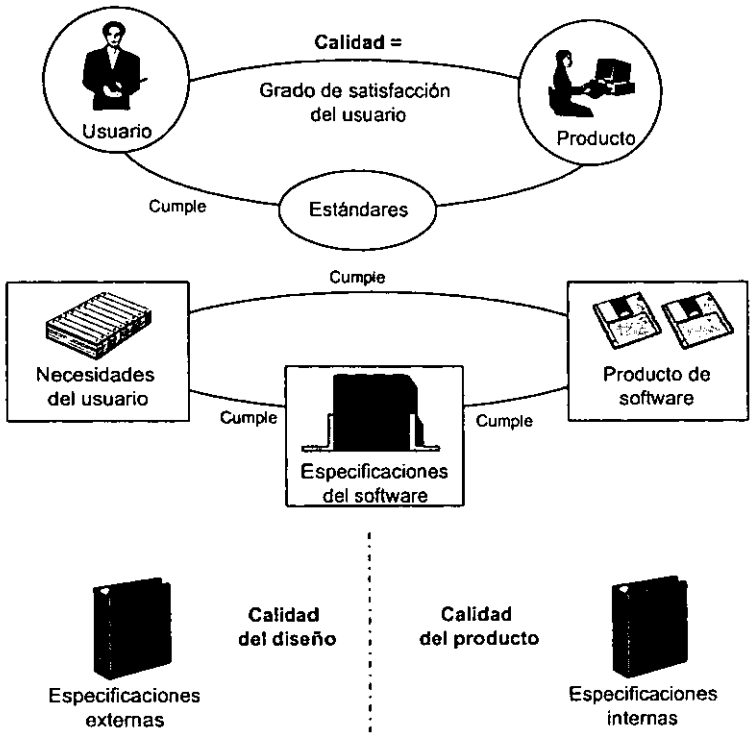


Figura 2-10 La percepción japonesa de la calidad del software

En lo que se refiere a la *producción* de software, el control de la calidad se aplica de formas cuyo origen es el proceso de manufactura. Dentro de este esquema, las funciones de diseño e implementación son unidades separadas, con funciones de control diferentes. Como una entidad separada, el departamento de inspección tiene el derecho de rechazar cualquier producto que no cumpla con los estándares.

De esta manera todos los trabajadores perciben que son partícipes del control de la calidad en el software y son estimulados para presentar y discutir cualquier problema que aparezca.

El concepto de la fábrica de software para el control de la calidad se ejemplifica en la Figura 2-11.

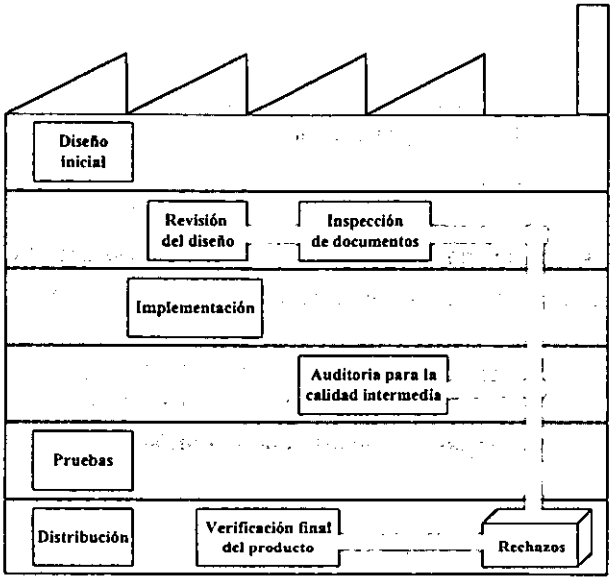


Figura 2-11 La fábrica de software

El aseguramiento de la calidad se define por la JIS como:

> *"Una serie sistemática de actividades hechas por el fabricante tendientes a asegurar que la calidad cumple completamente las necesidades del consumidor"*

Esta definición puede compararse con la correspondiente de la US IEEE/ANSI que enfatiza la conformidad a las especificaciones y reemplaza *satisfacción total* por *adecuada confianza*.

> *"Un esquema sistemático y planeado de todas las actividades para proveer una adecuada confianza de que el producto o artículo cumple con los requerimientos técnicos establecidos"* [35].

35 IEEE / ANSI

Los procesos japoneses de aseguramiento de la calidad se construyen en torno a tres etapas principales:

a) Revisión del diseño e inspección de la documentación

b) Auditoria intermedia para la calidad

c) Inspección del producto y del sistema

Revisión del diseño e inspección de la documentación

Desde los estudios de Boehm en 1981, se estableció que el costo de eliminar errores del software crece dramáticamente mientras más avanza el proyecto [36].

La revisión temprana del diseño pretende eliminar errores tanto como sea posible en las etapas iniciales del proyecto, y así minimizar costos. Yasuda apunta una lista de características requeridas para una revisión del diseño efectiva, entre las que están:

- Establecer fechas específicas para las revisiones
- Hacer un *checklist* completo de la revisión
- Llevar un registro de errores previos
- Conducir una adecuada investigación de cada uno de los hallazgos durante la revisión

Los documentos generados por estas actividades son vistos como productos del proceso de desarrollo de software, tan importantes como el código mismo. Estos documentos pueden considerarse como internos o externos. Un ejemplo de documentación externa son los manuales de usuario.

La calidad de la documentación está estrechamente ligada a la calidad del software y particularmente con la satisfacción del usuario hacia el producto final.

---

36 No sólo de manera empírica resulta cierta esta afirmación. Especialmente en proyectos complejos, con interacción de varias entidades de negocio o de sistemas, reparar errores en etapas avanzadas puede, incluso, representar un proyecto adicional.

Auditoria intermedia para la calidad

El objetivo de esta auditoría se divide en tres puntos:

- Predecir errores ocultos, haciendo una estimación del numero de errores que se detectarán.
- Comparar el número de errores real con el límite establecido.
- Analizar e investigar los errores.

La auditoría hace uso de la computadora para aplicar métodos estadísticos para la predicción de errores, basados en una técnica conocida como *prueba de calidad*. Si el número de errores encontrados o estimados excede el límite, entonces el producto es rechazado.

Durante la auditoría se pueden fijar ciertos límites de tolerancia con el fin de reducir el número de errores; sin embargo, si estos límites se exceden, el producto se rechaza nuevamente.

Inspección del producto y del sistema

Esta es la última etapa de inspección antes de la liberación del software. Los procedimientos usados son similares a los de la auditoría intermedia; nuevamente se utilizan métodos estadísticos de computadora.

La primera verificación consiste en asegurarse de que el producto cumple las especificaciones internas y externas. Esto constituye la *inspección del producto*. En esta etapa, se comprueba si el software cumple con los requerimientos del usuario dentro de su entorno real. Es por eso que las pruebas deben llevarse a cabo en un ambiente lo más cercano posible al real en que el usuario trabaja.

En esta etapa adicionalmente se pueden utilizar herramientas de simulación junto con casos de prueba construidos para tal fin.

La visión de los japoneses respecto al aseguramiento de la calidad es típicamente completa, sin embargo, existe cierta discrepancia entre sus objetivos

y sus procedimientos. El concepto de *fábrica* está orientado a generar cero defectos. Los procedimientos utilizados intentan garantizar que el producto estará libre de errores no sólo durante el proceso de producción, sino durante la operación por parte del usuario.

Esta situación es muy deseable, pero no satisface del todo los preceptos señalados por Yasuda para la calidad en el software. Cumplir con las necesidades del usuario es más que asegurarse de que el sistema trabaja correctamente bajo condiciones reales. Se refiere también a que el sistema contesta a las cuestiones correctas, al mismo tiempo de que se asegura de que las respuestas son las adecuadas.

Estas situaciones se enmarcan dentro de *Administración Total de la Calidad* (ATC).[37]

2.3 *Reuso de Software*

El reuso de software es una materia de la que se habla con frecuencia y que a pesar de su importancia salida a la luz en la década de los 60, hoy en día es raramente aplicada con eficiencia en las empresas. Este es un hecho preocupante pues su aplicación se relaciona con el camino hacia la calidad. Yourdon apunta:

> *"La reusabilidad del software pasará a la historia como una de las mayores contribuciones técnicas para la calidad y productividad en el software durante los años 90".* [38]

2.3.1 El problema y su entorno

Aun cuando los costos asociados con la creación de software tienden a cuidarse, la productividad asociada tiende a descender con la demanda de nuevos sistemas. Con frecuencia, el software desarrollado es de calidad pobre y muy difícil y caro de mantener.

---

37 Conocida también como TQM por sus siglas en inglés de *Total Quality Management.*

38 Yourdon, *op. cit.* p.214

Las mismas tendencias se perciben en todos los niveles de la industria del software. Sin embargo, "el reuso de software tiene el potencial de incrementar la productividad, reducir los costos y mejorar la calidad del software".[39]

¿Por qué se hace necesario reutilizar componentes de software? Algunas estadísticas indican, por ejemplo, que de todo el código escrito en 1983, menos del 15% fue realmente nuevo y específico para alguna aplicación. El reuso de software aparece como una área de oportunidad para aumentar la productividad y reducir costos, con el objetivo de aprovechar al máximo todo o parte del 85% del software redundante, creado con anticipación.

Por muchos años, el reuso de software ha ocurrido en el desarrollo de bibliotecas con subrutinas matemáticas, y actualmente ha alcanzado bibliotecas para sistemas operativos, procesadores de lenguaje, generadores de reportes o desarrollo de compiladores.

El reuso de software debe, sin embargo, ampliar su rango de alcance hasta llegar a los tamaños y complejidades del software actual. Su éxito no se basa en el aislamiento, por el contrario: el reuso debe aplicarse dentro de una estructura efectiva de ingeniería de software que incluya, invariablemente, desarrollo organizacional, procesos de ciclo de vida (políticas, métodos, herramientas, etc.) así como personal altamente capacitado.

2.3.2 Definiciones y conceptos

Los productos candidatos para ser reusados se caracterizan por relacionarse con actividades de las etapas de desarrollo y mantenimiento. Esas actividades se pueden caracterizar como:

- Análisis de dominio
- Especificación de requerimientos
- Diseño de alto nivel

39 Hooper, J., *et. al.* (1991) Software Reuse, EU: Plenum Press, p. 1

- Diseño detallado
- Codificación y pruebas
- Documentación
- Mantenimiento

En principio se esperan grandes beneficios al reusar productos surgidos de actividades donde se requiere un mayor grado de abstracción. Así, una especificación de requerimiento reutilizada representa un mayor apalancamiento respecto al reuso de un módulo de código.

Solo una parte de la inmensa cantidad de información manejada durante el ciclo de vida del sistema se recuerda o se almacena como antecedente. El conocimiento adquirido durante el proyecto suele exceder la capacidad de retención. Ésta es una razón que apoya el reuso de software.

Ya que el personal altamente experimentado retiene naturalmente mucho más conocimiento de su trabajo previo, es beneficioso el reuso del personal en el desarrollo de software, en una área especifica. También el reuso de procesos de desarrollo y mantenimiento en áreas no solamente de sistemas es recomendable dentro de las organizaciones. El personal aprende a aproximarse a la solución de los problemas de manera sistemática, y se beneficia de la aplicación repetida de los procesos.

Tracz (citado por Hooper & Chester) divide el reuso de software en tres áreas

a) Reuso de productos

b) Reuso de procesos

c) Reuso de personal

Basili, et. al. adoptan esta división y enfatizan dentro de ella, el *reuso del conocimiento* que existe sólo en las mentes de las personas (conocimiento informal), *el reuso de planes* que digan la forma de hacer ciertas actividades o que especifiquen la manera de estructurar y documentar ciertos productos

(conocimiento estructurado) y el *reuso de herramientas y productos* (conocimiento orientado a la producción).

En este contexto, el éxito en el reuso de software de debe a las características del componente reutilizado [40], del proceso reusado y al ambiente en el que tiene lugar el reuso.

¿Qué significa reuso de software? Aquí se muestran dos definiciones:

a) El límite hasta el cual un componente de software puede ser utilizado (con o sin adaptaciones) como la solución a múltiples problemas.

b) El límite hasta el cual un componente de software puede ser utilizado (con o sin adaptaciones) como la solución a un problema diferente para el que fue originalmente desarrollado.

La definición b) sugiere que la reusabilidad es un resultado incidental del desarrollo de software: un componente puede resolver varios problemas. La definición a) apunta al esfuerzo planeado hacia la reusabilidad.

Dentro del contexto de las definiciones, el *reuso del software* es la meta, mientras que al *reusabilidad del software* representa lo necesario para alcanzarla. La primera definición impulsa la creación de componentes de software específicamente para ser reutilizados, lo que es diferente a utilizar componentes que no fueron concebidos específicamente para el reuso[41] o de modificar código para crear nuevas versiones de un producto.

Las siguientes definiciones de *reuso* son consistentes con la definición a):

- Reuso es un acto de sintetizar la solución a un problema basándose en soluciones predefinidas a subproblemas.

---

40 En la mayor parte de la literatura sobre reuso de software, un *componte reusado* se refiere a cualquier tipo de recurso de software: módulos de código, especificaciones de diseño, establecimiento de requerimientos, conocimiento adquirido, experiencia de desarrollo, documentación, etc.

41 Más puntualmente, cuando se utiliza un componente no concebido originalmente para ser reusado, se estará haciendo un *salvamento o rescate de software*.

- Reuso es el proceso de implementación de nuevos sistemas de software a pertir de software pre-existente.
- Reuso es la reaplicación de una variedad de tipos de conocimiento de un sistema, a otro sistema similar para reducir el esfuerzo de desarrollo y mantenimiento de éste nuevo sistema.
- Reuso es el proceso por el cual el trabajo alrededor del software (código fuente, documentación, diseños, datos de prueba, especificaciones) son tomados y usados en un nuevo desarrollo, preferiblemente con un mínimo de modificaciones.

Existen numerosas implicaciones para el reuso de software, incluyendo la disposición para retener una solución predefinida para reusarla; codificar y retener las soluciones (por ejemplo, código); reconocer la disponibilidad y potencial aplicable a los componentes durante la solución a un problema; y, adaptar o modificar los componentes en software que provean una solución válida al problema. Alcanzar el reuso del software implica trabajar con estas implicaciones.

2.3.3 Beneficios del reuso de software

¿Qué se entiende en general por un grado alto de reusabilidad? De acuerdo con Yourdon la regla es simple: las mejores empresas de desarrollo alcanzan entre el 70 y 80 por ciento, mientras que las organizaciones típicas se quedan en niveles que oscilan entre el 20 y 30 por ciento.

Así, si un proyecto debe generar un producto de 100,000 líneas de código, una buena empresa programará sólo de 20,000 a 30,000 líneas, y utilizará el resto a partir de bibliotecas de componentes de reuso. Una empresa común desplegará el esfuerzo intelectual de escribir entre 70,000 y 80,000 líneas y sólo el resto las tomará de componentes reusables. Para hacer el panorama peor, la empresa común no tendrá control sobre los componentes que reutiliza; hace copias simples del código que requiere y lo modifica.

Este ejemplo se refiere a un componente inmediato como es el código fuente. Sobre él, los participantes en el desarrollo de software tienen un control más estrecho. ¿Cómo entender entonces el reuso de componentes cuyo nivel de abstracción es mayor, como las especificaciones, los diseños, etc.? El panorama se complica infinitamente.

a) Productividad

El primer beneficio del reuso de software es el aumento de la productividad. En un sentido esquemático, una organización que alcanza el 80% de reusabilidad es cuatro veces más productiva que aquélla con 20% de reusabilidad.

Sin embargo, los ahorros rara vez son tan notorios, ya que los beneficios de la reusabilidad implican:

- *Una inversión para crear componentes reusables.* Este capital de inversión se amortiza entonces entre el número de sistemas o programas que utilizarán estos componentes Obviamente, mientras más frecuentemente se reutilicen los componentes, menor será el capital de inversión inicial.
- *Una inversión en rigurosos procesos de prueba y aseguramiento de la calidad.* Los niveles altos de pruebas y QA se requieren por, y se justifican por, los altos niveles de uso que tendrán los componentes. La mayoría de las organizaciones encuentran que necesitan de dos a cuatro veces más pruebas para un componente reusable que para uno aislado.
- *Una inversión para mantener las bibliotecas de reuso.* Esta inversión aplica también para *browsers* y utilerias para que los participantes del desarrollo puedan encontrar los componentes reusables cuando los necesiten. Esto puede variar desde un simple índice, hasta un sistema experto en línea para hacer búsquedas complicadas.

b) Calidad

Otra razón para enfatizar la reusabilidad es el incremento en la calidad. Como se mencionó, los componentes reusables de software siempre requieren de más

pruebas y de un control de calidad más estricto, simplemente porque las consecuencias de un error son mucho más serias.

Esto que parece ser un inconveniente, se revierte cuando, por definición, los módulos con un alto grado de reuso tienen mayor calidad que los módulos ordinarios. En términos técnicos, los *bugs* son eliminados más rápida y exhaustivamente. Si estamos creando un nuevo producto a partir de componentes reusables, estaremos garantizando a la vez que es un producto sólido y aprobado previamente.

c) Prototipos

Otro beneficio que suele pasarse por alto es que la reusabilidad puede crear mecanismos para el desarrollo de prototipos. Incluso, miembros del SPC[42] apuntan que *"la reusabilidad y el desarrollo de prototipos son dos caras de la misma moneda"*.

Muchos de los esfuerzos actuales para hacer prototipos dependen de lenguajes de alto nivel y/o de herramientas gráficas para crear rápidamente un *esqueleto* de un nuevo sistema. El problema es que mientras la fachada del nuevo sistema impresiona al usuario, no existe nada detrás de esta fachada: no existe ninguna funcionalidad que permita al usuario probar que el sistema haga lo que debe.

Un prototipo construido a partir de componentes reusables, por el contrario, puede proveer una funcionalidad completa, ya que es creado a partir de componentes completamente codificados y probados. Otra ventaja: como los componentes reusables ya existen, el prototipo debe ser tan fácil y rápido de construir como cualquier otra forma de prototipo.

42 Software Productivity Consortium

2.3.4 Los obstáculos para la reusabilidad

Si la reusabilidad es altamente beneficiosa. ¿Porqué su aplicación no está ampliamente expandida? Hay algunas razones para ello:

- Los libros de texto de ingeniería de software enseñan a los nuevos practicantes a construir sistemas a partir de *principios básicos*; la reusabilidad no es promocionada o peor aún, no es discutida.
- El síndrome NIH[43] (no inventado aquí) y el desafío intelectual que implica el resolver un problema interesante de software de una sola manera, interfiere cuando hay arrogancia para no utilizar un componente de software de alguien más.
- La aplicación no exitosa en experiencias pasadas con el reuso de software convence a dirigentes de organizaciones y participantes en el proceso de desarrollo, que el concepto *reuso*, no es práctico.
- La mayoría de las empresas no *recompensan* el reuso de software. La productividad se mide en función del número de líneas *nuevas* de código, mientras que las líneas reusadas de código se descuentan normalmente en un 50 por ciento o más.

En los apartados siguientes se aborda brevemente cada uno de estos aspectos.

2.3.4.1 La reusabilidad vista por la academia

Casi cualquier libro de texto o curso universitario sobre diseño, análisis o desarrollo de sistemas enseña al estudiante la manera de resolver el problema desde cero.

En raros casos se le enseñará al alumno una forma inversa *(bottom-up).* de diseño, en la que arranque con la existencia de bibliotecas y módulos para componer una solución agregada al problema.

---

43 *Not Inveted Here*, Utilizado ampliamente en la literatura especializada.

En la mayoría de los casos se le dará al estudiante una hoja en blanco (o una herramienta CASE vacía) y se le instruirá en la forma lineal *(top-down)* de analizar, diseñar o codificar, por ejemplo, identificando funciones mayores y descomponiéndolas en subrutinas.

Nadie le dice al estudiante que el problema que está tratando de resolver ha sido resuelto con anterioridad cientos de veces y que quizá la solución óptima sea encontrar una solución aproximada *existente* y refinarla para cumplir con sus necesidades especiales.

De hecho, el rigor académico de las universidades considera esto como un plagio y penaliza al estudiante por seguir prácticas no apegadas a la ética. El reuso de software no es ni promovido ni enseñado en las escuelas, en detrimento del comportamiento profesional que posteriormente tendrán los estudiantes.

No es extraño que estos hábitos escolares sean traspasados por la persona cuando pasa al campo de trabajo formal. Alguien cuya práctica haya sido la de resolver todos los problemas desde cero, verá con reservas la posibilidad de crear soluciones con valor agregado a partir de soluciones ya existentes, sin contar la complicación para crear módulos exclusivos para el reuso. El impacto cultural que implica enfrentarse a ambientes no previstos por la carrera universitaria es enorme.

En este sentido, las universidades deben ser sensibles a la problemática que actualmente presenta el desarrollo de software. Como vínculos con el campo productivo es importante la inclusión en los programas de estudio los aspectos de calidad que serán puntos críticos en el desarrollo profesional.

2.3.4.2 El hilo negro

No sólo en al ámbito del desarrollo de software, sino en muchos más, persiste la necesidad quizás innata de las personas, para reinventar la rueda y resolver una y otra vez, problemas antiguos.

En el caso del desarrollo de software, el síndrome NIH es particularmente agudo porque para el nuevo desarrollador, un problema siempre se le presentará como una oportunidad de diseñar algo absolutamente nuevo para un problema nunca antes resuelto.

Esto es válido, pues para un problema especialmente de software, habrá soluciones rápidas o lentas, unas más inteligentes que otras, algunas muy sofisticadas o que sean reutilizadas en mayor o menor grado. Sin embargo, una vez dentro de un proceso de desarrollo, con tiempos y costos asociados, las consecuencias de tales prácticas pueden ser adversas.

Lo que es importante reconocer es que para el novel ingeniero de software deriva un gran placer intelectual de resolver problemas familiares —un *sort* binario, el cálculo de una nómina, o el pago de impuestos— por sí solo, con la posibilidad de, *posiblemente*, encontrar una mejor solución.

En casos extremos, algunas personas dedican mucho esfuerzo a dar soluciones a problemas que ellos mismos ya han resuelto con anterioridad. Tarde o temprano la novedad y la emoción ganan.

Por supuesto, no se debe fallar en propiciar la creatividad y curiosidad en el entorno del desarrollo de software, que por demás son características innatas del ser humano, pero es importante que las organizaciones promuevan, incentiven y capaciten a su personal para enfocar su energía intelectual en resolver prioritariamente problemas que ***no*** hayan sido resueltos.

2.3.4.3 Experiencias fallidas con la reusabiliadad

Dado que el concepto de reuso de software no es un concepto nuevo, presupone que algunas organizaciones lo aplicaron, especialmente en las décadas de los años 60 y 70, pero que los resultados fueron con frecuencia tan pobres que el esfuerzo fue abandonado.

¿Porqué existen fallas en la aplicación del reuso de software? La experiencia ha mostrado que estas fallas tienen su origen en la administración, en los recursos técnicos o en una combinación de ambos factores.

Fallas en la administración

- *Recursos o inversión inadecuada.* No se espera que el reuso ocurra a partir sólo de slogans o peticiones directivas. Los recursos requieren construir, mantener y administrar una biblioteca de componentes reusables, cuya inversión monetaria puede ser bastante considerable al principio, y que lleva consigo la dificultad de justificar el retorno de la inversión.
- *No se definen grupos de trabajo específicos para crear componentes reusables.* Obviamente, es irreal asumir que los ingenieros de software tengan el tiempo, la energía y lucidez para crear componentes reusables de software cuando se encuentran desarrollando un software específico para un proyecto. Se requiere un grupo por separado, para tal fin.
- *No se estimula el reuso de software.* La productividad en el desarrollo de software puede medirse por el número de líneas de código escritas. Es una medida válida. Sin embargo, considera líneas de código *nuevas*. En tal caso, las líneas de código reusado aparecen como producto de un esfuerzo mucho menor.

  Las implicaciones son inmediatas: si a los ingenieros de software se les mide y estimula por el número de líneas nuevas contenidas en un nuevo producto, crearán mas y más código nuevo; si el esfuerzo de reusar código existente les afecta personalmente, evitarán hacer el esfuerzo.

Fallas técnicas y administrativas

- *Inadecuado control de versiones.* Si la organización no puede llevar un registro veraz de qué versión del componente reusable se utilizó en cuál sistema, el esfuerzo del reuso de software se saldrá de control y eventualmente creará un colapso.

- *Mecanismos inadecuados de búsqueda y consulta.* Si no se sabe cómo y en dónde buscar un componente que cubra ciertas necesidades, aunque exista, no se utilizará.

  Desarrollar herramientas de búsqueda y consulta de componentes es un problema técnico y representa un problema potencial nada trivial. Al mismo tiempo es un problema administrativo, pues si no se destinan recursos a la creación de tales herramientas y de las bibliotecas de reuso mismas, éstas nunca serán construidas.

- *Poco control de lo que se pone en las bibliotecas.* El resultado es frecuentemente cantidades considerables de módulos con poca calidad, módulos con funcionalidad duplicada, características de los módulos no documentadas, etc.

Fallas técnicas

- *Inclusión de componentes muy grandes o con mucha funcionalidad.* Un error desde que el concepto de *reuso* apareció, fue la tendencia a crear módulos de propósito general que trataban de hacer todo, para todo tipo de personas y bajo todo tipo de condiciones.

  El resultado era con frecuencia que tales módulos no hacían nada extremadamente bien para todos y generaban efectos secundarios no deseados. La tendencia debería ser crear componentes pequeños sin efectos colaterales.

- *Interfaces no documentadas.* Si el ingeniero de sistemas que está evaluando un componente para reutilizarlo no sabe qué parámetros de entrada requiere, ni qué resultados produce, será prácticamente imposible que lo tome y lo aplique en su desarrollo.

- *Inflexibilidad para manejo de excepciones y cambio de parámetros.* Muchas veces se encuentran módulos aceptables, pero que se quisiera que fueran *un poco* diferentes. Ese *"poco"* puede implicar un cambio en la funcionalidad, diferentes parámetros, diferentes secuencias de ejecución, etc.

Con frecuencia los desarrolladores tienen que afrontar el hecho de que si se encuentra un componente reusable, se deben utilizar todos sus módulos o ninguno.

En nuestros días, por ejemplo con la expansión en el uso de GUI[44], el desarrollador puede cortar y pegar porciones de código y así hacer un reuso parcial; lo mismo pasa con ciertas bibliotecas orientadas a objetos. Sin embargo el esfuerzo por crear clases y subclases de objetos que permitan manejar excepciones debe continuar.

- *Se requiere un retrabajo adicional para compilar, ligar y ejecutar un módulo reusable.* Si se encuentra un módulo reusable de hace 20 años, es casi imposible un gasto adicional de esfuerzo de recursos de CPU o memoria para compilar, editar, ligar y ejecutar el módulo. Aunque este retrabajo existe (y no debería), hoy es más tolerable debido a las velocidades y capacidades actuales de las máquinas.

2.3.4.4 Falta de motivación para el reuso de software

Muchos directores de organizaciones enfatizan la productividad hoy en día. De hecho, la productividad, sea medida en número de líneas de código o puntos de función, ha sido una de las métricas más importantes en la efectividad del desarrollo de software por tres décadas o más.

Lo relevante es que la mayoría de las mediciones se hacen tomando en cuenta sólo el numero de líneas de código *nuevas*; no se les da crédito alguno a las líneas que provienen de módulos reusables. Así, no se le reconoce ningún esfuerzo a la persona que propicia un alto grado de reusabilidad al implantar en su código componentes de reuso.

En el mejor de los casos, algunas empresas cuentan las líneas reusadas, pero en una fracción del 30 al 50 por ciento respecto al peso de las líneas nuevas. A éste respecto Yourdon señala:

---

44 *Graphic User Interface*

"(...) lo que la organización debe hacer es medir la productividad de la funcionalidad liberada al usuario final, no el número líneas de código nuevo que frecuentemente repite un esfuerzo intelectual hecho por docenas de ingenieros de sistemas anteriores" [45]

La falta de impulso institucional al reuso de componentes, así como la medición de la productividad sin considerar el reuso, pueden generar casos extremos en las empresas.

Si los desarrolladores de software no son estimulados para el reuso y por lo contrario, son recompensados en función del número de líneas nuevas que escriban, entonces seguirán escribiendo nuevo código. Si el esfuerzo de reusar código existente penaliza personalmente al desarrollador, evitará hacer el esfuerzo.

En algunos casos, las empresas destacarán la importancia del reuso de software y darán mayor crédito a los desarrolladores que reusan software. Sin embargo, un problema permanece: ¿de dónde provinieron los componentes de software reusables la primera vez?

Como se mencionó antes, la creación de un componente reusable implica un esfuerzo quizá doble respecto a un módulo normal, porque debe desarrollarse y probarse al máximo detalle. ¿Dónde están entonces los recursos (tiempo, presupuesto, etc.) destinado a la creación de bibliotecas de componentes reusables?

Más que un problema académico, es un problema de inversión. SI no hay presupuesto destinado a la creación de bibliotecas de reuso, no habrá bibliotecas y sin ellas no se puede hablar de reusabilidad.

El problema es básicamente hacia dónde enfocan las organizaciones. Muchas tienden a enfocarse a mejorar la *productividad individual* con el uso de herramientas, técnicas o capacitación. Algunas se enfocan en la *productividad del proyecto*, usando por ejemplo herramientas CASE.

45 Yourdon, E., *op. cit.* p. 220

Sin embargo, la reusabilidad del software debe apuntar a la *productividad de la empresa*, y la organización debe tratarla como una inversión capital y no esperar a que los ingenieros de software tengan la inspiración o energía en su tiempo libre para crear componentes reusables para bien de toda la empresa.

Una vez más, como ocurre con la aplicación de planes de calidad o el impulso del cambio cultural hacia la misma, es imperiosa la existencia de una *visión institucional*, que no sólo avale sino también propicie los cambios. Sin ella los esfuerzos personales no encontrarán eco, y la empresa no evolucionará para bien.

2.3.5 Niveles de reusabilidad

Mucha de la discusión expuesta apunta a que el elemento más importante a reutilizar es el código, esto es, instrucciones en lenguajes como C o PASCAL. Como se ha expuesto, sin embargo, el reuso implica componentes diversos, no sólo el código.

Los siguientes apartados muestran el reuso de componentes diversos.

2.3.5.1 Reuso de código

Cuando se habla de reutilizar código, se crea una imagen mental de hacer llamadas a subrutinas que residen en alguna biblioteca, pero el concepto puede tomar alguna de las variantes siguientes:

- *Cortado y pegado de código fuente*. Desde que se perforaban tarjetas, el uso de porciones de código de otros programas ha sido práctica común. En algunos casos el código tomado se usa intacto, mientras que en otros puede sufrir alguna modificación adicional para cubrir una funcionalidad específica.

  Esta es una forma *primitiva* de reuso, que es mejor que no tener nada. Tal práctica lleva implícita el riesgo de generar errores durante la copia y/o modificación del código original. Peor aún lo relacionado con el control de

versiones. Para la organización será prácticamente imposible llevar un histórico de las múltiples versiones de código mutadas bajo este esquema.

Aunque este esquema está ampliamente expandido actualmente —aún en empresas grandes de desarrollo—, no debe considerarse como un ambiente de *reuso de software* sino como un *salvamento o rescate de software*.

- *"Includes" a nivel de código*. Muchos de los lenguajes de alto nivel tienen mecanismos para copiar o *"incluir"* código dentro de los programas, a partir de alguna biblioteca.

  Por ejemplo, el uso de la sentencia COPY en COBOL o la instrucción INCLUDE en C, cumplen esta función.

  Este tipo de sentencias se usan más frecuentemente para definiciones de datos o parámetros, pero también puede usarse para copiar instrucciones de programas ejecutables.

- *Ligas binarias*. La mayoría de los lenguajes de programación permiten los llamados *"externos"* a subrutinas, procedimientos o funciones contenidas en alguna biblioteca de componentes.

  Esas funciones externas han sido ya compiladas o ensambladas, pero deben ser incorporadas a los programas que las llaman. Esto se hace durante la fase de *"load"* o *"link-edit"* de la compilación del programa principal.

  Una ventaja de ésta práctica es que se incluye sólo una vez el componente en la versión ejecutable, sin importar cuantas veces se utilice dentro del mismo.

- *Invocación en tiempo de ejecución*. El problema con las tres formas anteriores de reuso de código, es que el programador debe conocer, en el momento en que escribe el código, qué componente desea reutilizar. El componente reusado se debe adicionar al momento de la codificación, de la compilación o de la link-edición.

  En algunos casos, el programador requerirá la flexibilidad para permitir al programa determinar, *en tiempo de ejecución*, qué componente debe invocar.

Algunos lenguajes de programación facilitan esta labor al permitir la definición de funciones o procedimientos dinámicos con las direcciones de los componentes que serán resueltas durante la ejecución del programa.

2.3.5.2 Reuso de datos

De la misma forma que las porciones de código incluidas en los programas, hacen evidente el reuso de código, los datos pueden ser reutilizados no solamente en forma de parámetros globales o constantes dentro de los programas, sino en forma de accesos a un repositorio de datos.

Con las herramientas CASE el uso del repositorio de datos reusables, se hace imprescindible. Con éste, no solamente se pueden utilizar declaraciones de datos sino todo tipo de diagramas de flujo, diagramas de entidad, diseños de bases de datos o diagramas estructurales.

2.3.5.3 Reuso de diseños

Las formas más sencillas e inmediatas de reuso se refieren al código, ya que es lo más cercano a los programadores; sin embargo, las formas más importantes de reuso son el reuso de diseños y el reuso de análisis, pues implican un grado de abstracción mayor.

En esencia, el reuso de código tiene lugar después de que la parte más complicada del proyecto se llevó a cabo: el análisis y el diseño. Una regla empírica indica que la programación representa sólo del 10 al 15 por ciento del tiempo dedicado al desarrollo de un sistema, así que cualquier esfuerzo para incrementar la productividad solamente en este rubro, ya sea por reuso de código o cualquier otro medio, sólo estará impactando a una pequeña parte del total.

En la actualidad, las organizaciones de punta en el desarrollo de software se han dado cuenta que obtienen resultados más significativos a través del reuso de los diseños o las especificaciones.

El reuso de código usualmente ocurre en los niveles más bajos de la jerarquía del diseño de sistemas; sin embargo, el reuso de diseños tiene lugar en ramas completas del árbol de la estructura de reuso que se esté utilizando. Gráficamente se muestra en la Figura 2-12.

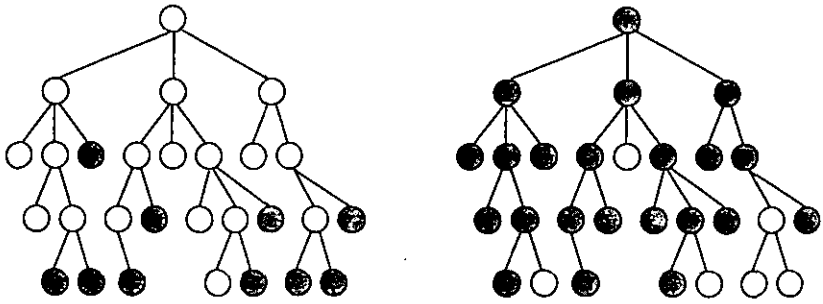


Figura 2-12 Reuso de código vs. reuso de diseño

Reusar componentes abstractos no es una tarea fácil porque en ellos está plasmado el conocimiento y la experiencia previa de las personas. El reusar diseños o especificaciones se traduce en la cobertura de más actividades —y de mayor relevancia— que se desarrollan desde el principio de un proyecto.

2.3.5.4 Reuso de especificaciones

Si el reuso del diseño es bueno, el reuso del análisis (especificaciones) es aún mejor porque permite eliminar el esfuerzo invertido en el diseño, la programación y las pruebas de las especificaciones.

En los años 70, el reuso de las especificaciones era virtualmente imposible, ya que las empresas de software dependían de amplios documentos de texto que describían los requerimientos del usuario.

En los años 90, con la expansión de las herramientas gráficas tales como el análisis estructurado, emergió el concepto de reuso a nivel de las

especificaciones. Sin embargo, su aplicación comenzó siendo poco práctica debido a que el análisis de los sistemas consistía básicamente en documentos en papel que generalmente no estaban actualizados.

En los años recientes, las organizaciones de software han apuntado más agresivamente al reuso de componentes a nivel de las especificaciones, debido a que sus correspondientes modelos (diagramas de flujo, diagramas de relación, diagramas de transición, etc.) se mantienen actualizados en repositorios CASE.

Reutilizando componentes en este nivel y durante la fase de análisis de un proyecto, el analista debe ser capaz de acceder a un repositorio institucional de proyectos previamente desarrollados para buscar alguno con especificaciones similares que puedan ser usadas como punto de partida para el nuevo proyecto.

Una de las formas de reuso más atractivas en este nivel es el *designware*. Este concepto se ha utilizado para designar el reuso de programas aplicativos, puestos en el mercado en forma de repositorios CASE.

Este concepto revoluciona el panorama del reuso ya que promete una forma más agresiva y de mayor nivel en esta práctica, ofreciendo la posibilidad de personalizar paquetes comerciales de software, mediante adecuaciones a los programas fuente.

2.3.5.5 Otros tipos de reuso

Mientras que el código, los diseños y las especificaciones parecen ser los candidatos inmediatos para el reuso, no son los únicos. Las empresas progresistas de software siempre están buscando todos los tipos de componentes en el desarrollo del sistema para analizar si pueden ser capturados, organizados y clasificados para su subsecuente reuso.

Algunos otros componentes que pueden reusarse son:

- Estimaciones y cálculos de costo-beneficio

- Documentación de usuario
- Estudios de factibilidad
- Casos de prueba

Finalmente, también se debe considerar la importancia de los componentes que *no pueden* ser almacenados en un repositorio de uso institucional: la gente que conforma el equipo de trabajo.

La experiencia, infraestructura y camaradería alcanzadas durante el desarrollo de un proyecto deben ser aprovechadas —esto es, reusadas— en proyectos siguientes y siempre que sea posible.

Aunque parece de sentido común, no lo es en la típica empresa de desarrollo de software: los equipos de trabajo son deshechos al final del proyecto y los participantes son atomizados y asignados a diferentes áreas de trabajo.

El reaprovechamiento del talento y capacidades humanas como base para el incremento de la productividad en el software por lo general resulta más beneficioso que el reaprovechamiento de elementos técnicos.

2.3.6 ¿Cómo abordar la reusabilidad?

No existe una receta de cocina que le permita a una empresa implementar un ambiente de reuso que resulte eficiente. Para que esto ocurra deben implementarse una serie de medidas básicamente en dos rubros: el tecnológico y el administrativo.

Brevemente se abordan cada uno de estos puntos.

2.3.6.1 El acercamiento tecnológico a la reusabilidad

Desde el punto de vista tecnológico, la reusabilidad del software puede estar acompañada de alguno de los mecanismos de la Figura 2-13.

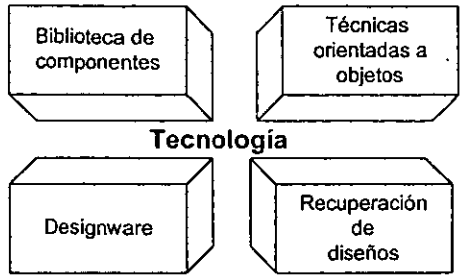


Figura 2-13 Visión tecnológica de la reusabilidad

Bibliotecas de componentes

El uso cada vez más expandido de las herramientas CASE apunta a que los repositorios asociados se convertirán gradualmente en los almacenes de componentes reusables de software. Pero en nuestros días muchas empresas relacionan reusabilidad con "bibliotecas" —bibliotecas de programas fuente o funciones, subrutinas, clases—, archivos de parámetros, bibliotecas de COPY o lay out de archivos.

Si una empresa hace un esfuerzo serio para la reusabilidad, debe considerar la relevancia de su biblioteca asociada. Para que esta biblioteca sea digna de aplicarse debe cumplir algunos puntos básicos:

- Buena documentación
- Debe asociarse con una gestión eficiente de la configuración (control de versiones) para guardar registro de las modificaciones sucesivas que se hacen de un componente
- Debe disponer de mecanismos adecuados de búsqueda y recuperación de componentes
- Debe contar con un esquema de seguridad que garantice la aplicación de normas de inserción o actualización de nuevos componentes
- Debe asegurar que los componentes incluidos han sido completa y eficientemente probados

¿Qué tan grande debe ser una biblioteca? Si una empresa se enfoca el reuso de código, es bueno saber que las empresas exitosas en este rubro tienen entre 200 y 400 componentes. Es altamente probable que haya poco uso de los componentes si la biblioteca contiene más de 1000.

Si bien el desarrollador no está obligado a conocer lo que cada uno de los componentes de la biblioteca hace, por lo menos debe tener una cierta idea de que algo parecido a lo que necesita está efectivamente dentro de la biblioteca.

Es muy probable que este conocimiento se complete en un universo de 200-400 componentes y una certeza de que se olvida si el numero es mayor.

Es característico de las bibliotecas muy usadas que sus componentes sean pequeños. Estos componentes tienen funciones muy específicas que cumplen con requerimientos muy puntuales. Los componentes multipropósito y muy grandes parecen no tener éxito debido al tiempo que se les debe invertir en analizarlos y determinar qué se puede utilizar de ellos, amén de que con frecuencia causan efectos secundarios no deseados.

Técnicas de desarrollo Orientadas a Objetos

Una de las características que prometen las metodologías de desarrollo orientadas a objetos, es el alto nivel de reusabilidad que tienen.

Además de los beneficios de encapsulamiento de datos y funciones integradas como un todo, las técnicas orientadas a objetos utilizan la *herencia* para facilitar el reuso: un objeto de nueva creación se puede definir como una *subclase* de un objeto ya existente y automáticamente heredar los atributos y funciones (o métodos) del componente a partir del cual es creado (padre).

La característica de *herencia* es particularmente útil cuando se requiere crear un módulo con funcionalidad adicional a la que posee uno ya existente. Las funciones provistas por el objeto en la biblioteca no son reimplementadas sino reutilizadas en una nueva implementación.

Lenguajes como C++, Eiffel y Smalltalk incorporan este concepto.

La *herencia múltiple* permite a varios objetos actuar como base para la creación de otro. Así, las características de varios objetos se combinan en el nuevo.

Por ejemplo, digamos que existe la clase AUTO que encapsula información relacionada con automóviles y la clase PERSONA que contiene información de gente. Ambas pueden usarse para una nueva clase de objetos llamada PROPIETARIO-AUTO que combine los atributos de AUTO y PERSONA.

Cuando un lenguaje de programación soporta la herencia, se autoconstruye una especie de telaraña para definir la nueva clase (Figura 2-14). Esta red muestra las clases superiores para cada clase dentro del sistema.

Mientras más clases se reutilicen, la red se volverá mas y más compleja.

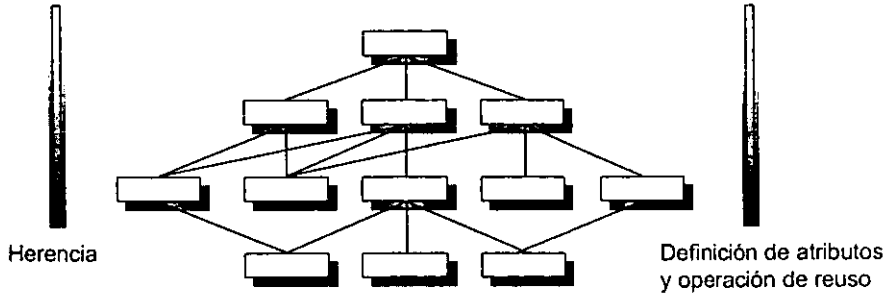


Figura 2-14 La red de clases

La herencia es un mecanismo útil para aplicar la reusabilidad de una manera práctica. Sin embargo, la red de subclases puede volverse muy compleja por la inclusión de nuevos componentes, mismos que progresivamente se harán más difíciles de comprender.

Con la herencia, el código de un componente no es ubicado en un solo lugar, sino esparcido en la red de clases lo que obliga a la persona que reusa, a examinar un cierto número de clases para que el componente que le interesa sea completamente entendido. Como consecuencia, se corre el riesgo de reutilizar componentes muy robustos, para aplicar de ellos sólo una pequeña parte.

La herencia no representa una respuesta inmediata a la reusabilidad, pero juega un papel muy respetable para soportar el desarrollo mediante el reuso.

Herramientas CASE y Designware

El repositorio de componentes indispensable en las herramientas CASE puede no sólo incluir porciones de código sino también *programas aplicativos reusables.*

El mercado de herramientas de software se ha visto poblado de repositorios de este tipo. Al establecimiento y uso de este tipo de repositorios se le conoce como *designware.*

Esta variante del reuso representa una oportunidad atractiva en la industria del software para una aplicación más agresiva del reuso. Una aplicación puede ser personalizada a través no de parámetros sino de adecuación de módulos (componentes reusables) para dar una solución particular.

Por ejemplo, considérese un sistema común —un sistema de nómina—. Por lo general es más barato comprar un paquete comercial que desarrollar uno propio. Pero para ajustar el paquete a las necesidades propias de mi empresa, el vendedor garantiza cierto nivel de parametrización que no me es suficiente. Entonces debo acceder al código y modificarlo para lograr mi objetivo.

¿Porqué en lugar de comprar un paquete y su código no se compra un repositorio que contenga datos, diagramas de flujo, diagramas de relación y entidad y especificaciones?

Es muy probable que esta opción cumpla con el 90% de las especificaciones y diseño que necesito y que el 10% restante lo pueda cubrir no en el nivel de programación, sino a nivel de análisis y diseño. Una vez que tome los componentes, mediante la herramienta CASE adjunta al producto, puedo generar la versión correcta de la aplicación que cubra mis expectativas.

El *designware* permite una reutilización que involucra componentes con un nivel de abstracción mayor (los generados en los niveles de análisis y diseño) y representa una opción muy atractiva para la creación de nuevas aplicaciones.

Recuperación de diseños

Para una empresa que se enfrenta al desafío del reuso, se le presenta un primer problema ¿Cómo poblar la biblioteca de componentes vacía? ¿De dónde tomar los componentes? ¿Cuánto tiempo llevará llenar la biblioteca?

Una solución recomendada es la de designar un grupo de trabajo exclusivamente dedicado a la creación de componentes de reuso. Este equipo será designado, sin embargo, *después* de que la empresa haya decidido seriamente implementar un esquema de reuso.

Sólo entonces y a partir de ese momento, se desarrollarán los componentes reusables. Se comienza entonces, desde cero.

Otra solución implica utilizar código existente en la organización, utilizar o desarrollar herramientas y técnicas para encontrar y extractar porciones de código importante contenido en el mar de aplicaciones ya implementadas e incluirlo sistemáticamente en la biblioteca institucional de componentes reusables.

Esta segunda opción, conocida como *recuperación de diseños*, ofrece la posibilidad de aprovechar la experiencia de la empresa contenida en sus sistemas en operación.

2.3.6.2 El acercamiento administrativo a la reusabilidad

Mientras que el aspecto tecnológico relacionado con la reusabilidad es obviamente importante, el ingrediente adicional, no menos importante para alcanzar altos niveles de reusabilidad en una organización, es la *administración.*

El compromiso de la administración para con la reusabilidad se resume en la Figura 2-15.

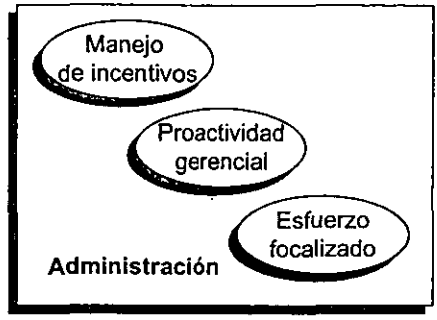


Figura 2-15 Visión administrativa de la reusabilidad

Establecimiento de mecanismos de incentivos

Como es de suponerse, los ingenieros de software no sentirán motivación para buscar oportunidades de reusar componentes, si no encuentran beneficio en tales acciones; la Dirección debe generar reconocimientos e incluso incentivos para promover una mayor participación.

Primero que nada, se le debe conceder igual peso al número de líneas de código utilizadas a partir de las bibliotecas de reuso, que el ofrecido a las líneas de código nuevas. Al ofrecer la misma calificación y dado que —se supone—, existen mecanismos eficientes de búsqueda de componentes de reuso, se encauza al personal en ir por la dirección correcta.

La implementación de campañas institucionales, y la capacitación continua en temas relacionados con el reuso de componentes, será también útil. Lo importante

es garantizar que los ingenieros de software no crean que tales acciones son sólo una moda en la organización que terminarán en la próxima crisis financiera. Por el contrario, debe haber la certeza de que es un esfuerzo a largo plazo y de que es parte integral de la cultura de la empresa.

En el mejor de los casos, el ingeniero de software se comenzará a sentir culpable siempre que escriba código nuevo, sin haberse preguntado antes si existe un módulo candidato a ser reusado en las bibliotecas institucionales.

Las compensaciones monetarias son también utilizadas como incentivo para reusar componentes. En las empresas donde opera este esquema, generalmente se reconoce a quienes utilizan con mayor frecuencia las bibliotecas de reuso, así como a quienes crean componentes nuevos que son aprovechados posteriormente.

La práctica de incentivos monetarios conlleva problemas de administración adicionales que pueden ir desde inconformidades personales que crean un ambiente de trabajo adverso, hasta problemas gremiales que deben ser resueltos legalmente.

En cualquier caso, lo esencial es incorporar la práctica del reuso de componentes a la cultura y a la mística de trabajo de la empresa. Si bien no es una tarea fácil, si representa la posibilidad de crear un entorno orientado a la calidad, garantizado primero que nada, por la participación e interés profesionales de los miembros de la organización.

Creación de una administración proactiva

Muchas empresas tienen una actitud pasiva hacia la reusabilidad: fuera de algunos exhortos o propagandas, a nadie le importa el concepto hasta que el proyecto termina. Es entonces cuando generalmente se evalúa el trabajo hecho y surgen preguntas por aclarar.

¿Qué nivel de reusabilidad se alcanzó? Si se alcanzó un nivel de 34%, esto sorprenderá sin duda al equipo de trabajo, quienes se voltearán a ver el sistema completo y se preguntarán ¿cómo lo hicimos?

En las organizaciones donde el reuso es premisa de trabajo, se establecen objetivos al respecto. Alcanzar un 50 o 60 por ciento de reusabilidad será una meta a alcanzar. Los niveles que se establezcan serán producto de un análisis serio y minucioso del comportamiento de la organización y su medición aplicará para *todos los proyectos* y se conocerán *desde el principio* de su desarrollo.

De esta manera, los equipos de desarrollo saben qué metas lograr y cómo alcanzarlas. Los niveles de reusabilidad representan entonces la meta, pero existe en la organización una infraestructura y una cultura de reuso sólidas que respaldan su consecución.

Las estimaciones de las metas de reusabilidad a alcanzar deben surgir de un programa global de métricas en la organización. Inicialmente se pueden revisar proyectos ya terminados y generar niveles a partir de ellos. La siguiente tarea es validar que los niveles establecidos no sean sólo un sueño. ¿Cómo se puede hacer ésta validación? Al respecto, Yourdon señala:

> *"(...) incorporando el estimado de reusabilidad dentro del presupuesto y plan del proyecto! Si el 80-85 por ciento del código del nuevo sistema puede ser obtenido de una biblioteca de reuso, entonces el tiempo requerido para diseñar, codificar y probar los nuevos módulos se debe reducir considerablemente, y el plan y presupuesto del proyecto deben reflejar esto."* [46]

Una actitud colaborativa en la organización que derive de la visión global de la Dirección cobrará frutos en el campo de la reusabilidad. Lo señalado por Yourdon es una práctica común en Japón; su aplicación junto con un ambiente general de calidad ha llevado al establecimiento de niveles requeridos de reusabilidad que no sólo se cumplen, sino que permanentemente se reflejan en los planes para la calidad.

---

46 *Ibid.*, p.231

Conformación de un grupo específico para el reuso

Como se ha expuesto, no es razonable suponer que los desarrolladores tendrán el tiempo, energía o capacidad para crear componentes de reuso generales, mientras se encuentren desarrollando un proyecto específico bajo presiones de presupuesto y calendario.

En este caso, la aplicación del reuso implica la existencia de componentes creados inicialmente para un fin específico, y en el mejor de los casos, revisados para hacerlos de propósito general, esto es, arrancar con una figura de rescate de software.

Un mejor esquema —que es el único capaz de crear una biblioteca estable con componentes útiles— es destinar un grupo de trabajo separado, cuya única misión sea la de crear los componentes reusables.

En una empresa con 100 ingenieros de sistemas, se pueden destinar 2 o 3 personas para integrar el nuevo equipo que al menos debe ser de dos personas. Esta alineación permitirá cierta flexibilidad, respaldo de personal y un sentido de trabajo institucional en equipo.

El —o los— equipos dedicados a la creación de partes reciclables deben integrar algo así como un *Departamento de Partes de Software*, insertado en una estructura organizacional parecida a la mostrada en la Figura 2-16.

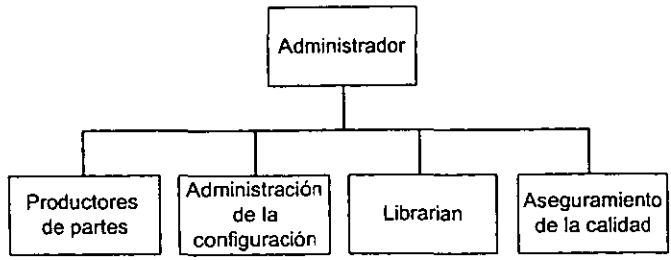


Figura 2-16 El reuso dentro de la estructura organizacional

El esquema no implica que el *Departamento de Partes se Software* se niegue a recibir componentes del resto de la organización, pero sí implica que debe revisar cuidadosamente los envíos y garantizar que sean candidatos para su inclusión en las bibliotecas. Debe también revisar que todos los atributos de calidad definidos, sean cumplidos por los componentes.

2.4 Confiabilidad del Software

La industria de la información, de incuestionable importancia en nuestros días y en el futuro próximo, se expande a pasos agigantados. Esta expansión se debe en parte al incremento acelerado de la relación costo-efectividad del hardware, cuyo incremento se calcula en 1000 por ciento por década.

Mientras esta tasa se mantenga, la cantidad de actividades controladas por computadora crecerá inevitablemente y con ella, la necesidad de crear software confiable y de gran calidad, entregado a tiempo y a un costo razonable.

La complejidad que los sistemas han alcanzado a través de los años, junto con el sentido de urgencia para su desarrollo, son el marco de la actividad actual. Esto provoca una problemática particular.

Existe le necesidad de hacer sistemas de calidad, altamente complejos y rápido. Los efectos de una falla en sistemas complejos pueden representar una catástrofe: ¿Qué pasaría si fallan sistemas de reservaciones de aerolíneas, bancos, hospitales o sistemas de defensa militar? Las consecuencias de una falla no sólo serían monetarios, sino de credibilidad en los sistemas.

En un apartado anterior se expuso la problemática de la métrica del software relacionada con la calidad. Calidad, costos y tiempo con elementos presentes en todo proyecto. Los dos últimos son cuantificables, mientras que la calidad es difícil de medir.

Un factor fuertemente relacionado con la determinación de calidad en los sistemas y cuya ausencia determina en gran medida una calidad pobre, es la

confiabilidad del sistema. Este concepto tiene que ver *con qué tan bien funciona el software* para cumplir los requerimientos del usuario. Tiene que ver con las fallas —un requerimiento del usuario no cumplido por alguna razón— que presenta el sistema en operación.

La confiabilidad no representa el número de errores o reparaciones hechas al sistema *antes* de su liberación. Su naturaleza no es estática sino *dinámica*, y no tiene que ver con el sistema en las fases de desarrollo, sino una vez que se encuentra funcionando. Su objetivo es manejar la frecuencia con que las fallas causan problemas.

2.4.1 Términos básicos y conceptos

Una *falla del software* es la desviación de los resultados externos de un programa, con respecto a sus especificaciones. El programa debe estar en ejecución para que una falla tenga lugar.

En este contexto una falla no debe confundirse con un *bug* o con un error de programa. Existe gran variedad de fallas, desde el cese momentáneo en el servicio de un sistema o las pequeñas discrepancias en la operación, hasta los excesivos tiempos de respuesta.

Un *error* es un defecto en el programa que cuando se ejecuta bajo ciertas condiciones produce una falla. Los errores pueden ser la causa de múltiples fallas dependiendo de los datos de entrada que se utilicen. Un error es más una propiedad del programa, que de la ejecución del mismo y es introducido durante las fases de diseño o programación.

Los cuantificadores de la confiabilidad están definidos en función del tiempo, aunque se pueden usar otras variables. El factor tiempo se divide en:

- *Tiempo de ejecución*. Para un programa, es el tiempo requerido por el procesador para ejecutar las instrucciones del programa.
- *Calendario*. Es el horario habitual al que estamos acostumbrados.

- *Tiempo de reloj.* Se usa ocasionalmente y representa el tiempo transcurrido entre el inicio y el final de la ejecución de un programa en la computadora.

Existen cuatro formas generales de caracterizar la ocurrencia de fallas a lo largo del tiempo:

a) Tiempo de falla[47]. Es el tiempo esperado para que una falla ocurra. Normalmente se establece con base en sondeos de producción. Así por ejemplo, se puede determinar que un foco durará 1000 horas encendido, antes de que se funda.

b) Intervalo de tiempo entre fallas[48]. Es el tiempo promedio esperado entre dos fallas del sistema.

c) Número de fallas acumuladas hasta un momento dado.

d) Fallas ocurridas en un intervalo específico de tiempo.

Las variables anteriores son *random o aleatorias*[49], y sus valores no son conocidos con certeza. Dado que existe un rango de valores posibles, cada rubro se asocia con una probabilidad de ocurrencia. La ocurrencia de una falla puede verse influenciada por una serie de factores y su aparición en el sentido aleatorio, no es producto de la distribución uniforme de una función.

Existen dos razones de peso para determinar las variables de tiempo como aleatorias. Primero, la introducción de errores por los programadores representa un proceso complejo e impredecible. Segundo, la función a ser ejecutada por el sistema generalmente es impredecible. Por ejemplo, un sistema de switcheo telefónico ¿cómo se podría saber qué tipo de llamada será la siguiente que se reciba?

---

47 Se conoce también como MTTF (*mean time to failure*).

48 MTBF ó *mean time before failure*.

49 El término *random* no tiene la connotación de irracional o impredecible. Significa "impredecible" sólo en el sentido de que su valor exacto no se puede conocer.

Un *proceso aleatorio* se puede ver como un conjunto de variables aleatorias, cada una correspondientes a un punto en el tiempo. Los procesos pueden ser *discretos* o *continuos* dependiendo de la forma en que se considere el tiempo.

Las variaciones pueden verse como *funciones de valor promedio*, o *funciones de intensidad de falla* que representan las fallas acumuladas en cada punto del tiempo.

Los principales factores que afectan las fallas son:

- *El número de errores contenidos en el software que se está ejecutando.* Los errores adicionados a los programas generan fallas. Una rama de la confiabilidad del software se preocupa de remover los errores y adecuar los índices de falla.
- *El ambiente de operación del sistema.* Todos los elementos que hacen posible la operación del software —sistemas operativos, redes de cableado, interacción con otros sistemas, componentes analógicos, etc.— pueden incidir en la presencia de una falla. Tales ambientes pueden incluso —para un mismo servicio— ofrecer niveles de confiabilidad diferentes. Por ejemplo, una compañía de teléfonos puede tener un nivel de confiabilidad más alto en una área urbana que en una rural y ofrecer niveles de servicio correspondientes a esos niveles.

2.4.2 Perfil de operación

Para entender el concepto de *perfil de operación* —o ambiente de operación—, se requieren algunos conceptos secundarios. La *ejecución de un programa* generalmente se ve como el conjunto de varias *corridas* del mismo, en las que cada una se asocia con distintas funciones del programa.

Las corridas que son ejecuciones idénticas del programa son *corridas tipo*, entre las que se encuentran las transacciones de abono a una cuenta de cheques, o la impresión de un documento en un procesador, por ejemplo.

Otro concepto necesario es el de *variable de entrada*. Esta es una variable que existe fuera del sistema, pero que puede ser utilizada por él, por ejemplo, el nombre de una emisora dentro de un sistema de bolsa. Un conjunto de valores que toma la variable de entrada forma una *entrada de estado*, que determina la corrida tipo que será usada.

En un sistema de bolsa, una entrada de estado puede ser el total de datos que identifican al corredor de bolsa, al comprador, al vendedor, etc. El conjunto de los posibles valores que pueden tomar las variables de estado se conoce como *espacio de la entrada* (Figura 2-17).

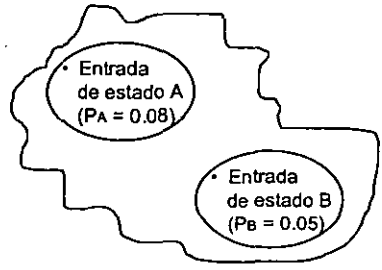


Figura 2-17 El espacio de entradas

Similarmente, *una variable de salida* es una variable que existe externa al programa y que es establecida por éste. Una *salida de estado* es un conjunto de valores para todas las variables de salida asociadas con la corrida de un programa. En el ejemplo, las salidas de estado pueden ser confirmaciones de compra-venta o los estados de cuenta mensuales.

El ***perfil de operación*** se define como el conjunto de corridas tipo que el programa puede ejecutar dentro de sus probabilidades de ocurrencia. La Figura 2-17 muestra dos posibles entradas de estado —A y B—, con sus probabilidades de ocurrencia. La parte del perfil operacional sólo para estos dos estados se muestra en la Figura 2-18 . Un esquema realista para el perfil de operación se muestra en la Figura 2-19.

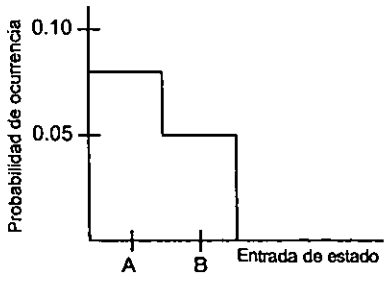


Figura 2-18 Porción del perfil operacional *

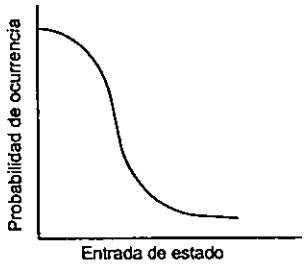


Figura 2-19 Perfil operacional

Los entradas de estado han sido arregladas sobre el eje horizontal en orden de probabilidades de ocurrencia. Con el gran número de entradas de estado que ocurren en la realidad, el perfil de operación parece representar una curva continua.

Aunque el concepto de perfil de operación es muy importante, puede no ser muy práctico por la dificultad de enumerar todas las entradas de estado y sus probabilidades a detalle. Algunas investigaciones proponen utilizar *particiones*, que son grupos de entradas de estado con características similares cuya probabilidad se considera igual para todos sus elementos. Esta convención simplifica el esquema y acelera el proceso de pruebas.

2.4.3 La confiabilidad y el entorno

Un ambiente confiable en el que se opere un sistema depende de muchos factores, Ya que la confiabilidad del software está estrechamente relacionada con las fallas que éste presenta durante su operación, las fallas pueden provenir tanto de los programas mismos como del equipo en que corren los programas. Los siguientes apartados exponen la relación que un entorno de confiabilidad guarda con el software y con el hardware.

* Tomado de Musa & Iannino, (1990) Software Reliability. EU: Academic Press, p.97

2.4.3.1 El software

La *confiabilidad del software* es la probabilidad de operar sin fallas un programa de computadora en un periodo de tiempo y en un ambiente específico (perfil de operación).

Por ejemplo, un sistema de tiempo compartido puede tener una confiabilidad de 0.92 para 8 horas de CPU, siendo operado por un usuario habitual. Cuando se ejecuta el sistema, se espera que opere sin fallas 92 de 100 de tales periodos de tiempo. Adicional al concepto de falla, el concepto de confiabilidad del software puede incluir que el *performance*[50] sea satisfactorio.

Para tener una correcta medición de la confiabilidad durante las pruebas, se deben elegir las corridas aleatoriamente y con la misma probabilidad, tal como se espera que ocurra en la operación real.

Se pueden elegir casos de prueba que ataquen una parte de la funcionalidad del software, con el objetivo de remover errores más rápido. Ya que la curva de intensidad de falla reflejará la parte del programa examinada, ésta decrecerá al inicio más rápidamente de lo que se esperaría. Cuando el programa toque otra parte de código, la curva probablemente crecerá debido al número de errores aún no encontrados.

La definición implica que el espacio de entradas (casos de pruebas) debe estar *bien cubierto*. Para medir la cobertura del espacio de entradas se deben asociar probabilidades a las corridas de prueba seleccionadas.

Al principio, la cobertura suele ser pequeña si el espacio de entradas es muy grande. La suma da la probabilidad de que ocurra cierta corrida que ya haya sido probada. Así, cualquier error en la medida de la confiabilidad imputable a la falta de cobertura, debe estar entre 1 y la cobertura.

---

50 El término *performance* se utiliza para determinar el tiempo de respuesta de un sistema. Responde a la pregunta ¿Cuánto tarda un sistema en contestarme una vez que introduje los datos necesarios?

La *intensidad de falla* es una alternativa para expresar la confiabilidad. En el caso anterior de un sistema de 0.92 para 8 horas de CPU es equivalente a una intensidad de falla de 0.01 por hora de CPU. Así,

$R(\tau)=\exp(-\lambda\tau)$

donde,

R es la confiabilidad

λ es la intensidad de falla

τ es el tiempo de ejecución

Cada medida —confiabilidad y de la intensidad de falla— tiene sus ventajas. La intensidad de falla es más económica pues sólo se requiere un número para su cálculo. La medida de confiabilidad es más conveniente cuando se habla de la combinación de confiabilidades de los componentes, para obtener la confiabilidad de un sistema.

Si existe un gran riesgo de falla en un sistema —tal como una planta nuclear— la intensidad de falla puede ser la medida apropiada. Cuando se trate de la operación apropiada de un sistema para cumplir una función en un tiempo dado —tal como un vuelo de avión—, la medida de confiabilidad será la mejor opción.

La Figura 2-20 muestra como la intensidad de falla y la confiabilidad varían durante un periodo de prueba según los errores sean removidos. La relación entre ambas depende del modelo de confiabilidad empleado si los valores van cambiando.

Cuando los errores son removidos, la intensidad de falla tiende a decrecer mientras que la confiabilidad tiende a aumentar. Cuando se introducen errores normalmente no se reflejan como cambios al diseño, provocando un salto en la intensidad de falla y una caída en la confiabilidad.

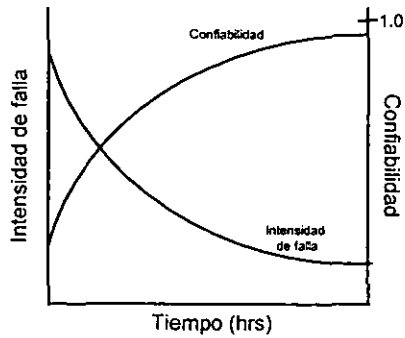


Figura 2-20 Confiabilidad e intensidad de falla *

2.4.3.2 El hardware

Ya que la confiabilidad en el equipo de cómputo es una práctica bastante expandida, resulta lógico preguntarse cómo es que se relaciona con la confiabilidad del software. De acuerdo con Musa y Iannino[51] esta división resulta un tanto artificial ya que la confiabilidad debería verse como una sola.

Ambas dependen del entorno. La fuente de los errores de las fallas del software generalmente es el diseño, mientras que las principales fallas del hardware se refieren al deterioro del equipo. Las teorías de la confiabilidad de software son aplicables a cualquier actividad de diseño, incluido el diseño de hardware.

La confiabilidad en el diseño es un concepto que históricamente no se ha puesto en práctica en el hardware, porque se ha considerado que el índice de fallas se debe con mayor frecuencia a razones físicas que a problemas de diseño desconocidos. Tradicionalmente resultó más fácil restar importancia a las fallas de diseño, porque el hardware resultaba lógicamente menos complejo que el software.

---

* Tomado de Musa & Iannino (1990) p. 99

51 Musa, J.D., *et al.* (1990) Advances in computers, vol 30. Academic Press, EU.

Adicionalmente, la corrección de errores de diseño mientras la producción de hardware se generaba, resultaba ser muy costosa. El énfasis hacia la confiabilidad del hardware ha cambiado en nuestros días. Este interés se ha reforzado por el desarrollo en paralelo cada vez más frecuente, de software y diseño de chips o precesadores.

Una característica adicional es que la confiabilidad del software tiende a variar mientras éste se desarrolla debido a que los cambios en el diseño o corrección de errores le afectan directamente. La confiabilidad del hardware puede variar por periodos de tiempo, comúnmente al inicio de su operación o al final de su vida útil, pero en general se mantiene en un valor más constante que en el software.

El término *tiempo promedio para fallar*, es usado en relación al hardware para medir el tiempo promedio en que la siguiente falla aparezca. Llevado al software es un término atractivo porque *más largo* significa *mejor.*

Sin embargo, el tiempo promedio para fallar puede resultar en una indefinición para muchos aspectos del software. La intensidad de falla es preferible porque siempre existe.

A pesar de las diferencias expuestas, la teoría de la confiabilidad en el software ha sido desarrollada de cierta forma compatible con la del hardware.

2.4.4 Prevención de errores

Todos los ingenieros de software tienen como objetivo producir programas libres de errores. Un proceso de desarrollo basado en el descubrimiento y modificación de errores más que en la prevención de los mismos, resulta en un proceso pobre y poco recomendable.

En este contexto, el software libre de errores es aquél que cumple con las especificaciones. Por supuesto pueden existir errores en las especificaciones que no reflejen las necesidades del usuario así que un software libre de errores no necesariamente significa que el software siempre se comportará anticipadamente como el usuario requiere.

Desarrollar software libre de errores es costoso y mientras los errores se eliminen de los programas, el costo de encontrar y remover los errores restantes tiende a elevarse exponencialmente (Figura 2-21). Una organización puede decidir que cierto nivel de errores residuales es aceptable; puede ser más costoso pagar las consecuencias de estos errores, que buscarlos y corregirlos antes de liberar el producto.

Figura 2-21 Costo de eliminar errores residuales

Dado un sistema libre de errores, parecería no haber necesidad para incluir facilidades de tolerancia de errores. Sin embargo, debido a que puede haber errores de diseño (especificaciones), aún los sistemas libres de errores deben contar con un esquema de tolerancia de errores que garantice la confiabilidad.

La prevención de errores y el desarrollo de software libre de errores dependen de:

a) El establecimiento de una precisa (preferentemente formal) especificación del sistema.

b) La adopción de alguna metodología de diseño de software basada en encapsulamiento.

c) El uso extenso de revisiones en el proceso de desarrollo, que a la vez validan al sistema.

d) La adopción de una filosofía organizacional donde la calidad sea el eje del proceso de desarrollo.

e) La planeación cuidadosa en las pruebas del sistema para exponer errores que no hayan sido descubiertos durante las revisiones y para establecer la confiabilidad del sistema.

Es esencial que un lenguaje de programación de alto nivel con una escritura impecable, sean introducidos al desarrollo del sistema. Desarrollar un software libre de errores es virtualmente imposible si se utilizan lenguajes de bajo nivel con poca validación de su código.

2.4.4.1 Programación estructurada

La *programación estructurada* es un término acuñado al final de los años 60, que expone un estilo de programar sin sentencias GO TO y utilizando en su lugar sólo sentencias WHILE o IF. La adopción de este estilo fue importante porque marcó el inicio de una manera disciplinada de programar.

Con este estilo, los programadores se ven obligados a pensar detenidamente acerca del programa que desarrollan, reduciendo la posibilidad de error durante el desarrollo. La programación estructurada implica programas que son fáciles de leer y por lo tanto de inspeccionar y de entender. Sin embargo, evitar sentencias riesgosas es sólo el primer esfuerzo para la confiabilidad.

Uno de los promotores de errores es el uso de la sentencia GO TO porque puede romper la secuencia del programa, pero existen otros que no son menos riesgosos:

- *Números de punto flotante.* Este tipo de números son imprecisos por diseño y pueden conducir a problemas, especialmente al hacer comparaciones que pueden resultar incorrectas.
- *Punteros.* Son construcciones de bajo nivel que permiten referenciar directamente áreas específicas de memoria. Su uso puede causar efectos devastadores porque pueden tener varios *alias* —diferentes nombres— y

hacer referencia a la misma área: los programas se vuelven difíciles de entender.

- *Paralelismo.* El riesgo de correr procesos paralelos puede incluso no ser evidente durante las pruebas y parece inevitable. Sin embargo, una determinación oportuna de la interdependencia de procesos será útil al evitar errores.
- *Recursión.* La recursión se presenta cuando una subrutina se llama a sí misma o ejecuta otra que contiene un llamado a la subrutina que la llamó inicialmente. Su uso puede conducir al desarrollo de programas bastante concisos pero por lo general son difíciles de entender porque su lógica suele ser compleja.
- *Interrupciones.* Una interrupción implica el paso forzado del control de un programa a alguna sección del mismo sin importar lo que se esté ejecutando. Los riesgos son obvios pues pueden causar que una tarea crítica termine anormalmente su operación.

Es obvio que las prácticas anteriores son útiles y que no pueden ser prohibidas determinantemente, pero su uso debe ser cuidadosamente aplicado por los programadores.

2.4.4.2 Clasificación de datos

Un principio de seguridad adoptado por organizaciones militares es el principio de *conocer lo necesario.* Sólo los individuos que requieren conocer cierta pieza de información para su trabajo lo pueden hacer. La información que no les es relevante es retenida y se considera como *información clasificada.*

En la programación de un sistema se puede aplicar un principio análogo para controlar el acceso a los datos. A cada programa o componente se le debería permitir accesar sólo los datos que requiere para completar su función.

La práctica de ocultar información garantiza que la misma no sea corrompida por programas que se suponen no deben utilizarla. La representación de los datos se puede cambiar sin cambiar los componentes que la usan.

La clave para clasificar o tipificar datos es el uso de todo el potencial del lenguaje de programación utilizado. En menor o mayor grado, los lenguajes actuales permiten la implementación de esta facilidad cuyo uso hará más claros los programas y limitará la modificación indeseada de datos.

Para cada tipo de datos existente en el mundo real, debe existir un tipo de datos en el programa. Por ejemplo, considérese un sistema de control de semáforos escrito en ADA. Las siguientes instrucciones pueden ser codificadas:

type ColorDeLuz **is** (rojo, amarillo, verdeintermitente, verde);

ColorActual, ColorSiguiente : ColorDeLuz;

Los objetos ColorActual y ColorSiguiente que modelarán el cambio de luces en los semáforos sólo podrán tener los valores rojo, amarillo, verdeintermitente, o verde.

Cuando se trate no de valores escalares sino de valores dentro de un rango se puede utilizar como:

type NumeroPositivo **is** Integer **range** 1..MAXINT;

Asi NumeroPositivo podrá tomar valores entre 1 y el numero entero más grande que soporte el equipo.

El uso de técnicas como éstas que delimitan los valores que pueden tomar las variables, Asi como la aplicación de las mismas en técnicas más avanzadas como el empaquetamiento de tipos o la implementación de tipos abstractos de datos, crean un vínculo del programa con la situación real que representa, y por lo tanto, ofrece una visión lógica y establece límites de error menores.

2.4.5 Tolerancia de errores

Aunque un sistema esté libre de errores, se deben incluir facilidades para la tolerancia de errores cuando el sistema se instale en un ambiente donde se requiera un nivel alto de confiabilidad.

La razón de lo anterior es que *libre de fallas* significa sólo que el programa cumple sus especificaciones; pero éstas pueden incluir a su vez errores u omisiones y pueden estar basadas en apreciaciones erróneas del ambiente.

Un sistema con tolerancia a errores debe incluir las siguientes actividades:

a) *Detección de errores.* El sistema debe detectar que un estado en particular o que una combinación de estados puede resultar en una falla.

b) *Reconocimiento de daños.* Las partes del sistema que hayan sido afectadas por una falla deben poder ser detectadas.

c) *Recuperación de errores.* El sistema debe ser capaz de restaurarse a un estado "seguro". Esto puede hacerse corrigiendo el estado dañado (recuperación adelantada) o restaurando el sistema al estado previo antes de la falla (recuperación invertida).

d) *Reparación de errores.* Significa modificar el sistema para que el error no reaparezca. En muchos casos las fallas del software son transitorias y se deben a una combinación particular de entradas. En tales casos no es necesaria una reparación pero sí una recuperación inmediata de la falla.

Cuando una falla no es transitoria, su reparación requiere un sistema de reconfiguración dinámica donde el componente reemplazado sea sustituido por el erróneo sin detener el sistema, o requiere que el sistema sea suspendido para corregir el error.

Existe una técnica de tolerancia a errores aplicada al hardware conocida como TMR (*triple modular redundancy*) que consiste en replicar tres veces cada unidad de hardware. La salidas de cada unidad son entonces comparadas. Si una de las unidades falla y no produce el mismo resultado que las otras, su salida es ignorada y el sistema trabaja sólo con dos unidades (Figura 2-22).

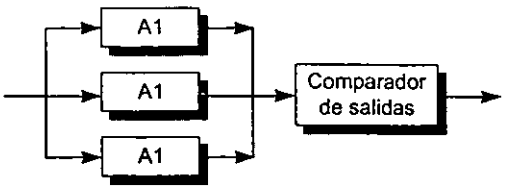


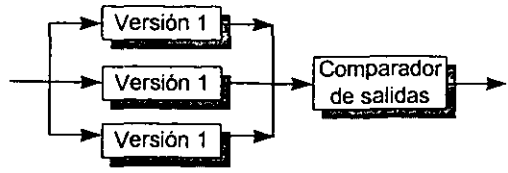
Figura 2-22 Redundancia triple modular

Esta visión de la tolerancia a errores se basa en que las fallas del hardware se deben más a fallas de sus componentes que a fallas de diseño. Hay una probabilidad muy baja de que dos componentes fallen simultáneamente. Existe una probabilidad asociada de que dos componentes contengan un error de diseño que cause la misma falla; esta probabilidad se reduce al introducir bajo este esquema, componentes hechos por diferentes fabricantes.

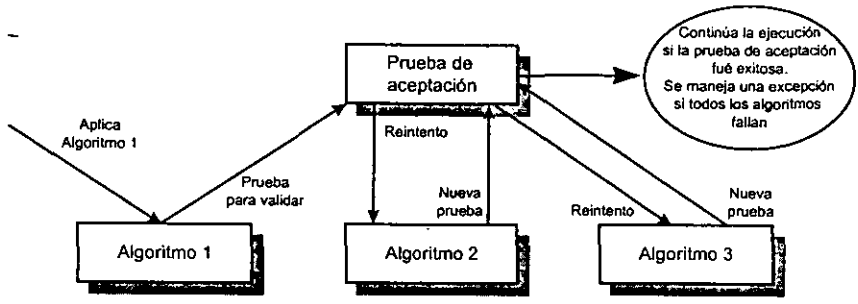
Aplicado a la tolerancia de errores en el software existen dos esquemas comparables a los usados en el hardware (Figura 2-23).

a) ***Programación de N-versiones.*** Bajo este esquema el sistema es implementado en cierto número de versiones desarrolladas por diferentes equipos de trabajo y son ejecutadas en paralelo. Las salidas de cada versión se comparan usando un sistema de calificación y las salidas inconsistentes son eliminadas. Se requieren al menos tres versiones del programa.

b) ***Recuperación de bloques.*** Este método de tolerancia de errores es más fino. En él existe un programa que verifica la falla y existen adicionalmente otros programas de respaldo que le permiten al sistema repetir el proceso y manejar la falla. A diferencia del modelo anterior, las ejecuciones de los diferentes programas no se hacen en paralelo, sino en serie, permitiendo la aparición de fallas en cada etapa. Una falla que ocurra simultáneamente en cada módulo del esquema de N-versiones y que por tanto sea aceptada, es rechazada por la recuperación de bloques.



a) Programación basada en N-versiones



b) Recuperación de bloques

Figura 2-23 Tolerancia a fallas en el software

El establecimiento de un software con tolerancia de errores requiere de un sistema de control que se asegura de ejecutar los pasos necesarios para tolerar los errores.

El esquema más usado para la tolerancia de errores es el N-versiones. Sin embargo, recientes investigaciones han mostrado que el asumir la baja probabilidad de que diferentes equipos de desarrollo incurran en los mismos errores, puede no ser válida. De hecho el esquema no cubre la tolerancia a errores cuando hay errores de especificación o inconsistencias en el análisis.

Para cubrir ésta tolerancia a errores se debe asumir que existen errores en las especificaciones y se deben implementar técnicas de *programación defensiva*. Estas técnicas incluyen verificaciones del sistema después de que ha sido modificado, para asegurarse de que el cambio es consistente. Si existen

inconsistencias, la modificación se rechaza y el sistema sigue en el estado anterior.

2.4.6 Manejo de excepciones

Cuando un error de cualquier tipo o un evento inesperado ocurre durante la ejecución de un programa, hablamos de una *excepción*. Las excepciones pueden ser causadas por errores de software o de hardware. Cuando una excepción no ha sido previamente definida, el control se transfiere a un sistema o mecanismo para el manejo de excepciones. Si una excepción ha sido vislumbrada con anticipación, se incluye dentro del código una rutina para detectarla y manejarla[52].

La mayoría de los lenguajes de programación no incluyen facilidades para la detección y manejo de excepciones. Las facilidades que provee cada lenguaje deben ser utilizadas para construir dentro del software, las herramientas necesarias para detectar y resolver qué hacer dentro del programa cuando una excepción se presente.

Cuando un programa se está ejecutando de manera lineal, la aparición de una excepción puede ser manejada desviando la ejecución del programa hacia otra parte del código —una zona de seguridad—;pero cuando la excepción ocurre en una secuencia anidada de llamadas a procedimientos, no existen mecanismos seguros para pasar de un procedimiento a otro garantizando el manejo adecuado de la excepción.

Considérese un programa con cierto número de procedimientos anidados, donde el procedimiento A llama al B que a su vez invoca al C (Figura 2-24). Si ocurre una excepción durante la ejecución de C, puede ser tan seria que la ejecución de B no debe continuar en tales circunstancias. Es necesario entonces que B transmita la excepción a A para que tome una acción apropiada.

---

52 El manejo de excepciones permite al programa manejar un universo de errores posibles previamente tipificados. Su manejo implica que el control del programa no se perderá como producto de la aparición —en tiempo de ejecución— del error que causa la excepción.

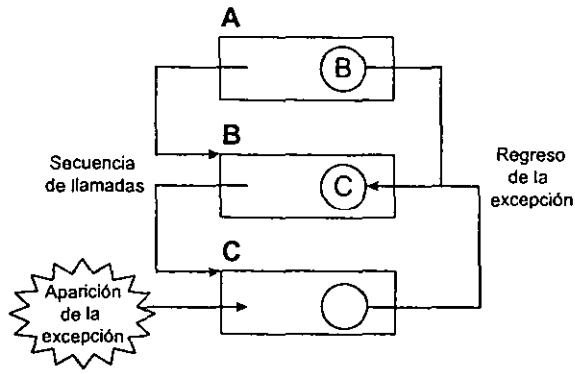


Figura 2-24 El regreso de una excepción en un llamado a procedimientos anidados

Por complicado que pueda ser y las restricciones técnicas en las que deba operar, el manejo de excepciones debe ser una práctica implícita de desarrollo. Su uso expandido al máximo posible será producto del análisis detallado del sistema y su manejo redundará en una confiabilidad mayor del sistema una vez liberado.

3

EL MANEJO DE LA CALIDAD DEL SOFTWARE

"Haz todo tan sencillo como te sea posible, pero nunca simple"

Albert Einstein

3.1 Metodologías de software

"Las empresas de software más competitivas a nivel mundial, no se distinguen por el nombre de la metodología que utilizan" [53].

Este hecho crea un nuevo paradigma en el área de las metodologías para el desarrollo de software, donde todavía hay mucho por conocer y evolucionar: refinación de técnicas estructuradas, avances sorprendentes en la ingeniería de la información, el *boom* del las técnicas orientadas a objetos.

53 Yourdon, E. *op. cit.*, p. 92

Sin embargo, las organizaciones que mantienen el uso estricto de metodologías tradicionales superan a aquéllas que utilizan, casi siempre de manera desorganizada y ambigua, las más modernas metodologías.

Las empresas de software pueden ser clasificadas, según su propia evolución en el desarrollo de software, en niveles (ver ANEXO B). Una empresa que se encuentra en el nivel 3 de evolución se encuentra *definida*. Ha definido procesos; con ellos asegura una correcta implementación y provee bases para el entendimiento general. La tecnología en este nivel es aprovechada útilmente.

¿Qué tienen que ver las metodologías con las empresas que se encuentran en este nivel intermedio de evolución? ¿Cuál es el cometido de las metodologías en la presente década, y en relación con la calidad?

Una razón que hace importantes a las metodologías en nuestros días es que rápidamente se están volviendo automatizadas. Las herramientas CASE han venido a sustituir a los tomos de estándares y manuales que conformaban las metodologías precedentes.

Las metodologías evolucionan, cambian de acuerdo a un numero de factores como el grado de evolución de las empresas que la usan, el nivel de compromiso que haya en su uso o incluso el tipo de software que desarrollan. Sin embargo, las metodologías no son una solución para la calidad. Como Yourdon señala:

> *"(...) las herramientas CASE típicamente automatizan metodologías* viejas — —*esto es, metodologías que anteceden a las herramientas por diez años o más— así que esperaremos ver un cambio en las metodologías mismas en los próximos años para poder aprovechar las capacidades de las herramientas."* [54]

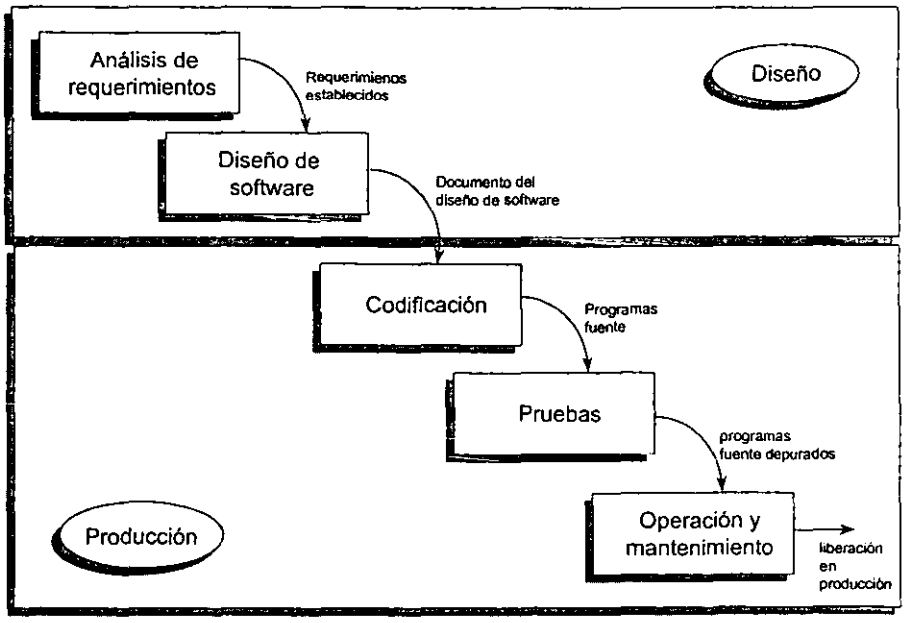
El panorama de las metodologías debe verse como un proceso evolutivo más que como una materia terminada. En la historia de éstas se puede contar una variedad que va desde el tradicional modelo de cascada, hasta las emergentes metodologías orientadas a objetos. Ellas suelen plantear un cambio de paradigma: cambiar las metodologías, evolucionarlas dentro de la organización, acorde a su propia historia.

3.1.1 Definición: Ciclo de vida en cascada

Desarrollado en el paso de la década de los 60 a los 70, el ciclo de vida en cascada al final de los 90 el esquema de desarrollo de software más utilizado.

Dentro del proceso de desarrollo de software existe una ambigüedad entre las técnicas para el desarrollo y para la producción. Específicamente dentro del área de la calidad del software, la necesidad de dar una solución que cubra las necesidades del usuario es considerada frecuentemente como *calidad del diseño*, mientras que asegurarse de que las especificaciones se cumplan se conoce como *calidad de la manufactura.*

Esta ambigüedad la disuelve el modelo clásico de desarrollo de sistemas diferenciando diseño y producción en etapas diferentes del proceso (Figura 3-1).



El ciclo de vida en cascada

Figura 3-1 El ciclo de vida en cascada

54 *Ibid.*

Las fases de implementación y pruebas están estrechamente relacionadas a la manufactura[55] del software. En la perspectiva del proceso de producción, uno de los objetivos es asegurar que el producto cumpla con las especificaciones de diseño. Más adelante las especificaciones de diseño serán la referencia para apoyar la conformidad del trabajo y mostrar el cumplimiento de la calidad.

El punto de referencia en que se convierten las especificaciones, no tiene su similar en la fase de diseño. Ni el análisis de requerimientos ni el diseño tienen punto de comparación, por lo que la calidad del diseño es más difícil de estimar.

Existen trabajos serios centrados en mejorar la implementación de software a partir de especificaciones de diseño preestablecidas. Esto puede ser ineficaz cuando las etapas críticas para determinar la calidad global pertenecen a la fase de diseño. La división rígida de *diseño* y *producción*, útil para los desarrolladores, puede establecer restricciones en la asertividad con la que el producto final cumple los requerimientos del usuario.

3.1.2 Métodos y metodología

Los términos método, metodología y el ciclo de vida de sistemas suelen utilizarse indistintamente, aunque en las empresas donde se ocupen es frecuente encontrar los manuales respectivos por separado.

Algunas definiciones para cada uno de ellos se presenta a continuación.

Metodología: un plan general detallado paso a paso para lograr algún resultado deseado. Una metodología de software generalmente identifica las actividades globales —por ejemplo análisis, diseño, codificación y pruebas— a ser desarrolladas e indica que personas (usuarios, líderes, analistas) deben realizar qué tarea y las funciones que deben desempeñar.

Las metodologías con frecuencia describen criterios de:

---

55 La visión de *fábrica de software* originada en Japón, tiende a expandirse en los 90 y siglo venidero. De ella se toma el concepto *manufactura* en el sentido de la escritura y validación del código.

- *Entrada.* Por ejemplo, las condiciones que se deban satisfacer antes de comenzar la fase de diseño.
- *Salida.* Son muy utilizados documentos en cada fase, que respaldan la ejecución de las tareas que conforman al proyecto global. Por ejemplo, sin los documentos de diseño, no se puede considerar terminada esa fase.
- *Puntos de revisión (checkpoints).* Ayudan a tomar decisiones en proyectos de desarrollo. Por ejemplo, es común detenerse, después de la fase de análisis, a evaluar con los usuarios si es viable continuar con el proyecto.

Ciclo de vida: Sinónimo de metodología

Método: una ayuda técnica, paso a paso para ejecutar una o más de las actividades identificadas en la metodología completa. Así, al análisis estructurado, es un método para trabajar la fase de análisis del proyecto, de la misma forma en que el diseño orientado a objetos es un método para desarrollar la fase de diseño.

En muchos de los casos, métodos y metodologías van de la mano, especialmente en empresas donde las metodologías se desarrollan de acuerdo a necesidades propias, o bien, cuando ambas forman parte de la misma herramienta CASE.

Al respecto, es común adquirir herramientas CASE que se proclaman como "independientes de metodología", pero esto no es posible ya que tales herramientas forman parte de la metodología misma.

> *(...) el ciclo de vida en cascada, el abuelo de todas las metodologías, es en si mismo,* completamente *independiente de los métodos; éste puede ser usado con métodos de ingeniería de información, métodos de análisis y diseño estructurados ó métodos de análisis y diseño orientado a objetos."* [56]

Asi, por ejemplo, es posible utilizar un ambiente orientado a objetos dentro de la metodología de desarrollo en cascada. El hecho de plantear su imposibilidad radica en que tales ambientes exigen una metodología que permita

56 *Ibid.* p. 94

implementación de prototipos rápidamente, cubriendo incluso varias fases del proyecto.

3.1.3 Problemática del desarrollo en cascada

A pesar del uso bastante expandido del ciclo de vida en cascada, vale la pena mencionar algunos problemas y debilidades que conlleva.

- ***Se basa por completo en papel.*** Debido al desarrollo histórico que ha presentado, el modelo en cascada se basa en formas impresas, documentos, diagramas en papel. Hoy es impensable el uso de diagramas de flujo con la importancia que se les dio en las primeras etapas de la programación.

  Los documentos escritos son mejores que no tener nada; sin embargo, un documento *simple*, sin una herramienta que lo apoye, no puede ser validado para evitar errores, ni tampoco puede ser transformado automáticamente en código ejecutable.

- ***Hay que esperar tiempo para ver resultados.*** Nada es ejecutable o demostrable hasta que se escribe el código. Los resultados que todos esperan ver de un desarrollo de software tiene la forma de código fuente depurado y corriendo; para alcanzar este punto se debieron seguir varios pasos anteriores.

  Se ha entendido que el modelo de cascada es secuencial, lo que implica que deben pasar meses o incluso años antes de apreciar un avance real.

- ***Depende de requerimientos correctos y estables***. En el proceso de desarrollo de software, la calidad de una fase depende en gran medida de la calidad de las fases anteriores. De esta manera, "si los requerimientos de usuarios han sido malinterpretados o malentendidos, o si los usuarios cambian sus requerimientos originales durante subsecuentes fases de diseño o implementación, el ciclo de vida en cascada puede producir una brillante solución a el problema equivocado" (Yourdon, 1992).

  Por desgracia, esto no es fácil de detectar sino hasta que el código está listo; la aplicación de esta metodología no garantiza que los requerimientos sean

validados en fases posteriores; aún peor, gran parte de los errores en los requerimientos, conscientemente, se postergan como actividades de mantenimiento.

- ***Es difícil traducir los requerimientos al código de los programas.*** La misma gráfica del modelo de cascada ayuda a explicar la razón. La información que pasa del analista al diseñador puede estar distorsionada y así la pasará al programador quien, con afán de dar los mejores resultados, puede nuevamente modificar el mensaje original.

  El *teléfono descompuesto* es lo que se juega de este modo. Especialmente en proyectos grandes es vital asegurarse de que las especificaciones están completamente reflejadas en el código; esta actividad es extremadamente difícil con el ciclo de vida en cascada.

- ***Se deja la detección de errores hasta el final.*** La detección de errores en el esquema de cascada, se reserva para la fase formal de pruebas del proyecto.

  Para entonces quizá sea demasiado tarde. Hay tanta presión para liberar a producción el software, que se hacen pruebas insuficientes. Además si se detectan errores de análisis y diseño, corregirlos será una tarea difícil y muy costosa (Figura 3-2). Los errores tienen un momento en el que ocurren y otro en el que las personas notan su existencia. Mientras más avanzado se encuentre el proyecto, más difícil será su reparación.[57]

- ***No promueve el reuso de software.*** Por razones históricas, tanto en el campo del desarrollo de sistemas como en el de calidad del software, el modelo de cascada no ignora el concepto *reuso*; simplemente no lo promueve ni lo alienta.

---

[57] Este problema no es exclusivo del modelo en cascada. Si bien las herramientas CASE pueden disminuir el problema detectando errores de lógica en el diseño, no harán nada por mostrarlos cuando el analista no entendió los requerimientos de los usuarios.

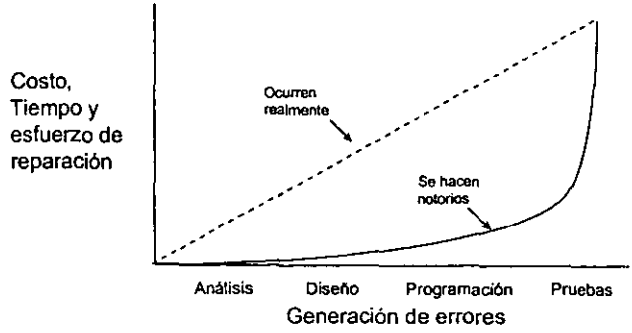


Figura 3-2 El crecimiento de un problema en software.

- ***No promueve el uso de prototipos.*** Por su naturaleza, el modelo considera que la fase de análisis debe ser cubierta sólo una vez. Actualmente, se ha expandido el uso de prototipos donde los usuarios validan y llegan al software que necesitan a través de un proceso de prueba y error.
- ***Generalmente no se practica en un sentido formal.*** Debido a el volumen de documentos que implica, muchas organizaciones no tienen el tiempo ni los recursos suficientes para aplicar el modelo en cascada rigurosamente. Este modelo se practica en general, por una asimilación cultural que transforma el modelo en versiones *ad-hoc*, que pueden conducir a resultados impredecibles.

3.1.4 Tipos de ciclos de vida

Muchas de las implementaciones del ciclo de vida en cascada asumen que una actividad debe finalizar para comenzar la siguiente; sin embargo, no hay nada escrito para apoyar tales restricciones. Incluso algunas de las actividades pueden traslaparse sin causar problemas.

En un extremo, todas las actividades del ciclo de vida pueden realizarse en paralelo. Del otro lado, el líder de proyecto puede decidir organizar el proyecto de manera totalmente secuencial.

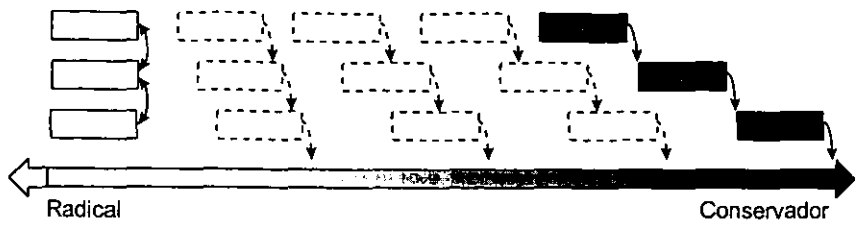


Figura 3-3 Ciclos de vida radicales y conservadores

Separando ambos extremos, se puede decir que una aproximación *radical* al ciclo de vida en cascada impulsa el inicio de todas las actividades en paralelo al inicio del proyecto; en contraste, la aproximación conservadora no permite el inicio de una actividad sin que la inmediata anterior haya terminado (Figura 3-3).

En la práctica es extraño que un líder decida tomar alguno de los extremos para desarrollar un proyecto. Lo importante es reconocer que la aproximación radical y la conservadora marcan dos extremos de un rango de múltiples opciones.

Con esta gama de patrones por tomar, no existe una regla definida que utilice el líder de proyecto para decidir cuál seguir. Generalmente, la balanza se inclinará a uno u otro lado dependiendo de factores como los siguientes:

- El nivel de cambios en las especificaciones que haga el usuario, y su grado de vulnerabilidad.
- La presión que exista para que el Líder de proyecto entregue resultados tangibles.
- La presión existente para cumplir restricciones de tiempo, costo y uso de recursos técnicos y humanos.

Ninguno de los puntos anteriores tiene una medición exacta que permita arreglar el modelo de antemano. Éste se construye en la práctica y con la experiencia del Líder en el campo de desarrollo. Así será posible saber qué tipo de usuario participará, qué nivel de compromiso tiene, cuán propenso es a cambios de

opinión; mientras más experiencia hay, existe más sensibilidad para conocer el entorno en general; se perciben los problemas potenciales, se aprovechan los talentos especiales que la gente posee y se anticipan soluciones oportunamente.

Un factor importante para decidir qué modelo construir es la necesidad de mostrar resultados tangibles. Existen proyectos que obligan a etapas de análisis muy largas, de donde un conjunto de especificaciones producen etapas de programación cortas. En tales casos una aproximación conservadora puede ser factible; por el contrario, puede existir por necesidades del usuario o del entorno de mercado, la necesidad de que un desarrollo termine en cierta fecha específica. En tal caso se precisa una aproximación más radical.

Por supuesto, en todo proyecto existe la presión (incluso dinámica) de mostrar resultados tangibles. Si las condiciones varían en el curso del proyecto, así cambiará también el modelo que se utilice. Por ejemplo, si un proyecto tiene holgura para finalizar, arrancará con un esquema paso a paso. Si las fechas se adelantan para terminar, el proyecto tomará una forma más radical que se adapte a las nuevas condiciones.

Otro factor son los requerimientos de tiempo, presupuesto, uso de recursos, etc., que viven en todas las organizaciones. Cuando estos requerimientos se presentan de manera desorganizada e informal, quizá sea conveniente una aproximación radical que absorba los impactos por sesgos en las estimaciones. Por el contrario, en organizaciones grandes con proyectos largos, se tiene un especial cuidado en hacer las cosas sólo después de minuciosos análisis, investigaciones y autorizaciones; esto las lleva a implementar esquemas muy conservadores.

En resumen, una aproximación radical conviene para desarrollos que potencialmente presenten cuestiones por clarificar. Es bueno para ambientes donde todo o parte del software debe estar funcionando en fechas inamovibles o cuando la percepción de los usuarios acerca de lo que quieran que haga el sistema esté sujeta a cambios.

Por otro lado, la aproximación conservadora tiende a ser usada en proyectos grandes donde se gastan gruesos presupuestos que obligan a la ejecución de análisis y diseños muy cuidadosos para evitar desastres posteriores. En todo caso, cada proyecto es diferente y requiere su propia combinación de implementaciones radical y conservadora. Para manejar la naturaleza individual de cada proyecto, se debe estar preparado para cambiar la aproximación en algún punto intermedio si es necesario.

3.1.5 Modelos incrementales y en espiral

Las limitaciones del modelo en cascada han quedado expresadas en secciones anteriores. Para hacer más flexible el modelo, Boehm y otros han propuesto que el desarrollo de software puede ser manejado como una serie de *incrementos*. De esta manera el modelo puede ser visto como una sucesión de ciclos de vida en cascada, uno por cada incremento. La Figura 3-4 muestra el modelo.

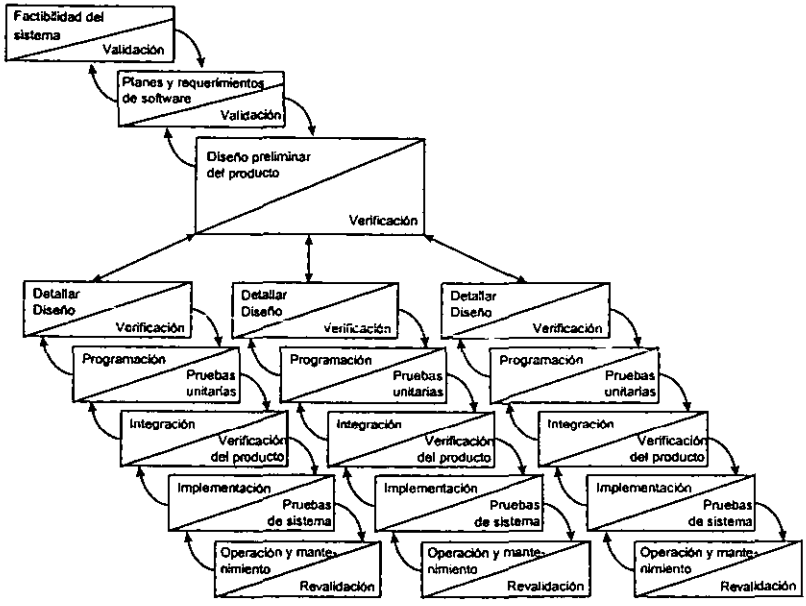


Figura 3-4 El ciclo de vida incremental

A partir de este esquema, Boehm considera posible también ver al desarrollo de software como una *serie de espirales*. Este esquema puede verse en la Figura 3-5. Hay que destacar que el modelo incorpora el uso de prototipos, que se discutirán más adelante.

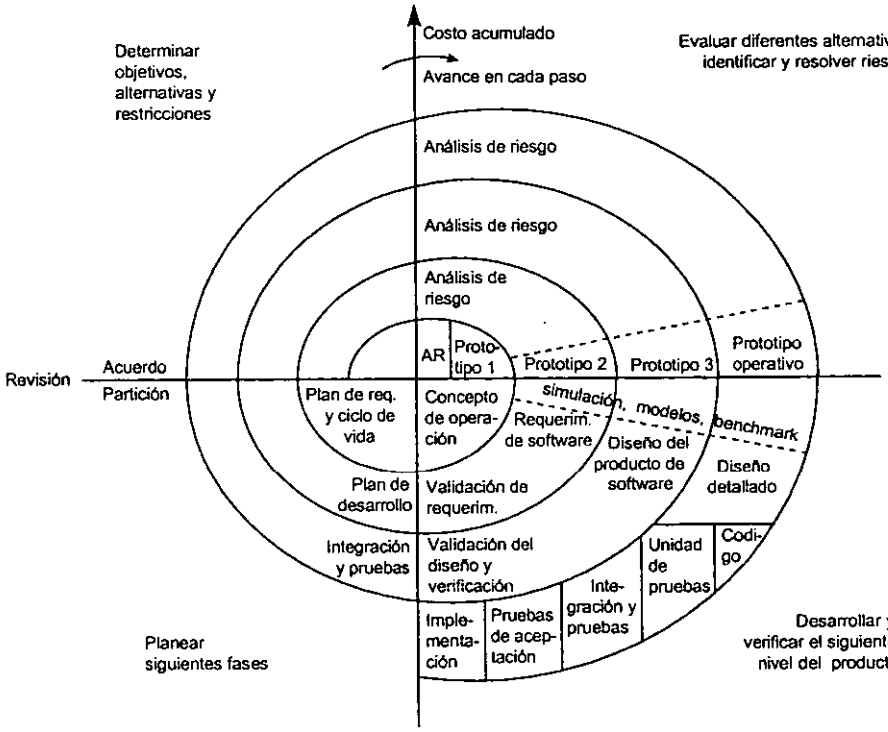


Figura 3-5 El ciclo de vida en espiral.

3.1.6 Transformación automática de modelos.

De la discusión precedente se pude inferir que el ciclo de vida en cascada no es inservible en si mismo, pero tiene limitaciones que en buena medida dependen de la forma en que el modelo se implementa.

Con el auge de las herramientas CASE que utilizan generadores de código, el ciclo de vida en cascada se ve desde una nueva perspectiva (Figura 3-6).

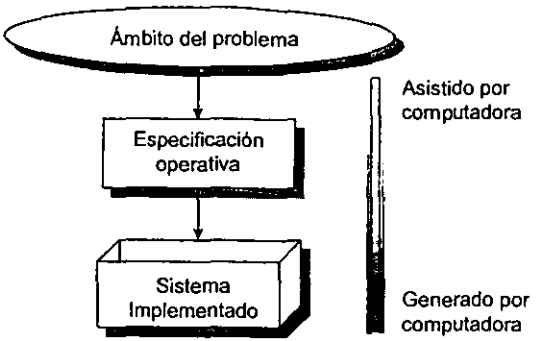


Figura 3-6 La versión moderna del ciclo de vida en cascada.

Las ventajas de desarrollar de ésta forma, conocida como *ciclo de vida basado en transformaciones*, son varias:

- El mantenimiento puede hacerse en el momento de las especificaciones. Los proyectos que utilizan el modelo clásico en cascada, realizan todo el mantenimiento sólo cuando se escribe el código, pues el manejo de documentos en papel durante el análisis y diseño hacen imposible mantenerlos actualizados.

  Virtualmente los documentos pueden ser obsoletos cuando termina la fase de desarrollo o aún al termino de la misma fase que los produjo. En contraste, la versión transformada del ciclo de vida genera código automáticamente, cuando se establecen las especificaciones. Esto significa que el mantenimiento a las especificaciones durante el diseño es viable y garantiza su veracidad en siguientes fases.

- Verifica errores anticipadamente. Si el diseño de un sistema y sus requerimientos se traducen en modelos asistidos por computadora, donde se genera código, entonces es factible la verificación y análisis de errores. De hecho, ésta es una de las ventajas principales que ofrecen las herramientas CASE. Como se ha comentado, mientras más pronto se detecten errores, más fáciles y baratos serán de corregir.

Sin embargo, las herramientas CASE puntualizan los errores mecánicos producidos por la herramienta misma. El factor humano sigue siendo inevitable en la validación de errores lógicos, o de diseño redundante.

- Habilita el rastreo de requerimientos. Los distintos modelos surgidos uno a partir de otro mediante transformaciones son controlados por computadora, Esto facilita el rastreo de los requerimientos a través de los diferentes modelos para asegurarse que correspondan al código producido.

- Fomenta el reuso de software. Si se piensa en el ciclo de vida como una serie de transformaciones hechas por computadora, existe la oportunidad de aprovechar las bibliotecas ya existentes. Estas bibliotecas no sólo pueden ser de código sino también de diseños o especificaciones.

- Impulsa las especificaciones orientadas al problema. Debido a que el código es generado automáticamente, el ciclo de vida moderno pone más énfasis en representar los requerimientos del usuario con la óptica de la representación del problema mismo. El modelo clásico en cascada ofrece poco en esta área y los requerimientos del usuario se expresan en términos de la máquina se utiliza o del lenguaje en que se desarrolla [58] .

3.1.7 El ciclo de vida con prototipos.

Informalmente los prototipos han sido practicados desde el desarrollo de la primera computadora, pero pasó a un segundo plano cuando el ciclo de vida en cascada dominó en el campo de la creación de software en los años 60 y 70.

En esos tiempos se debía asumir que la forma correcta de hacer las cosas era analizar meticulosamente los requerimientos de usuario primero, para después dedicarse a los detalles del diseño y la implementación.

---

58 Hablando de la calidad de los procesos de creación de software se puede notar un gran vacío en el área de la administración. Muchos administradores de hoy se hicieron en la práctica, lo que los ha llevado a administrar en las áreas que más conocen, por ejemplo: "eso lo haría en BASIC en media hora". Con el uso de modelos modernos donde el código no es lo más importante, sólo nos queda reflexionar acerca de nuestro papel en el proceso, cómo asumirlo y cómo mejorarlo.

Bernard Boar fue un pionero que hizo del prototipo en el desarrollo de software un camino legítimo para clarificar los requerimientos del usuario. La Figura 3-7 muestra este ciclo de vida.

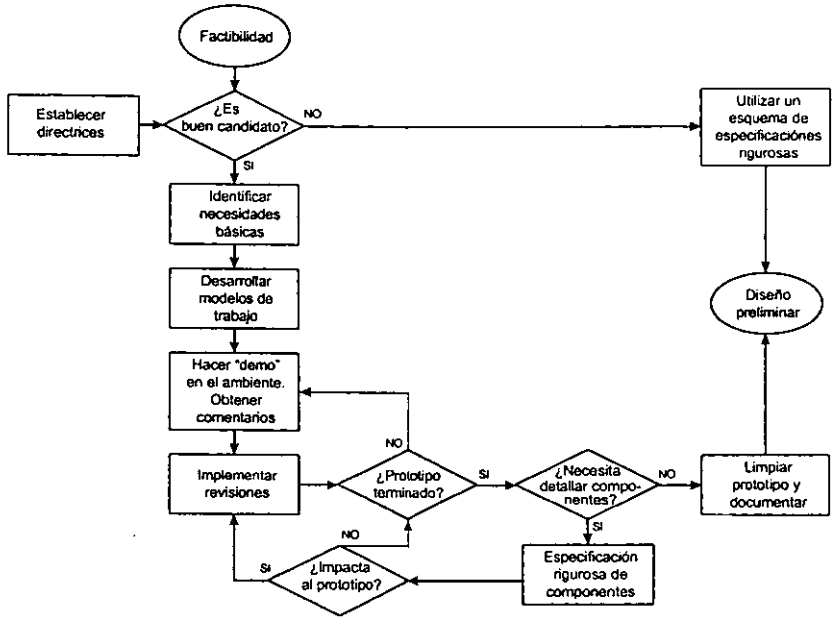


Figura 3-7 El ciclo de vida con prototipos

Los prototipos han sido reconocidos como un medio efectivo de explorar interfaces humanas alternativas para el sistema, por ejemplo para explorar formatos de entrada, diseño de pantallas o reportes. Cuando el usuario no tiene claro el tipo de sistema que desea, este modelo puede ser el único viable para dilucidar los requerimientos.

Aunque suena atractivo hacer prototipos para desarrollar la interfaz humana apropiadamente, como indica el diagrama del modelo, no todos los proyectos son buenos candidatos para crear prototipos. El ejemplo más claro es cuando los desarrolladores no cuenten con buenas herramientas técnicas que produzcan el prototipo.

La forma radical del ciclo de vida en cascada es, en sí misma, una forma de prototipo. Se puede hacer una versión inicial del sistema a partir de las fases de análisis y diseño. Si resulta inaceptable para el usuario, pude cambiarse; las especificaciones cambian y se genera un nuevo sistema. Así, la decisión entre modelo en cascada o prototipo puede no ser binaria; se puede escoger uno entre la versión conservadora del ciclo de vida y la radical.

3.1.8 Metodología y cultura

En la forma más simple, una metodología en la ingeniería de software puede ser vista como una lista de reglas y normas combinadas con alguna notación gráfica para el modelado de requerimientos y diseño del sistema.

En un sentido más profundo una metodología requiere y promueve cierta disposición; su uso en una organización tiende a crear una cultura tecnológica que es radicalmente diferente a la cultura de otra organización que utilice una metodología diferente.

3.2 Las herramientas CASE y las metodologías

Una parte importante en la creación de software son las metodologias utilizadas durante los proceso de análisis, diseño y desarrollo. El uso de las mismas adecuadamente —suficientes en relación con el entorno tecnológico y aplicativo— será útil en el camino hacia la calidad del software.

En los recientes años, la aplicación y avance en este campo, así como en el desarrollo de herramientas CASE abren un vasto horizonte donde la calidad del software puede verse beneficiada.

3.2.1 El boom de las metodologías en la ingeniería de software

Como en toda actividad de ingeniería, los procesos que se van haciendo complejos requieren de un método para ser llevados a cabo. El software no

escapa a la regla y la utilización de las metodologías de desarrollo se ha vuelto parte substancial de la ingeniería de software.

Las técnicas estructuradas hicieron su aparición a finales de los años 60, con la introducción de la programación estructurada. En los años 70 se utilizó el diseño y el análisis estructurado.

El análisis y el diseño estructurado se caracterizaron por el uso de dos modelos gráficos: los diagramas de flujo y las tablas estructurales. Ambas enfatizaban las *funciones* ejecutadas por el sistema. Existen variaciones más modernas que han incorporado diagramas de entidad-relación y diagramas de estado-transición como ayuda al ingeniero de software para modelar los *datos* en el *comportamiento dependiente del tiempo* del sistema.

Los aspectos metodológicos de estas técnicas pasaron de ser una simple representación física del modelo, a un nuevo modelo lógico que incorpora la importancia de los datos y el tiempo en el desarrollo del sistema. Cabe señalar que este tipo de metodologías han sido incorporadas recurrentemente en herramientas CASE.

A partir de la importancia que toman los datos para los sistemas, las metodologías de la ingeniería de la información establecen un énfasis: los *datos* juegan un papel determinante, mientras que las *funciones* del sistema se subordinan a ellos.

Con esta extensión, los diagramas de entidad-relación por sobre los diagramas de flujo, no es la cuestión esencial. Se incluyen al mismo tiempo relaciones entre las funciones de negocio con las entidades definidas, para mostrar los vínculos que unen a las diferentes funciones del negocio dentro de la empresa con las entidades que usan y modifican.

Quizá lo más importante de estas metodologías sea el *nivel* establecido por el modelo. Aunque las metodologías estructuradas han sido utilizadas a nivel institucional dentro de las organizaciones, en la mayoría de los casos se usan sólo para modelar programas o sistemas individuales. La ingeniería de la información,

por el contrario, generalmente percibe la metodología en un nivel mayor, en el que la metodología intenta primero modelar a la empresa para después modelar programas y sistemas individualmente.

El énfasis en los datos en lugar de las funciones, destacando el nivel de la organización a la cual la metodología debe involucrar, se puede representar como una gráfica bidimensional en la Figura 3-8.

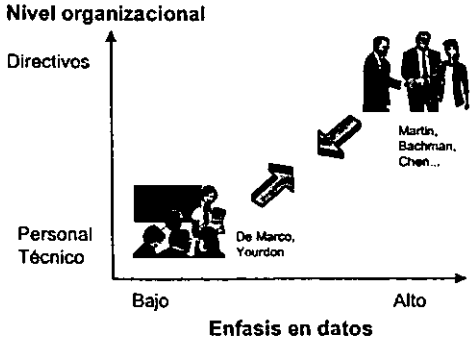


Figura 3-8 Diferencias entre ingeniería de la información y análisis estructurado

Lo más significativo de la gráfica anterior es que ambos esfuerzos se mueven hacia una meta común que combina datos y funciones, así como herramientas apropiadas tanto para personal técnico como para directivos.

Las metodologías evolucionan con el tiempo, pero la presencia de diversas figuras en este campo, como individuos, consultores y vendedores de servicios hacen que esta evolución no siempre ocurra de la forma más conveniente.

En recientes años han tomado auge las metodologías orientadas a objetos. Por novedosas que parezcan, su nacimiento se registra en los años 60 con el desarrollo de lenguajes como SIMULA y posteriormente Smalltalk. Al principio ciertas metodologías de desarrollo —especialmente la popular Jackson—, que se relacionaban con la programación estructurada tendieron pronto hacia una visión

de objetos, que evolucionó en análisis orientado a objetos que ha tomado gran auge.

Aunque las técnicas orientadas a objetos no son nuevas, su uso se ha expandido y cobrado importancia. Este cambio se ha debido al cambio gradual en las prioridades y en la tecnología. Entre algunos puntos que vale la pena destacar se encuentran:

- Los conceptos que engloban la visión orientada a objetos han tenido una década para madurar en donde la atención ha variado de revisar asuntos de programación, a discutir temas de diseño y finalmente de análisis.
- La tecnología existente para construir sistemas se ha vuelto más poderosa. Desafortunadamente la forma en que se analiza se influencia por las ideas preconcebidas de cómo se diseñará el sistema. Al mismo tiempo, el diseño es influenciado por nuestras ideas preconcebidas de la programación. A su vez, la programación puede influenciarse por los lenguajes existentes. Es difícil pensar en diseño y análisis estructurados cuando se dispone solamente de ensamblador para el desarrollo. La aparición de lenguajes orientados a objetos facilita esta tarea.
- La complejidad de los sistemas actuales es mayor que hace años. Abarcan más aspectos y están sujetos a continuos cambios. El uso del análisis estructurado hace más estables a los sistemas, al mismo tiempo que cubren las enormes demandas de interfaces de usuario.
- El modelado de datos ha cobrado importancia. La complejidad funcional que un sistema pueda tener suele ser menor que la importancia de los datos y sus relaciones.

Las metodologías orientadas a objetos evolucionan de las técnicas de programación estructurada. La línea divisoria no es clara del todo y los expertos no se han puesto de acuerdo en la definición exacta de lo que representa una metodología orientada a objetos. Sin embargo, un acuerdo común implica que las siguientes cuatro características deben ser cubiertas por una metodología orientada a objetos:

- ***Abstracción de datos***. En lugar de la importancia de la funcionalidad, o el establecimiento de procedimientos típicos de las técnicas estructuradas, las metodologías orientadas a objetos imprimen importancia a la conceptualización de los datos. Aquí se incluyen los términos clase y subclase que definen la relación entre las entidades —representación de los datos— que son base del análisis.

- ***Encapsulamiento***. Las técnicas estructuradas consideran a los datos y a las funciones que los procesan como entes separados. Cada uno tiene sus características e incluso pueden ser trabajadas por personas diferentes. La esencia del la visión orientada a objetos es el concepto de *empacar* o *encapsular* los datos y las funciones que los ocupan, juntos de tal manera que la única forma de acceder (o modificar) un dato sea invocando su función asociada.

- ***Herencia***. Las metodologías orientadas a objetos permiten a un objeto heredar tanto los atributos de datos como de funciones pertenecientes a objetos "padres" de nivel superior.

- ***Comunicación a través de mensajes***. En las técnicas estructuradas la comunicación entre módulos era por medio de llamadas a funciones o subrutinas. En un sistema orientado a objetos la comunicación cambia a una arquitectura donde los objetos se comunican enviando mensajes unos a otros.

No importa la metodología que se use, o de la cual se tengan referencias para apoyarla, se debe considerar que en los sistemas existen tres dimensiones que deben modelarse. Estas son funciones, datos y tiempo de respuesta.

Las metodologías de décadas anteriores se focalizaban sólo en una de esas tres dimensiones porque estaban orientadas a ser usadas por sistemas de esos tipos. La Figura 3-9 muestra que ciertos tipos de sistemas pueden cubrir sólo una dimensión en su complejidad; de esta manera podrían estar bien cubiertos por metodologías que proporcionen herramientas suficientes para esa dimensión.

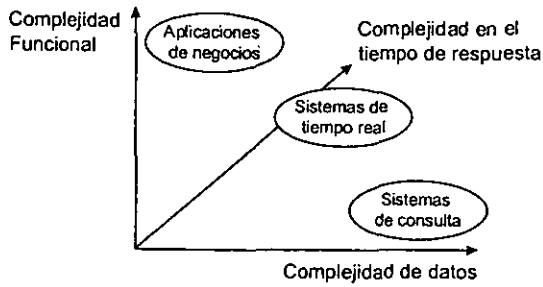


Figura 3-9 Relación entre tipos de sistemas y complejidad

Aun en la actualidad algunos sistemas son complejos sólo en algún rubro, sea funcionalidad, datos o tiempo de respuesta. Algunos otros tienden a ser complejos en dos o tres de estos rubros. Así se pueden ver sistemas de control con estructuras de datos complejas y funcionalidad amplia en adición con lo crítico que puede llegar a ser el tiempo de respuesta. Sistemas bancarios, de aerolíneas o de seguros son de este tipo, y deben soportar miles y miles de transacciones en línea hechas en una gran cantidad de terminales. En ellos coincide una especial complejidad en datos, lo mismo que en la funcionalidad y en el tiempo, factor importante en la ejecución y el servicio de tales sistemas.

Lo anterior nos lleva a pensar en que lo importante no es qué dimensión modelar en el sistema, sino cuál modelar primero. Las metodologías de sistemas más avanzadas deben considerar al menos estos tres aspectos, pues cualquier metodología —especialmente en nuestros días—, que ignore una de estas partes, no generará un modelo adecuado del sistema y por tanto no contribuirá a su creación con calidad.

Varias metodologías se han incorporado como parte de herramientas CASE. Su objetivo se explicará más adelante. Baste decir que, como todo sujeto de evolución, las metodologías tienden a cambiar, a caminar de un estado a otro dependiendo de los avances tecnológicos y del tipo de sistemas que se deben crear.

El reto es utilizar una metodología adecuada para el caso en que se trabaje, y tener en cuenta que las metodologías no son en sí mismas la solución a todos los problemas relacionados con la calidad del software. Si bien constituyen un aliado necesario, su utilización —y aún el cambio de metodología si es necesario— debe responder al entorno tecnológico y aplicativo en el que se utilice.

3.2.2 Herramientas CASE

Las herramientas *CASE* son herramientas basadas en computadora para asistir el proceso de la ingeniería de software. En la práctica, cualquier CASE se compone de un conjunto de módulos o *toolkit*.

Estas herramientas han cobrado gran popularidad en la solución a los problemas de calidad, especialmente debido a la experiencia que con ellas han tenido los desarrolladores de software.

Las ventajas de las herramientas *CASE* incluyen:

- ***Productividad.*** Los buenos desarrolladores de software son escasos. Uno de los objetivos de las herramientas CASE es maximizar la productividad para aprovechar eficientemente el recurso humano.
- ***Consistencia.*** Los *CASE toolkit* cuentan con un diccionario de datos central al que todos los desarrolladores involucrados en el proyecto se debe referir. Esto permite que varias personas trabajen separadamente conservando la consistencia en cuanto a variables, datos o sintaxis. Especialmente en proyectos grandes, esta característica puede justificar por si sola el uso de tales herramientas.
- ***Metodología automatizada.*** Muchas herramientas se asocian con el uso de una metodología. El CASE se asegura de que el desarrollador se apegue a tal metodología. Esto asegura la consistencia, pero de cierta manera restringe la creatividad.

- ***Promueve prácticas estándar.*** Debido al apego automatizado a cierta metodología, los CASE también se aseguran de que se implementen prácticas buenas en la organización, tales como el uso de la programación estructurada.
- ***Documentación.*** Este punto es tradicionalmente menospreciado en el desarrollo de sistemas. Sin embargo, la importancia de la documentación queda clara en lo expuesto en este trabajo. Las herramientas CASE proveen formas automatizadas de asistir el proceso de documentación, manteniendo las características de consistencia, verosimilitud y estándares establecidas por la metodología asociada.
- ***Mantenimiento.*** Una de las principales razones para la introducción de herramientas CASE son los costos asociados con el mantenimiento. Las herramientas pueden mejorar la calidad del software creado y hacer que los cambios sean más baratos y fáciles de implementar.

Las herramientas CASE se dividen en tres tipos. Esta clasificación se basa en la parte del ciclo de desarrollo que cubre específicamente la herramienta. La relación entre los diferentes tipos de herramientas CASE y el ciclo de vida del sistema se muestra en la Figura 3-10.

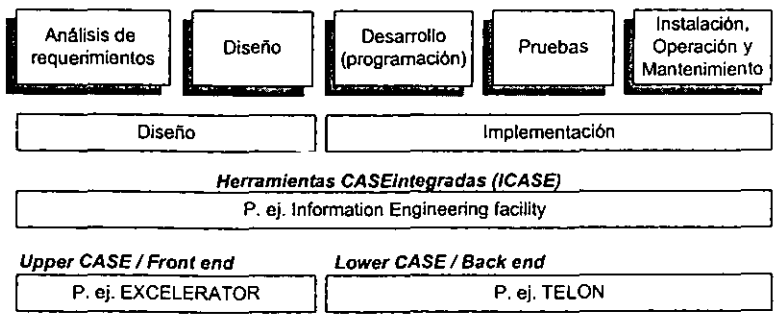


Figura 3-10 Tipos de herramientas CASE

Herramientas CASE *Superiores* o de *FRONT-END*

Estas herramientas tienen que ver con las fases de diseño del ciclo de vida de sistemas. Pueden estar relacionadas con una metodología en particular o pueden permitir el uso de alguna metodología propia del usuario.

Ejemplo de este tipo de CASE es el producto EXCELERATOR. [59]

Herramientas CASE *Inferiores* o de *BACK END*

Estas herramientas tienen que ver con las etapas de implementación en el ciclo de vida, usualmente la codificación, las pruebas y la documentación. Apuntan a incrementar la confiabilidad, la adaptabilidad y la productividad asociadas con el código escrito. Los lenguajes de cuarta generación 4GLs pueden considerarse como herramientas CASE de back end. Ejemplo es el producto TELON [60].

Herramientas CASE integradas

Las herramientas CASE integradas (*ICASE*) se dirigen a soportar el ciclo de desarrollo completo y están ligadas a una metodología específica. Con frecuencia las *ICASE* resultan complicadas y caras, pero ofrecen a los desarrolladores la mejor integridad posible, pues contienen una enciclopedia de datos única que se utiliza de principio a fin del ciclo de desarrollo

Las herramientas de este tipo se relacionan estrechamente con las metodologías de desarrollo exhaustivas tales como IEM (Information Engineering management). Las herramientas CASE basadas en IEM incluyen IEF (Information Engineering Facility) y IEW (information Facility workbench).

---

59 Producto de la firma Intersolv. WWW.Intersolv.com y www.albany.ede/acc/gangolly/a381exel.html

60 Producto de la firma Computer Associates. WWW.cai.com/products/telon

3.2.3 Desarrollo evolutivo

El aparente entusiasmo por el modelo en cascada para el desarrollo de software y las metodologías rígidas basadas sobre éste no parecen, sin embargo, ser universales.

En la práctica se pueden detectar casos donde la adopción de tales modelos tradicionales conducen a problemas serios en la solución propuesta. Gilb, por ejemplo, habla de ejemplos donde los atributos críticos son difusos, en lugar de ser resaltados una vez que se han analizado los requerimientos.

Por ejemplo, es más común de los que se piensa, que sistemas corporativos fallen porque el atributo crítico de capacidad de procesamiento no se cumple. Como la necesidad de cierta capacidad de proceso no se detectó ni se especificó a tiempo, una vez rebasado el límite, se genera el problema. En tal caso, la aparición tardía de la falla tiene la característica de ser costosa y de alto impacto para la organización.

La aplicación de un modelo evolutivo en el caso anterior, acompañado con un énfasis en el establecimiento de los atributos críticos hubiera hecho posible la detección de la falla anticipadamente.

En el proceso evolutivo se asume que es imposible establecer todos los requerimientos del usuario desde el principio del proyecto que se desarrollará con el enfoque del ciclo de vida en cascada:

> *"Si tan solo tuviéramos la capacidad intelectual y el conocimiento necesario para hacer las cosas correctamente. En realidad, debemos admitir que no podemos abordar estas tareas adecuadamente mas que en proyectos trivialmente pequeños"* [61]

Esta aproximación al desarrollo de software ve a la especificación de requerimientos como una entidad en constante evolución, que pretende estar lo más cerca posible del requerimiento real, pero que nunca está allí. Aunque los

61 Gilb, T (1988) Principles of Software engineering management. *Op cit.* en Gillies, Software quality, p. 125

requerimientos originales se alcanzaran, al momento de cumplirse ya hubieran cambiado. Esto se ve gráficamente en la Figura 3-11 .

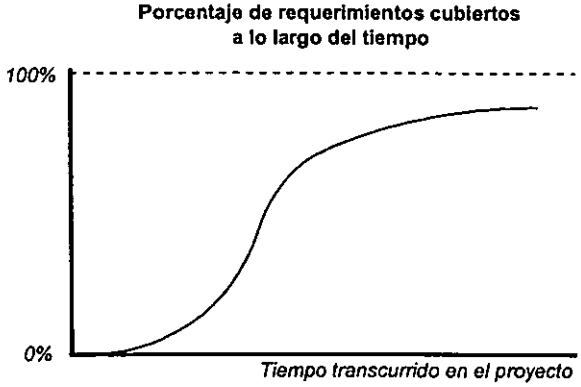


Figura 3-11 La evolución de los requerimientos en el tiempo *

La idea evolutiva del proceso de diseño propuesta por Gilb et. al. está inevitablemente basada en el los modelos de prototipos. Hacer prototipos no es una idea ampliamente expandida entre la comunidad de desarrolladores de software.

Esto puede parecer extraño, pues si se compara la ingeniería de software con alguna otra disciplina de la ingeniería, en las otras el uso de prototipos es una técnica ampliamente aceptada.

En el capítulo anterior se resumió el concepto de proceso evolutivo en el diseño del software, desarrollado por Gilb. El código se implementa como una serie de mini proyectos que pueden ser administrados de la misma forma que cualquier otro proyecto de desarrollo de software.

Ya que estos mini proyectos son más pequeños en comparación con los rimbombantes proyectos de software a gran escala, existen ciertas ventajas, especialmente con la reducción de la complejidad.

* Tomado de Sommerville, I. (1992) Software Engineering. Inglaterra: Addison Wesley

El uso del proceso evolutivo y las ventajas asociadas con la disminución de la complejidad, llevan a algunos beneficios:

- Los requerimientos de usuario son más fáciles de establecer
- Los usuarios se sienten más involucrados y por ende, motivados
- Los atributos críticos se pueden determinar tempranamente
- Los problemas dentro del equipo de desarrollo salen a la luz oportunamente
- Cualquier entrega parcial de código genera credibilidad
- Prueba el modelo, desde su diseño, en situaciones reales antes del fin del proyecto
- Enfatiza la importancia de la documentación de usuario, que se considera impostergable

Sin embargo quienes critican este posible modelo de desarrollo tienen objeciones. En general éstas se caracterizan por la resistencia al cambio. Existe una inversión substancial en los modelos secuenciales y cualquier esfuerzo nuevo de capacitación implicará una inversión adicional. Cambiar viejos hábitos y actitudes siempre costará un tiempo significativo.

Los equipos de trabajo pueden objetar el cambio de paradigmas en el hecho de que no hay necesidad de cambios mientras las cosas funcionen correctamente y no aparezcan fallas. Sin embargo, está probado que un cambio de actitud es importante para que procesos de este tipo puedan implementarse como parte de un programa de calidad total. El programa apunta a un cambio cultural en la organización, que la orienta hacia el logro de la calidad.

3.2.4 Las herramientas CASE y la calidad

La selección de la herramienta CASE a utilizar en una organización es un paso importante en el plan global para adquirir y explotar la tecnología CASE. Sin importar el proveedor que se elija, el riesgo de falla siempre estará presente. Yourdon (1992) apunta: "es muy posible gastar millones de dólares en

herramientas CASE para después de algunos años descubrir que no ha tenido ningún impacto en la productividad o en la calidad". [62]

El fracaso en la implementación y uso de herramientas CASE puede ser influido por una serie de factores:

- Que no haya metodología ni estándares en donde se implementa.
- Que se ignore la importancia que tiene la administración.
- Que se haga demasiado énfasis en el CASE como la solución mágica.
- Que haya confusión de lo que hace la herramienta CASE.
- Que no haya un uso expandido y adecuado de la herramienta.
- Resistencia y poca importancia al cambio.
- Poca certeza y falta de consenso a cerca de lo que la herramienta CASE trata de resolver ¿Productividad? ¿Calidad? ¿velocidad en el desarrollo de nuevos productos?.
- Poca integración de las herramientas.
- Funcionalidad inadecuada.
- Documentación y capacitación deficientes.

De ésta forma, las cuestiones técnicas como la falta de funcionalidad o la pobre integración, pueden ser una causa evidente de falla, pero en estricto sentido, la mayoría de los puntos de la lista se refieren a cambios *culturales o administrativos*.

Según Yourdon[63], para evitar problemas como los expuestos se requieren tres cosas:

62 Yourdon, E. *op. cit.*, p 156

63 *Ibid.*

a) *Soporte administrativo de la dirección, con expectativas razonables.* Las herramientas CASE no producirán milagros de un día para otro, y esto debe ser entendido desde los niveles más altos de dirección.

b) *Un nivel moderado de sofisticación en el desarrollo de software de la organización.* Las herramientas CASE deben utilizarse apropiadamente; no es adecuado utilizarlas para fines distintos a los establecidos por la dirección. En este sentido, por ejemplo, los *hackers*[64] no harán un buen uso del CASE.

c) Un plan para la implementación del CASE. Es recomendable que el plan permita un arranque mesurado, tomando proyectos pequeños como piloto, antes de que el CASE sea ampliamente difundido en toda la organización. El plan debe garantizar que las herramientas CASE escogidas sean las apropiadas para resolver el problema; por ejemplo, la implementación de CASE superiores puede ser una gasto excesivo si lo único que se pretende resolver es el mantenimiento de los sistemas.

Las herramientas CASE, lo mismo que las metodologías solamente o los más avanzados procesos de desarrollo, no representan soluciones mágicas ni inmediatas para la calidad. No hay razón para suponer que la simple utilización de una herramienta cambiará las cosas. Si la herramienta anterior no ofreció resultados, no es previsible que la nueva lo logre. Más aún, que lo logre en el corto plazo.

3.2.5 El cambio de cultura

Probablemente la parte más significativa en la evolución hacia el uso de CASE sea el cambio gradual en la cultura de la ingeniería de software dentro de la organización. El desarrollo de software basado en CASE debe ser un proceso orientado a las especificaciones más que al código; deberá enfatizar la construcción del sistema a partir de partes reusables, con un proceso automatizado y continuo de verificación.

---

64 Un *Hacker* es un técnico altamente entrenado en sistemas de cómputo, herramientas y todo tipo de utilerías y lenguajes de programación que utilizan sus conocimientos para fines inadecuados. Ellos se atribuyen la introducción de poderosos virus o la modificación de páginas WEB dentro de internet.

Los participantes en el proceso deberán trabajar en un entorno interactivo y de respuesta inmediata con una interfaz humana aún más sofisticada que la que existe en nuestros días. Asimismo el ambiente CASE debe incorporar expertos en áreas tales como la selección e implementación de metodologías.

Carma McClure[65] caracteriza el cambio cultural en términos de entornos *pre-CASE* y *post-CASE* en la organización:

Pre-CASE	Post-CASE
Enfasis en codificación y pruebas	Enfasis en análisis y diseño
Modelos en papel	Prototipos rápidos
Codificación manual	Generación automática de código
Documentación manual	Documentación automática
Pruebas del código	Pruebas de especificaciones y diseño
Mantenimiento al código	Mantenimiento a las especificaciones y al diseño

Las organizaciones de software de *primer mundo* poseen una especial preocupación por las diferencias entre esos dos ambientes y trabajan exhaustivamente para moverse del primero al segundo. Desde el punto de vista filosófico, la mayor parte de las organizaciones deberían estar de acuerdo en que el ambiente post-CASE es preferible, pero he aquí una prueba para determinar si esto es sólo bagaje filosófico o un cambio *real*: diga a los programadores que no podrán hacer ningún cambio al código fuente.

En este esquema *cualquier* cambio debe estar acompañado de una modificación a la especificaciones o al modelo del sistema generado y entonces, regenerar el código fuente.

McClure afirma también que el cambio cultural estará acompañado de un ambiente dramáticamente mejorado para los ingenieros de software:

65 McClure, Carma, (1989) CASE is software automation. EU: Englewood CLIFFs,. Op cit en Yourdon, p. 168

Ambiente actual	Ambiente inteligente post-CASE
Interfaces amigables para el usuario	Interfaces personalizadas
Orientado al diagnóstico	Correctivo
Orientado al soporte	Directivo
Ayudante	Instructivo
Responde cuando se requiere	Reacciona ante los eventos que detecta

Esta transición es tecnológicamente atractiva pero supone que se sabe mucho más de la forma de trabajar de los ingenieros de software, de lo que en realidad se conoce. De hecho, las herramientas CASE han probado ser inútiles si no se acomodan a la forma de trabajo de las personas, a menos de que resulte factible cambiar la forma de trabajo de la gente.

La tarea más difícil en la organización es crear un balance al adaptar las herramientas a la forma de trabajo de la gente y adaptar los hábitos de trabajo de la gente a las capacidades y limitaciones de las herramientas.

Un cambio cultural, como se ha mencionado varias veces en este trabajo, no es fácil pero es indispensable en el camino hacia la calidad. La inteligencia, habilidad e interés con que se promueva dentro de la organización, redundará en un proceso evolutivo donde hablar de calidad, procesos de verificación o creación de software, sean partes de un mismo entorno,

3.3 Sistemas para el manejo de la calidad

Los sistemas de gestión de la calidad se consideran una serie de herramientas y procedimientos establecidos en una organización para la consecución y mejora de la calidad. Un factor relevante en ellos son las personas y el grado de participación y compromiso que tengan hacia su trabajo y por consecuencia, la actitud que muestren hacia la calidad.

Conceptos importantes como aseguramiento de calidad, círculos de calidad, sistemas de control, etc. aparecen dentro del espectro del complejo mundo de la calidad. Estos conceptos permiten en la práctica el establecimiento de criterios de calidad, su medición y finalmente la verificación de su cumplimiento. Todo esto con la idea de la mejora continua.

En el terreno del desarrollo de software también aparecen estos conceptos y formas de administración. Especialmente importante resulta el desarrollo de una cultura para la calidad que incluya a todos los participantes en el desarrollo de software.

3.3.1 Perspectiva histórica de los sistemas de manejo de la calidad

El área de la gestión de la calidad es dominada por las ideas de algunos personajes que se han convertido en *gurús*. Sus aportaciones han contribuido enormemente en la asimilación del termino *calidad*, a los procesos productivos. Los teóricos más reconocidos son Deming, Juran y Crosby.

Al Dr. Edward Deming generalmente se le atribuye haber establecido las bases de la *gestión de la calidad*. Después de la segunda guerra mundial ofreció sus ideas sobre la calidad al gobierno de Estados Unidos y a líderes empresariales. Por entonces sus ideas no fueron bien recibidas; este acontecimiento, sin embargo, marcó el destino en su carrera. Deming se trasladó a Japón y se puso en contacto con la JUSE[66]. El resto de la historia es bien conocida.

Junto con las ideas de Juran sobre *adecuación para el uso*, las ideas de Deming formaron las bases de la recuperación económica japonesa y de su posterior colocación en el mercado mundial.

Una vez que quedaron probadas en la práctica dentro de la industria japonesa las aportaciones de Deming y Juran, sus ideas fueron rápidamente aceptadas, especialmente en Estados Unidos, donde Crosby estableció su concepto de *cero defectos* y de que la *calidad es gratis*.

En recientes décadas se ha acuñado el término *administración total de la calidad* —TQM por sus siglas en inglés—, que engloba las aportaciones más importantes hechas sobre el campo de la calidad, pero que lo hacen parecer sólo un conjunto de modas que llegan y se van. Los hallazgos de los *gurús*, sin embargo, con frecuencia son blanco de críticas y son interpretados sólo como un conjunto de frases como las mostradas en la Figura 3-12 .

Figura 3-12 Opiniones comunes sobre la calidad

Una manera simplista de ver las cosas conducirá por ejemplo, en creer que resulta sencillo crear un sistema de administración donde todos sientan alegría por su trabajo, y que esto es sólo una percepción alejada de la realidad que Deming percibió al establecer sus preceptos.

Sin embargo, esta idea es, como otras, aplicable y especialmente útil en el campo de la calidad del software.

Las ideas básicas de los teóricos más importantes se resumen en la Figura 3-13 .

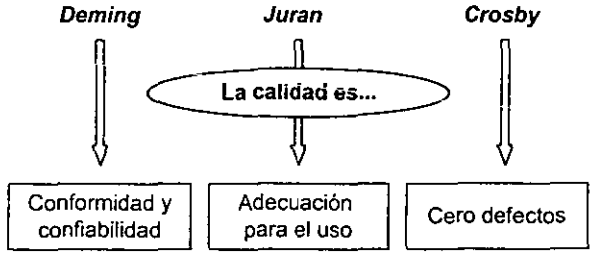


Figura 3-13 La calidad para Deming, Juran y Crosby

66 Unión de Ciencia e Ingeniería Japonesa

Deming

Para el Dr. Deming la aplicación de la estadística a la calidad es crucial. Su definición de calidad es:

> *"Un grado predecible de uniformidad y confiabilidad a bajo costo y adecuado al mercado."*

Sus ideas están orientadas al control estadístico de la calidad y a la participación de los empleados en la toma de decisiones. Argumenta que para conseguir la calidad, no basta con que los empleados hagan su mejor esfuerzo, *deben saber qué hacer.* Por tal razón, él no es partidario de las campañas publicitarias —posters, exhortos— establecidas en las empresas para alcanzar la calidad. Argumenta que suelen estar mal dirigidas y pueden causar frustración y resentimientos en las personas.

Para la gestión de la calidad, Deming sugiere la práctica de catorce puntos, mostrados en la Tabla 3-1, que deben ser usados tanto por la empresa como por sus proveedores.

1	Crear constancia en el propósito de mejorar el producto y servicio
2	Adoptar la nueva filosofía
3	Dejar de depender de la inspección para lograr la calidad
4	No hacer negocios sobre la base de precios bajos solamente. En vez de ello, minimizar el costo total trabajando con un solo proveedor
5	Mejorar constante y continuamente todos los procesos de planificación, producción y servicio
6	Implantar la capacitación en el trabajo
7	Adoptar e implantar el liderazgo
8	Desechar el miedo
9	Derribar las barreras entre las áreas de staff
10	Eliminar los slogans y exhortaciones
11	Eliminar las cuotas para la producción y los objetivos numéricos para la dirección
12	Permitir que el personal se sienta orgulloso de su trabajo. Eliminar calificaciones anuales y sistemas de méritos
13	Implantar programas de educación
14	Crear estructuras en la empresa para conseguir la transformación

Tabla 3-1 Los 14 puntos de Deming para la gestión de la calidad

Deming es un defensor de establecer una relación estrecha con un solo proveedor, pues a largo plazo resulta más benéfica la cooperación. Adicionalmente introduce el uso de técnicas estadísticas para el control de los procesos, para asegurar la calidad de los insumos entregados por los proveedores.

Juran

La fama de Juran aparece con Deming después de la Segunda Guerra Mundial. A él se le debe el término *adecuación para el uso*, que tiene gran relevancia en la gestión de la calidad y especialmente en el desarrollo de software.

Su punto de vista es que las definiciones de calidad basadas en la *conformidad con las especificaciones*, son inadecuadas. Sus ideas no son contrarias a las de Deming, y en los puntos donde difieren, generalmente son complementarias; junto a las aportaciones de Crosby, las de Juran no siempre son coincidentes.

Por ejemplo, Deming y Juran están en contra de campañas de exhortos para lograr la perfección; ambos se orientan al control estadístico de los procesos. Sin embargo, Juran rechaza dos principios básicos de Crosby de *cero defectos* y *conformidad con las especificaciones*. Argumenta que la ley de disminuir los rechazos durante la producción aplica al control de la calidad y que *la calidad NO es gratis*. Juran ha establecido diez pasos para alcanzar la calidad (Tabla 3-2).

Las ideas de Juran son mucho más orientadas a las personas. Así, concede gran énfasis a los equipos de trabajo y el establecimiento de proyectos.

Crosby

Las aportaciones de Crosby generalmente difieren de las anteriores, especialmente de las ideas de Juran. Crosby establece el concepto de *cero defectos* y forma parte importante de su ideario el título de su libro *La calidad no cuesta*.

1	Crear conciencia de la necesidad y oportunidad para la mejora
2	Establecer objetivos para las mejoras
3	Organizar para alcanzar los objetivos
4	Proveer capacitación
5	Llevar a cabo proyectos para resolver problemas
6	Reportar avances
7	Reconocer los esfuerzos realizados
8	Comunicar los resultados
9	Registrar los movimientos
10	Mantener la fuerza haciendo de las mejoras anuales parte de los procesos regulares de la compañía

Tabla 3-2 Diez puntos de Juran para mejorar la calidad

Sus aportaciones deben resumirse como prevención más que como métodos tradicionales de inspección y pruebas. Él iguala la perfección con la prevención, punto de vista ampliamente aceptado en nuestros días, particularmente en el área de la manufactura, donde las ideas de Crosby son muy apropiadas.

Él sugiere tres puntos como "vacuna para la calidad" que intentan prevenir la no conformidad. Esta vacuna se compone de determinación, educación e implementación. Propone, al mismo tiempo, cuatro máximas para la calidad:

- Definición: Conformidad con los requerimientos
- Sistema: Prevención
- Estándar de producción: cero defectos
- Medición: el precio de la no conformidad

Al igual que Deming, establece catorce puntos para la mejora, orientados hacia la gestión (Tabla 3-3).

Particularmente, las dos primeras aportaciones parecen ser más acertadas que las de Crosby, especialmente porque se orientan más a las personas que a los procesos —considerando el enorme peso del factor humano en la creación de software, esto toma relevancia—, y porque no incluyen definiciones absolutas de calidad, como lo hace Crosby.

1	Dejar claro que la administración se orienta a la calidad
2	Formar equipos de mejora de la calidad, donde todos los departamentos estén representados
3	Determinar los orígenes de los problemas actuales y potenciales
4	Evaluar el costo de la calidad y explicar su uso como herramienta
5	Inculcar la conciencia y preocupación sobre la calidad a todos los empleados
6	Tomar acciones para resolver los problemas identificados
7	Establecer un comité para el programa de *cero defectos*
8	Capacitar a los supervisores para que lleven a cabo su papel en la mejora de la calidad
9	Designar un *día de cero defectos* para que todos los empleados detecten los cambios
10	Estimular a las personas para establecer objetivos para las mejoras
11	Alentar la comunicación con la Dirección respecto a los obstáculos para la mejora
12	Reconocer y apreciar a los participantes
13	Establecer Consejos para la calidad que ayuden en la comunicación
14	Hacer todo lo anterior y demostrar que nunca termina

Tabla 3-3 Los 14 puntos de Crosby para la mejora de la calidad

Un Breve resumen que compara las contribuciones en el campo de la calidad, se presenta en la Tabla 3-4.

3.3.2 Aseguramiento de la calidad del software

El aseguramiento de la calidad en el software (ACS) está estrechamente ligado a las actividades de verificación y validación establecidas en cada etapa del ciclo de vida del sistema. Aunque en muchas organizaciones no se hacen distinciones entre estas actividades del ACS, éste debe diferenciarse. El aseguramiento de la calidad es una función de administración, mientras que la validación y la verificación son procesos técnicos del desarrollo de software.

Dentro de las organizaciones, el ACS debe llevarse a cabo por un grupo específico separado de la estructura y que le reporte al nivel superior del líder de proyecto que desarrolla el software. El equipo de AC no debe estar asociado a ningún grupo desarrollador en particular pero debe ser responsable del aseguramiento de la calidad en toda la organización (Figura 3-14).

	Crosby	Deming	Juran
Definición	Conformidad con requerimientos	Grado predecible de uniformidad y confiabilidad a bajo costo	Adecuación para el uso
Responsabilidad de la Alta Dirección	Responsable de la calidad	Responsable del 94% de los problemas	Responsable al menos del 80% de los problemas
Estándar de producción	Cero defectos	Varias escalas de acuerdo al control estadístico de la calidad; NO cero defectos	Evitar las campañas que exhorten la perfección
Característica principal	Prevención	Reducción de los cambios: mejora continua	Enfasis en la administración de los recursos humanos
Estructura	14 pasos	14 puntos	10 pasos
Control estadístico del proceso	Rechaza los niveles estadísticos aceptables para la calidad	El control estadístico debe ser usado	Recomienda el control estadístico, con precaución del uso de instrumentos
Bases para la mejora	Un proceso, no un programa	Continuidad: eliminar metas	Orientado al proyecto: establecer metas
Equipos de trabajo	Equipos para mejora de la calidad; Consejos para la calidad	Participación de los empleados en las decisiones	Equipo o circulo de calidad
Costo de la calidad	La calidad es gratis	No hay costo óptimo. Es un proceso continuo	Hay un costo óptimo; la calidad NO es gratis
Compras	Los proveedores son extensiones del negocio	Usa control estadistico para establecer un fuerte nivel de cooperación	Problemas complejos. Aplicación de muestreos formales
Indices de ventas	Si	No	Si, pero trabajando con proveedores
Fuente única de insumos o un solo proveedor	-	Si	No

Tabla 3-4 Comparación de ideas sobre la calidad

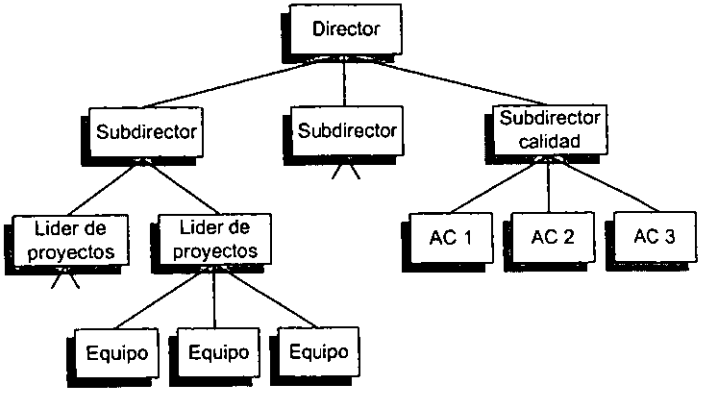


Figura 3-14 Estructura en una organización de software

Otro aspecto que marca la diferencia es que la verificación y la validación tienen que ver con la detección de fallas. El rango de acción del ACS es más amplio, pues tiene que ver con una serie de atributos del software como son confiabilidad, facilidad de uso, facilidad de mantenimiento, etc.

Un papel muy importante del equipo de aseguramiento de calidad es el de dar soporte para todos los periodos del ciclo de vida de la calidad. La calidad debe ser incluida en el producto (el software) y debe ser conducida a lo largo del proceso. La calidad no es un atributo que se le adicione al producto, por lo que se hace un objetivo lograr los más altos niveles de calidad en cada etapa del desarrollo. En este sentido, Sommerville apunta:

> *"La función del aseguramiento de la calidad es la de favorecer el logro de la calidad, no la de juzgar o criticar a los equipos de trabajo".* [67]

Una definición de aseguramiento de calidad puede ser la siguiente: el AC consiste en los procedimientos, técnicas y herramientas aplicadas por profesionales para asegurarse de que el producto cumple o excede los estándares pre-especificados durante un ciclo de desarrollo del producto; sin el establecimiento de estándares específicos, el aseguramiento de la calidad garantiza que un producto cumple o excede un cierto nivel comercial o industrial aceptable de excelencia.

El problema que aparece cuando se habla de estándares para el aseguramiento de la calidad en el desarrollo de software, tiene que ver con la naturaleza de éste y con los múltiples criterios que son propios del software y que fueron expuestos en el capítulo uno.

La planeación para la calidad debe comenzar desde las etapas iniciales del proceso de software. Un plan de calidad debe establecer los niveles deseados de calidad y debe definir como serán valorados. Un error muy común en el aseguramiento de la calidad es creer que existe un entendimiento común de lo que significa "gran calidad" en el software. Tal concepto común no existe porque los atributos que se definan para medir la calidad son los que deben involucrarse en su aseguramiento y pueden variar de proyecto en proyecto.

Por tanto, el plan de calidad debe establecer cuáles son los criterios de calidad aplicables para el producto que se desarrolla. Esto es de crucial importancia, pues al establecer estos criterios, los involucrados pueden cooperar para cubrirlos. El plan también debe definir el proceso de valuación de la calidad; no tiene mucho sentido tratar de desarrollar un plan que persiga calidad si no le es posible valorar si ésta existe o no.

3.3.3 Cultura para la calidad: clave para la administración de la calidad

Dos partes son las importantes en un sistema de gestión de la calidad. Por un lado, las herramientas y procedimientos, y por el otro las personas. La razón de ser de las herramientas y procedimientos es la ayuda para que la gente pueda tener un resultado de calidad.

La aceptación de los preceptos de calidad por parte de la Dirección es crucial y no se da por si misma. La gestión del cambio es crítica para que el proceso tenga éxito. El riesgo que se corre al establecer un sistema de gestión de calidad, es que pase como una imposición de nuevas normas de trabajo.

El sistema de calidad funcionará sólo en la medida en que la Dirección perciba la trascendencia de los cambios. Estos beneficios potenciales incluyen:

- Una mejor satisfacción en el trabajo
- Menos tiempo gastado en actividades no relevantes
- Mayor orgullo del trabajo realizado
- Mayor participación del grupo
- Mas retroalimentación del trabajo

Es importante hacer notar que los equipos de trabajo no estarán motivados si no ven un compromiso real, acciones de la Dirección, un clima organizacional sobre la calidad y un trabajo en equipo orientado a los problemas relacionados con la

[67] Sommervile, Ian, (1992) Software Engineering. Inglaterra: Addison Wesley, p. 95

calidad. Resulta de particular relevancia la comunicación en dos sentidos, pues las ideas de los grupos de trabajo deben ser recogidas para hacer la diferencia en el proceso.

Los círculos de calidad

Una de las principales formas de conseguir que el staff se integre, es a través de la implementación de círculos de calidad (Figura 3-15). Un círculo de calidad (CC) es un grupo de personas a las que se les pide, —no se les dice—, que se unan para trabajar. Generalmente tienen un líder que los entrena y guía. Adicionalmente debe existir un supervisor que coordina el programa de círculos de calidad como un todo dentro de la organización.

Figura 3-15 Esquema del círculo de calidad

La administración debe estar comprometida con el programa. Mientras que existe la obligación y el derecho de administrar, se debe tener el estilo de no rechazar recomendaciones sin buenas razones, o no permitir la aparición de tales ideas.

Los círculos de calidad se componen de entre tres y quince personas. Los círculos más grandes tienden a fragmentarse y a crear varios sub-grupos, lo cual suele no ser conveniente. Los círculos deben reunirse periódicamente —el periodo puede variar entre una y tres semanas— para resolver los problemas pendientes.

El equipo de trabajo decide sobre qué problema trabajar, aunque existan sugerencias por parte de la dirección. Asimismo, el círculo puede convocar a expertos que juegan el papel de consultores y que ofrecen alternativas de solución; la responsabilidad de las acciones a tomar, sin embargo, recae en el circulo mismo.

Los círculos de calidad no aparecen espontáneamente. Un ingrediente vital para el éxito del proceso es la capacitación en todos los niveles de la organización. El líder necesita entrenarse para motivar a otras personas a que contribuyan a estructurar la discusión y asegurarse de que el grupo no sea dominado por uno o dos participantes. Los miembros de los equipos necesitan entrenamiento en técnicas básicas de calidad para saber detectar los problemas y la manera de resolverlos.

Un esquema alternativo para organizar los esfuerzos hacia la calidad —aparentemente desarrollado por Crosby—, es el *equipo para la mejora de la calidad (EMC)*. Este equipo representa el conocimiento, las habilidades y la experiencia multidisciplinaria dentro de la organización.

Un EMC tiene una vida limitada, —a diferencia de un círculo de calidad, que es continuo— pues su objetivo es la solución de un problema específico. Al igual que en un CC, en un EMC es importante crear un ambiente estimulante para la discusión y para promover a las personas a pensar creativamente. Debido a que el EMC se crea exprofeso para resolver un problema, las diferentes soluciones no óptimas debieron ser ya analizadas y rechazadas en otras instancias. De las ideas vertidas por los integrantes de este equipo en particular, seguramente saldrá la alternativa que sea el inicio de la solución.

Relacionado con el proceso de software, la conformación de equipos de mejora son particularmente útiles, especialmente en momentos de crisis, cuando la relevancia de un problema puede poner en riesgo no sólo al sistema, sino a la organización entera.

Por ejemplo, la operación de equipos centrales de cómputo suele ser bastante estable y bien soportada técnica y aplicativamente. Sin embargo, la complejidad de los sistemas operativos para tales equipos mainframe es alta y por tanto, muy especializada. Cuando existe un problema que afecta técnicamente al equipo, sin duda afectará por completo la integridad de los servicios que sustenta.

Así, un problema en el equipo central de un Banco puede repercutir de inmediato en los servicios que ofrece: cajeros automáticos, inversiones, proceso de cheques, etc., con las ineludibles consecuencias financieras. El problema presentado puede no ser inmediatamente detectable y una vez que se hayan puesto en práctica —o sólo sugerido— soluciones no exitosas, la conformación de un equipo de mejora de la calidad se hace indispensable.

Las soluciones ofrecidas por en EMC con frecuencia parten de *pensamientos laterales* que aparentemente no son la solución y que se confunden con aportaciones irrelevantes a la solución. De acuerdo con Gillies, muchos problemas intrincados son difíciles de resolver porque para llegar a la solución se necesita dar un paso en lo que aparentemente es la dirección equivocada (Figura 3-16).

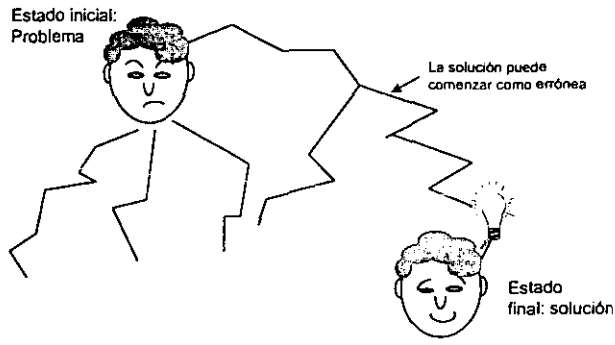


Figura 3-16 Las estrategias de solución a problemas no siempre son obvias

Juran es un ferviente promotor del trabajo en equipo como medio para motivar a la gente hacia la calidad. Esta práctica tiene algunas ventajas:

- Permite la solución de una variedad mayor de problemas
- Se dispone de una gran variedad de habilidades, conocimientos y experiencia
- Genera mayor satisfacción e incrementa la moral en el equipo
- Los problemas interdepartamentales pueden tratarse más fácilmente
- Las recomendaciones tiene mayor peso cuando provienen de un equipo

Como cualquier otra técnica, los EMC se ofrecen como una herramienta adicional. Por si mismos no causarán una revolución pero serán útiles cuando se conforme un programa global. Deben ser vistos como complemento de los círculos de calidad, y no como opciones en conflicto (Tabla 3-5).

Propiedad	Equipo para la mejora de la calidad	Circulo de calidad
Propósito	Convocar a expertos específicos para resolver un problema particular	Permitir a los trabajadores contribuir con sus ideas a la solución de los problemas
Componentes	De cinco a diez expertos interdisciplinarios	Hasta quince trabajadores de línea
Guiados por	Las personas más calificadas para resolver exitosamente el problema	Gerente o supervisor de línea
Tiempo de vida	Limitado	Permanente
Necesidades de capacitación	Necesita trabajar en equipo	Métodos básicos de calidad

Tabla 3-5 Cuadro comparativo de EMC y CC

Uno de los mayores problemas que aparecen en las organizaciones —basadas en quipos de trabajo o no—, es la falta de apreciación de la interdependencia de cada contribución a la solución. Por ejemplo, si una empresa se organiza en grupos de análisis de negocio e implementación de software, será muy necesario para ambos consultar al grupo de negocios para saber qué proyectos son factibles de desarrollar.

Esto apunta al concepto de clientes *internos* y *externos*. A partir de la concepción de Juran de adecuación para el uso, la calidad ha sido reconocida

como el cumplimiento de las necesidades del cliente, donde el cliente es una figura externa que hace los requerimientos.

Al mismo tiempo, existe una avanzada en el concepto de *cliente interno*, referenciado frecuentemente a una *cadena de calidad*. Un cliente interno es cualquiera que recibe un producto o servicio de otro grupo o individuo.

Si se considera el esquema tradicional de desarrollo en cascada, éste se representa como una cadena de calidad conformada por clientes internos, y finalizada por el cliente externo (Figura 3-17). Sin embargo, algunas veces la situación puede ser aún más compleja. Continuando con el ejemplo de los grupos de análisis de negocio e implementación mencionado antes, ellos desarrollan una relación cliente/proveedor en dos sentidos (Figura 3-18).

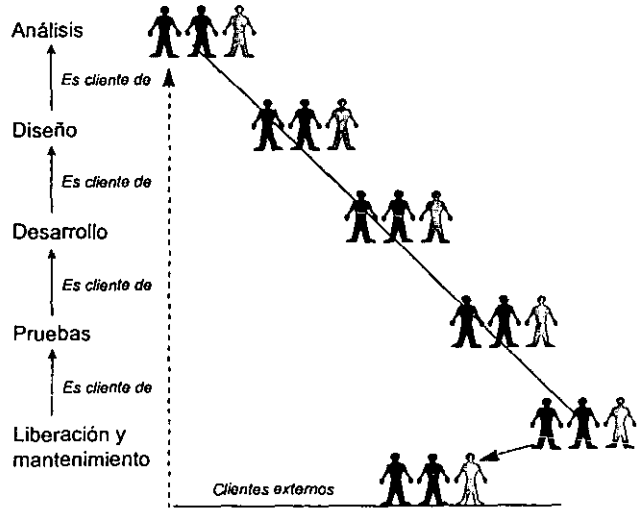


Figura 3-17 La cadena de calidad en el desarrollo tradicional de software

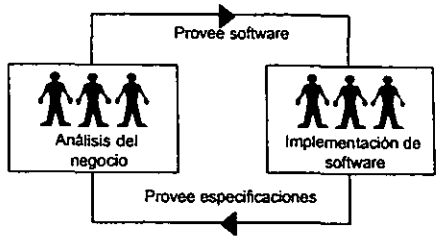


Figura 3-18 Los equipos pueden ser clientes uno del otro

Cuando se maneja cualquier situación, buena parte del éxito no está conectada con los procedimientos utilizados, sino con el día a día establecido en la organización. Los procedimientos pueden ayudar, pero también pueden constituir una traba para la solución; en cualquier caso no garantizan el éxito. El éxito en la aplicación de los procedimientos depende en buena medida de la efectividad de las personas que lo aplican. Los factores ambientales también influyen; éstos deben considerarse para luego tomar uno de dos posibles estereotipos que ayuden en el análisis de la situación.

¿Administración hacia abajo o invertida?

En cualquier organización donde se esté estableciendo una cultura para la calidad, existe tensión. La tensión existe entre una fuerza que proviene de la Dirección y otra que se genera desde los niveles inferiores de la organización. La fuerza proveniente de la Dirección es el deseo de administrar. La administración es absolutamente necesaria; no es posible alcanzar la calidad por consenso. Sin una administración firme no habrá políticas, estrategias ni consistencia en la toma de decisiones y se producirá el caos.

Al mismo tiempo se hace necesario nutrir a la organización con las ideas provenientes de los niveles inferiores. Una cultura para la calidad contribuirá para el logro de esto abriendo canales más fluidos. Una administración totalmente vertical puede convertirse en una autocracia, donde las ideas provenientes de los individuos no sólo sean frenadas sino, en muchos casos, reprimidas. Las

personas que tengan otras ideas diferentes a las de la Dirección pueden ser vistas como problemáticas, y no promoverán la generación de ideas en otras personas.

Se hace necesario un balance entre estructura, dirección y políticas por un lado, e innovación, pensamiento lateral y creatividad por el otro. Las visiones de la calidad que enfatizan el cumplimiento en los componentes, pueden fácilmente apuntar al cumplimiento cuando se maneja al staff.

Primer estereotipo: administración de personal

El primer carácter estereotipo que se considera es el cínico. Todos lo hemos visto, hecho u oído con anterioridad. Este tipo de personas pueden ser bastante destructivas en muchas situaciones, pero pueden también ofrecer algunos elementos:

- Generalmente es personal muy experimentado con una enorme carga de conocimiento que puede ser útilmente explotada.
- El papel de abogado del diablo puede ser extremadamente útil, especialmente donde se han empleado consultores externos.
- Pueden convertirse en eficaces defensores de una causa si están convencidos.

Una estrategia es tratar de guiar el papel de tales personas para explotar sus fortalezas de experiencia tratando de aislar a miembros más jóvenes del staff de las actitudes negativas.

Segundo estereotipo: el entusiasta

El entusiasmo es una característica muy valiosa pero suele causar tantos dolores de cabeza como los que resuelve. El entusiasmo suele ser efímero. La gente que se vuelve entusiasta tiende a aburrirse y adoptar la siguiente idea de moda. El entusiasmo puede conducir a las personas a no ser críticas y a no detectar peligros potenciales hasta que es demasiado tarde.

Parece atractivo poner a un cínico y a un entusiasta juntos, para que actúen en una balanza, y encontrar el justo medio. Este proceso, sin embargo, puede ser muy destructivo antes de que pueda dar frutos. En vez de encararlos, es más conveniente dejar que sus influencias sean mediadas a través de respectivos grupos de trabajo que se balanceen uno con el otro. Si este cambio puede darse sin hacer enfrentar directamente a estas dos figuras, el efecto será beneficioso.

El papel del entusiasta será el de alimentar a otros con ideas y entusiasmo. El grupo que se alimenta descarta muchas ideas al principio, pero tarde o temprano adoptará y desarrollará algunas ideas como suyas. Este proceso es muy importante pues se traduce en un entusiasmo creativo.

Afortunadamente la mayoría de las personas caen en un punto intermedio entre estos dos estereotipos. El balance entre promover la creatividad y mantener la estructura puede ser muy engañoso. Asegurarse de que cada quien tiene un papel en una organización orientada a la calidad representa un desafío y refuerza las visiones de los tres gurús en el sentido de que este desafío debe ser aceptado por la alta Dirección.

3.3.4 El problema de los requerimientos de usuario

El campo del desarrollo de software parece siempre estar en crisis, producto de su acelerada evolución. De hecho, se ha hablado de crisis del software desde 1960. Sea o no un panorama constante de crisis, lo cierto es que sí se puede detectar cuando la complejidad del software comienza a sobrepasar los métodos de diseño aplicados y los errores comienzan a evidenciarse a niveles inaceptables.

Esta situación ha sido un punto histórico para el software cuya resolución la aportó la ingeniería de software. Se puede hablar entonces, de una crisis que tuvo que ser resuelta porque los clientes (usuarios del software) no aceptaron la situación.

En los años recientes se ha comenzado nuevamente a hablar de tal crisis, donde los usuarios no han estado dispuestos a soportar la carga de errores mientras paga considerables cantidades de dinero por sus sistemas. Con el continuo avance tecnológico se ha creado una especie de cortina de humo para el usuario quien detecta que el error no es que el sistema no trabaje, sino que no hace lo que necesita. En términos de Juran, el sistema no encaja para el propósito previsto.

En nuestros días ha tomado gran importancia un aspecto que hace una década no era punto de discusión: el costo del software. Hoy las empresas hacen fuertes inversiones no sólo en tecnología sino en aplicaciones y con frecuencia se preguntan el tiempo necesario para que la inversión retorne. Si no existe evidencia financiera que sustente el desarrollo de software, con seguridad no se asignarán recursos a los proyectos. Hoy, más que nunca antes, el costo de los sistemas es una figura determinante en el proceso, y por tanto, en la calidad del mismo.

Gillies señala a éste respecto:

> *"Los días de los grandes ahorros a través de la automatización de tareas manuales repetitivas, se han ido, y el retorno de la inversión se proyecta ahora en términos de márgenes competitivos y otros conceptos nebulosos (...)".* [68]

Los clientes se han vuelto más perspicaces y escépticos y por tanto exigen una mayor calidad en el producto por el que pagan. Esta nueva crisis puede comprarse con su predecesora en la Tabla 3-6.

	Software en los años 60	Software en los años 90
El problema	la complejidad sobrepasa el diseño tecnológico	Los requerimientos de usuario sobrepasan la calidad con que se distribuye
La solución	Métodos estructurados conducen a herramientas y metodologías	¿Gestión de la calidad? ¿Herramientas de análisis para el negocio? ¿Prototipos? ¿Cultura para la calidad?

Tabla 3-6 Comparación. Crisis de los 60 y actual

68 Gillies, A., *op. cit.*, p.154

Muchas soluciones han sido propuestas pero es demasiado pronto para decir en la actualidad cuál de estas propuestas, si la hay, ha probado ser de ayuda para resolver la presente crisis.

El primer problema que aparece cuando se trata de analizar el problema es que se percibe diferente desde el panorama del usuario y del desarrollador. Las razones citadas por los desarrolladores incluyen:

- Los usuarios no saben lo que quieren
- Los usuarios no pueden expresar lo que necesitan
- Lo que los usuarios quieren no es factible
- Los requerimientos cambian demasiado rápido para ser cubiertos

Los usuarios, por el otro lado, ven las cosas diferente. Ellos enfatizan los presupuestos erogados en el software, las peticiones a los vendedores y la actitud adversa de muchos desarrolladores. Es crucial por lo tanto, llegar a una definición conjunta del problema que elimine actitudes adversas entre los protagonistas.

Los desarrolladores de herramientas y métodos han tratado de hacer de las representaciones gráficas un lenguaje común. Por ejemplo, han incorporado diagramas de flujo, de relación o jerárquicos. El uso de tales facilidades ha limitado el flujo de comunicación entre los participantes de un proyecto, aunque resulta claro que es más sencillo entrenar a un novato en la lectura de estos diagramas que en la escritura de los mismos en C.

El desarrollo de prototipos rápidos se ha sugerido también como una solución de cierto éxito[69]. El problema con los prototipos es saber cuando parar. Una vez que el usuario ve que el desarrollo puede ser alterado, siempre existe la tentación de ajustar una cosa más y la persona que aparece para detener ésta dinámica es vista como un ogro.

---

69 El éxito de esta técnica se ve afectado por el ambiente en el que se desarrolle. Por ejemplo, el desarrollo de sistemas grandes para equipos mainframe suele no ser un candidato de desarrollo de prototipos, aunque tales sistemas sean dignos candidatos para hablar de la calidad necesaria en el software.

En la práctica se puede argumentar que todas las técnicas han mostrado no ser generalmente aplicables o efectivas. Al final, cualquier técnica basada en los procedimientos terminará debatiéndose entre la falta de entendimiento del problema por todos los participantes, y el rápido incremento en el cambio de los requerimientos.

En otras palabras, es necesario no sólo proveer herramientas en el trabajo, sino incrementar la motivación para alcanzar la meta.

La gestión de la calidad adecuadamente aplicada ofrece una expectativa optimista para la mejora de la calidad del software porque apunta a actitudes tanto como a procesos. Si un sistema de gestión de calidad apunta solamente a procesos e ignora las actitudes del staff, no alcanzará su cometido.

3.4 Estándares de calidad

¿Cómo saber si un sistema de gestión de calidad incluye ciertos criterios o no? ¿Cómo garantizar que algunas tareas, como las revisiones o la documentación se hagan? ¿Cómo plasmar organizadamente las tareas ejecutadas para posteriores consultas?

Los estándares ayudan en la respuesta a estas cuestiones. No importa que estándar se utilice, éste se basa en un modelo que considera ciertos criterios a revisar y evaluar para garantizar su cumplimiento y por consiguiente, certificar su ejecución.

Los estándares dentro de los procesos de desarrollo de software —como en cualquier otro proceso de manufactura— ayudarán a establecer criterios que garanticen la ejecución de actividades tendientes a cubrir los niveles requeridos de calidad.

3.4.1 El propósito de los estándares

Los estándares se definen generalmente en términos de un modelo ideal que puede ser comparado con otros. Esto también aplica para los estándares de la calidad del software. En general, los estándares en este campo se basan en dos tipos de modelos:

- *Modelos generales.* La serie ISO9000 de estándares es de éste tipo. El modelo que se emplea no es específicamente para el desarrollo de software pero puede servir en un espectro amplio de aplicaciones desde la arquitectura hasta las líneas de manufactura.
- *Modelos de ingeniería de software.* Este tipo de estándares se basan específicamente en modelos de desarrollo de software, focalizándose en la adherencia a alguna metodología de desarrollo específica o a algún compilador o lenguaje en particular.

Los estándares establecen el modelo a ser empleado para luego ser certificado de alguna manera y así asegurar la aplicación de los mismos. La acreditación va desde las tareas de monitoreo internas en la misma empresa, hasta la certificación por entidades externas que revisan el cumplimiento. Esta acreditación no sólo crea confiabilidad en la empresa que la aprueba, sino que valida que sus procedimientos de gestión de la calidad están siendo efectivos.

Es importante aclarar y entender que los estándares no mejoran la calidad directamente, ni aseguran la perfección. Lo que si garantizan es que los procedimientos correctos están en el lugar adecuado y se llevan a cabo.

La preparación de un estándar es un proceso largo e interactivo que tiene el compromiso de condensar detalles. En muchos casos describen el común denominador de las técnicas y los métodos para las actividades básicas de aseguramiento de la calidad más que describir la tecnología misma.

Los estándares dan poca o nula información de *cómo* deben ejecutarse las actividades de aseguramiento de la calidad, pero si establecen *qué* debe hacerse.

A pesar de las sus deficiencias, los estándares para la calidad de software actuales son una contribución importante al proceso de gestión del software. Definen en un sentido práctico las tareas de aseguramiento para niveles directivos, desarrolladores de software, y en general todos los individuos responsables de llevar a cabo las tareas de aseguramiento de la calidad.

Establecen asimismo, un nivel mínimo de actividades que deben llevar las organizaciones que tengan un aseguramiento de la calidad del software. Son, al mismo tiempo, promotores de la consistencia de diferentes esfuerzos en el aseguramiento de la calidad en diferentes proyectos y organizaciones.

Un problema al que se enfrentan las empresas de desarrollo de software es el de la selección del estándar en particular que utilizará de entre los existentes. Generalmente los clientes determinan qué estándares deben aplicar sus proveedores, pero los clientes se enfrentan al mismo problema. En todo caso, la empresa de software debe decidir cuál estándar es aplicable al tipo de cliente (usuario final, proveedor de servicios, etc.) y utilizar el más conveniente.

3.4.2 La serie ISO9000: una administración genérica de la calidad

La serie de estándares ISO9000 es una serie de cinco estándares de calidad desarrollado por la ISO —*International Organization for Standarization*—. Estas series datan de 1979, pero han sufrido adecuaciones desde entonces siendo liberadas en 1987. La Tabla 3-7 muestra una breve descripción de cada serie de estándares.

En general, esta serie de estándares es óptima para aplicaciones donde el elemento diseño es importante. Dado que es relevante el diseño en el desarrollo de software, el ISO9001 es el más utilizado para el caso. ISO9002 se orienta al ambiente de manufactura donde el producto se fabrica bajo una especificación previa mientras que el ISO9003 es un subconjunto del ISO9002 y es útil para aplicaciones pequeñas donde la calidad puede ser determinada por una simple inspección final dentro de procedimientos de prueba.

ISO9000	Es una guía para seleccionar el estándar apropiado para un sistema de gestión de la calidad.
ISO9000-3	Gestión de la calidad y estándares para el aseguramiento de la calidad. Parte 3: Guía para la aplicación del ISO9001 al desarrollo y mantenimiento de software.
ISO9001	Es la especificación de un sistema de gestión de la calidad para el diseño, desarrollo, producción, instalación y servicio.
ISO9002	Especifica un sistema de gestión de la calidad para la producción e instalación.
ISO9003	Especifica un sistema de gestión de la calidad para la inspección final y las pruebas.
ISO9004	Es una guía para el cumplimiento de los estándares ISO9001/2/3

Tabla 3-7 La serie ISO9000 de estándares para la gestión de la calidad

Esta serie provee dos estándares de guía y tres contractuales orientados a ser una guía para la gestión de la calidad y el aseguramiento de la calidad contractualmente por una segunda figura.

El Comité Técnico 176 de la ISO (TC 176) ha producido varias directrices para aplicar la serie ISO9000 para el software. Este comité ha destacado primero la aplicación de y certificación de la ISO9000-3, que es una guía para la aplicación de la ISO9001 para el desarrollo, mantenimiento y provisión de software. La Figura 3-19 muestra gráficamente la posición de los estándares.

La adopción de un estándar ampliamente aceptado como el ISO9001[70] para el caso del desarrollo de software, tiene sin embargo, algunas complicaciones. El ISO9001 es percibido erróneamente como un estándar de manufactura y los proveedores de software no creen —generalmente— en la acreditación como una ventaja competitiva.

70 Varios países han adoptado la serio ISO9000 como suya para estándares locales. En esencia contienen las mismas secciones, pero adecuadas al entorno particular.

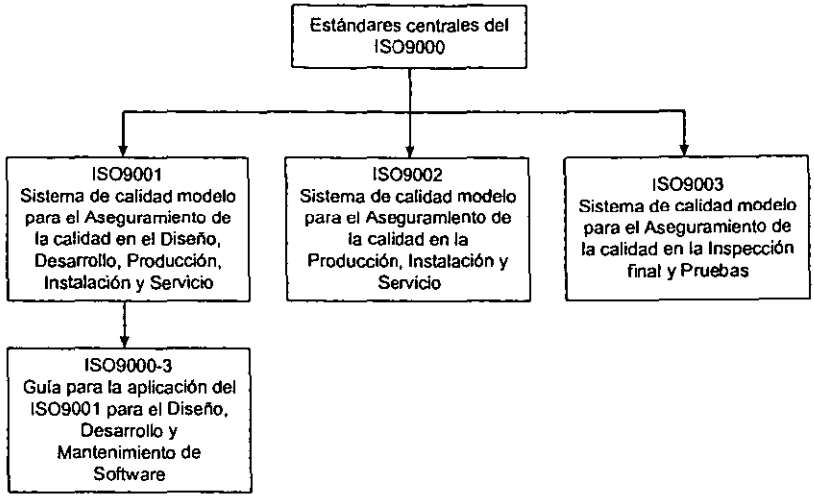


Figura 3-19 Estándares centrales ISO9000

El contenido del ISO 9000-3

Por la relevancia que éste estándar tiene en la aplicación del ISO9001, se presenta en forma resumida el contenido del mismo.

1. Alcance.
2. Definiciones normativas.
3. Definiciones.
4. Sistema de calidad. Estructura.

    4.1 Responsabilidades de la administración.

        4.1.1 Proveedores.

        4.1.2 Compradores.

        4.1.3 Revisiones conjuntas.

4.2 Sistema de calidad.

4.2.1 General.

4.2.2 Documentación del sistema de calidad.

4.2.3 Plan de calidad.

4.3 Auditorias internas al sistema de calidad.

4.4 Acciones correctivas.

5. Sistema de calidad. Ciclo de vida.

5.1 General.

5.2 Revisión contractual.

5.3 Especificación de requerimientos de los clientes.

5.4 Plan de desarrollo.

5.5 Planeación de la calidad.

5.6 Diseño e implementación.

5.7 Pruebas y validación.

5.8 Aceptación.

5.9 Copiado, distribución y aceptación.

5.10 Mantenimiento.

6. Sistema de calidad. Actividades de soporte..

6.1 Gestión de la configuración.

6.2 Control de la documentación.

6.3 Registros de calidad.

6.4 Medición.

6.5 Reglas, prácticas y convenciones.

6.6 Herramientas y técnicas.

6.7 Compras.

6.8 Inclusión de productos de software.

6.9 Capacitación.

El ISO9001

El estándar se basa en la especificación de un modelo para un sistema de gestión de la calidad. El modelo en general, se basa en dos principios básicos:

- Hacer las cosas desde la primera vez
- Cumplir con los propósitos

El estándar intenta ser realista e implementable, por lo que no establece cuotas de calidad, siendo más bien un acuerdo entre las partes donde el cliente acepta los términos establecidos. El estándar se enfoca en que los procedimientos se llevan a cabo de manera sistemática y que los resultados son documentados también de forma sistemática.

La cláusula 4 del estándar contiene los requerimientos principales y contiene 20 subcláusulas. La Tabla 3-8 muestra los encabezados de cada cláusula y se indica si se encuentran también en los ISO9002 y 9003.

El modelo reconoce la importancia que tiene la responsabilidad de los puestos de administración para que la calidad se implante a través de toda la organización. Mientras que para la alta dirección es imposible revisar personalmente todo, el

estándar designa explícitamente a un representante que es responsable directo de la calidad y que le reporta a la Dirección.

Cláusula	ISO9001	ISO9002	ISO9003
4.1	Responsabilidad de la administración	✔	✔
4.2	Sistema de calidad	✔	✔
4.3	Revisión del contrato	✔	
4.4	Control del diseño		
4.5	Control de la documentación	✔	✔
4.6	Compras	✔	
4.7	Producto entregado al cliente	✔	
4.8	Identificación del producto	✔	✔
4.9	Control de procesos	✔	
4.10	Inspección y pruebas	✔	✔
4.11	Inspección, medición y prueba de equipo	✔	✔
4.12	Inspección y pruebas	✔	✔
4.13	Control de las no-conformidades del producto	✔	✔
4.14	Acciones correctivas	✔	
4.15	Manejo, almacenaje, empaque y distribución	✔	✔
4.16	Registros de calidad	✔	✔
4.17	Auditorías internas para la calidad	✔	
4.18	Entrenamiento	✔	✔
4.19	Servicio		
4.20	Técnicas estadísticas	✔	✔

Tabla 3-8 Comparación de los requerimientos en los tres principales estándares

Cláusula 4.1: Responsabilidad de la administración.

Este apartado apunta los principios para el establecimiento del sistema de calidad dentro de la organización y establece de manera general sus funciones, que luego se detallan en secciones posteriores del mismo estándar.

Cláusula 4.2: Sistema de calidad.

El modelo requiere que la organización cuente con un sistema de calidad. El sistema se debe documentar y se debe preparar un plan y un manual para la calidad. El alcance de este plan se determina por las actividades relacionadas y en consecuencia se aplica el ISO9000/1 /2 /3. El objetivo del plan es asegurarse de que las actividades se llevan a cabo sistemáticamente y que se documentan.

Cláusula 4.3: Revisión del contrato.

Este apartado especifica que cada pedido del usuario debe tratarse como un contrato. Se deben establecer y documentar procedimientos para la recepción de los requerimientos. El objetivo de estos procedimientos es:

- Asegurarse de que los requerimientos del usuario han sido claramente definidos por escrito.
- Señalar las diferencias entre la orden y la petición inicial, para llegar a acuerdos.
- Asegurarse de que los requerimientos pueden ser atendidos.

El objetivo de esta cláusula es asegurarse que tanto cliente como desarrollador entienden los requerimientos especificados para cada orden y documentar los acuerdos para prevenir conflictos y malos entendidos posteriormente.

Cláusula 4.4: Control del diseño.

Los procedimientos para el control del diseño se establecen para verificar las actividades de diseño, tomando resultados de investigaciones de mercado. Las actividades clave que comprende son:

- Planeación para la investigación y el desarrollo.
- Asignación de actividades al personal calificado.
- Identificación de interfaces entre grupos relevantes.
- Preparación de un resumen del diseño.

- Producción de datos técnicos.
- Verificación de que las salidas de la fase de diseño cumplen con los requerimientos de entrada.
- Identificación y documentación de todos los cambios y modificaciones.

El objetivo es asegurarse de que la etapa de diseño se lleve a cabo correctamente así como asegurarse de que las salidas cumplan con lo especificado en las entradas. Esta es una parte importante en el desarrollo del software, que de ninguna manera debe subestimarse.

Cláusula 4.5: Control de la documentación.

El estándar reconoce tres niveles de documentación:

- Nivel 1: Documentos de políticas y planeación
- Nivel 2: Procedimientos
- Nivel 3: Instrucciones detalladas

El nivel más alto de documentación opera en el plan de calidad y establece políticas en temas relacionados con la calidad. Los niveles siguientes agregan detalle a los documentos, que deben ser incorporados siempre que sea posible.

La documentación debe ser sistemática, cuidando de no emitir documentos sólo porque sí. Los documentos tienen que ser consistentes con los del nivel superior y tendrán más detalle mientras más bajo sea al nivel de los mismos.

Tal parece que los estándares requieren una cantidad inalcanzable de documentación que, sin embargo, puede actualmente ser reducida al mínimo removiendo documentos cuya información sea redundante. Asimismo, la documentación ya existente puede reutilizarse cuando se crea un nuevo sistema ya que el estándar no requiere un formato especifico sino un documento que cumpla con lo establecido.

Cláusula 4.6: Compras.

El sistema de compras se implementa para asegurarse de que todos los productos y servicios que adquiere la organización cumplen con sus estándares. Se hace énfasis en verificar los procedimientos de calidad de los proveedores. Esta tarea se simplifica si los proveedores tienen alguna certificación en algún estándar. Como en los otros procedimientos, éste también se documenta.

Cláusula 4.7: Producto entregado al cliente.

Todos los productos y servicios ofrecidos al cliente deben ser verificados de la misma forma que se hace cuando los productos o servicios los entrega un proveedor. Para lograr esto, los procedimientos se deben poner en marcha y documentar, para que tales productos y servicios puedan ser rastreados a lo largo del proceso.

Cláusula 4.8: Identificación del producto.

Para asegurar un control de procesos efectivo y corregir cualquier no conformidad es necesario establecer procedimientos para identificar y rastrear materiales (o servicios) desde la entrada hasta la salida. Esto permite establecer las raíces de los problemas de calidad; si se puede rastrear un material, puede que la falla radique en él y por lo tanto el problema se encuentre fuera del sistema de calidad en sí. Lo importante es habilitar el camino que ha de recorrerse para cada producto.

Cláusula 4.9: Control de procesos.

Esta parte requiere un pleno conocimiento del proceso. Este debe ser documentado, generalmente de manera gráfica, como un diagrama de flujo o algo equivalente. Los procedimientos de inicialización o de calibración deben también registrarse.

La documentación de cada proceso debe estar disponible para todo el personal para asegurarse de que las tareas se lleven a cabo de la manera especificada. En este punto cabe mencionar que con frecuencia las

organizaciones no tienen bien entendidos sus procesos. La disciplina de documentarlos, constituye un paso educacional para la certificación.

Cláusula 4.10: Inspección y pruebas.

La inspección y pruebas se requieren para garantizar la conformidad en tres estados:

- Materiales o servicios de entrada
- En proceso
- Producto o servicio final

Todos los servicios o productos de entrada deben ser verificados de alguna manera. El método variará de acuerdo al status de los procedimientos de gestión de la calidad del proveedor y pueden ir desde una revisión exhaustiva de los bienes o servicios entregados.

Es necesario el monitoreo *en proceso* para asegurarse de que todo va de acuerdo al plan. Al final del proceso se debe efectuar la documentación de toda prueba e inspección final.

Cláusula 4.11: Inspección, medición y prueba de equipo.

Todo equipo que se utilice para medir y probar debe estar sujeto a mantenimiento y calibración. Los procedimientos que aseguren que estas actividades se cumplan deben estar implementados y documentados, estableciendo las métricas requeridas y la precisión de las mismas.

Las actividades de calibración deben ser incorporadas como parte del mantenimiento regular. La administración debe asegurarse de que estas actividades se efectúen en intervalos regulares y que se registren los resultados.

Cláusula 4.12: Inspección y pruebas.

Todo material o servicio debe clasificarse en una de tres categorías:

- Esperando inspección de pruebas
- Inspección aprobada
- Inspección fallida

El status debe ser identificable claramente en cada estado. Es importante que los materiales o servicios que esperen inspección no pasen por ésta y deje de detectarse una no conformidad.

Cláusula 4.13: Control de las no-conformidades del producto.

El estándar define la no conformidad del producto como todos los productos o servicios que caen fuera de los límites de tolerancia acordados de antemano con los clientes. Los productos o servicios con no conformidad deben ser claramente identificados, documentados y si es posible, separados físicamente de los productos que si cumplen la conformidad.

Cláusula 4.14: Acciones correctivas.

Las acciones correctivas son la clave para la mejora continua. Tales acciones deben ser implementadas por medio de un programa sistemático que provea una guía y defina las obligaciones de las partes involucradas. Se debe llevar un registro de las acciones tomadas para que futuras auditorías puedan investigar su efectividad.

Cláusula 4.15: Manejo, almacenaje, empaque y distribución.

El manejo y las actividades asociadas se deben diseñar para proteger la calidad lograda en el producto. Si existen sub-contratos para transporte, por ejemplo, deben contar con los mismos procedimientos documentados como si fueran empleados internos. El alcance de esta cláusula está determinado por el contrato establecido con los clientes y cubre todas las actividades que son las obligaciones contractuales del proveedor.

Cláusula 4.16: Registros de calidad.

Los registros son vitales para asegurarse de que las actividades relacionadas con la calidad han sido llevadas a cabo realmente. Estos registros son la base para las auditorías tanto internas como externas. Si bien no se apegan a un formato específico, si deben cumplir con su cometido. Ya que mucha información puede existir incluso antes de que se establezca un sistema para la calidad, el objetivo es sistematizar y asimilar las prácticas existentes, siempre que sea posible para reducir el esfuerzo de reproducir el trabajo anterior.

Cláusula 4.17: Auditorías internas para la calidad.

El sistema para la calidad debe estar revisado desde dentro de la organización y no depender de inspecciones externas. Los procedimientos deben establecer auditorías internas regulares como parte de los procedimientos generales de administración. El papel de las auditorías internas es el de identificar problemas anticipadamente, para minimizar el impacto y el costo.

Cláusula 4.18: Entrenamiento.

Las actividades de capacitación deben implementarse y documentarse. En especial se requieren documentos escritos para :

- Establecer necesidades de capacitación.
- Llevar a cabo las actividades de entrenamiento.
- Contar con un registro de los requerimientos de capacitación y el avance que se ha tenido en este rubro para cada miembro del staff.

Un requerimiento del estándar es que en todos los niveles, el personal requerido para desarrollar una función particular, cuente con las habilidades y el conocimiento necesario para hacer el trabajo adecuadamente. En este sentido la capacitación no sólo se refiere a cursos formales, sino también a la transmisión de los conocimientos de manera personal.

Cláusula 4.19: Servicio.

Donde se requieran actividades de servicio, los procedimientos deben documentarse y verificarse. Los procedimientos deben garantizar que los servicios se lleven a cabo adecuadamente y que existan suficientes recursos para desarrollar tales actividades. Es necesario establecer interfaces efectivas con el usuario para que la actividad de servicio —o mantenimiento, en el caso de sistemas— se complete de manera satisfactoria.

Cláusula 4.20: Técnicas estadísticas.

Las técnicas estadísticas deben aplicarse donde sea apropiado. El estándar no especifica técnicas o métodos en particular a ser utilizados, pero si establece que los que se utilicen deben ser adecuados para el propósito establecido. El uso de la estadística será necesario para satisfacer otros requerimientos, en especial el control de procesos mencionado en la cláusula 4.9.

3.4.3 Certificación en ISO9000[71]

Hay una serie de razones que las empresas han visto para certificarse bajo un sistema ISO9000. Estas razones incluyen mejoras internas, posicionamiento en el mercado, control de proveedores o cumplimiento de requerimientos legales.

A pesar de tales razones bien fundadas, el número de firmas que se han certificado en EU es todavía menor. En Europa y Australia apenas son cientos las empresas certificadas mientras que en Inglaterra se ha implementado un premio codiciado entre las mejores firmas con un sistema de aseguramiento de la calidad del software. Empresas como Oracle y Hitachi Software han ganado este premio que establece como requerimiento que la empresa este certificada bajo ISO9001.

En Europa la certificación es vista como una herramienta de mercadeo y como una forma para mejorar los procesos del software.

---

[71] Las ideas expuestas han sido tomadas de "The way to ISO9001 Certification for Software Firms", por Stan Magge y publicado en Noviembre de 1996 por Datapro.

En EU la visión tiende a ser sólo mercantilista impulsada por ímpetu económico.

Muchas de las grandes empresas de software en EU requieren que sus asociados estén certificados en ISO. Tal es el caso de AT&T e IBM. Dentro de la Unión Europea, los Gobiernos solicitan la certificación para el concurso de contratos. Lo mismo ocurre en algunos proyectos del Gobierno de EU, donde solicita que las empresas participantes cuenten con un sistema de calidad que sea equivalente al ISO9001.

Como punto de venta, varias empresas que han sido certificadas, utilizan esto como una credencial en sus estrategias de venta en las publicaciones. Tanto ellos como sus clientes ven la certificación como un factor que los distingue de sus competidores.

El ISO9001 puede ser un incentivo poderoso para que las organizaciones tengan sus procedimientos de calidad correctos. La acreditación constituye una evidencia externa de este hecho.

Aparte de los problemas que se han presentado en la comunidad de desarrollo de software por la aceptación del estándar como válido o apropiado, existen algunas limitaciones del mismo. Por ejemplo, debido a su naturaleza, el estándar no es prescriptivo, es decir, no especifica ningún método o herramienta a ser utilizado en cada etapa.

Sin embargo, la mayor preocupación del modelo de un sistema de calidad que sea el corazón del ISO9001, debe ser el énfasis sobre los procedimientos de control de calidad. Hay muy poco en el estándar relacionado con el establecimiento de una cultura humana para la calidad. Con frecuencia se argumenta que esta cultura está implícita dentro del modelo y que es menester cubrir los otros requerimientos.

El factor humano es demasiado importante como para ser considerado un requerimiento implícito: su omisión deja al estándar abierto a las acusaciones en el sentido de que es simplemente un sistema de registro. Sin el establecimiento

de una cultura para la calidad y requerimientos formales traducidos en procedimientos que faciliten esta causa, el proceso vital de mejora continua que lleve a la gestión de la calidad más allá del registro de errores y performance, corre el riego de ser omitido (Figura 3-20).

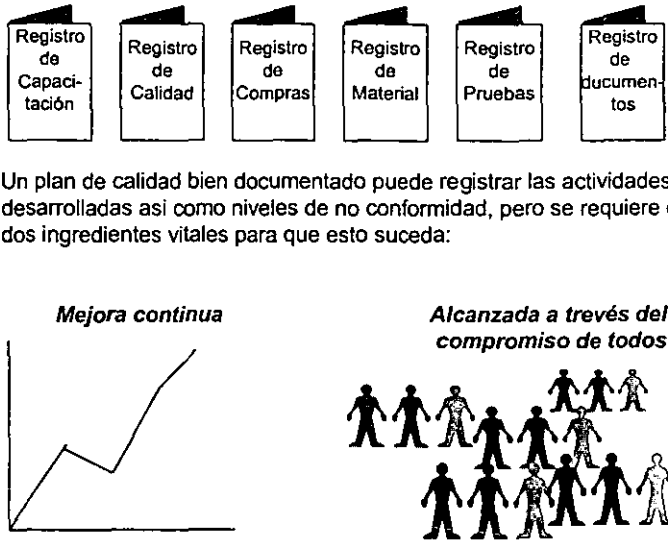


Figura 3-20 Sin el compromiso para la mejora, el ISO9001 puede devaluarse

3.4.4 Desarrollos recientes e impacto

Una de las barreras más importantes para la aceptación del ISO9001 entre la comunidad de software, es su naturaleza genérica como un estándar de manufactura. Aunque el estándar ha sido aplicado en el área de servicios, muchos encuentran que es inapropiado y difícil de aplicar en el área del desarrollo se software.

El ISO9000-3

Para responder a esto se han incluido algunas notas para la aplicación del estándar ISO9001 en el desarrollo de software. Estas notas no invalidan al estándar, sino que lo amplifican para explicar la manera en que el estándar debe aplicarse en el contexto del software. Estas notas son conocidas como ISO9000-3 y fueron establecidas en 1991.

Las notas ISO9000-3 están dirigidas a la comunidad relacionada con el software. Se considera como un solo documento y su estructura difiere del ISO9001. La estructura del ISO9000-3 es como sigue:

- *Introducción.* Las tres primeras cláusulas del estándar definen el alcance del mismo, hacen referencia a otros estándares y definen términos utilizados en el ISO900-3
- *Sección 4. Sistema de calidad - estructura.* Esta parte contiene cuatro sub-secciones: responsabilidad de la administración, sistema de calidad, auditoría interna para la calidad y acciones correctivas.
- *Sección 5. Sistema de calidad - actividades del ciclo de vida.* Esta parte contiene nueve secciones relacionadas con una o más partes del ciclo de vida. Aquí las actividades son definidas en detalle y a diferencia del ISO9001, son substanciales cuando se aplican al software.
- *Sección 6. Sistema de calidad - actividades de soporte.* Contiene nueve elementos que cubren el resto de las actividades. Algunas importantes, como la gestión de la configuración, son apenas mencionadas en el ISO9001. Otras actividades que incluye son mediciones, reglas, prácticas y convenciones, así como técnicas y herramientas. Mucho del contenido de esta sección hace explícitos los requerimientos implícitos del ISO9001.

La Tabla 3-9 resume los encabezados para el ISO9000-3 y los relaciona con sus correspondientes para el ISO9001, clasificando el grado de guía mostrado, como ninguno, menor, significativo o mayor.

Encabezado de sección	Sub-Sección	Titulo	Cláusula en el ISO9001	Adición al ISO9001
	4.1	Responsabilidad de la administración	4.1	Significativo
4. Sistema de	4.2	Sistema de calidad	4.2	Significativo
Calidad -	4.3	Auditorias internas de calidad	4.17	Menor
estructura	4.4	Acciones correctivas	4.14	Menor
	5.1	Revisiones contractuales	4.3	Significativo
	5.2	Especificación de los requerimientos del comprador	a, 4.4	Significativo
5. Sistema de	5.3	Planeación para el desarrollo	4.4.2	Significativo
Calidad -	5.4	Planeación de la calidad	4.2	Significativo
actividades del	5.5	Diseño e implementación	4.4, 4.9	Significativo
ciclo de vida	5.6	Pruebas y validación	4.10, 4.13	Significativo
	5.7	Aceptación	4.10, 4.13	Significativo
	5.8	Copiado, distribución e instalación	4.15	Significativo
	5.9	Mantenimiento	4.19	Mayor
	6.1	Gestión de la configuración	4.4, 4.8, 4.5	Mayor
	6.2	Control de documentos	4.5	Significativo
	6.3	Registros de calidad	4.16	Ninguno
6. Sistema de	6.4	Mediciones	4.20	Mayor
Calidad -	6.5	Reglas, practicas y convenciones	4.9, 4.11	Significativo
Actividades de	6.6	Herramientas y técnicas	4.9, 4.11	Significativo
soporte	6.7	Compras	4.6	Menor
	6.8	Productos de software incluidos	4.7	Significativo
	6.9	Capacitación	4.18	menor

Tabla 3-9 ISO9000-3 notas para la aplicación del ISO9001 al desarrollo de software

Las áreas centrales del ISO9000-3 son la definición de los requerimientos, la definición del ciclo de vida, la gestión de la configuración y las mediciones. El estándar considera como caso especial al software porque:

- Es un objeto intelectual.
- El proceso de desarrollo tiene sus propias características.
- Replicar el software produce copias exactas.
- Una vez que una falla se repara, no vuelve a ocurrir.
- Como producto, el software es único.

A pesar de las diferencias, las notas ofrecidas por el ISO9000-3 no constituyen un estándar diferente y son concebidas como una ayuda para que los usuarios del ISO9001 lo apliquen en el ambiente de software. Incluso, todavía se certifican los sistemas de calidad con el ISO9001.

ISO/IEC 12207

La aplicación del ISO 9000-3 como caso particular para el desarrollo de software llevó a varios países a establecer continuas revisiones sobre el estándar y al mismo tiempo establecer adecuaciones particulares. Especialmente Australia, Inglaterra y Japón hicieron contribuciones que complementaron lo establecido en el ISO9001, pero para el caso del software.

En 1993, cuando aparentemente se dejaría de revisar el ISO 9001, se generó un largo debate para decidir si el ISO 9000-3 debía ser actualizado para reflejar cambios hechos al ISO 9001 o si debía ser reformado por completo para seguir la idea del ISO 9001 y revisar cláusula por cláusula.

La decisión final se tomó en 1996 para realinear el documento del ISO 9000-3 con el 9001 y al mismo tiempo alinearlo con el ISO/IEC 12207 (1995) conocido como Ciclo de Vida para los Procesos de Ingeniería de Software. Esto se hizo dando algunas guías en las cláusulas del ISO 9001, siempre que fuera posible.

Donde se requería información a detalle, el documento del ISO 9000-3 dirige al lector a la cláusula correspondiente el ISO/IEC 12207.

El trabajo que se ha tenido sobre los estándares se ha conjuntado en un proyecto llamado *Software Process Improvement and Capability dEtermination* (SPICE). El modelo creado en este esfuerzo es similar al modelo de madurez de procesos mostrado en el Anexo B. Cada uno de estos estándares y guías tienen como objetivo dar a conocer a la industria del software y profesionales relacionados la manera de mejorar la calidad en sus productos.

Aunque en la composición futura de los documentos de la serie 9000 se encuentra actualmente en discusión, esta serie se convertirá en la documentación genérica para los sistemas de calidad de software e incorporará los requerimientos y guías dadas en el 9001, 9002 y 9003. Los requerimientos genéricos de calidad en 9000-3 se incorporarán en la nueva 9001, y otros requerimientos para el software quedarán en el ISO/IEC 12207 como las *mejores prácticas* en los procesos del ciclo de vida del software, una vez que termine su aprobación en el año 2000.

La Figura 3-21 muestra la relación de los estándares 9000 con otros estándares a lo largo del tiempo, mientras que la Figura 3-22 muestra el desarrollo progresivo de los mismos.

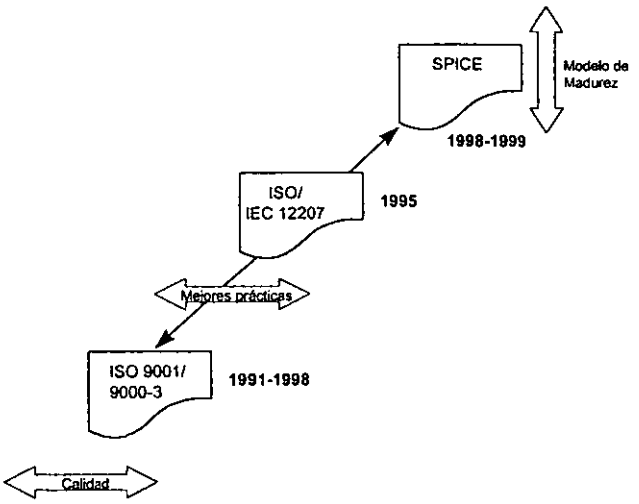


Figura 3-21 Relación del ISO 9000-3 con otros estándares

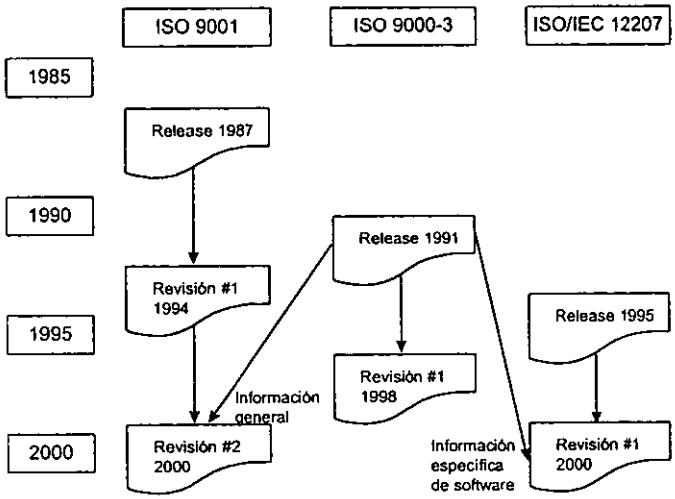


Figura 3-22 Cronología de ISO 9001, 9000-3 y 12207

4

FUTURO DE LA CALIDAD DEL SOFTWARE

"El costo de la calidad del software, es realmente el costo total de la NO calidad del mismo"

W.E. Deming

4.1 La producción de software

En la creación del software cobran importancia los elementos de ingeniería involucrados, especialmente los procesos y el nivel de control de los mismos. La combinación de ambos establece un nivel de madurez en que se encuentra la organización que produce software, permitiendo ubicarse dentro del marco de la calidad.

¿Cómo se traducen los esfuerzos de mejora continua de la calidad en el proceso creativo del software? El concepto de *fábrica de software* retoma el camino hacia la calidad en el futuro cercano del desarrollo de software.

La fábrica de software, como organización conocedora de la esencia de sus procesos, motivadora del cambio y la participación de sus integrantes, promotora

de la mejora continua y cuidadosa de los costos de producción representa un modelo donde la calidad es fundamental.

4.1.1 La fábrica de software

En la ingeniería de software puede hablarse de tres estados de control, cuya diferencia radica en el objeto del control mismo. Estos estados pueden llamarse *control del desarrollo*, *control del producto* y *control multiproducto*.

En el control del desarrollo, el objeto de control se limita únicamente a las fases de desarrollo de sistemas. El control del producto extiende el objeto de control al ciclo de vida de sistemas completo, es decir, incluye las fases de puesta en marcha, instalación y mantenimiento continuo del sistema, hasta su obsolescencia.

Finalmente, el objeto de control para el control multiproducto se extiende de uno, a varios productos. En este caso caen las organizaciones o departamentos que no solamente son responsables de un solo sistema, sino de varios en su ciclo de vida completo. La organización debe ser capaz, en este estado, de soportar los sistemas de una manera óptima, para lo cual, el reuso de software juega un papel relevante[72].

La Figura 4-1 muestra gráficamente estos tres estados.

Una organización que se encuentra en el estado de control multiproducto se ha tipificado como *fábrica de software*.

En la actualidad muchas organizaciones donde se desarrolla software, tienden a la aplicación del concepto de fábrica de software, ya que no desarrollan solamente un producto sino varios, dándole soporte continuo durante todo el ciclo de vida del sistema, desde su análisis hasta su reemplazo por uno nuevo. Sin

---

[72] Van Genuchten, de hecho, considera al reuso de software como promotor del control multiproducto, pues permite no sólo la posibilidad de crear diferentes sistemas a partir de componentes reutilizables que garanticen de alguna manera el nivel de calidad, sino la garantía de un mejor servicio a todos los sistemas maximizando la funcionalidad de los mismos a costos bajos.

embargo, se debe considerar que especialmente en el control multiproducto, el uso de recursos se optimiza, involucrando un cambio total en la organización. El manejo de presupuestos que garanticen el menor costo y produzcan los sistemas de más calidad, junto con un intensivo uso de herramientas —metodologías, reuso de componentes— como parte de la cultura, son inherentes a las empresas en este estado.

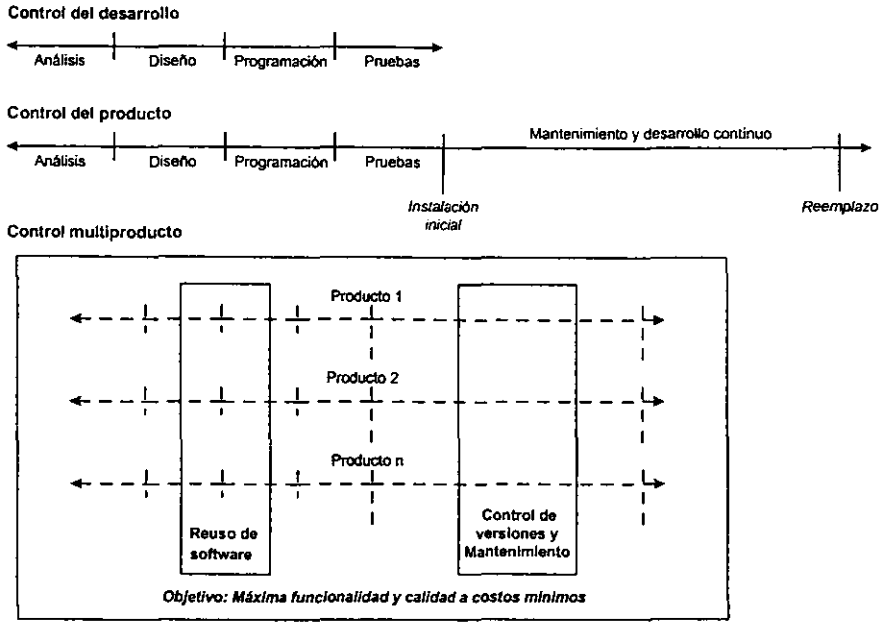


Figura 4-1 Estados en el control del software

Podría pensarse que el desarrollo en una organización inercialmente provoca el cambio de un control del desarrollo hacia el control del producto y posteriormente al control multiproducto y que es sólo cuestión de tiempo. El tiempo por sí solo no determina la evolución, pero por el contrario, si puede influir en retroceder cuando los esfuerzos hacia el control multiproducto dejan de ser constantes.

4.1.2 Procesos de software

El Anexo B muestra el modelo del SEI con cinco estados que diferencian la calidad de los procesos involucrados en la creación de software. En esta sección se detallan los niveles de control de procesos como medio para llegar al modelo de fábrica de software.

Humprey[73] define un proceso como un conjunto de tareas que al ejecutarse adecuadamente, producen el resultado deseado. Al mismo tiempo, destaca que el primer paso importante para consignar problemas en el software, es tratar a la ingeniería de software como un proceso que puede ser medido, controlado y mejorado.

El control estadístico establecido por Deming puede ser llevado a la ingeniería de software. Deming establece que una vez que el proceso está bajo control estadístico, se pueden alcanzar resultados consistentemente mejores sólo con la mejora del proceso. Deming aplicó estas ideas en la manufactura; Humprey asegura que aunque existen diferencias evidentes en los campos de la manufactura y del software, no existe razón que impida la aplicación de las ideas de Deming en el campo del software.

El modelo de madurez organizacional del SEI se estableció para tipificar las capacidades de las organizaciones que desarrollan software. Esta estructura también es utilizada para ubicar el estado de la organización e identificar las áreas más importantes para mejoras. La Tabla 4-1 resume los niveles establecidos en el modelo, incluyendo la característica principal necesaria para pasar al siguiente estado.

Una explicación detallada de cada nivel se encuentra en el Anexo B. Aquí se mostrará un resumen con las características y problemática de los diferentes niveles, en la Tabla 4-2.

---

73 Humprey, W.S., (1989), Managing the software process, Inglaterra: Addison Wesley.

Nivel del control del proceso	Requerimiento para pasar al siguiente nivel
Optimizado	
Administrado	Control de procesos
Definido	Medición de procesos
Repetible	Definición de procesos
Inicial	Control básico administrativo

Tabla 4-1 Los niveles de control del proceso

Nivel	Característica	Areas problemáticas
Optimizado	Mejora y retroalimentación de procesos	Automatización
Administrado	(Cuantitativo) Procesos medidos	Análisis de problemas Prevención de problemas
Definido	(Cualitativo) Procesos definidos e institucionalizados	Medición de procesos Análisis de procesos Planes de calidad cuantitativos
Repetible	(Intuitivo) Los procesos dependen de los individuos	Capacitación Prácticas técnicas (revisiones, pruebas) Focalizar procesos (estándares, grupos de procesos)
Inicial	(Ad hoc, caótico)	Administración de proyectos. Planeación de proyectos Gestión de la configuración Aseguramiento de la calidad del software.

Tabla 4-2 Características de los niveles de control

La distinción entre los diferentes niveles no debe ser interpretada en un sentido absoluto. Por ejemplo, puede ser que la medición de procesos se aplique en el primer nivel, o que la gestión de la configuración sea todavía un problema durante el nivel de procesos repetible. Obviamente no se espera que las empresas cambien uno o varios niveles de inmediato tomando en cuenta las características mostradas arriba, pero si da una idea de los diferentes niveles de

control de procesos en la ingeniería de software y ofrece elementos para que una organización avance de un nivel a otro.

Considerando los estados de control de desarrollo, control del producto y control multiproducto ¿cuál es el camino a seguir para avanzar de un estado al siguiente?. La ruta que se debe seguir hacia el modelo de fábrica de software considera varios preceptos. Inicialmente, no es sólo cuestión de tiempo avanzar de un nivel inferior a uno superior, aunque bajar de nivel, puede sí serlo. En el avance de niveles se destaca la acción de dos importantes factores: una decisión explícita de la organización para el cambio, y un cierto nivel de control de procesos alcanzado en términos del modelo.

La decisión necesaria para avanzar de un estado a otro define la esencia de la organización. Puede no ser indispensable para todas las organizaciones avanzar del control del desarrollo hacia el control del producto o control multiproducto. Por ejemplo, una empresa puede limitarse a controlar el desarrollo, ofreciendo servicios en ese universo exclusivamente, y atacar así ese nicho del mercado.

Una organización pude circunscribirse al desarrollo y mantenimiento de un producto, o de una familia ya definida de productos, por lo que el estado de control del producto sería suficiente.

Otro requerimiento importante en el avance hacia el modelo de fábrica de software es contar con cierto grado de control de procesos. Así, por ejemplo, una empresa que no cuente con un control del desarrollo para un solo producto, seguramente no podrá controlar el desarrollo y el mantenimiento al mismo tiempo para el mismo producto.

El nivel más básico que se ha definido es el de desarrollo. Este estado requiere que la organización opere en el nivel repetible de control de procesos. El nivel inicial se caracteriza por ser caótico. Las empresas que operan en el nivel inicial de control de procesos con frecuencia trabajan los proyectos sin planes, con un costeo de los mismos después de hechos y con procedimientos de trabajo aceptados y sin cambios.

Por consiguiente, no se evalúan los proyectos conforme a los planes, dejando de lado el control en lo que se refiere a calidad, tiempo y costo. La introducción de habilidades de administración es un requerimiento mínimo para alcanzar un nivel repetible de control de procesos. Ejemplos de esta administración son el control de cambios y la planeación de proyectos. El mismo tipo de habilidades se requieren para controlar proyectos de desarrollo en el estado que se ha llamado control del desarrollo.

El nivel repetible de control de procesos puede ser apropiado para controlar proyectos de desarrollo, pero es insuficiente para llevar el control de todas las actividades de la ingeniería requeridas durante el ciclo de vida de un producto. Las organizaciones que operan en un nivel repetible de control de procesos "típicamente tienen sus costos y calendarios bajo un control razonable. Generalmente no tienen métodos ordenados para registrar, controlar y mejorar la calidad de sus procesos de software." [74]

Asimismo, los niveles de calidad son poco conocidos y controlados en el nivel repetible. Una organización que se hace responsable del desarrollo y mantenimiento no se puede permitir abandonar su responsabilidad para con la calidad. Por tal motivo, el nivel de control de procesos tendrá que avanzar para ser capaz de controlar tanto el desarrollo como el mantenimiento

El control multiproducto extiende la responsabilidad de una organización aún más. La meta de controlar el producto es maximizar la calidad al mínimo costo durante la vida de un producto dado, bajo restricciones de tiempo establecidas. La meta del control multiproducto es minimizar el costo y maximizar la funcionalidad y calidad de cierto rango de productos de software. La meta de una organización que lucha por un control multiproducto es un tanto ambigua. Se requiere un nivel alto de control de procesos.

Una empresa que emplea control multiproducto, intenta explotar el potencial del reuso de software. Ha identificado un rango de productos sobre los que opera

---

[74] Van Genuchten, M. (1992) Towards a software factory. Holanda: Kluwer, p. 100

e invierte en componentes para los que promueve el reuso. Esto significa que ve más allá de los productos que se encuentran en desarrollo y posee una visión de futuro clara para el rango de productos que maneja.

Una organización que se proponga ver a futuro sus proyectos y productos actuales tiene que controlar sus operaciones sin confrontarlas con los problemas del día a día que acorralan continuamente a los procesos de software.

Es cuestionable que el nivel definido de control de procesos sea suficiente para el control multiproducto. En este nivel, el conocimiento de los procesos es básicamente cualitativo. Para acceder a un control multiproducto se requiere un conocimiento más cuantitativo de los procesos de la ingeniería de software. El avance a los niveles administrado u optimizado se hace necesario.

Las relaciones entre los estados de control y los niveles de control de procesos, aparecen en la Figura 4-2.

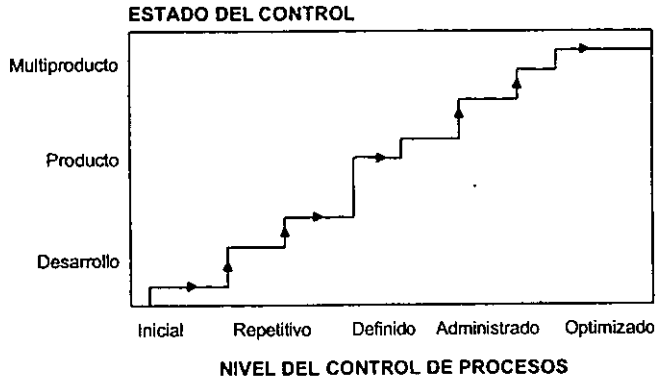


Figura 4-2 Relación entre el nivel del control de procesos y el estado del control

La figura anterior no es normativa, sino indicativa y sólo visualiza los pasos que existen hacia la fábrica de software. Destaca que mejorar los procesos es necesario para avanzar desde el control del desarrollo hasta el control

multiproducto. La figura escalonada representa la mejora incremental de los procesos al mismo tiempo que el foco del control se expande.

Un ejemplo de avance incremental sería el siguiente: una organización que está en el estado de control del desarrollo tiene la oportunidad de acordar el mantenimiento con uno de sus clientes. Después de haber adquirido cierta experiencia, la organización puede llegar a acuerdos similares con varios de sus clientes, luego de haber pasado por una etapa de control del producto. Simultáneamente, la empresa estaría avanzando de un nivel inicial hacia el nivel repetitivo de control de procesos.

Hablar del control de procesos es importante. Si el proceso no se conoce sencillamente no se puede mejorar y sin esta oportunidad, la calidad no aparece. En la actualidad este modelo de madurez organizacional es ampliamente aceptado y puede utilizarse en la definición del estado de las empresas y en el manejo de la calidad del software que presentan.

4.1.3 El costo de la calidad

Se ha hablado teóricamente de la calidad en el software como una serie de características del mismo que deben cumplirse en determinadas circunstancias de tiempo y costo. Asociado ya el desarrollo de software a los procesos de la ingeniería de software involucrados en su creación, se ha descrito el concepto de fábrica de software como un conjunto de procesos cuyo nivel de control se asocia con el modelo de madurez organizacional descrito anteriormente.

Esta relación es útil para ubicar a una organización dentro del marco de la calidad. Las restricciones asociadas de tiempo y costo, están presentes en el análisis de la calidad. Así, el tiempo de un proyecto determina la oportunidad del usuario para su uso, mientras que el costo (directo) restringe de cierta manera el nivel de calidad —los criterios del software definidos— esperado por el usuario.

Pero ¿cuál es el costo de la calidad del software?. Deming responde directamente basado en sus estudios:

"(...) el costo de la calidad del software es realmente el costo de la NO calidad del mismo, es decir, lo que despilfarramos por hacer mal las cosas. De ahí que cualquier costo —mejor, despilfarro— originado por un componente de software en especificaciones y diseño no realizadas correctamente desde el mero principio, represente, simple y llanamente, un costo de no calidad". [75]

La exposición es clara: las actividades realizadas mal, representan un costo de los recursos mal empleados, que se relacionan directamente con el costo pagado por la calidad.

Estos costos pueden ser de cuatro tipos: de prevención, de evaluación o por fallas internas o externas.

Costos de prevención. Estos suelen no ser los de mayor cuantía, pues la experiencia en desarrollo indica que no se gasta en la prevención de fallas. Sin embargo, el hablar de una organización consciente del compromiso con la calidad implica que ésta educará a sus miembros en el marco de la calidad, dotándolos con materiales de soporte tales como directrices y ayudas a la ejecución, mientras se planifican los proyectos de manera tal que todos se mentalicen con la prevención del error.

Los costos de prevención de no-conformidades incluyen educación para la calidad, planeación, la implementación de la ingeniería de software y la formación de gerencias de proyectos; inversión en sistemas de soportes (tales como las herramientas CASE) y materiales de soporte (metodologías, estándares, procedimientos de trabajo).

Costos de evaluación. La práctica común de los desarrolladores es invertir en la prueba del producto una vez finalizado. Es frecuente que esto se haga sin que la gerencia tenga conocimiento de los resultados que se espera. Según Deming, los costos de evaluación de esta forma se convierten en puro derroche. Las pruebas pueden representar incluso menos esfuerzo y ser más eficaces tanto más planificadas y rigurosamente realizadas sean. Como se ha reiterado a lo largo de este trabajo, es preciso unir esfuerzos en las etapas iniciales de los proyectos para la detección precoz de las no conformidades.

---

75 Deming, W.E. Los 14 principios de mi filosofía de software, en *FORUM Calidad* 13/90. P 27

Las actividades de evaluación implican inspecciones, revisiones pruebas iniciales e incluso la instalación de programas en alguna fase inicial. Sus costos se asocian con medidas y auditorías a componentes y sistemas de acuerdo a especificaciones y estándares.

Costos por fallas internas. Estos representan los costos de reparación de las fallas ya incluidas en los programas durante las fases de análisis y diseño que no se detectan a tiempo. Estos costos además de ser considerables, exigen mucho tiempo y esfuerzo en reparaciones y rectificaciones durante el desarrollo y fases posteriores. [76]

Cuando se incurre en costos por fallas internas se habla de la no-conformidad del producto antes de ser entregado al usuario final e incluye actividades de reparación, revisiones de documentación, resolución de problemas y pruebas de regresión.

Costos por fallas externas. En la mayoría de los casos estos costos son inaceptablemente altos, siendo el de personal el más grande. De aquí que los sistemas de administración para la gerencia sean más costosos en su mantenimiento que en su desarrollo.

Cuando se incurre en costos una vez que el cliente ya está usando el producto, se habla de costos por fallas externas. También incluyen reparaciones, solución a problemas y pruebas de regresión, pero se excluye cualquier extensión al sistema.

Los costos de la calidad del software con frecuencia no son del todo claros, especialmente cuando se trata de productos de software que no son utilizables del todo, por su mala calidad. En estos casos las repercusiones no sólo de costos, sino a nivel organizacional en la toma de decisiones pueden ser peligrosas. Con cierta frecuencia se ven presupuestos de departamentos que en más de un 50 por ciento representan el pago por el uso de un software deficiente. Estos

[76] De acuerdo con su experiencia, Deming considera que los costos de reparación de fallas internas pueden multiplicar fácilmente por diez el costo del software.

presupuestos se traducen en un alto grado de tareas de mantenimiento, el pago de personal de soporte en exceso o el tiempo invertido para llegar a un software utilizable.

En este sentido Deming apunta que "ninguna empresa pública o privada puede permitirse el lujo de pagar a su equipo de sistemas de información para la gerencia, por construir un software y luego seguir pagándolo para corregir sus defectos". Por simple y trivial que parezca, lo mejor es hacer las cosas bien desde el principio.

4.2 La calidad del software

La calidad del software debe verse en un contexto institucional donde todos los participantes estén comprometidos con la calidad y donde la visión de los sistemas sea compatible con la visión estratégica del negocio cuyas funciones automatiza. En el presente apartado se desarrollan esta idea y se vislumbra el futuro de la calidad para el desarrollo de software.

4.2.1 Una visión estratégica de la calidad

Durante los años 70, la calidad de software estuvo fuertemente relacionada con qué tan bien el software cumplía los requerimientos contenidos en las especificaciones, haciendo énfasis a la visión de la calidad de la conformidad con las especificaciones.

En los años 80, la idea de la adecuación para el uso de Juran estuvo ampliamente expandida en el campo de la investigación de la calidad. La esencia en el campo del desarrollo de software empezó a basarse en la perfección del negocio. Más tarde su existencia se justificó en términos de recursos invertidos o de la tecnología utilizada.

Al mismo tiempo, la manera en que la tecnología de la información es implementada dentro de las organizaciones, ha cambiado. En la actualidad, existe

un énfasis en la visión estratégica en el campo de la tecnología de la información, que soporte asimismo la visión estratégica de negocio de las organizaciones.

Esto apunta a la alineación del camino que las empresas tienen, como negocio, al tipo de sistemas que deben desarrollarse o implementarse para conseguir los objetivos de negocio, siempre dentro de la misma visión institucional.

Estos factores sugieren que el alcance de la calidad del software debe insertarse dentro de una visión más estratégica e institucional de la calidad. Si la calidad del software se ve como la adecuación para el uso, una visión estratégica de la calidad debería tratarla como el grado en el que la función de la tecnología de la información cumple el objetivo de hacer posible lograr la estrategia de negocio.

Así, el propósito de la tecnología de la información se define dentro de la estrategia de la tecnología de la información. La adecuación para el uso es entonces el emparejamiento entre las estrategias de sistemas y de negocio. La Figura 4-3 muestra esta visión de la calidad.

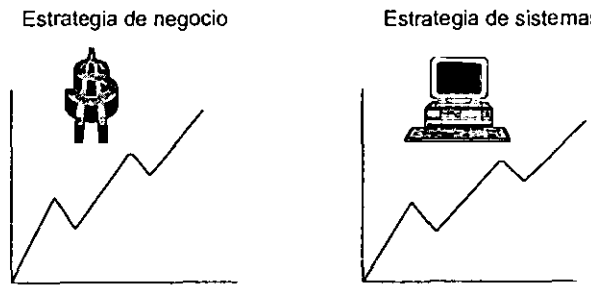


Figura 4-3 Visión estratégica de la calidad

En los años recientes se han hecho investigaciones al respecto bajo el título de *efectividad en la tecnología de la información*. Aunque existe la definición del problema, hay también una escasez de buenas soluciones. Los problemas que se presentan para dar soluciones pueden dividirse en dos grandes grupos. El primero trata con cuestiones como:

- ¿Qué criterios se deben usar?
- ¿Los criterios son universales o deben definirse localmente?
- ¿Cómo deben ser medidos?
- ¿Es posible dar una medida general de la efectividad?

Una de las corrientes teóricas de la calidad de software tiene que ver con estos temas de investigación.

El segundo problema que una organización enfrenta al intentar hacer compatibles las estrategias de sistemas y del negocio, es separar el impacto de la tecnología de otros factores que influencian el éxito de la estrategia de negocio, por ejemplo, el estilo y la estrategia de administración, factores económicos, políticos, etc. Así, la efectividad en los sistemas va más allá para insertarse dentro de la efectividad de la organización completa. Ésta representa qué tan bien una organización está haciendo uso de la tecnología, y cómo se refleja esto en el cumplimiento de los objetivos del negocio.

Considerar la efectividad en la tecnología de la información más que simplemente la calidad del software refleja una cierta madurez en el aprovechamiento de la tecnología. Tradicionalmente se justifican inversiones tecnológicas en términos de incremento en la eficiencia —¿cuántas personas son necesarias para hacer redundante una actividad?—. Actualmente, con el establecimiento de una visión más amplia de la tecnología de la información dentro de los negocios, la justificación para las inversiones descansa sobre argumentos más elaborados que forman parte de la toma de decisiones estratégica de la organización.

Muchos de los beneficios de la aplicación de los sistemas de cómputo no aparecen explícitamente en el balance de las organizaciones, por lo que se hace necesario hacer una diferencia entre la efectividad y la eficiencia en la tecnología de la información.

La eficiencia representa la facilidad tecnológica que posibilita a la organización a tener beneficios financieros que explícitamente puedan verse como pérdidas o ganancias. Bajo este rubro se encuentran, por ejemplo, los ahorros en los costos asociados al desarrollo, el ahorro en los gastos operativos o el incremento de proyectos desarrollados por un mismo equipo de trabajo.

La efectividad en la tecnología de la información se relaciona con conceptos menos inmediatos, tales como margen de competencia, imagen corporativa o calidad en la toma de decisiones. Por su naturaleza, estos conceptos son más difíciles de medir. La distinción equivale a la diferencia entre productividad y calidad en términos de software. La productividad es visible y cuantificable, mientras que la calidad no lo es tanto. Otra distinción es que la calidad y la efectividad son categorías a largo plazo, mientras que la productividad y la eficiencia pueden verse como beneficios inmediatos.

4.2.2 La visión de Deming

El hecho de medir los costos de la calidad del software no implica en sí haber mejorado el software. En primera instancia, conocer los costos asociados ayuda a dos cosas: uno, son una medida de análisis de las actividades de los departamentos de sistemas y dos, sensibilizan a la Dirección del problema de la calidad. Ambos elementos coadyuvan a la evaluación intrínseca de la magnitud del problema.

Según Deming, analizar los costos de la calidad es el primer paso en el camino hacia la mejora de la Calidad en el software. Su filosofía se basa en un método gerencial de mejora continua de la calidad que conduzca simultáneamente al aumento de la productividad y a la reducción de costos.

Sus catorce principios, así como los obstáculos que se presentan, son aplicables al desarrollo de software. Estos puntos vistos en la aplicación por la Gerencia de sistemas son:

1º. Crear una constancia y conciencia del propósito firme para la mejora continua del desarrollo con el objetivo de alcanzar la excelencia y la satisfacción del usuario.

Este esfuerzo implica una innovación constante, planificación a largo plazo así como investigación, formación y mejora continuas. Para lograr la constancia en el propósito de la calidad, la Gerencia debe llevar a cabo tareas como las siguientes:

- Establecer definiciones prácticas para identificar cada etapa del proceso de desarrollo. Parte importante es definir lo que se entiende por servicio al cliente.
- Definir claramente la personalidad de los clientes internos y externos.
- Desarrollar mecanismos que mejoren los servicios y sistemas de trabajo, empleando tiempos más cortos e involucrando medios más limitados.
- Invertir en técnicas y herramientas a fin de lograr un mejor desarrollo de sistemas

2º. Comprometerse con la nueva filosofía de calidad. En nuestra época la calidad dentro de la ingeniería de software no sólo es premisa de existencia, sino de sobrevivencia. Las fallas y actitudes negativas no tienen cabida en este proceso. La Gerencia debe asumir su responsabilidad de dirección, ya que es ella la única que puede resolver la mayoría de las fallas que surgen en el desarrollo de sistemas.

Una vez asumido el compromiso de la Calidad, la Gerencia debe comunicar claramente sus planes y propósitos a su equipo de trabajo con acciones que engloben a todos los participantes en el propósito compartido de la calidad.

3º. Acabar con la dependencia de las pruebas finales, ya que esta inspección es tardía y por consiguiente, poco fiable. La calidad debe insertarse orgánicamente dentro de los procesos, o no se consigue.

4º. El valor intrínseco del producto o servicio es la calidad. Por tal razón no se deben basar los negocios de software sobe el precio de venta porque es poco significativo.

No tener esto claro empuja a ofrecer el negocio al mejor postor, quien con frecuencia es el que peor calidad involucra. La Gerencia debe procurar relaciones de confianza a largo plazo y tratos limpios con sus proveedores. Esto promueve la reducción de costos y redunda en beneficios para el cliente.

5°. Mejorar continuamente el proceso de desarrollo de software, pues la calidad no significa un esfuerzo único sino mantener constante el empeño por conseguirla.

En este sentido la Gerencia debe insistir en la mejora de la metodología del desarrollo de sistemas y estándares, con prácticas que han de ser revisadas y actualizadas continuamente.

6°. Instaurar una formación profesional de calidad que haga honor a su nombre. La calidad comienza con la formación y educación continua, lo que obliga a la contratación de expertos en todas las áreas del desarrollo de sistemas.

Es de suma importancia la capacitación —incluso de la Gerencia— que desarrolle el pensamiento analítico y estadístico que promueva la comprensión de la naturaleza de las variaciones presentadas en los procesos.

7°. Ayudar a los integrantes de la organización. El papel de la Gerencia debe ser dirigir, no castigar. Quienes dirijan un proyecto deben hacerlo de tiempo completo porque resulta laborioso encauzar al factor humano hacia la calidad, siendo constante en la consecución de las metas propuestas.

Los realizadores de proyectos de calidad deben adoptar los métodos estadísticos y las bases analíticas imprescindibles para detectar y localizar los orígenes de un problema dado. Asimismo deben estar capacitados para detectar la naturaleza de cualquier variación; esto apoya la toma de decisiones a la vez que obliga a disponer de información exacta que describe el estado de los procesos.

8°. Eliminar toda fuente de temor que coarte el desarrollo de las verdaderas capacidades del personal. La Dirección debe responsabilizarse de los errores que atenten contra el medio ambiente.

Si una persona no se siente segura en su puesto, jamás pedirá ayuda y jamás preguntará, a pesar de no haber comprendido bien la tarea que se le haya encomendado.

9°. Eliminar las trabas entre departamentos. Para insertar el proceso de desarrollo de software dentro de la calidad se debe asumir la problemática existente entre los distintos grupos de la organización, para establecer canales de comunicación efectivos tanto entre los departamentos, como con los clientes.

10°. Eliminar slogans y exhortaciones del tipo "trabaja mejor" porque no constituyen por sí solos sistemas de calidad. En lugar de esto, la Gerencia debe concentrar esfuerzos en el estudio y la respuesta a sugerencias que coadyuven a la mejora de la calidad.

Solo desde arriba, desde la Dirección, es posible realizar un cambio hacia la mejora de la cultura empresarial del medio ambiente.

11°. Suprimir las cuotas y objetivos puramente numéricos, reemplazándolos por el liderazgo puro. Estos objetivos numéricos conducen a pensar en cantidades, no en métodos y en calidad. Quienes se guían irrestrictamente por el cumplimiento de tales cuotas incurren en derroches e ineficiencia, fuente de la no-calidad.

Por muy meticuloso que parezca a primera vista, un plan que provoque estrés, tiempos y fallas no es benéfico para nadie.

12°. Apoyar todo lo que signifique legítimo orgullo entre los miembros del equipo de trabajo. En lugar de una planeación estresante de los proyectos, el gerente debe hacer un autoexámen de sus propios sistemas y procesos operativos para averiguar si los mismos promueven o frenan la mejora constante de la calidad.

13°. Establecer programas de automejora y autoeducación para cada uno de los miembros de la organización. Tanto la Gerencia como los equipos de trabajo deben formarse en calidad e ingeniería de software con la certeza de la seguridad personal y progreso en el trabajo.

Para construir mejores sistemas se hace necesario los mejores profesionales, que son los que comprenden y asumen que el proceso de aprendizaje no puede ni debe cesar jamás.

14°. Unir esfuerzos y voluntades de equipo para lograr la transformación de la cultura organizacional en pro de la calidad, tarea que incluye a todos sin excepción.

Estos puntos resumen una serie de actividades que deben implementarse como camino hacia la calidad. Principalmente enfatizan la importancia de la cultura organizacional entendida como una serie de prácticas personales y de administración como el ambiente en que la calidad debe darse.

Si bien asumir el compromiso para la calidad que Deming ha resumido en catorce puntos es una tarea prioritaria, no está exenta de obstáculos que traben su desarrollo y que amenazan la calidad dentro de las organizaciones.

Respecto al software, los puntos que se deben evitar son los siguientes:

1	La inercia en la creación de sistemas para el usuario
2	La mala planificación
3	La evaluación y revisión del personal
4	La rotación de personal
5	El análisis incorrecto de indicadores
6	Los presupuestos de personal inflados
7	El aumento en los costos de mantenimiento

La inercia en la creación de sistemas para el usuario. El hecho de concebir sistemas para el usuario por el simple hecho de continuar y aun crear puestos de trabajo es inconsistente con los preceptos de la calidad. La calidad del software

no puede mejorarse si se considera un lujo. El esfuerzo debe cumplir una línea consecuente con las metas siendo constante en las intenciones y no obedeciendo a situaciones coyunturales.

La mala planificación. Cuando se planifica exclusivamente a corto plazo se vulnera la seguridad en la actuación, creando una preocupación extrema de crear software de mala calidad. En este sentido, la calidad debe sobreponerse a los planes de corto plazo, mientras que la gerencia debe razonar una ley de la calidad: "cada cual ha de disponer del tiempo suficiente como para hacer las cosas bien".[77]

La evaluación y revisión del personal. Los efectos que pueden provocar las evaluaciones y revisiones a los equipos de trabajo, suelen ser contraproducentes para lograr la calidad y crean un clima organizacional adverso. Un ambiente de trabajo adecuado debe excluir presiones que supongan un mayor estrés e inseguridad en los individuos.

La rotación de personal. La fluctuación del personal de sistemas dificulta mantener el propósito constante y la planificación para la calidad. Al mismo tiempo, un empleado que se va se lleva consigo conocimientos que son importantes para el proyecto. No se trata de que los buenos profesionales deban ser sobrepagados para retenerlos, sino crear un ambiente de trabajo idóneo, donde los elementos de la calidad —capacitación, participación en equipo, comunicación efectiva, etc.— promuevan una estadía larga para los profesionales involucrados.

El análisis incorrecto de indicadores. Con mucha frecuencia se toman decisiones sobre indicadores incorrectos que reflejan el avance de las metas hacia el cumplimiento de las cuotas y la productividad. La Dirección puede tomar decisiones deslumbrada por el impacto de cifras que cubren a otras más cruciales, como la satisfacción del usuario. Basar las decisiones sólo en el aumento de la productividad y en los costos puede crear éxitos sólo en el corto plazo. El software

77 Deming, E., *op. cit.* p.13

de calidad implica menos tiempo y recursos en su construcción que un software de baja calidad y es el producto de un esfuerzo continuo donde la calidad redunda necesariamente en el aumento de la productividad.

Los presupuestos de personal inflados. Muchos de los proyectos de software involucran costos asociados con el personal elevados —contratación, enfermedad, prestaciones, etc.— sin contar la subutilización de las capacidades de las personas para proyectos determinados. Una política de administración al respecto debe analizar minuciosamente la necesidad de personal real que debe cubrirse y sobre todo, considerar el valor agregado que cada individuo ofrece para la satisfacción de las necesidades del usuario.

El aumento en los costos de mantenimiento. El costo de vida útil de un sistema que contiene errores de fondo es por general muy alto. La mejora en la calidad del mantenimiento es una fuente importante de ahorro, considerando que quienes desarrollan software dedican la mitad o más de su tiempo en esas actividades. En todas las fases del desarrollo de los sistemas debe prevenirse, como se ha dicho, la inclusión de errores que posteriormente se reflejarán en complicadas actividades de mantenimiento.

4.2.3 La efectividad de la tecnología de información

Establecer mecanismos de análisis de la efectividad en la aplicación de la tecnología de la información implica que se deben alcanzar un par de objetivos: la reducción de la administración al mínimo y el convencimiento del personal de que los beneficios serán visibles.

El análisis de la efectividad hace necesario el análisis exhaustivo de las actividades que cada persona hace para determinar cuáles son clave y al mismo tiempo clasificar los tipos de actividad desarrollada.

Este análisis tiene algunas ventajas para la compañía:

- Provee datos de las áreas de sistemas que están dando un servicio efectivo al resto de la organización.

- Hace evidente cuando una área de sistemas cumple los objetivos establecidos.

Lo que sigue haciendo falta es información respecto a la efectividad en el empatamiento entre las estrategias de negocio y de sistemas y el impacto que la tecnología de la información tiene sobre el trabajo interno de otros departamentos.

Una manera de atacar el problema es enfocarse en las estrategias de negocio y de sistemas. En general, esta forma de solución se divide en tres etapas: la planeación, la implementación y la evaluación.

La primera fase es un ejercicio de escritorio para verificar que la estrategia de sistemas sirve a las necesidades del negocio. Debido a que la estrategia del negocio puede verse como un proceso continuo, es conveniente tomar fotografías de los requerimientos a sistemas para verificar cuántos de ellos y en qué forma han sido cubiertos por la estrategia de sistemas.

Los factores críticos de éxito deben identificarse dentro de la operación del negocio. En cada uno de los puntos donde se detecte un factor crítico de éxito los requerimientos de la tecnología de información deberán ser cubiertos, mientras se establece una lista de requerimientos críticos. La estrategia de la tecnología de información debe entonces evaluarse en términos de su habilidad para proveer los recursos críticos en el tiempo apropiado.

Una vez que la estrategia tecnológica ha sido evaluada en términos de la estrategia de negocio, es necesario garantizar que la implementación real se haga de acuerdo a un plan. Los cambios en los planes del negocio debidos a cambios circunstanciales, deben ser cubiertos por consiguientes cambios en la estrategia de sistemas.

Es factible establecer procedimientos para asegurarse de que la efectividad en la tecnología de la información se alcanza en la práctica tanto como se ha establecido en la teoría. Esto se puede lograr con el uso de algunos procedimientos mencionados en el capítulo anterior. Un sistema de gestión de calidad debe establecerse para las funciones relacionadas con la tecnología de la información. Por supuesto, esto suele involucrar no solamente a un departamento

sino a varios, ya que es muy frecuente encontrar tales actividades esparcidas en diferentes áreas de la organización.

Algunos factores de análisis no aparecen sino hasta el término de los proyectos. Se hace necesaria entonces, una evaluación retrospectiva. La experiencia adquirida debe retroalimentar a los miembros de la organización para el futuro. En este sentido se habla del establecimiento de un ciclo continuo de mejora tal como lo sugiere Deming. Este proceso de planeación, implementación y evaluación, con actividades de monitoreo en cada fase, se resume en la Figura 4-4. Ésta refleja sólo un esquema general que muestra la idea; el alcance del presente trabajo no profundiza en el detalle del esquema.

4.2.4 El desafío del futuro

Muchos desarrolladores parecen estar muy contentos con el estado actual de la calidad del software. Normalmente se resisten a los cambios de paradigmas, a la aplicación de nuevos conceptos —metodologías, técnicas de programación, etc.— y a la introducción de nuevas ideas que son vistas como una afrenta a su integridad y a su profesionalismo. El argumento básico es con frecuencia que las técnicas de gestión de calidad son sólo ideas pasajeras de moda.

La tendencia actual en el área de sistemas hace la tarea de la calidad más difícil de lo que hace tiempo podía ser. Ya no es suficiente con medir el software exclusivamente en términos técnicos ni es aceptable considerar una aplicación aislada de su entorno.

En la medida en que el propósito del desarrollo de sistemas sea percibido como el soporte de los objetivos de negocio de las organizaciones, el punto de vista de la calidad del software deberá cambiar.

El compromiso con la calidad se amplia entonces a un rango de factores más allá que simplemente la calidad del software y depende fuertemente de la calidad de la información sobre la que se toman decisiones. La calidad tanto del software

como de la información depende de factores humanos que involucran recolección de datos, factores organizacionales y estructuras administrativas.

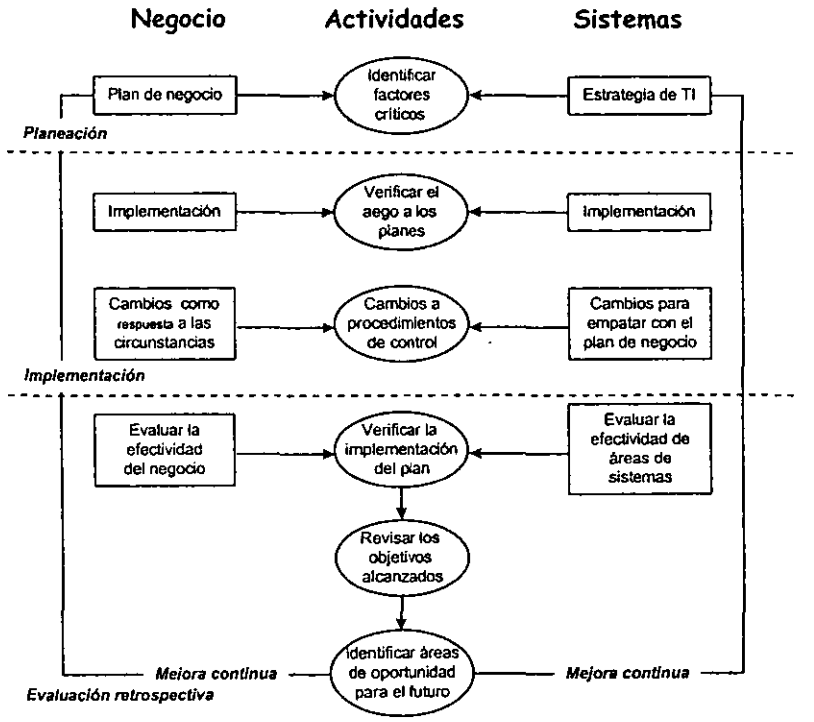


Figura 4-4 Estrategia de efectividad

Mientras más integrados se encuentren las áreas de sistemas con la administración, los aspectos de calidad se vuelven más interdependientes. El desafío es incrementar continuamente la calidad mientras que los requerimientos se tornan más complejos y rigurosos, entendiendo a la calidad del software como un proceso a largo plazo que es el resultado del trabajo en el que todos toman la parte de responsabilidad que les corresponde.

En el pasado la calidad significó hacer que los programas funcionaran. En tiempos recientes ha significado hacer lo que los usuarios quieran. En el futuro

próximo y aun ahora significará hacer para las personas lo que necesitan ahora, con una visión de lo que a futuro necesitarán.

La clave es la visión estratégica que posea el software que se construya para los usuarios. Este debe no solamente ser visto como el producto de las necesidades actuales, sino que debe satisfacer las necesidades a futuro, al mismo tiempo que debió crearse por medio de procesos donde, de principio a fin, sus creadores se comprometieron con la calidad en todos los niveles de la organización, comenzando por la Dirección y sin dejar escapar a nadie.

4.3 Recomendaciones en torno a la calidad del software

La calidad del software representa una problemática que empieza con la naturaleza efímera del mismo y se complica con la necesidad de incluirle calidad. Lo que se hable o escriba en torno a la calidad de software no servirá de mucho si siguen existiendo, como hasta hoy, sistemas de seguridad violados, transacciones comerciales inconclusas o sistemas operativos fallando que crean un ambiente de desconfianza y enojo en sus usuarios.

¿Qué hacer al respecto? En el trabajo se ha hablado de un cambio de cultura que debe estar fuertemente impulsado por la Dirección, y debe ser adoptado orgánicamente por los individuos. Este es el primer paso, hacer conciencia de la necesidad de la calidad para vivir con ella y por consiguiente, mejorarla continuamente.

Entender que asumir el compromiso de la calidad es un proceso que lleva tiempo no se debe perder de vista. No se puede creer en soluciones instantáneas, puesto que no hay una receta que diga paso a paso qué hacer para crear software de calidad. La única solución es, basándose en conocimientos sólidos, tener la determinación y disposición de trabajar duramente.

La calidad no se instala cuando se implementa un programa para la calidad, sino que es una tarea diaria que se consigue sometiendo al proceso a un control estadístico, bajo la visión de una mejora constante. Para líderes de proyecto

impacientes, la calidad es una cuestión inalcanzable porque sus frutos no son visibles a primera vista. Por el contrario, la inclusión de control estadístico, la educación del personal, el desarrollo de programas para la calidad, son actividades que requieren tiempo y que implican que la mejora de la calidad y la productividad son cuestiones a plantear a largo plazo.

Según Deming, la calidad y la productividad son consecuencia del empeño y del trabajo humanos y no son creados por el hardware o software, por excelentes y avanzados que sean. En este sentido se debe tener mucho cuidado en la aplicación irrestricta de las más modernas técnicas y herramientas de la ingeniería de software. Si las actuales no han resuelto el problema de la calidad ¿porqué las nuevas si lo harían?

La mayoría de los problemas que se enfrentan en el desarrollo de software son similares de un proyecto a otro, independientemente del software que se construya. Así, la administración de recursos humanos y técnicos, los problemas presupuestales, el vacío de comunicación entre desarrolladores y usuarios, son temas que todos los involucrados en el desarrollo de sistemas conocemos. Entender que los problemas son comunes hará ponernos a trabajar en lo realmente importante, lejos de seguir escudándonos en que nuestros problemas son diferentes a los de otros.

Por difícil que parezca, hacer bien las cosas desde el principio es un paso significativo hacia la calidad. Esto implica a todos los involucrados en los procesos, desde los programadores hasta la Dirección, cada uno con sus responsabilidades, ya que al término de los proyectos o durante el mantenimiento a los sistemas suele atribuirse a la ineficiencia de los programadores las fallas encontradas.

Indicar culpables es delicado porque para la Dirección resulta difícil responder a la pregunta ¿quién los contrató, quién los dirige? Mientras cada quien no asuma su responsabilidad, la calidad será una ilusión. Así por ejemplo, las organizaciones que cuentan con un departamento de calidad suelen delegarles

todos los problemas relacionados ignorando que si la Gerencia no se hace cargo de su responsabilidad, la calidad simplemente no sucede.

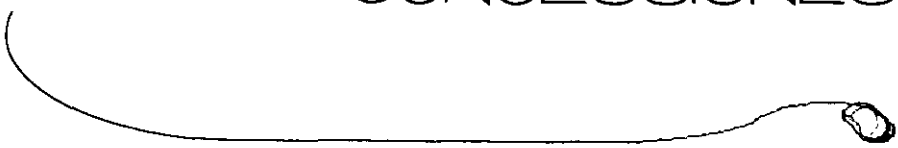
El manejo del factor tiempo es muy delicado y estresante. Trabajar siempre con fechas límite pone a todos en un estado extremo donde la calidad es fácilmente olvidable. Por la premura de cumplir se dejan de lado procedimientos y análisis requeridos por la calidad, además de que se van dejando errores para después al no existir tiempo suficiente ni para repararlos ni para prevenirlos. La corrección de errores como mantenimiento se convierte entonces, no sólo en un factor de costos excesivo sino en un indicador que evidencia la mala calidad de los sistemas.

La calidad significa un trabajo en equipo que debe ser reforzado con la colaboración de terceros, especialistas objetivos y bien adiestrados que contribuyan al progreso hacia la calidad. Puede ser un error de la Dirección suponer que nadie puede ayudar a menos que conozca a fondo los sistemas. Quienes conocen los sistemas son los involucrados en sus procesos, especialistas en ellos que aunque teniendo conciencia de la calidad, no tienen idea de como mejorarla.

Asi como los deportistas se apoyan en psicólogos para conseguir su meta, los creadores de software pueden apoyarse en especialistas en calidad. Los sicólogos son especialistas en su ramo y no necesitan ser expertos en el deporte de la persona para poder trabajar en equipo y lograr resultados de excelencia.

Finalmente, el cumplimiento de las especificaciones no debe ser el límite. Hay que trabajar en la entera satisfacción del cliente. El trabajo de los involucrados en el desarrollo y mantenimiento de sistemas debe reflejar las necesidades reales de los usuarios, revisando y actualizando constantemente las especificaciones. Este proceso continuo es a la vez un proceso de aprendizaje en donde usuarios y desarrolladores trabajan conjuntamente para establecer requerimientos reales apegados a necesidades concretas y no solamente deseos o caprichos solicitados por usuarios que generalmente redundan en costos y esfuerzos excesivos.

CONCLUSIONES



El siglo 20 ha dejado en la humanidad una huella imborrable en su desarrollo y evolución. Siglo de guerras, es al mismo tiempo el siglo de los grandes inventos y de la avanzada tecnológica más significativa de la historia del hombre. El conocimiento adquirido a lo largo de siglos se sintetiza en aplicaciones que lo han puesto muy cerca de sus sueños: el espacio, la comunicación a distancia, los viajes ultrasónicos, la medicina.

Todo ha tenido que ver con el gran invento: la computadora. Como ente compuesto de hardware y software, su importancia se acrecenta en la medida de su exactitud, de su confiabilidad, de su facilidad de transporte, etc. Si gran parte del entorno actual de la sociedad del mundo globalizado depende de las computadoras ¿es trivial hablar de niveles de calidad en las mismas? ¿Resulta importante destacar el papel que tiene la producción de computadoras —hardware y software— con niveles altos de calidad, considerando que de ellas dependerá en buena medida la seguridad, la salud, las relaciones económicas de las personas?

Las respuestas parecen ser obvias. Por supuesto que producir computadoras cada vez más avanzadas tecnológicamente que resuelvan problemas de la vida cotidiana con mayor exactitud y que contribuyan al desarrollo se hace importante. Es evidente que si las computadoras controlarán procesos vitales, contribuirán al desarrollo económico o serán indispensables en la investigación científica, deban ser las mejores.

Sin embargo, en la práctica la situación no es tan clara. Gillies afirma que la calidad no se reconoce cuando está presente, pero es fácilmente reconocible en su ausencia. Aquí aparece la primera gran controversia de la calidad en el software: la relación que ésta guarda con el hardware, la casa que habita.

Hablando de procesos de manufactura, de sistemas de control de calidad, de inspecciones y de control estadístico del proceso, la producción de hardware tiene la ventaja de ser un proceso cuantificable, controlable y por lo tanto, perfectible. Más aún: se habla de que el diseño del hardware es vital en la definición de nuevas tecnologías que superen constantemente a sus antecesoras. Por mucho tiempo se consideró que si se producia hardware defectuoso con procesos de producción bajo control estadístico, las fallas debían atribuirse al diseño. En la actualidad el diseño también es sujeto de revisión y por lo tanto, de mejora en la calidad.

Si el hardware que actualmente se fabrica está sujeto a controles de calidad, ¿porqué los sistemas fallan, están fuera de servicio, o producen errores? El papel que juega el software es determinante. Aún la más avanzada tecnología o la computadora más actual puede sucumbir ante un software defectuoso. Sin un software adecuado, de nada sirve un hardware avanzado. En relación con el avance tecnológico, Yourdon sostiene que "éste avanza más rápido de lo que se prevé, pero la gente cambia más lentamente. Dada una nueva tecnología, la gente tiende a hacer las cosas como antes, quizá sólo un poco más barato y rápido. A las personas les toma un poco más tiempo encontrar nuevas aplicaciones para nueva tecnologia".

El software, como producto intelectual fruto del talento y conocimientos humanos, significa gente. En el contexto de lo expresado por Yourdon, el desarrollo de software quizá vaya un paso atrás de la tecnologia, pero siempre debe avanzar pues le es indispensable. Este es el gran desafío del futuro para el software: responder a niveles de complejidad altos, abarcar más aplicaciones, dar

soluciones más rápidas, ser altamente eficiente. Hablo entonces, de hacer software de calidad.

Un problema a vencer en el cambio de paradigmas es la inercia. Si bien la tecnología parece ser lo más importante a primera vista, no hay que olvidar que ésta avanza más rápidamente de lo que uno piensa. La tecnología esta allí; ya existe y hay que utilizarla de la mejor manera. Los aspectos más significativos en el desarrollo de software no tienen que ver con la tecnología por si misma sino con las personas.

Un problema actual es el de la transferencia de tecnología; el conocimiento aplicado en la vanguardia de la técnica. Lo que deba hacerse al respecto —acerarse a consorcios como MCC, SEI, ESPRIT, asistir a conferencias— debe comenzar desde ahora. Nunca es demasiado tarde para iniciar porque el camino es largo e intrincado.

El software no debe verse sólo como una serie de instrucciones, sino en primera instancia como una aplicación que es el producto de algunos procesos y en un sentido amplio, inserto dentro de una estrategia de negocio de las empresas. Deming afirma que un software será tan bueno o tan malo como el proceso de su construcción y desarrollo, que ha de ser iniciado con el riguroso análisis de las efectivas necesidades del usuario.

En el complejo panorama el desarrollo de software, tres factores son de crucial importancia para la calidad:

- Costos
- Tiempo
- Detección de necesidades reales

El manejo de la calidad del software tiene que ver con estos tres rubros y es imperativo que los involucrados en los procesos de la ingenieria de software conozcamos de ellos, para poder mejorarlos.

La calidad del software tiene un costo, no solamente monetario. El costo son todas las repercusiones y consecuencias que puede producir la operación de un software de baja calidad, lo cual significa que se gasta para hacer mal las cosas.

Evitar los costos asociados en el desarrollo de software que redundan en subsidiar el hacer mal las cosas, es más que necesario, una tarea urgente. Se incurre en gastos de personal excesivo, en costos de operación —tiempo de CPU, herramientas, etc.— que no siempre justifican los resultados o en largas etapas de prueba que no logran evitar la aparición de errores.

El factor tiempo es una restricción muy delicada en el desarrollo de software. Siempre se trabaja con fechas fijas y contra el reloj. Esto evidentemente no es deseable, pero es práctica común. Cuando una organización o un equipo de gente involucrada en el software —personal técnico, usuarios, áreas de apoyo— trabajan conscientes de su compromiso con la calidad, el tiempo no debería ser un límite porque todos trabajan para hacer las cosas bien, y eso toma tiempo.

En la práctica el establecimiento de una cultura que garantice desarrollar software con calidad sin restricción de tiempo representa un escenario difícil y poco frecuente. Sin embargo, estoy seguro que si se adquiere el compromiso con la calidad y se siguen sus preceptos, el tiempo de desarrollo se reduce, no solamente para la entrega de un producto, sino en la atención a requerimientos, prevención de fallas y mantenimiento.

Un tema en discusión hasta nuestros días y que normalmente se ve disminuido por el peso de la tecnología, es el conocimiento de las necesidades del usuario. Aunque parezca trivial, el desarrollo de software no siempre cumple con las necesidades reales de los usuarios. Tal vez se haya creado con las herramientas más modernas, el lenguaje más avanzado y la tecnología de punta, pero no sirve de nada si no responde al problema real.

El software es, como he dicho, el producto de un equipo de personas, en donde los usuarios, los clientes, representan un sector muy importante. Conocer

a los usuarios y entender sus necesidades reales también es una cuestión imperativa.

¿Qué hacer para lograr la construcción de software de calidad? Es importante destacar los siguientes puntos antes de continuar.

- La calidad es un proceso que toma tiempo. Al tener relación con todos los elementos en el desarrollo de software, y sobre todo con las personas, los cambios en hábitos son paulatinos.
- No existe una solución mágica e instantánea al problema de la calidad en el software. La teoría en este tema muestra el camino a seguir, pero de nada sirve si no se toma desde hoy. Nunca es tarde para comenzar.
- La calidad no debe verse como una añadidura al software; no se imprime una vez que se escribe el código de un sistema. La calidad se trabaja en todas las etapas del software, incluso desde la propia idea de hacerlo. El cambio de percepción de la calidad como una adición debe partir del compromiso de todos los involucrados para con la calidad.
- En las organizaciones la calidad debe ser un compromiso de todos, impulsado y asumido primeramente por la Dirección. Los esfuerzos personales no fructifican donde no existe una cultura de calidad y donde el compromiso con la calidad no es asumido por todos.
- La calidad no debe verse como un lujo. El software no puede mejorarse si se piensa en la calidad como algo costoso y que implica esfuerzo adicional e innecesario.
- La planeación en la creación de software debe verse más a largo plazo. Si se establecen objetivos a corto plazo, generalmente se estimula el trabajo a marchas forzadas. Esto repercute en la creación de software de mala calidad, con las consecuencias que esto implica. Establecer horizontes más amplios[78]

78 Yourdon señala que un periodo de 10 años puede ser factible, al entender que los cambios se producen lentamente.

redundará en relaciones más sólidas entre las figuras de la Ingeniería de Software.

- La calidad del software no implica solamente el aumento en la productividad, en la óptima relación tiempo-costo. Tomar cifras sólo de estos factores puede deslumbrar y por tanto desvirtuar los resultados, limitando metas sólo a corto plazo. Cuando la calidad es máxima, la productividad se incrementa no como producto del presupuesto de tiempo y costo sino por el esfuerzo continuo de todos por la calidad.

Con lo anterior, un camino hacia la calidad del software implica:

- Crear una conciencia de la calidad con el propósito firme de alcanzar la excelencia y satisfacer las necesidades del usuario.
- Propiciar la investigación, la capacitación y mejora continua de las personas inmersas en el desarrollo de software. La calidad comienza con una formación y educación continua. La formación profesional que debe tenerse en mente involucra temas propios del desarrollo de software y calidad, pero también áreas que desarrollen el pensamiento estadístico y analítico.
- Los objetivos deben plantearse a largo plazo, y todos deben trabajar en la mejora constante para lograrlos.
- Invertir en técnicas y herramientas que permitan un desarrollo de sistemas mejor. Si bien las herramientas no son la solución per-se al problema de la calidad del software, si constituyen un apoyo.
- Aceptar la idea de una nueva cultura para la calidad. La figura de Dirección es de vital importancia en este punto. Especialmente en las organizaciones, el compromiso con la calidad que parta de la Dirección redunda en todos los niveles en relación con la creación de la cultura para la calidad.
- Erradicar el concepto de pruebas como sinónimo de calidad. Las pruebas consumen un tiempo adicional y vistas al margen del desarrollo, no siempre garantizan la calidad. Si la calidad no se implanta orgánicamente a lo largo de todo el proceso, no se consigue.

- Mejorar los procesos del desarrollo de sistemas para afianzar la productividad y perfeccionar la calidad, a la vez que se reducen costos y tiempo. Mejorar la calidad no significa hacer un solo esfuerzo, sino mantener constante el empeño de conseguirla.

- Mejorar la metodología de desarrollo de sistemas y estándares. La elección de estos elementos depende del rumbo que tome la Organización; lo relevante es saber que deben existir para que se pueda evolucionar en el camino a la calidad.

- Promover y practicar los métodos estadísticos y las bases analíticas que permitan encontrar las causas originales de un problema dado. Tan importante puede llegar a ser esto como la habilidad que se tenga en las herramientas y técnicas de la ingeniería de software. Alguien que dirija un programa de software para la calidad debe ser capaz de detectar la naturaleza de las variaciones.

- Suprimir las metas numéricas. Pensar en alcanzar objetivos basados en cantidades conduce a pensar en números y no en calidad. Se debe promover un ambiente donde se disminuyan los errores y el trabajo para la reparación de fallas.

- Hacer bien las cosas desde el principio. Esto puede resultar obvio, pero en el desarrollo de software no lo es. Lo que se ha hecho mal en etapas iniciales, surgirá posteriormente en forma de errores que significan problemas potenciales en el proceso general. Los costos y las repercusiones de no tener esta practica, redundan en la reparación de fallas que no debieron originarse.

- Desarrollar el papel de la Dirección. La Dirección debe ser consciente del apoyo a las personas, la elección de estándares y la metodología a usar. Debe hacerse responsable de las fallas en la organización y del clima de trabajo que genere en las personas el gusto por lo que esta haciendo. Sólo desde la Dirección es posible realizar cambios hacia la mejora de la cultura empresarial, teniendo siempre en mente la revisión de métodos y procedimientos que garanticen la mejora continua de la calidad.

- Apegar las metas de sistemas con las metas estratégicas de negocio. Este concepto lleva a una visión global empresarial[79] que se preocupa por una solución general más que por el desarrollo y establecimiento de sistemas individuales. Un modelo global, influenciado por la estrategia de la organización guía los planes de negocio que finalmente promueven el análisis, diseño e implementación de sistemas individuales.

Varios de los puntos antes expuestos tienen que ver con la formación profesional de las personas que se insertarán en la maquinaria de las empresas creadoras de software. Quiero destacar la importancia que tienen las Universidades en éste contexto, pues es allí donde los futuros profesionales no sólo adquirirán conocimientos, sino una cultura de trabajo que posteriormente marcará su ejercicio profesional.

Queda no sólo para las Universidades, sino también para las organizaciones que se nutren de ellas, tomar el desafío de calidad. Esta comienza con la formación y sólo así se podrá vislumbrar un desarrollo de software acorde con las exigencias de los años venideros.

La calidad del software no debe ser más un tema ignorado por quienes nos dedicamos a él. No debe ser comentario al margen del oropel de la tecnología ni tema adicional y mínimo al lado de los temas de moda en la ingeniería de software. No se debe asumir la existencia de la calidad sólo por creer que se está consiguiendo. Cada día es más imperativo dejar de creer que la inercia e intuición profesional inducirá en el esfuerzo de los individuos la creación de software de calidad. La calidad es un compromiso, un proceso. Hay que tomarlo.

79 *Enterprise modeling* es un concepto que ha tomado importancia en los años recientes y que seguramente será tomado por las organizaciones cuyo camino sea el de la calidad.

ANEXOS

Anexo A: Instituciones dedicadas a la tecnología de software

Organismos relacionadas con la problemática de la tecnología del software y temas relacionados con la calidad.

MCC

- ***Microelectronics and Computer Tecnology Corporation***

  http://www.mcc.com/

  Consorcio de 20 compañías estadounidenses proveedoras de hardware y software. Conocida como MCC, también desarrolla interfaces humanas y hardware.

SOFTWARE PRODUCTIVITY CONSORTIUM

- ***Software Productivity Consortium.***

  http://www.software.org/

  Esfuerzo cooperativo de 15 contratistas en los campos de componentes electrónicos y desarrollo aeroespacial. Este esfuerzo conjunta a miembros de la industria de EU y Canadá, gobierno y universidades para la creación de

herramientas, métodos y procesos que ayuden a sus afiliados en la creación de sistemas de la más alta calidad.

- ***Software Engineering Institute (SEI)***

  http://www.sei.cme.edu/

  Nace como resultado del ambicioso proyecto del *DoD* (Department of Defense) llamado *Software Technology for Adaptable and Reliable Systems* (*STARS*). El Instituto está asociado con el Carnegie-Mellon Institute.

  Fuera de los Estados Unidos, existen iniciativas para la investigación y la transferencia tecnológica. Entre ellos destacan:

- ***European Strategic Program for Research and Development in Technology (SPRIT)***

  http://www.cordis.lu/esprit/home.html

  Institución formada por la Comunidad Económica Europea. Diversas compañías en los países miembros son subvencionadas para el desarrollo de tareas del Programa.

- ***Alvey***

  Iniciativa de software independiente, domiciliada en Reino Unido, estrechamente relacionada a *SPRIT*.

- ***Metkit Project*** [80]

  http://www.sbu.ac.uk/~csse/metkit.html

  Centre for Systems and Software Engineering,

  South Bank University,

  London SE1 0AA,UK

  Tel +44 (0)171-815 7415

  Proyecto europeo desarrollado con el objetivo de establecer las métricas del software en un contexto más sistemático.

- ***Society for Software Quality***

  http://www.ssq.org/

  Esta institución tiene como objetivo conjuntar a todos aquéllos interesados en promover la calidad como meta universal del software. Su sitio cuenta con ligas a otras entidades relacionadas con el tema.

- ***Software Quality Institute (SQI)***

  http://utwired.engr.utexas.edu/sqi/

80 Citado en Fenton, N. (1991) Software metrics: A rigorous approach. E.U.: Chapman & Hall

Instituto de la Universidad de Texas dedicado a la educación e investigación en temas relacionados con la calidad del software.

- ***Software Quality Partners***

  http://www.sqp.com/

  Este organismo ofrece servicios de consultoría y pruebas para la construcción de software de calidad.

- ***SR Institute's Software Quality Hotlist***

  http://www.soft.com/Institute/HotList/

  Este sitio ofrece una serie de ligas interesantes a otros sitios relacionados con la calidad del software. La organización de la pagina permite accesos por temas y contiene una buena cantidad de sitios serios sobre el tema.

Anexo B: Procesos de software: el modelo del SEI

En la carrera de las organizaciones por incrementar su productividad y niveles de calidad en la creación de software, se suelen aplicar soluciones técnicas (herramientas CASE, nuevas metodologías) minimizando el cambio que debe gestarse en el factor humano. El uso de tecnología para resolver tales problemas no es la solución óptima cuando la organización no se ha planteado la pregunta ¿estamos listos para la nueva tecnología?

De acuerdo con Yourdon (1992) *"la mayoría de las organizaciones de desarrollo de software actuales son, desafortunadamente, muy primitivas para aprovechar la CASE, nuevas metodologías, o virtualmente cualquier tecnología compleja"*.

A principios de los 90 se desarrolló el concepto de *madurez organizacional*, en relación con el *proceso* usado por la organización para desarrollar sistemas. El concepto ha sido acuñado principalmente por el Software Engineering Institute (SEI) y es conocido como *Capability Maturity Model* (CMM).

Se debe aclarar que aunque el modelo del SEI sea el más popular, no es el único. Otras compañías de consultoría han ofrecido modelos con buenos resultados; incluso este modelo de reciente concepción está sujeto a revisiones y adecuaciones.

El modelo

La esencia del modelo del SEI es que las organizaciones de desarrollo de software pueden existir en uno de cinco niveles de madurez:

- Nivel inicial.
- Nivel repetitivo.
- Nivel definido.
- Nivel administrado.

- Nivel de optimización.

Esquemáticamente el modelo puede verse como una sucesión de niveles (Figura B-1), cuya importancia no radica en el número, sino en las diferencias reconocibles que hay entre las organizaciones que pasan de la anarquía a la madurez.

El modelo requiere el llenado de algunos cuestionarios de cierta complejidad que engloban más de cien preguntas; con sus ventajas y desventajas, pretende mediante el proceso, mostrar en qué punto de evolución se encuentra la organización para tomar acciones determinantes tendientes a su crecimiento.

Nivel 1: El nivel inicial.

Las características de este nivel son:

- Prevalece la anarquía.
- No existen reglas ni procedimientos comunes.
- Puede haber estándares, pero son generalmente ignorados.
- Puede haber herramientas, pero se usan irregularmente.
- Puede ser oficial el empleo de una metodología tal como el análisis y diseño estructurado, pero se aplica libremente a la conveniencia de los desarrolladores.

En este nivel, no significa que la organización tenga sólo gente incompetente, sino simplemente que los resultados son *impredecibles*. No se garantiza la entrega de productos a tiempo, no se sabe con certeza cuantas personas son necesarias, no se pueden medir riesgos.

El éxito o fracaso de los proyectos no depende de la naturaleza del proceso utilizado (pues no hay tal) o del profesionalismo del líder; depende enteramente de las capacidades —estados de ánimo y temperamento— de los individuos participantes.

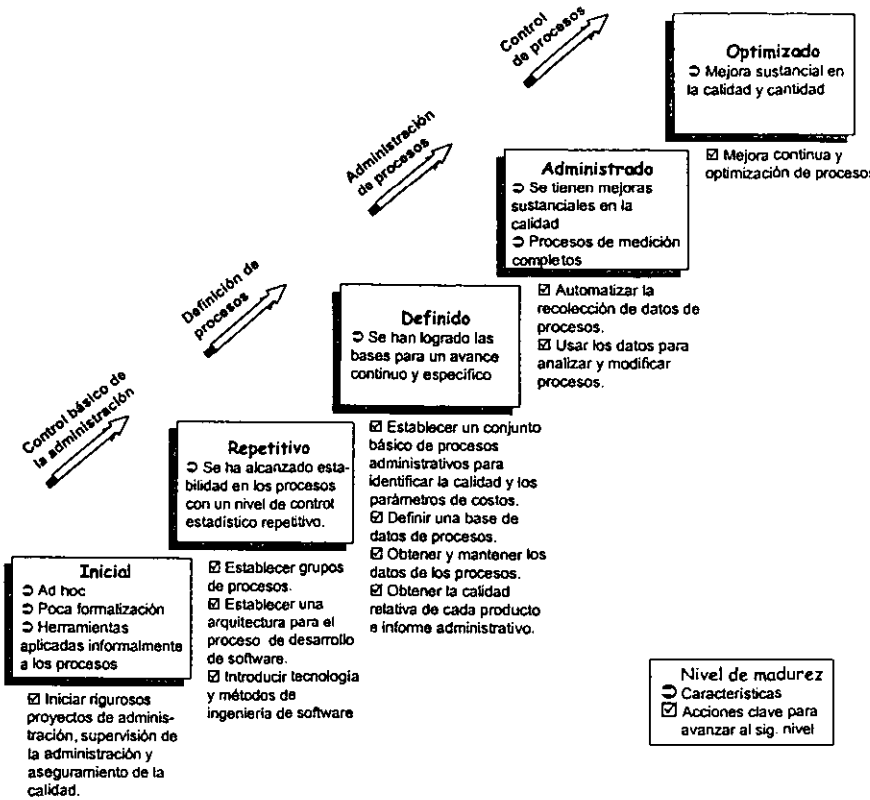


Figura B-1 Los niveles de madurez de procesos del SEI

Irónicamente, las empresas que viven en este nivel lo saben. Perciben el desorden cuando todos se quejan del estado de las cosas pero tienen poca idea de las acciones que deben tomar para mejorar. Cometen frecuentemente el error de contratar más personas para resolver cualquier tipo de problema.

Algunos puntos útiles que deben tomarse en cuenta para pasar al nivel 2, son los siguientes.

- *Estimar el tamaño del software*. Si no se conoce el tamaño del trabajo a realizar, no tiene ningún efecto el uso de herramientas o metodologías. Las

organizaciones en éste nivel tienden a minimizar la importancia del tamaño del software.

- *Evaluar el impacto de los cambios al software.* Frecuentemente se solicitan cambios aparentemente sencillos que se programan en las fases intermedias o finales del proyecto; si éstos no son evaluados y autorizados por la organización, habrá desvíos en tiempo, costos y esfuerzo. Las organizaciones en nivel 1 no saben el impacto de estos cambios y generalmente son asumidos por personal técnico de rango inferior.
- *Dar seguimiento a errores de código y de prueba.* Las organizaciones más avanzadas detectan errores en las fases iniciales del proyecto, mientras que las del nivel 1 esperan hasta la fase de pruebas, después de la programación. Aunque se detecten y reparen, las organizaciones de nivel 1 no dan seguimiento a los errores.
- *Calendarizar y estimar.* La organizaciòn de nivel 1 ocupa estimaciones informales, dadas comúnmente por personal técnico de bajo rango. El cálculo de tiempos de desarrollo es igualmente informal y no es habitual el uso de paquetes de administración que produzcan diagramas de Gantt o PERT.
- *Revisar el alcance de los proyectos.* Si existe demasiada presión, la organización acordará desarrollar el proyecto aunque se comprometa a fechas imposibles de cumplir y no cuente con suficiente presupuesto. De hecho, los factores politicos y personales inciden directamente en la toma de decisiones de una empresa de nivel 1, que por falta de un proceso de decisión, puede ser incapaz de saber si cumple sus cometidos.

Nivel 2: El nivel repetitivo.

La organización en este nivel ha logrado un proceso estable a un nivel repetitivo de control estadístico, alcanzado a través de la rigurosa administración de los proyectos, los costos, los tiempos y los cambios.

En el nivel 2 la empresa:

- Es estable y está bajo control.
- Cumple los calendarios de desarrollo y los presupuestos con una varianza estadísticamente aceptable.
- Logra sus éxitos empleando un estilo clásico de administración de proyectos: no utiliza metodologías avanzadas de ingeniería de software o herramientas CASE.
- Utiliza alguna metodología en la que se han capacitado todos los involucrados.
- Basa el éxito de los proyectos en el administrador del proyecto y no en el personal técnico.
- No depende del personal técnico para continuar con el desarrollo; sin embargo puede depender del líder del proyecto.

En este nivel, las empresas dirigen puntualmente el curso del desarrollo. Con esta dirección, sin embargo, se sabrá que fue equivocada sólo hasta que aparezca un error; repararlo será complicado y costoso.

En el nivel 2 la organización es razonablemente exitosa, no sufre severa rotación de personal y frecuentemente desarrolla software del mismo tipo. Para alcanzar la estabilidad de este nivel, se aconseja lo siguiente:

- *Hacer válida la administración del software*. La organización debe tener un proceso racional que pueda decir *NO* a metas que no pueda cumplir. De otra forma los líderes de proyecto estarán obligados a trabajar destinados al fracaso.
- *Planear el desarrollo de software y estimar costos*. Debe existir una forma más formal y sofisticada de planear y estimar que la usada en el nivel 1.
- *Controlar cambios y administrar la configuración del sistema*. Las modificaciones a los sistemas durante el desarrollo, así como los mantenimientos en esta fase, deben hacerse de manera ordenada.
- *Capacitar en ingeniería de software*. Esto es porque se debe contar con métodos formales, conocidos y aplicados por todos los individuos. La

experiencia dice que este tipo de capacitaciones se hicieron años antes, pero deben ser repetidas.

- *Ejecutar pruebas de regresión.* Como en el nivel 2 se cuenta con un control de cambios, se deben hacer pruebas que permitan rastrear el impacto de los cambios durante el desarrollo que aseguren que la funcionalidad ya instalada no se altere.
- *Recolectar datos sobre errores.* Los datos sobre defectos del software deben capturarse para una análisis posterior, pues los errores son más costosos y difíciles de reparar si no se descubren en etapas previas.
- *Establecer un grupo de procesos de software.* Esto con la finalidad de normar y difundir los procesos que se establezcan.
- *Establecer una organización de aseguramiento de la calidad del software.* Para comenzar a focalizarse en la calidad del sistema desarrollado y para ajustarse a tiempos y costos.

Nivel 3: El nivel definido.

En este nivel, la organización ha definido un proceso que asegura la implementación consistente de sus preceptos y provee las bases para el entendimiento del proceso mismo. En este punto se puede introducir mesuradamente tecnología avanzada.

Las características de este nivel son:

- Institucionalización del proceso. Se codifica "la nueva forma de hacer las cosas" a todos niveles de la organización.
- Se puede mejorar el proceso, ya que es universalmente conocido y se encuentra formalmente escrito.
- Se asume que los cambios al proceso se pueden dar sólo si todos los involucrados conocen y siguen rigurosamente el proceso actual.
- Para alcanzar este nivel se debe:

- *Establecer estándares formales*. Esto para diferenciar la manera informal del uso de estándares en el nivel 2.
- *Conducir inspecciones formales*. Para asegurarse de que el proceso se sigue y como un mecanismo general de aseguramiento de la calidad.
- *Adecuar políticas de pruebas más formales*. El establecimiento de casos de prueba puede ser parte de un trabajo formal, diferenciado del nivel 2, donde si bien existen pruebas, sus reglas pueden no ser muy rigurosas.
- *Utilizar formas más avanzadas de administración de la configuración del software*. En este nivel la organización aplica control a las actividades de análisis y diseño, implementa planes, establece herramientas automatizadas y provee facilidades para la administración de cambios y auditorias.
- *Establecer modelos formales de desarrollo*. Es importante pensar en el proceso de desarrollo de software como un *modelo* que pueda ser tratado como una abstracción mental. Las herramientas CASE pueden proveer diferentes modelos de procesos ajustables a las características propias de la organización.
- *Reforzar el grupo de ingeniería de procesos de software*. Este grupo es importante para la documentación, evolución y diseminación del proceso en la organización.

Nivel 4: El nivel administrado.

En este nivel la organización ha iniciado procesos de medición exhaustiva, más allá del costo y del cumplimiento de calendarios. Aquí es donde comienzan las mejoras substanciales a la calidad. En una organización de este nivel se distingue:

- El arranque de un programa de medición del software.
- El uso de las mediciones de los niveles anteriores —tamaño del software, personal involucrado, tiempo de desarrollo, esfuerzo desplegado, etc.—

adicionalmente a la medición del *proceso* mismo —tiempo utilizado en cada fase de desarrollo, tiempo estimado para revisiones, etc. —

- El énfasis en interpretar las datos recolectados para mejorar la calidad tanto del producto como del proceso que lo creó.
- El funcionamiento permanente de un grupo de aseguramiento de la calidad.

En este nivel se enfatiza la calidad del software y se establecen los siguientes postulados que el desafío que la calidad implica:

1. A menos de que se establezcan agresivas metas de calidad, nada cambiará.
2. Si esas metas no son numéricas, el programa de calidad será sólo un buen deseo.
3. Sin planes para la calidad, muy pocos se comprometerán con ella.
4. Los planes para la calidad no son efectivos si no están sujetos a seguimiento y revisión.

Nivel 5: El nivel de optimización.

En este nivel la administración de la organización y la instrumentación de sus procedimientos son retroalimentados para su mejora. Se cuentan con las bases para la mejora continua y la optimización de los procesos.

Así, la característica del nivel 5 es el énfasis formal en la continua mejora de los procesos con base en las mediciones hechas en el nivel 4.

La organización en este nivel debe tener un mecanismo que permita saber cómo el proceso es modificado.

Implicaciones del modelo.

Con el resumen anterior se puede tener cierta idea de la forma en que el modelo opera, introduciendo el concepto de madurez organizacional. Sin embargo, el modelo del SEI evidencia algunas implicaciones:

- No se pueden saltar niveles.
- Toma cierto tiempo pasar de un nivel de madurez a otro.
- Pocas organizaciones se encuentran más allá del nivel 1.
- En los niveles inferiores se debe evitar el uso de nuevas tecnologías.
- Las nuevas empresas de software no pueden comenzar en el nivel 3.
- El establecimiento del nivel de la organización puede repercutir en la firma de nuevos contratos de desarrollo.

Anexo C: El proyecto COQUAMO

Alguno del los trabajos de mayor importancia en el Reino Unido realizado por Kitchenham y otros, ha resultado en la herramienta ***COQUAMO***. Bajo el auspicio de los programas de investigación ALVEY y ESPRIT, se creó este modelo constructivo de calidad —*COnstructive QUality MOdel*— nombrado así luego de la aparición del modelo COCOMO —COnstructive COst MOdel—.

Este modelo forma la base de un paquete de software que asiste al desarrollador en su objetivo de crear un sistema de alta calidad.

La teoría usada en la herramienta incluye a la calidad en cinco visiones: trascendental, basada en el producto, basada en el usuario, basada en la manufactura y basada en el valor.

Para acomodar estas visiones de la calidad en un solo entorno, Kitchenham adiciona el concepto de *perfil de calidad*, haciendo una diferencia entre medidas objetivas y subjetivas de la calidad.

El perfil de calidad (Figura C-1) ve a la calidad del sistema como un todo y contiene las siguientes partes:

- *Propiedades trascendentales*. Se refiere a factores cualitativos de calidad que son difíciles de medir y sobre los que las personas pueden tener diferentes percepciones. La usabilidad del sistema es un ejemplo.
- *Factores de calidad*. Son características del sistema que son medibles (métricas de calidad) así como atributos de calidad. Los factores de calidad pueden ser características objetivas o subjetivas tales como confiabilidad y flexibilidad.
- *Indicadores de mérito*. Definen subjetivamente funciones del sistema. Las funciones se miden por índices de calidad cuyos valores son producto de la apreciación subjetiva.

El modelo COQUAMO apunta en tres direcciones:

a) Predecir la calidad del producto final

b) Monitorear el avance hacia la calidad

c) Retroalimentar con los resultados, para mejorar las predicciones en proyectos futuros.

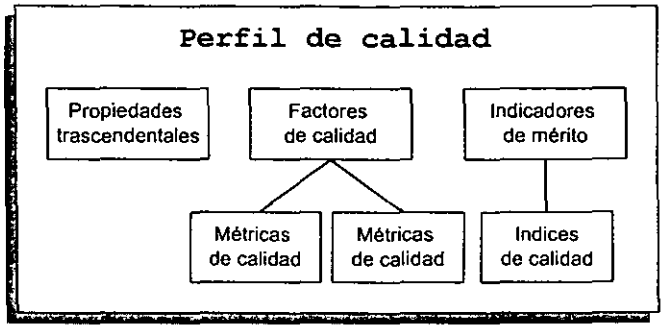


Figura C-1 El perfil de calidad definido por Kitchenham

COCOMO utiliza un ambiente similar a COQUAMO y se conforma de tres módulos: COQUAMO-1 para la calidad al principio del proyecto; COQUAMO-2 monitorea el manejo y cumplimiento de la calidad durante el desarrollo y COQUAMO-3 que determina la calidad del producto final. Una vez completado el ciclo, las mediciones tomadas retroalimentan al sistema para futuros proyectos. El esquema de la Figura C-2 muestra la relación entre los módulos.

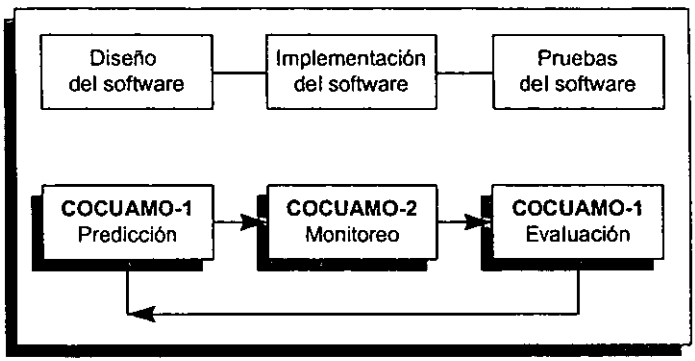


Figura C-2 El paquete COQUAMO

Los módulos predictivo y de evaluación reflejan la perspectiva de calidad del usuario, mientras que el módulo de monitoreo representa la visión de la calidad del desarrollador. Brevemente se describen los módulos:

COCUAMO-1: Predicción

La entrada de este módulo es un nivel de calidad *promedio* para cada factor de calidad considerado, derivado de proyectos similares. Estos valores se ajustan para determinar los factores que influencian los niveles de calidad, definiendo así los *quality drivers*.

Estos *quality drivers* son de cinco tipos:

- *Atributos de producto*, tales como requerimientos de calidad, que hacen crítico o difícil el desarrollo del producto en cuestión.
- *Atributos de proceso*, tales como maduración del proceso, uso del producto y maduración del método.
- *Atributos del personal*, tales como la experiencia y motivación de las personas involucradas en el proyecto.
- *Atributos del proyecto*, se refiere a normas de calidad y estilo de liderazgo.

- *Atributos organizacionales* son por ejemplo la gestión de la calidad y el ambiente físico establecidos para el proyecto.

Este módulo considera sólo los factores de la calidad que son comunes a la mayoría de las aplicaciones. Considera los siguientes conceptos:

a) *Confiabilidad.* Se define como el tiempo esperado para que ocurra una falla una vez que se libere el producto.

b) *Facilidad de mantenimiento.* Es el tiempo promedio esperado para encontrar una falla, una vez que se haya reportado.

c) *Facilidad de extensión.* Es la productividad promedio alcanzada en la modificación de código.

d) *Usabilidad.* Representa el tiempo esperado en el que no ocurren fallas.

e) *Reusabiliadad.* Se define como el esfuerzo necesario para crear módulos en la biblioteca de reuso, que potencialmente representan ahorros.

Las entradas de COQUAMO deben ser valores estimados, pero una vez que el proyecto se termina, los datos que emite COQUAMO-3 se retroalimentan y las estimaciones se empiezan a ajustar a la realidad.

COCUAMO-2: Monitoreo

Este módulo se basa en una guía de tareas necesarias para asistir el monitoreo de la calidad durante un proyecto. Esta guía apunta a:

- Identificar las métricas adecuadas a cada etapa del proceso de desarrollo.
- Indicar los métodos para establecer niveles para las métricas del proyecto.
- Sugerir métodos de análisis para identificar componentes inusuales.
- Indicar posibles causas de componentes inusuales y derivaciones en el *performance* .

- Indicar posibles acciones correctivas.

La automatización de éste módulo es una tarea compleja que se encuentra en proceso de desarrollo.

COCUAMO-3: Evaluación

Este módulo intenta dar datos acerca de la calidad del producto final. Está hecho tanto para validar las predicciones hechas en COQUAMO-1 como para proveer datos reales para COQUAMO-1 en futuros proyectos.

Esta herramienta representa una ayuda para el manejo y consecución de la calidad en un proyecto determinado pero tiene ciertas limitaciones:

- Requiere un registro de proyectos pasados, sobre los que basa las predicciones. Puede que estos datos no existan o se vean impactados por el uso de ciertas prácticas para incrementar la productividad, tales como las herramientas CASE.
- Depende mucho de apreciaciones subjetivas, que pueden variar según la experiencia.
- Excluye criterios de calidad comunes, tales como el nivel de *performance* o la portabilidad. Oros criterios que si se incluyen —como la usabilidad— son definidos en un sentido local.
- La efectividad de la herramienta no puede ser validada empíricamente.

Las limitaciones presentadas reflejan los problemas encontrados en la medición de la calidad del software. Sin embargo el esfuerzo es bueno: mientras exista la intención de hacer algo al respecto, aparecerán nuevas y más acertadas herramientas.

BIBLIOGRAFIA

BEHUNIAK, Jhon A.; AHMAD, Iftikhar A; COURTRIGHT, Ann M., **"Production software that works"**, Digital Press, Estados Unidos, 1992, 204 pp.

CARD, David N. & GLASS, Robert L., **"Measuring software design quality"**, Prentice Hall, Estados Unidos, 1990, 129 pp.

DUNN., Robert H., **"Software quality: concepts and plans"**, Prentice Hall, Estados Unidos, 1990, 196 pp.

DEMING, W. Edwards, **"Calidad, productividad y competitividad"** Ediciones Diaz de Santos S.A., España, 1989, 391 pp.

EVANS, Michael W., **"The software factory: a fourth generation software engineering environment"**, John Wiley & Sons, Estados Unidos, 1989.

FAIRLEY, Richard,**"Ingenieria de software"**, Mc Graw Hill, México, 1987, 389 pp.

FENTON, Norman E., **"Software metrics: a rigorous approach"**, Chapman & Hall, 1991, 416 pp.

GHEZZI, Carlo; JAZAYERI, Mendi; MANDRIOLI, Dino; **"Fundamentals of software engineering"**, Prentice Hall, Estados Unidos, 1991, 592 pp.

GILLIES, Alan C., **"Software quality, theory and management"**, Chapman & Hall, Londres, 1992, 258 pp.

HOPPER, James & Chester, Rowena O., **"Software reuse"**, Plenum Press, Nueva York y Londres, 1991, 180 pp.

INCE, Daniel, Editor, **"Software quality: reliability tools and methods"**, Chapman & Hall, Unicom Seminaries, Londres, 1991.

LARIOS, J. José, **"Hacia un modelo de calidad"**, Grupo Editorial Iberoamérica, México, 1990, 160 pp.

PFLEEGER, Shari, **"Software engineering: the production of quality software"**, Mc. Millan Press, Inglaterra 1991.

PRESSMAN, Roger S., **"Ingeniería del software: un enfoque práctico"**, McGraw-Hill, España, 1993, 884 pp.

SHILLER, Larry, **"Software excellence"**, Yourdon Press, Computer Series, Estados Unidos, 1990, 239 pp.

SOMMERVILLE, Ian, **"Software engineering"** 4a. ed., Addison Wesley, Inglaterra, 1992, 649 pp.

UDAONDO Duran, Miguel, **"Gestion de calidad"**, Ediciones Diaz de Santos S.A., España, 1992, 343 pp.

VAN GENUCHTEN, Michiel, **"Towards a software factory"**, Kluwer Academic Publishers, Holanda, 1992, 173 pp.

VAN VLIET, Hans, **"Software engineering, principles and practice"**, John Wiley & Sons, Estados Unidos, 1993, 748 pp.

VON MAYRHAUSER, Anneliese, **"Software engineering, methods and management"**, Academic Press, USA, 1990, 864 pp.

YOURDON, E., **"Decline and fall of the american programmer"**, Prentice Hall, Estados Unidos, 1992, 352 pp.

HEMEROGRAFIA

BOITEN, E. A., et. al., **"How to produce correct software, an introduction to formal specification and program development by transformation"**, en *The*

Computer Journal, Cambridge University Press, Inglaterra, vol. 35, num. 6, diciembre de 1992.

CONWAY, B. et. al., **"Just enough process for AD"**, en *Datapro Research notes*, febrero de 1997, 3 pp.

CONWAY, B. et. al., **"The ASQ market: what the heck is going on?"**, Research notes en *Datapro*, mayo de 1997, 4 pp.

DEMING, E., **"Los 14 principios de mi filosofía de software"**, en *Forum Calidad* vol.13, num. 90.

GEZZI, Carlo & FUGGETTA, Alfonso, **"State of the art and open issues in PSEEs"**, en *Abstract del Politecnico di Milano and CEFRIEL*, Italia, 1995, 13 pp.

HOCHSTRASSER, B., **"Quality engineering: a new framework applied to justifying and prioritising it investments"**, en *European Journal of Information Systems*, Londres, vol 2., num. 3, julio de 1993.

HOTLE, R., **"Climbing the CMM: how long can you tread water?"**, Monthy research review en *Datapro*, noviembre de 1998, 2 pp.

HUNTER, R., et. al., **"Methodology and productivity study: the analysis"**, Research notes en *Datapro*, junio de 1997, 2 pp.

HUNTER, R. et. .al., **"From knowledge management to tought management"**, en *Datapro Research notes*, agosto de 1998, 3 pp.

KNIGHT, John C. & MYERS E. Ann, **"An improved inspection technique"**, en *Communications of the ACM*, publicación mensual de The Association for Computing Machinery, Estados Unidos, vol. 36, num. 11, noviembre de 1993.

MAGGE, Stan, **"Software life cycle cost"**, en *Datapro*, enero de 1996, 14 pp.

MAGEE, Stan, **"The way to iso 9001 certification for software firms"**, en *Datapro*, 15 de noviembre de 1996, 30 pp.

MAGGE, Stan, **"Software process improvement programs (SPIP)"**, en *Datapro*, marzo de 1997, 16 pp.

MAGGE, Stan, **"ISO 9000-3"**, en *Datapro*, marzo de 1998, 25 pp.

MAGGE, Stan, **"implementing 12207"**, en *Datapro*, junio de 1998, 7 pp.

MC.DERMIT, John & Rook, Paul, **"Software developement process models"**, en *Bulletin of the University of York*, 1995, 35 pp.

MILLS, Harlan, D., **"Zero defect software: cleanroom engineering"**, en *Advances in Computers*, Yovits, M., editor, Academic Press, Estados Unidos, vol. 36, 1993, p. 1

MIYOSHI Takeshige & AZUMA, Motoei, **"An emprirical study of evaluating software development environment quality"**, en *IEEE Transactions on software Engineering*, Estados Unidos, vol. 18, num. 5, Mayo de 1992, p. 425.

MUSA, John D. & IANNINO, A., **"Software reliability"**, en *Advances in Computers*, YOVITS, M., editor, Academic Press, Estados Unidos, vol. 30, 1990, p.85.

RAYMOND, J. Rubey, **"Aseguramiento de la calidad del software"**, Notas del curso *Software Quality Assuance (SQA)* impartido por Tecnology Training, México, febrero de 1994.

PLISKIN, N. et. al., **"Presumed versus actual organizatoinal culture: managerial implications for implementation of infotmation systems"**, en *The Computer Journal*, Cambridge University Press, Inglaterra, vol. 36, num. 2, abril de 1993.

SHNEIDEWIND, Norman F., **"Methodology for validating software metrics"**, en *IEEE Transactions on software Engineering*, Estados Unidos, vol. 18, num. 5, Mayo de 1992, p. 410.

SUTCLIFFE, A.G. et. al., **"Integrating human computer interaction with Jackson system developement"**, en *The Computer Journal*, Cambridge University Press, Inglaterra, vol. 34, num. 2, abril de 1991.

VOSS, A. **"The real importance of software quality and the role of quality systems"**, en Ince, D. (Ed) (1991). *Software Quality, reliability tools and methods*. EU: Chapman.