CAPÍTULO 11
CALIDAD DEL SOFTWARE

ACRÓNIMOS

CMMI	Capability Maturity Model Integrated
COTS	Commercial Off-the-Shelf Software
PDCA	Plan, Do, Check, Act
SQA	Software Quality Assurance
SQM	Software Quality Management
TQM	Total Quality Management
V&V	Verificación y Validación

INTRODUCCIÓN

¿Que es la calidad de software, y por qué es tan importante como para estar omnipresente en la Guía SWEBOK? Durante años, los autores y organizaciones han definido el término "calidad" de manera diferente. Para A Phil Crosby (Cro79), fué " la conformidad a las exigencias de usuario. " Watts Humphrey (Hum89) se refiere a calidad como " el alcanzar los niveles excelentes de salud para el empleo" mientras IBM acuñó la frase " la calidad conducida por el mercado, " frase basada en el objetivo de alcanzar la satisfacción de cliente total.

Los criterios Bladridge para la calidad organizacional utilizan una frase similar “calidad conducida por el cliente,” e incluye la satisfacción del cliente como una consideración mayor. Más recientemente, la calidad se ha definido en (ISO9001-00) como “el grado en que un conjunto de características inherentes cumple requisitos.”

Este capítulo estudia los aspectos relativos a la calidad de software los cuales transcienden a los procesos del ciclo de vida. La calidad de software es un aspecto ubicuo en la ingeniería de software, y por lo tanto también es tratado en mucho de los KAS. En el sumario, la Guía SWEBOK describe un conjunto de modos de alcanzar la calidad del software. En particular, este KA tratará las técnicas estáticas, es decir, aquellas que no requieren la ejecución del software para su evaluación, mientras que las técnicas dinámicas son cubiertas en el KA referido a Pruebas del Software.

DESGLOSE DE LOS TEMAS EN CALIDAD DEL SOFTWARE

1. Fundamentos de Calidad de Software

Un acuerdo sobre exigencias de calidad, así como trasladar a la ingeniería del software qué constituye calidad, requiere que muchos de los aspectos del concepto calidad sean formalmente definidos y tratados.

Un ingeniero de software debería entender los significados subyacentes en los conceptos y características de calidad y su relevancia en el desarrollo o mantenimiento de software.

El concepto relevante es que los requerimientos del software definen las características de calidad requeridas de ese software e influyen en los métodos de medición y criterios de aceptación para evaluar estas características.

1.1. Ingeniería del Software Cultura y Ética.

Los ingenieros de software esperan compartir un compromiso sobre calidad de software como una parte de su cultura. Una cultura sana sobre ingeniería del software se describe en [Wie96].

La ética puede jugar un papel significativo en la calidad de software, la cultura, y las actitudes de ingenieros de software. La IEEE Computer society y el ACM [IEEE99] han desarrollado un código de ética y práctica profesional basada en ocho principios con el objetivo de ayudar a los ingenieros de software a reforzar actitudes relacionadas con la calidad y con la independencia de su trabajo.

1.2. Valor y coste de la calidad.

El concepto de calidad no es tan simple como parece, para un ingeniero de productos hay muchas calidades deseadas relevantes para una perspectiva determinada de un producto, para que esto pueda ser tratado y determinado en el tiempo las exigencias de producto son puestas por escrito. Las características de calidad pueden requerirse o no, o se pueden requerir en un mayor o menor grado, y pueden hacerse compensaciones entre ellas. [Pfl01] El coste de calidad puede segmentarse en el coste de prevención, el coste de apreciación, el coste de fracaso interno, y el coste de fracaso externo. [Hou99]

La motivación latente tras un proyecto de software es el deseo de crear un software que tiene valor, y este valor puede o no puede ser cuantificado como un coste. El cliente tendrá en mente algún coste máximo, a cambio del cual espera que se cumpla el objetivo básico del software. El cliente también puede tener alguna expectativa en cuanto a la calidad del software. En ocasiones los clientes pueden no haber estudiado detenidamente las cuestiones de calidad o sus gastos relacionados. ¿La calidad es meramente decorativa, o es consustancial al software? Si la respuesta se sitúa en un punto intermedio, como es casi siempre el caso, es cuestión de hacer del cliente parte del proceso de decisión y concienciarle totalmente tanto de los costes como de los beneficios.

Idealmente, la mayor parte de estas decisiones serán adoptadas en el proceso de requerimientos de software (vd. El KA Requerimientos del software), sin embargo estas cuestiones pueden surgir en todas las etapas del ciclo de vida del software. No hay ninguna regla definida en cuanto a como deber ser adoptadas estas decisiones, pero el ingeniero de software debería ser capaz de presentar alternativas de calidad y sus correspondientes costes. Una discusión acerca del coste y el valor de los requerimientos de calidad puede encontrarse en [Jon96:c5; Wei96:c11].

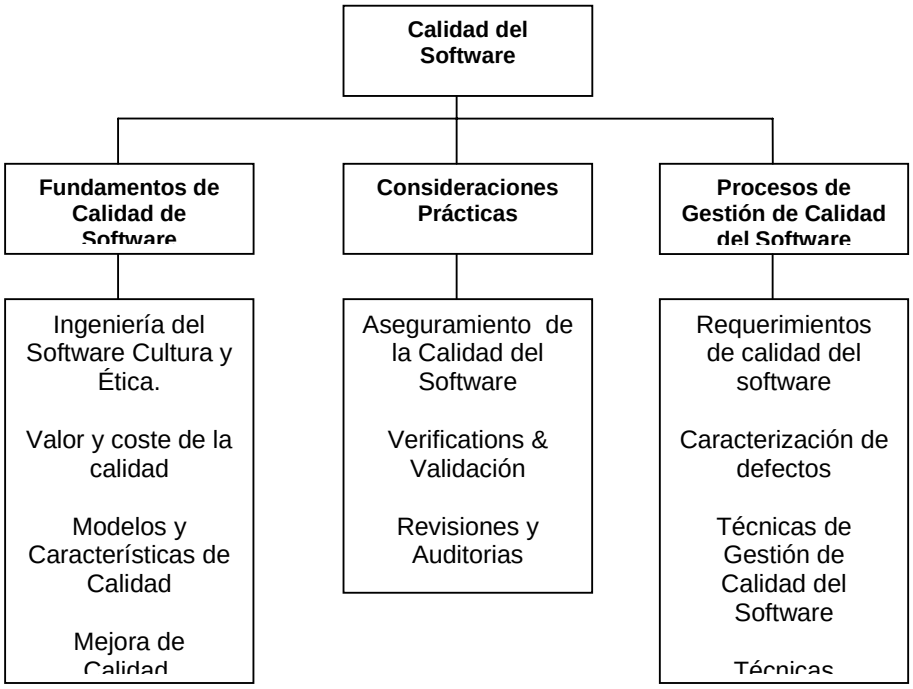

Figura 1 Desglose de los temas de Calidad del Software.

1.3 .Modelos y Características de Calidad.

[Dac01; Kia95; Lap91; Lew92; Mus99; NIST; Pre01; Rak97; Sei02; Wal96].

La terminología para las características de calidad del software difiere de una taxonomía (o modelo de calidad de software) a otra, cada modelo quizás tenga un número diferente de niveles jerárquicos y un número total diferente de características. Varios autores han enunciado distintos modelos de características de calidad de software o atributos que pueden ser útiles para la negociación, planificación, y tasación de la calidad de productos software. [Boe78; McC77] ISO/IEC ha definido tres modelos relacionados de calidad de productos software (la calidad interna, la calidad externa, y la calidad en el empleo) (ISO9126-01) y un conjunto de partes relacionadas (ISO14598-98).

1.3.1. La calidad del proceso en la ingeniería del software.

La gestión de la calidad de software y la calidad de proceso en la ingeniería de software guarda relación directa con la calidad del producto software.

Los modelos y los criterios que evalúan las capacidades organizacionales en software son esencialmente la organización de proyecto y consideraciones de gestión, y, como tales, son tratados en los KAs relativos a Gestión en Ingeniería del Software y el Proceso en Ingeniería de Software.

Desde luego, no es posible distinguir completamente la calidad del proceso de la calidad del producto.

La calidad de proceso, tratada en el KA, de esta Guía, el Proceso en Ingeniería de Software, afecta a las características de calidad de los productos software, que a su vez repercuten en la calidad-en-el-uso tal y como es percibido por el cliente.

Dos importantes estándares de calidad son TickIT [Llo03] y uno con impacto sobre la calidad de

software, el estándar ISO9001-00, con sus directrices para su aplicación al software [ISO90003-04].

Otro estándar industrial en calidad del software es el CMMI [SEI02], también tratado en el KA Proceso en la Ingeniería de Software. CMMI pretende proporcionar directrices para mejorar procesos.

Específicamente las áreas de procesos relacionadas con la gestión de calidad son: a) Aseguramiento de la calidad en el proceso y el producto, (b) la verificación de proceso, y c) la validación de proceso. CMMI clasifica revisiones y auditorias como los métodos de verificación, y no como procesos específicos como (IEEE12207.0-96).

Hubo inicialmente algún debate sobre si ISO9001 O CMMI deberían ser usado por ingenieros de software para asegurar la calidad. Este debate ha sido profusamente publicado, y, como resultado, se ha concluido que los dos resultan complementarios y que tener la certificación ISO9001 puede ayudar enormemente par alcanzar los niveles de madurez más altos del CMMI. [Dac01].

1.3.2. Calidad de producto software.

El ingeniero de software, ante todo, necesita determinar el Objetivo verdadero del software. En cuanto a esto, es de capital importancia tener presente los requerimientos del cliente y aquellos que estos incluyen como requerimientos de calidad, no únicamente los requerimientos funcionales. Así, el ingeniero de software tiene como responsabilidad obtener los requerimientos de calidad, que pueden no estar explícitos en un principio, tratar su importancia así como el nivel dificultad para alcanzarlos. Todos los procesos asociados a la Calidad de software (como por ejemplo, construcción, pruebas, mejora de la calidad) serán diseñados con estas exigencias en mente, y ello conlleva gastos adicionales.

El estándar (ISO9126-01) define, para dos de sus tres modelos de calidad, las características de calidad mencionadas, las Sub-características, y las medidas que son útiles para Evaluación de calidad de producto de software. (Sur03)

El significado del término "producto" es ampliado para incluir cualquier artefacto que es la salida de cualquier proceso empleado para construir el producto de software final. Como ejemplos de un producto cabe incluir, aunque no con carácter limitativo, una completa especificación del sistema, una especificación de requerimientos de software para un componente de software de un sistema, un módulo de diseño, código, documentación de prueba, o los informes producidos como consecuencia de tareas de análisis de calidad. Mientras la mayor parte del tratamiento de la calidad es descrito en términos del software final y funcionamiento del sistema, una ingeniería práctica responsable requiere que los productos intermedios relevantes para la calidad sean evaluados a lo largo de todo el proceso de ingeniería de software.

1.4. Mejora de Calidad.

[NIST03; Pre04; Wei96]

La calidad de los productos software puede ser mejorada mediante un proceso iterativo de mejora continua que requiere control de dirección, coordinación, y retroalimentación de muchos procesos simultáneos: (1) los procesos de ciclo de vida de software, (2) El proceso de detección de error/defecto, retirada de los mismo y prevención, (y 3) el proceso de mejora de calidad. (Kin92) La teoría y conceptos presentes detrás de mejora de calidad, tales como la construcción en calidad, mediante la prevención y detección temprana de errores, mejora continua y enfoque en el cliente, son adecuados para la ingeniería de software. Estos conceptos están basados en el trabajo de expertos en calidad los cuales ha afirmado que la calidad de un producto está directamente conectada con la calidad del proceso empleado para crearlo.

Aproximaciones tales como Total Quality Management (TQM) process of Plan, Do, Check, and Act (PDCA) son Instrumentos mediante los cuales conocer los objetivos de calidad. El apoyo a la gestión sustenta el proceso y la evaluación del producto así como las conclusiones resultantes. Entonces se desarrolla un programa de mejora identificando acciones detalladas y proyectos de mejora para ser gestionados en un plazo de tiempo factible. El apoyo a la gestión implica que cada proyecto de mejora tiene suficientes recursos para alcanzar el objetivo definido. El apoyo a la gestión ha ser solicitado con frecuencia mediante la implementación proactiva de actividades de comunicación. La participación de los equipos de trabajo, así como el apoyo a la gerencia media y los recursos asignados en el nivel de proyecto, son tratados en el KA Proceso de Ingeniería de Software.

2. Procesos de Gestión de Calidad del Software

La gestión de calidad de software (SQM) resulta de aplicación a todas las perspectivas de procesos de software, productos, y recursos. Esto define procesos, propietarios de proceso, y requerimientos para aquellos procesos, medidas del Proceso y sus correspondientes salidas, y canales de retroalimentación. (Art93) Los procesos de gestión de calidad del software consisten en numerosas actividades. Algunos de ellos pueden encontrar defectos directamente, mientras otros indican donde pueden resultar valiosas más revisiones. Estos últimos también son conocido como actividades de " direct-defect-finding ". Muchas actividades a menudo sirven para ambos propósitos.

La planificación para la calidad de software implica:

(1) Definición del producto requerido en términos de sus características calidad (descrito más detalladamente en, por ejemplo, El KA Gestión en Ingeniería del Software).
(2) Planificación de los procesos para alcanzar el producto requerido (Descrito, por ejemplo, en los Kas, Diseño de Software y Construcción de Software).

Estos aspectos difieren de, por ejemplo, los procesos mismos de planificación SQM, que evalúan las características de calidad planificadas versus la implementación actual de esa planificación. **Los procesos de gestión de calidad de software deben dirigirse a como los buenos productos software van a satisfacer o satisfacen al cliente y las exigencias del personal implicado, a como proporcionan valor a los clientes y demás personal implicado, y proveen la calidad de software precisa para conocer los requerimientos del software.**

El SQM puede ser utilizado para evaluar productos intermedios así como el producto final.

Algunos de los procesos específicos SQM están definidos en el estándar (IEEE12207.0-96):

- Procesos de aseguramiento de calidad
- Procesos de verificación
- Procesos de validación
- Procesos de revisión
- Procesos de auditoría

Estos procesos incentivan la calidad y también permiten encontrar posibles problemas. Sin embargo presentan diferencias en cuanto a su énfasis.

Los procesos SQM ayudan a asegurar una calidad de software óptima en un proyecto dado. Además proveen, como un subproducto, información general sobre gestión, incluyendo directrices de calidad para todo el proceso de ingeniería del software. Las Kas Proceso de la Ingeniería del Software y Gestión en Ingeniería del Software tratan sobre la calidad de los programas para la organización que desarrolla el software. El SQM puede proporcionar retroalimentación relevante para estas áreas.

Los procesos SQM consisten en tareas y técnicas para indicar como los proyectos de software (por ejemplo, la gestión, el desarrollo, la gestión de configuración) están siendo puestos en práctica y la mejor manera para que los productos intermedios y los finales encuentren sus requerimientos especificados. Los resultados de estas tareas son recopilados en informes para la dirección antes de que sea tomada la acción correctiva. La gestión de un proceso SQM se desarrolla con la certeza de que los resultados de estos informes son exactos.

Como se describe en este KA, los procesos SQM están estrechamente relacionados; pueden solaparse y hasta, en ocasiones, estar combinados. En su mayor parte parecen de naturaleza reactiva ya que toman los procesos como practicados y los productos como producidos; sin embargo tienen un papel principal en la fase de planificación, con carácter proactivo en términos de los procesos y los procedimientos precisos para alcanzar las características y grados de calidad requerida por los sujetos implicados en el software.

La gestión del riesgo también puede jugar un papel importante en la entrega de software de calidad. La incorporación de un análisis de riesgo disciplinado y técnicas de gestión en los procesos de ciclo de vida de software puede incrementar el potencial para producir un producto de calidad (Cha89). Refiérase al KA Gestión en la Ingeniería del Software para el material relacionado sobre gestión de riesgos.

2.1. Aseguramiento de la Calidad del Software

[Ack02; Ebe94; Fre98; Gra92; Hor03; Pfl01; Pre04; Rak97; Sch99; Som05; Voa99; Wal89; Wal96]

Los procesos SQA proporcionan la garantía de que los productos software y los procesos en el ciclo de vida de proyecto son conformes a los requerimientos especificados por medio de la planificación, emitiendo, y realizando un conjunto de actividades para generar la confianza adecuada en que se está construyendo calidad dentro del software.

Ello significa asegurar que el problema está clara y suficientemente identificado y que los requerimientos de la solución están correctamente definidos y expresados. El SQA procura mantener la calidad a lo largo de todo el desarrollo y mantenimiento del producto mediante la ejecución de una variedad de actividades en cada etapa que pueden permitir identificación temprana de problemas, un rasgo casi inevitable de cualquier actividad compleja. El papel del SQA en lo que concierne al proceso es asegurar que procesos planificados son apropiados y posteriormente implementados de acuerdo con la planificación, y se proveen procesos de medición relevantes para una adecuada organización.

El plan SQA define el medio que será usado para asegurar que el software desarrollado para un producto específico satisface las exigencias del usuario y es de la máxima calidad posible dentro de las restricciones del proyecto. Con el objetivo de llevar esto acabo, primero debe asegurarse que el objetivo de calidad es claramente definido y entendido. En ello deben considerarse los planes de gestión, desarrollo, y mantenimiento para el software. Ver el estándar (IEEE730-98) para detalles.

Las actividades y tareas específicas de calidad se elaboran, con sus gastos y exigencias de recursos, sus objetivos totales de gestión, y su programa en relación con aquellos objetivos de gestión en la ingeniería, el desarrollo, o planes de mantenimiento. El plan SQA debería ser compatible con el plan de gestión de configuración de software (refiérase al KA

Gestión de Configuración de Software). El plan SQA identifica documentos, normas, prácticas, y convenciones que guían el proyecto y de qué manera serán comprobados y supervisados para asegurar adecuación y conformidad. El plan SQA también identifica medidas, técnicas estadísticas, procedimientos para el reporte de problemas así como la correspondiente acción correctiva, recursos tales como herramientas, técnicas, y metodologías, seguridad para el medio físico, formación, además de reportes y documentación SQA. Por otro lado, el plan SQA considera las actividades de garantía de calidad de software como cualquier otro tipo de actividad descrita en los proyectos de software, tales como la consecución de proveedor de software para el proyecto o el software de instalación comercial disponible (COTS), así como el servicio tras la entrega del software. También puede incluir criterios de aceptación así como reportes y actividades de gestión críticas para la calidad de software.

2.2. Verificaciones y Validación

[Fre98; Hor03; Pfl01; Pre04; Som05; Wal89; Wal96]

Con el propósito de ser breve, Verificación y Validación (V&V) son tratadas como un único asunto en esta Guía más que como dos asuntos separados tal y como se hace en el estándar (IEEE12207.0-96). La V&V del software es un acercamiento disciplinado a la evaluación de productos de software a lo largo de todo el ciclo de vida de producto. Un esfuerzo en V&V es esforzarse en asegurar que la calidad es construida dentro del software y que el software satisface exigencias de usuario " (IEEE1059-93).

La V&V trata directamente la calidad de producto software y emplea técnicas de prueba que pueden localizar defectos de tal manera que estos puedan ser tratados. También evalúa los productos intermedios, como, y, en esta capacidad, los pasos intermedios de los procesos de ciclo de vida de software.

El proceso V&V determina si productos de una actividad dada de desarrollo o mantenimiento se adecuan o no al correspondiente requisito de esa actividad, y si el producto final de software cumple o no cumple con su propósito fijado y converge o no con los requisitos del usuario. La Verificación es un intento para asegurar que el producto se construya correctamente, en el sentido que los productos resultantes de una actividad converjan con las especificaciones fijadas para los mismos en actividades previas. La validación es un intento por asegurar que se construye el producto correcto, es decir, que el producto satisface su propósito específico fijado. Tanto el proceso de comprobación como el proceso de validación empiezan temprano en la fase de desarrollo o mantenimiento. Proporcionan un examen de características claves del producto en relación con el producto inmediato predecesor y a las especificaciones con las que debe converger.

El propósito de la planificación V&V es asegurar que cada recurso, papel y responsabilidad está claramente asignada. Los documentos del proyecto V&V resultantes describen varios recursos y sus papeles y actividades, así como técnicas y herramientas para ser usados. La comprensión de los objetivos diferentes de cada actividad V&V ayudará en la planificación cuidadosa de las técnicas y los recursos precisos para alcanzar sus respectivos objetivos. Estándares específicos (IEEE1012-98:s7 y IEEE1059-93: El apéndice A) que generalmente incluye un plan de V&V.

El plan también considera la gestión, la comunicación, la política, y los procedimientos de las actividades V&V y su interacción, así como el reporte de defectos y exigencias de documentación.

2.3. Revisiones y Auditorias

Con el propósito de ser breve, revisiones y auditorías son tratadas como un solo tema en esta Guía, más que como dos temas separados tal y como se hace en (IEEE12207.0-96). La revisión y el proceso de auditoría son ampliamente definidos en (IEEE12207.0-96) y más detalladamente en (IEEE1028-97). Cinco tipos de revisiones o auditorias se presentan en el estándar IEEE1028-97:

- Revisiones de gestión
- Revisiones técnicas
- Inspecciones
- Walk-throughs
- Auditorias

2.3.1. Revisiones de gestión

El objetivo de una revisión de gestión es supervisar el progreso, determinando el estado de planes y programas, requerimientos confirmados y su sistema de localización, o evaluar la efectividad de los enfoques de gestión empleados para lograr la idoneidad del objetivo. [IEEE1028-97]. Ello apoya decisiones sobre cambios y las acciones correctivas precisas durante un proyecto de software. Las revisiones de gestión determinan la idoneidad de los proyectos, programas, y requerimientos y supervisan su progreso o inconsistencias. Estas revisiones pueden ser realizadas sobre productos tales como informes de auditoria, informes de progreso, informes V&V, y proyectos de muchos tipos, incluyendo la gestión de riesgo, gestión del proyecto, gestión de configuración del software, seguridad del software, y la evaluación de riesgo, entre otros. Refiérase para el material relacionado a los Kas Gestión en Ingeniería del Software y Gestión de Configuración de Software.

2.3.2. Revisiones técnicas

[Fre98; Hor03; Lew92; Pfl01; Pre04; Som05; Voa99; Wal89; Wal96]

"El propósito de una revisión técnica es evaluar el producto software para determinar si es idóneo para su correspondiente uso. El objetivo es identificar discrepancias con especificaciones aprobadas y estándares. El resultado debería proporcionar gestión con evidencias (o no) de que el producto converge con sus especificaciones y se adhiere a los estándares, y que los cambios están controlados. (IEEE1028- 97).

En una revisión técnica deben estar establecidos los roles específicos: el que adopta las decisiones, un líder revisor, un registrador, y un staff técnico para apoyar las actividades de revisión. Una revisión técnica requiere que las entradas obligatorias estén en su lugar con el objeto de proceder a:

- Exposición de objetivos
- Un producto software específico
- El plan específico de gestión del proyecto
- La lista de cuestiones claves asociadas al producto
- El procedimiento de revisión técnica

El equipo sigue el procedimiento de revisión. Un individuo técnicamente calificado presenta una descripción del producto, y el examen se lleva a cabo durante una o varias reuniones. La revisión técnica es completada una vez que todas las actividades catalogadas en el examen han sido completadas.

2.3.3. Inspecciones
[Ack02; Fre98; Gil93; Rad02; Rak97]

El propósito de una inspección es detectar e identificar anomalías en los productos software" (IEEE1028-97).Existen dos importantes elementos diferenciadores entre inspección y revisión, son los siguientes:

1. Un individuo que mantiene una posición de dirección sobre cualquier miembro del equipo de inspección no participará en la inspección.

2. La inspección ha de ser llevada por un inspector con formación en inspecciones técnicas

Las inspecciones de software siempre implican al autor de un producto intermedio o final, mientras otras revisiones puede que no. Las inspecciones también incluyen a un líder de inspección, un registrador, un lector, y un grupo (2 a 5) de inspectores. Los miembros de un equipo de inspección pueden poseer diferentes especializaciones, como especialización en el dominio, especialización en el método de diseño, o especialización en el lenguaje. Las inspecciones por lo general son llevadas a cabo sobre una relativamente pequeña sección del producto a la vez. Cada miembro del equipo debe examinar el producto software así como practicar otras revisiones de imputs con anterioridad a la reunión de revisión, quizás aplicando una técnica analítica (referida en la sección 3.3.3) en una pequeña porción del producto, o en todo el producto enfocando solo un aspecto, por ejemplo, interfaces. Cualquier anomalía encontrada se documenta y se envía al responsable de la inspección. Durante la inspección, el responsable de la inspección conduce la sesión y verifica que todos están preparados para la misma.

Un herramienta comúnmente usada en las inspecciones es una lista de comprobación, con anomalías y preguntas pertinentes sobre las cuestiones de interés. La lista resultante a menudo clasifica las anomalías (refiérase a IEEE1044-93 para detalles) y se revisa por parte del equipo su exactitud e integridad. La decisión sobre el final de la inspección se corresponde con uno de los tres criterios siguientes:

1. No aceptar re-trabajo o como mucho un re-trabajo menor
2. Aceptar con verificación del re-trabajo
3. Re-inspección

Típicamente las reuniones de inspección duran por lo general algunas horas mientras que las revisiones técnicas y auditorias son por lo general de mayor alcance y requieren más tiempo.

2.3.4. Walk-throughs
[Fre98; Hor03; Pfl01; Pre04; Som05; Wal89; Wal96]

"El objetivo de un Walk-throughs es evaluar un producto de software. Un Walk-throughs puede ser conducido para el objetivo de formar a una audiencia en cuanto a un producto de software." (IEEE1028-97) los objetivos principales son [IEEE1028-97]:

- Encontrara anomalías
- Mejorar el producto software
- Considerar implementaciones alternativas
- Evaluar la conformidad con estandares y especificaciones

El Walk-throughs es similar a una inspección, sin embargo, su desarrollo, por lo general, es menos formal. El Walk-throughs es organizado fundamentalmente por el ingeniero de software para dar a sus compañeros de equipo la oportunidad de repasar su trabajo, como una técnica de aseguramiento.

2.3.5. Auditorias
[Fre98; Hor03; Pfl01; Pre01; Som05;Voa99; Wal89; Wal96]

"El objetivo de una auditoría de software es proporcionar una evaluación independiente de la

conformidad de productos software y procesos a regulaciones aplicables, estándares, directrices, planes, y procedimientos " [IEEE1028-97]. La auditoría es una actividad formalmente organizada, con participantes que tienen papeles específicos, como el auditor jefe, otro auditor, un registrador, o un iniciador, e incluye a un representante de la organización auditada.

La auditoría identificará los casos de no conformidad y producirá un informe requiriendo al equipo que adopte la acción correctiva correspondiente.

Mientras pueden haber muchos nombres formales para revisiones y auditorias como los identificados en el estándar (IEEE1028-97), La clave es qué revisiones y auditorias pueden practicarse sobre casi cualquier producto en cualquier etapa del proceso de mantenimiento o el desarrollo.

3. Consideraciones Prácticas

3.1. Requerimientos de calidad del software

[Hor03; Lew92; Rak97; Sch99; Wal89; Wal96]

3.1.1. Factores de influencia

Varios factores influyen en la planificación, gestión, y selección de actividades de SQM y técnicas, incluyendo:

- El dominio del sistema en el cual el software residirá (seguridad crítica, misión crítica, negocio crítico)
- Requerimientos del Sistema y del software
- Los componentes comerciales (externos) o estándar (internos) que serán usados por el sistema
- Los estándares específicos de ingeniería del software específico aplicables
- Los métodos y herramientas de software para ser usados en el desarrollo y el mantenimiento así como para la evaluación de calidad y mejora
- El presupuesto, el personal, planes de organización, proyectos, y la planificación de todos los procesos
- Los usuarios implicados y el empleo del sistema
- El nivel de integridad del sistema

La Información sobre estos factores de influencia como los procesos SQM son organizados y documentados, son seleccionadas como actividades SQM específicas, que recursos son necesarios, y que impondrá límites a los esfuerzos.

3.1.2. Confiabilidad

En casos en los que el fracaso del sistema puede tener consecuencias sumamente severas, la confiabilidad total (en hardware, el software, y humanos) son la exigencia de calidad principal además de la funcionalidad básica. La confiabilidad del software incluye características tales como tolerancia al defecto, fiabilidad, seguridad, y usabilidad. La fiabilidad es también un criterio que puede ser definido en términos de confiabilidad (ISO9126).

El cuerpo de conocimiento para sistemas debe ser sumamente confiable (" alta confianza " " o alta integridad en sistemas"). La terminología para los sistemas tradicionales mecánicos y eléctricos que no incluyen software ha sido importada para tratar amenazas o peligros, riesgos, integridad del sistema, y conceptos relacionados, y puede ser encontrada en las referencias citadas para esta sección.

3.1.3. Niveles de integridad del software

El nivel de integridad se determina en base a las consecuencias posibles del fracaso del software y a la probabilidad de fracaso. Para el software en el que la seguridad o la fiabilidad son importantes, técnicas como el análisis de riesgo para la seguridad o el análisis de amenazas para la fiabilidad pueden ser usadas para desarrollar una actividad de planificación que se identificaría en donde se encuentren conflictos potenciales. Históricos de fracaso de software similar también puede ayudar en la identificación de qué técnicas serán las más útiles en el descubrimiento de defectos y evaluación de calidad. Se proponen niveles de integridad (por ejemplo, la gradación de integridad) en (IEEE1012-98).

3.2. Caracterización de defectos

[Fri95; Hor03; Lew92; Rub94; Wak99; Wal89]

Los procesos SQM encuentran defectos. Caracterizar estos defectos conduce a un entendimiento del producto, facilita correcciones al proceso o al producto, e informa al gestor del proyecto o al cliente del estado del proceso o el producto. Existes muchas taxonomías defectuosas, y, mientras se ha intentado obtener un consenso en una taxonomía de defectos o fallos, la literatura indica que hay bastantes en uso [Bei90, Chi96, Gra92], IEEE1044-93). La caracterización del Defecto (la anomalía) también es usada en auditorias y revisiones, el responsable de revisión a menudo presenta una lista de anomalías proporcionadas por miembros de equipo para su consideración en una reunión de revisión.

Nuevos métodos de diseño y lenguajes se desarrollan, junto con avances en todas las tecnologías de software, las nuevas clases de defectos aparecen, y requieren mucho esfuerzo para interpretar clases previamente definidas. Rastreando defectos, el ingeniero de software está interesado tanto en el número como en el tipo de defectos. La Información sola, sin ninguna clasificación, no es realmente de ninguna utilidad en la identificación de las causas subyacentes de los defectos, pues los tipos específicos de problemas tienen que ser agrupados juntos para determinar como se gestionan. El asunto es establecer una taxonomía de defectos que sea

significativa para la organización y los ingenieros de software.

SQM descubre la información en todas las etapas de desarrollo de software y mantenimiento. Típicamente donde se usa la palabra "defecto" se refiere "a un fallo" tal y como se detalla más abajo.

Sin embargo, diferentes culturas y normas pueden usar significados algo diferentes para estos términos, lo cual ha llevado a tentativas para definirlos. Definiciones parciales tomadas del estándar (IEEE610.12-90) son:

- Error: “Una diferencia entre un resultado calculado y un resultado concreto”
- Faultt: “Un paso, proceso, o definición de datos incorrecto en programa de computadora.”
- Failure: “El resultado [incorrecto] de un fault”
- Mistake: “Un error humano que produce un resultado incorrecto” fallo falla

Los fracasos encontrados en pruebas como consecuencia de fallos de software están incluidos como defectos en esta sección.

Los modelos de fiabilidad son construidos en base a los fallos recogidos durante pruebas de software o procedentes de software en servicio, y de esta manera pueden ser usados para predecir futuros fracasos y asistir en decisiones sobre cuando detener las pruebas. [Mus89]

Una probable acción resultante de las conclusiones SQM es eliminar los defectos del producto durante su inspección. Otras acciones permiten alcanzar el valor completo de las conclusiones SQM. Estas acciones incluyen el análisis y el resumen de las conclusiones, y técnicas de medida de utilización para mejorar el producto y el proceso así como para rastrear los defectos y su eliminación. La mejora de proceso principalmente es tratada en el KA Proceso de Ingeniería de Software, junto con el proceso SQM como una fuente de información.

Los datos sobre las insuficiencias y defectos encontrados durante la implementación de técnicas SQM pueden ser perdidos a no ser que sean registrados. Para algunas técnicas (por ejemplo, revisiones técnicas, auditorias, inspecciones), los registradores están presentes para poner por escrito tal información, incluyendo cuestiones y decisiones. Cuando se utilizan herramientas automatizadas, las salidas de la herramienta pueden proporcionar la información sobre defectos. Los datos sobre defectos pueden ser recogidos y registrados sobre un formulario SCR (software change request) y posteriormente puede ser trasladado a algún tipo de base de datos, a mano o automáticamente, desde una herramienta de análisis. Proporcionan informes sobre defectos a la dirección de la organización.

3.3. Técnicas de Gestión de Calidad del Software

[Bas94; Bei90; Con86; Chi96; Fen97; Fri95; Lev95; Mus89; Pen93; Sch99; Wak99; Wei93; Zel98]

Las técnicas SQM pueden categorizarse de diferentes formas: estáticas,
Personal-intensivas, analíticas, dinámicas.

3.3.1. Técnicas Estáticas

Las técnicas estáticas implican el examen de la documentación del proyecto y el software, y otra información sobre los productos de software, sin ejecutarlos. Estas técnicas pueden incluir actividades intensivas de personal (como se define en 3.3.2) o actividades analíticas (como se define en 3.3.3) conducidas por individuos, con o sin la ayuda de herramientas automatizadas.

3.3.2. Técnicas intensivas de personal.

La puesta en marcha de técnicas intensivas de personal, incluyendo revisiones y auditorias, puede variar de una reunión formal a una reunión informal o una situación de comprobación de escritorio, pero (por lo general, al menos) dos o más personas están implicadas. Puede resultar necesaria una preparación anticipada. Tanto los recursos como los items bajo examen pueden incluir listas de comprobación y son resultado de técnicas analíticas y pruebas. Estas actividades son tratadas en (IEEE1028-97) sobre revisiones y auditorias. [Fre98, Hor03] [y Jon96, Rak97]

3.3.3. Técnicas Analíticas

Un ingeniero de software generalmente aplica técnicas analíticas. A veces varios ingenieros de software usan la misma técnica, pero cada uno lo aplica a partes diferentes del producto. Algunas técnicas son llevadas a cabo por herramientas; las otras son manuales. Algunas pueden encontrar defectos directamente, pero generalmente son usadas para apoyar otras técnicas. Algunas técnicas también incluyen varias evaluaciones como la parte de análisis de calidad total. Ejemplos de tales técnicas incluyen el análisis de complejidad, controlan el análisis de flujo, y el análisis algorítmico.

Cada tipo de análisis tiene un objetivo específico, y no se aplican todos ellos a cada proyecto. Un ejemplo de una técnica de apoyo es el análisis de complejidad, que es útil para determinar si realmente el diseño o la implementación resultan demasiado complejos para desarrollar correctamente, realizar las pruebas, o el mantenimiento. Los resultados de un análisis de complejidad también pueden ser usados en test de desarrollo. Las técnicas para encontrar defectos como el análisis de flujo de control, también pueden utilizarse para apoyar otra actividad.

Para software con muchos algoritmos, el análisis algorítmico es importante, especialmente cuando un algoritmo incorrecto podría causar un resultado catastrófico. Hay demasiadas técnicas analíticas para listarlas todas aquí. La lista y referencias proporcionadas pueden ofrecer ideas en la selección de una técnica, así como sugerencias para posteriores lecturas.

Otros tipos, más formales, de técnicas analíticas se conocen como métodos formales. Son usados para verificar requerimiento de software y diseños. Las pruebas de corrección corresponden a las partes críticas de software. Han sido usadas, sobre todo, en la verificación de las partes cruciales de sistemas críticos, tales como la seguridad y exigencias de fiabilidad. (Nas97)

3.3.4. Técnicas Dinámicas

Las diferentes clases de técnicas dinámicas son ejecutadas durante todo el desarrollo y el mantenimiento de software.

Generalmente, son técnicas de testeo, pero técnicas tales como la simulación, la comprobación de modelo, y la ejecución simbólica pueden ser consideradas dinámicas. La lectura de código es considerada una técnica estática, pero ingenieros de software experimentados puede ejecutar el código tal y como lo leen. En este sentido, la lectura de código también puede considerarse una técnica dinámica. Esta discrepancia en la categoría indica que personal con papeles diferentes en la organización puede considerar y aplicar estas técnicas de manera diferente.

Algunas pruebas, así mismo, pueden ejecutarse en el proceso de desarrollo, el proceso SQA, o el proceso de V&V, de nuevo dependen de la organización de proyecto. Debido a que la planificación SQM incluye testeo, esta sección incluye algunos comentarios sobre pruebas.
El KA Pruebas de Software proporciona el tratamiento y referencias técnicas a la teoría, técnicas para pruebas, y automatización.

3.3.5. Pruebas

Los procesos de aseguramiento descritos en SQA y V&V examinan todos los output de la especificación de requerimientos del software con el objeto de asegurar su trazabilidad, consistencia, completitud, corrección y ejecución. Esta comprobación también incluye los output de los procesos de desarrollo y mantenimiento, recopilando, analizando y midiendo los resultados. SQA asegura la planificación, desarrollo e implementación de determinados tipos de pruebas, y V&V desarrolla planes de prueba, estrategias, casos y procedimientos.

Las pruebas son tratadas detalladamente en el KA. testeo de Software. Dos tipos de pruebas pueden incluirse en el ámbito de SQA y V*V, por razón de su responsabilidad por la calidad de los materiales usados en el proyecto:

- Evaluación y prueba de herramientas que serán usadas en el proyecto (IEEE1462-98).
- Prueba de Conformidad (o revisión de prueba de conformidad) de componentes y productos COTS que se usaran en el producto; allí ahora existe un estándar para paquetes de software (IEEE1465-98).

En ocasiones una organización independiente V&V puede ser requerida para monitorear el proceso de pruebas y a veces para atestiguar la ejecución actual con el objeto de asegurar que este es llevado a cabo conforme a los procedimientos especificados. También se puede apelar a V&V para evaluar las pruebas en sí mismas: adecuación a los proyectos y procedimientos, y suficiencia y exactitud de resultados

Otro tipo de pruebas que puede incluirse en el ámbito de la organización V&V es el de pruebas de terceros.

Este tercero no es el desarrollador, tampoco esta relacionado en modo alguno con el desarrollo del producto. En cambio, el tercero es un “Facility” independiente, por lo general acreditado por alguna autoridad. Su objetivo es el de probar un producto respecto de su conformidad con un conjunto específico de exigencias.

3.4. Medición de calidad del software

Los modelos de calidad del software a menudo incluyen métricas para determinar el grado de calidad de cada característica alcanzada por el producto.

Si estos modelos se seleccionan correctamente, las métricas pueden apoyar la calidad del software de muchas maneras (entre otros aspectos de los procesos de ciclo de vida de software).

Pueden ayudar en procesos de toma de decisiones de gestión. Pueden encontrar áreas problemáticas y cuellos de botella en procesos de software y pueden ayudar a los ingenieros de software a evaluar la calidad de su trabajo respecto de objetivos SQA y respecto de procesos a largo plazo de mejora de calidad.

Con el incremento de la complejidad del software las cuestiones sobre calidad van más allá de si el software es capaz de alcanzar objetivos de calidad mensurables. Existen algunas áreas en las que las métricas soportan SQM directamente. Esto incluye asistencias en la decisión de cuando detener las pruebas. Para ello los modelos de fiabilidad y

pruebas, resultan útiles usando datos de fallos y fracasos.

El coste de los procesos SQM es una cuestión que casi siempre se suscita cuando se adopta la decisión de cómo debería organizarse un proyecto. A menudo, los modelos genéricos de coste usados, están basados en cuando se encuentra y cuanto esfuerzo se precisa para fijar el defecto en relación con el hallazgo del defecto temprano en el proceso de desarrollo. Los datos de proyecto pueden ofrecer una mejor idea del coste. El tratamiento de este tema puede ser encontrada en [Rak97: pp. 39-50]. La información relacionada puede ser encontrada en los KAs el Proceso de Ingeniería de Software y Gestión en Ingeniería del Software.

Finalmente los informes SQM en sí mismos proveen información valiosa no sólo sobre estos procesos, sino también sobre como pueden perfeccionarse todos los procesos de ciclo de vida. El tratamiento de estos asuntos puede encontrarse en [McC04] (y IEEE1012-98).

Mientras las métricas para características de calidad y rasgos de producto pueden resultar útiles en sí mismas (por ejemplo, el número de requerimientos defectuosos o la proporción de requerimientos defectuosos), pueden aplicarse técnicas matemáticas y gráficas para ayudar en la interpretación de esa métricas. Ello se encuentra en las categorías siguientes y son tratados en [Fen97, Jon96, Kan02, Lyu96, Mus99].

- Basadas estadísticamente basado (por ejemplo, Pareto analysis, run charts, scatter plots, normal distribution).
- Pruebas Estadísticas (por ejemplo, binomial test, chisquared test).
- El análisis de Tendencia.
- Predicción (por ejemplo, modelos de fiabilidad).

Las técnicas basadas en estadísticas y las pruebas a menudo proporcionan una imagne de las áreas más problemáticas del producto software examinado. Los cuadros y gráficos resultantes resultan ayudas de visualización que los gestores con poder de decisión pueden utilizar para concentrar recursos donde resulten más necesarios. Los resultados del análisis de tendencia pueden indicar que una programación no ha sido respetada, tales como las pruebas, o que ciertas clases de fallos se intensificarán a no ser que se adopte alguna acción correctiva en el desarrollo. Las técnicas de predicción asisten en la planificación del periodo de prueba y en la predicción del fracaso. Más información sobre medición en general pude encontrarse en los Kas Proceso de Ingeniería de Software y Gestión en Ingeniería del Software. Información más específica sobre medición en pruebas se presenta en el Ka Testeo de Software.

Las referencias [Fen97, Jon96, Kan02, Pfl01] proporcionan un tratamiento del análisis del defecto, consistente en medir la aparición del defecto y posteriormente aplicar métodos estadísticos para entender los tipos de defectos que aparecen más frecuentemente, es decir, determinar su densidad. También ayudan a comprender las tendencias en cómo están trabajando las técnicas de detección, y cómo están progresando los procesos de desarrollo y mantenimiento.
La medición de la cobertura de la prueba ayuda a estimar cuanto esfuerzo en términos de prueba queda por hacer, y predecir los posibles defectos restantes. De estos métodos de medición, pueden desarrollarse los perfiles del defecto para un dominio específico de la aplicación.

Así, para el siguiente sistema de software dentro de esa organización, los perfiles podrán utilizarse para

guiar los procesos SQM, es decir, para focalizar esfuerzo allí donde sea más probable que puedan surgir problemas. Semejantemente, los puntos de referencia, o los defectos típicos de ese dominio, pueden servir como ayuda en la determinación de cuando estará el producto listo para su entrega.

El estudio de como usar datos de SQM para mejorar procesos de desarrollo y mantenimiento puede encontrarse en los Kas Gestión en Ingeniería del Software y Procesos en Ingeniería del Software.

MATRIZ DE TEMAS VS REFERENCIAS

	[Boe78]	[Dac01]	[Hou99]	[IEEE99]	ISO9001-00]	[ISO900 03-04]	[Jon96]	[Kia95]	[Lap91]	[Lew92]	[Llo03]	[McC77]	[Mus99]	[NIST03]	[Pfl01]	[Pre04]	[Rak97]	[Sei02]	[Wal96]	[Wei93]	[Wei96]
1. Fundamentos de Calidad de Software																					
1.1. Ingeniería del Software Cultura y Ética.				*																	*
1.2. Valor y coste de la calidad.	*		*				*							*	*	*				*	*
1.3 .Modelos y Características de Calidad.	*	*			*	*		*	*	*	*	*	*	*		*	*	*	*		
1.4. Mejora de Calidad.														*		*					*

	[Ack02]	[Ebe94]	[Fre98]	[Gil93]	[Gra92]	[Hor03]	[Lew92]	[Pfl01]	[Pre04]	[Rad02]	[Rak97]	[Sch99]	[Som05]	[Voa99]	[Wal89]	[Wal96]
2. Procesos de Gestión de Calidad del Software																
2.1. Aseguramiento de la Calidad del Software	*	*	*		*	*		*	*		*	*	*	*	*	*
2.2. Verificaciones y Validación			*			*		*	*				*		*	*
2.3. Revisiones y Auditorias	*		*	*		*	*	*	*	*	*		*	*	*	*

	[Bas84]	[Bei90]	[Con86]	[Chi96]	[Fen97]	[Fre98]	[Fri95]	[Gra92]	[Hor03]	[Jon96]	[Kan02]	[Lev95]	[Lew92]	[Lyu96]	[McC04]	[Mus89]	[Mus99]	[Pen93]	[Pfl01]	[Rak97]	[Rub94]	[Sch99]	[Wak99]	[Wal89]	[Wal96]	[Wei93]	[Zel98]
3. Consideraciones Prácticas																											
3.1. Requerimientos de calidad del software									*				*							*		*		*	*		
3.2. Caracterización de defectos		*		*			*	*	*				*			*					*		*	*			
3.3. Técnicas de Gestión de Calidad del Software	*	*	*	*	*	*	*		*	*		*				*		*		*		*	*			*	*
3.4. Medición de calidad SW					*			*		*	*			*	*		*		*	*							

REFERENCIAS RECOMENDADAS PARA CALIDAD SW

[Ack02] F.A. Ackerman, "Software Inspections and the Cost Effective Production of Reliable Software," *Software Engineering, Volume 2: The Supporting Processes*, Richard H. Thayer and Mark Christensen, eds., Wiley-IEEE Computer Society Press, 2002.

[Bas84] V.R. Basili and D.M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, vol. SE-10, iss. 6, November 1984, pp. 728-738.

[Bei90] B. Beizer, *Software Testing Techniques*, International Thomson Press, 1990. [Boe78] B.W. Boehm et al., "Characteristics of Software Quality," *TRW Series on Software Technologies*, vol. 1, 1978.

[Chi96] R. Chillarege, "Orthogonal Defect Classification," *Handbook of Software Reliability Engineering*, M. Lyu, ed., IEEE Computer Society Press, 1996.

[Con86] S.D. Conte, H.E. Dunsmore, and V.Y. Shen, *Software Engineering Metrics and Models*: The Benjamin Cummings Publishing Company, 1986.

[Dac01] G. Dache, "IT Companies will gain competitive advantage by integrating CMM with ISO9001," *Quality System Update*, vol. 11, iss. 11, November 2001.

[Ebe94] R.G. Ebenau and S. Strauss, *Software Inspection Process*, McGraw-Hill, 1994.

[Fen98] N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, second ed., International Thomson Computer Press, 1998.

[Fre98] D.P. Freedman and G.M. Weinberg, *Handbook of Walkthroughs, Inspections, and Technical Reviews*, Little, Brown and Company, 1998.

[Fri95] M.A. Friedman and J.M. Voas, *Software Assessment: Reliability, Safety Testability*, John Wiley & Sons, 1995.

[Gil93] T. Gilb and D. Graham, *Software Inspections*, Addison-Wesley, 1993.

[Gra92] R.B. Grady, *Practical Software Metrics for Project Management and Process Management*, Prentice Hall, 1992.

[Hor03] J. W. Horch, *Practical Guide to Software Quality Management*, Artech House Publishers, 2003.

[Hou99] D. Houston, "Software Quality Professional," *ASQC*, vol. 1, iss. 2, 1999.

[IEEE-CS-99] IEEE-CS-1999, "Software Engineering Code of Ethics and Professional Practice," IEEE-CS/ACM, 1999, available at http://www.computer.org/certification/ ethics.htm.

[ISO9001-00] ISO 9001:2000, *Quality Management Systems — Requirements*, ISO, 2000.

[ISO90003-04] ISO/IEC 90003:2004, *Software and Systems Engineering-Guidelines for the Application of ISO9001:2000 to Computer Software*, ISO and IEC, 2004.

[Jon96] C. Jones and J. Capers, *Applied Software Measurement: Assuring Productivity and Quality*, second ed., McGraw-Hill, 1996.

[Kan02] S.H. Kan, *Metrics and Models in Software Quality Engineering*, second ed., Addison-Wesley, 2002.

[Kia95] D. Kiang, "Harmonization of International Software Standards on Integrity and Dependability," *Proc. IEEE International Software Engineering Standards Symposium*, IEEE Computer Society Press, 1995.

[Lap91] J.C. Laprie, *Dependability: Basic Concepts and Terminology in English, French, German, Italian and Japanese, IFIP WG 10.4*, Springer-Verlag, 1991.

[Lev95] N.G. Leveson, *Safeware: System Safety and Computers*, Addison-Wesley, 1995.

[Lew92] R.O. Lewis, *Independent Verification and Validation: A Life Cycle Engineering Process for Quality Software*, John Wiley & Sons, 1992.

[Llo03] Lloyd's Register, "TickIT Guide," iss. 5, 2003, available at http://www.tickit.org.

[Lyu96] M.R. Lyu, *Handbook of Software Reliability Engineering*: McGraw-Hill/IEEE, 1996.

[Mcc77] J.A. McCall, "Factors in Software Quality — General Electric," n77C1502, June 1977.

[McC04] S. McConnell, *Code Complete: A Practical Handbook of Software Construction*, Microsoft Press, second ed., 2004.

[Mus89] J.D. Musa and A.F. Ackerman, "Quantifying Software Validation: When to Stop Testing?" *IEEE Software*, vol. 6, iss. 3, May 1989, pp. 19-27.

[Mus99] J. Musa, *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*: McGraw Hill, 1999.

[NIST03] National Institute of Standards and Technology, "Baldrige National Quality Program," available at http://www.quality.nist.gov.

[Pen93] W.W. Peng and D.R. Wallace, "Software Error Analysis," National Institute of Standards and Technology, Gaithersburg, NIST SP 500-209, December 1993, available at http://hissa.nist.gov/SWERROR/.

[Pfl01] S.L. Pfleeger, *Software Engineering: Theory and Practice*, second ed., Prentice Hall, 2001.

[Pre04] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, sixth ed., McGraw-Hill, 2004.

[Rad02] R. Radice, *High Quality Low Cost Software Inspections*, Paradoxicon, 2002, p. 479.

[Rak97] S.R. Rakitin, *Software Verification and Validation: A Practitioner's Guide*, Artech House, 1997.

[Rub94] J. Rubin, *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*, John Wiley & Sons, 1994.

[Sch99] G.C. Schulmeyer and J.I. McManus, *Handbook of Software Quality Assurance*, third ed., Prentice Hall, 1999. [SEI02] "Capability Maturity Model Integration for Software Engineering (CMMI)," CMU/SEI-2002-TR-028, ESC-TR-2002-028, Software Engineering Institute, Carnegie Mellon University, 2002.

[Som05] I. Sommerville, *Software Engineering*, seventh ed., Addison-Wesley, 2005.

[Voa99] J. Voas, "Certifying Software For High Assurance Environments," *IEEE Software*, vol. 16, iss. 4, July-August 1999, pp. 48-54.

[Wak99] S. Wakid, D.R. Kuhn, and D.R. Wallace, "Toward Credible IT Testing and Certification," *IEEE Software*, July/August 1999, pp. 39-47.

[Wal89] D.R. Wallace and R.U. Fujii, "Software Verification and Validation: An Overview," *IEEE Software*, vol. 6, iss. 3, May 1989, pp. 10-17.

[Wal96] D.R. Wallace, L. Ippolito, and B. Cuthill, "Reference Information for the Software Verification and Validation Process," NIST SP 500-234, NIST, April 1996, available at http://hissa.nist.gov/VV234/.

[Wei93] G.M. Weinberg, "Measuring Cost and Value," *Quality Software Management: First-Order Measurement*, vol. 2, chap. 8, Dorset House, 1993.

[Wie96] K. Wiegers, *Creating a Software Engineering Culture*, Dorset House, 1996.

[Zel98] M.V. Zelkowitz and D.R. Wallace, "Experimental Models for Validating Technology," *Computer*, vol. 31, iss. 5, 1998, pp. 23-31.

APÉNDICE A. LISTA DE LECTURAS COMPLEMENTARIAS

(Abr96) A. Abran and P.N. Robillard, "Function Points Analysis: An Empirical Study of Its Measurement Processes," presented at IEEE Transactions on Software Engineering, 1996. **//journal or conference?//**

(Art93) L.J. Arthur, *Improving Software Quality: An Insider's Guide to TQM*, John Wiley & Sons, 1993.

(Bev97) N. Bevan, "Quality and Usability: A New Framework," *Achieving Software Product Quality*, E. v. Veenendaal and J. McMullan, eds., Uitgeverij Tutein Nolthenius, 1997.

(Cha89) R.N. Charette, *Software Engineering Risk Analysis and Management*, McGraw-Hill, 1989.

(Cro79) P.B. Crosby, *Quality Is Free*, McGraw-Hill, 1979.

(Dem86) W.E. Deming, *Out of the Crisis*: MIT Press, 1986.

(Dod00) Department of Defense and US Army, "Practical Software and Systems Measurement: A Foundation for Objective Project Management, Version 4.0b," October 2000, available at http://www.psmsc.com.

(Hum89) W. Humphrey, "Managing the Software Process," Chap. 8, 10, 16, Addison-Wesley, 1989.

(Hya96) L.E. Hyatt and L. Rosenberg, "A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality," presented at 8th Annual Software Technology Conference, 1996.

(Inc94) D. Ince, *ISO 9001 and Software Quality Assurance*, McGraw-Hill, 1994.

(Jur89) J.M. Juran, *Juran on Leadership for Quality*, The Free Press, 1989.

(Kin92) M.R. Kindl, "Software Quality and Testing: What DoD Can Learn from Commercial Practices," U.S. Army Institute for Research in Management Information, Communications and Computer Sciences, Georgia Institute of Technology, August 1992.

(NAS97) NASA, "Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems, Volume II: A Practitioner's Companion," 1997, available at http://eis.jpl.nasa.gov/ quality/Formal_Methods/.

(Pal97) J.D. Palmer, "Traceability," *Software Engineering*, M. Dorfman and R. Thayer, eds., 1997, pp. 266-276. (Ros98) L. Rosenberg, "Applying and Interpreting Object- Oriented Metrics," presented at Software Technology Conference, 1998, available at http://satc.gsfc.nasa.gov/ support/index. html.

(Sur03) W. Suryn, A. Abran, and A. April, "ISO/IEC SQuaRE. The Second Generation of Standards for Software Product Quality," presented at IASTED 2003, 2003.

(Vin90) W.G. Vincenti, *What Engineers Know and How They Know It — Analytical Studies from Aeronautical History*, John Hopkins University Press, 1990.

APÉNDICE B. LISTA DE ESTANDARS

(FIPS140.1-94) FIPS 140-1, *Security Requirements for Cryptographic Modules*, 1994.

(IEC61508-98) IEC 61508, *Functional Safety — Safety-Related Systems Parts 1, 2, 3*, IEEE, 1998.

(IEEE610.12-90) IEEE Std 610.12-1990 (R2002), *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.

(IEEE730-02) IEEE Std 730-2002, *IEEE Standard for Software Quality Assurance Plans*, IEEE, 2002.

(IEEE982.1-88) IEEE Std 982.1-1988, *IEEE Standard Dictionary of Measures to Produce Reliable Software*, 1988.

(IEEE1008-87) IEEE Std 1008-1987 (R2003), *IEEE Standard for Software Unit Testing*, IEEE, 1987.

(IEEE1012-98) IEEE Std 1012-1998, *Software Verification and Validation*, IEEE, 1998.

(IEEE1028-97) IEEE Std 1028-1997 (R2002), *IEEE Standard for Software Reviews*, IEEE, 1997.

(IEEE1044-93) IEEE Std 1044-1993 (R2002), *IEEE Standard for the Classification of Software Anomalies*, IEEE, 1993.

(IEEE1059-93) IEEE Std 1059-1993, *IEEE Guide for Software Verification and Validation Plans*, IEEE, 1993.

(IEEE1061-98) IEEE Std 1061-1998, *IEEE Standard for a Software Quality Metrics Methodology*, IEEE, 1998.

(IEEE1228-94) IEEE Std 1228-1994, *Software Safety Plans*, IEEE, 1994.

(IEEE1462-98) IEEE Std 1462-1998//ISO/IEC14102, *Information Technology — Guideline for the Evaluation and Selection of CASE Tools*.