Calidad del software (II)

En la primera parte de este artículo, publicado en el número anterior de esta revista, se argumentaba la importancia que tiene el desarrollo del software en la actualidad. Por tanto, uno de los objetivos de muchas empresas tecnológicas es desarrollar un software de alta calidad.
En el artículo anterior, se presentaron los factores que afectan a la calidad del software y cuáles son las siete actividades principales para garantizarla.
Esta segunda parte analiza dos de estas actividades, como son el uso de métricas para medir esa calidad de una manera lo más objetiva posible y la utilización de estándares en el desarrollo del software. Además, se presentan también las restricciones que tienen algunas de las actividades analizadas. Finalmente, se presentan unas conclusiones generales sobre calidad y de uso real en las empresas de desarrollo software.

Yolanda González Arechavala
Licenciada en Informática por la Universidad del País Vasco. Es investigadora en Formación del Instituto de Investigación Tecnològica y profesora en el Departamento de Sistemas Informáticos, ambos de la ETS de Ingeniería de la UPCo. Este trabajo ha sido parcialmente financiado con una beca de la Asociación de Ingenieros del I.C.A.I.

Fernando de Cuadra García
Dr. Ingeniero Industrial del ICAI. Es Profesor Propio Agregado de la UPCo, Director de las Escuelas Técnicas del I.C.A.I. e Investigador del Instituto de Investigación.

Estándares y modelos de referencia

En primer lugar, es necesario destacar la diferencia que existe entre "proceso" y "producto" (Hatton, 1995). Un producto es un bien tangible que es el resultado de un proceso. Aunque el software tiene aspectos intangibles, un producto software es sin embargo un bien en sí mismo e incluye sus documentos asociados. La estandarización del proceso define la manera de desarrollar el producto software mientras que la estandarización del producto define las propiedades que debe satisfacer el producto software resultante.

Según el British Standards Institute se define en Pfleeger, Fenton, & Page, (1994) un estándar como:

"Una especificación técnica u otro documento disponible para el público, preparado con la cooperación y el consenso o la aprobación general de todos los interesados que pueden verse afectados por él, basado en resultados científicos, técnicos o experiencias consolidadas, y que buscan el beneficio de la comunidad".

El proyecto SMARTIE (Pfleeger et al., 1994) perseguía una validación objetiva de los distintos estándares para encontrar aquéllos cuyo uso reportara una mejora en el proceso de desarrollo y el producto software. Una de sus principales conclusiones fue que los estándares son enfoques cuya efectividad no ha sido ni rigurosa ni científicamente

demostrada. Además, no todos los estándares son aplicables en cualquier situación. Antes de decidirse a utilizar un estándar, es necesario plantearse si es el adecuado al problema que se tiene, así como estudiar los beneficios que va a reportar.

Existen diversos organismos que promueven estándares en el área de calidad. En general, los estándares proporcionan una serie de directrices que luego cada empresa, al crear su plan de calidad (y de seguridad si es necesario) debe particularizar de acuerdo a sus necesidades. En este apartado se recogen someramente los estándares más conocidos y utilizados actualmente.

Estandarización y medida del producto

Según Hatton, (1995) el aspecto más importante de la estandarización del producto software es que prácticamente no hay ningún estándar válido. Diversos estudios han destacado que el estado del arte en cuanto a métricas hoy en día no está lo suficientemente avanzado como para permitir una estandarización, aunque sí que existen algunos estándares sobre métricas como el ISO 9126 "Software product evaluation. Quality characteristics and guidelines" (ISO/IEC, 1991) que cubre aspectos sobre calidad del software y medida o el IEEE 1061 "Standard for a Software Quality Metrics Methodology" (IEEE, 1998) que presenta una metodología para obtener las métricas que puedan evaluar la calidad del producto software realizado. En el subapartado relacionado con las métricas se mostrará una visión general de las mismas.

Estandarición del proceso

Se trata de estandarizar el proceso de desarrollo del producto software. Hay estándares muy generales que luego se van particularizando en estándares para un campo de aplicación determinado, un área de influencia, etc. Además, aunque en este apartado únicamente se van a nombrar algunos estándares que cubren gran parte del proceso de desarrollo software, existen estándares para tareas más específicas, como la codificación, convenciones para poner nombres, gestión de la configuración, documentación, glosarios de términos, mantenimiento, etc. Un compendio hasta 1993 de todos ellos puede encontrarse en Thayer (1993).

ISO 9001 e ISO 9000-3

La ISO 9001 es la parte de la familia de sistemas de calidad ISO 9000-9004 que es relevante para los proyectos de ingeniería del software (junto con la ISO 9000-3 que muestra las líneas maestras de la aplicación de la ISO 9001 al proceso de desarrollo software). ISO 9001 es un estándar internacional de manera que cada país tiene su propio estándar equivalente que únicamente se diferencia en el lenguaje (en España, es el UNE 66900-4). Esencialmente, este estándar cubre dos aspectos:

- La definición y el soporte de los principios necesarios para un sistema de calidad que asegure que el producto cumple los requisitos durante cada etapa de su producción.
- La gestión apropiada de las responsabilidades y la autoridad para su uso efectivo.

Pero esta norma no define un sistema de calidad en particular sino que pretende que sea cada compañía la que defina su propio sistema de calidad tomando de la ISO 9001 las directrices para ello. Los puntos débiles de este estándar pueden ser:

- Algunas compañías la utilizan como algo muy puntual para conseguir sus propósitos y no como un sistema permanente de calidad. Este estándar debería ser un comienzo, no un fin.
- No es incremental: una compañía está certificada o no.
- Por las directrices que propone, es posible formalizar un proceso poco eficaz.
- Los aspectos del proceso de desarrollo software que cubre la ISO 9001 junto con la ISO 9000-3 son muy básicos.
- Está pensada para control de calidad de sistemas de fabricación tradicional y los sistemas software no se adaptan apenas a las características de éstos: un cliente establece un contrato con

un proveedor en el que se detallan las especificaciones detalladas del producto. Si el proveedor desarrolla y fabrica el producto de acuerdo a esas especificaciones, el cliente estará contento. En desarrollo de sistemas software, los sistemas son muy complejos, cambian sus requisitos, es necesario mantenerlos, verificarlos,…

Por tanto, parece ser que los resultados obtenidos con la ISO 9001 en el campo del desarrollo de procesos software muestran que no es la mejor opción (Alarcón Rodríguez, 1999), resultando más adecuados los modelos de referencia que se verán a continuación.

CMM "Capability Maturity Model for Software"

El SEI ("Software Engineering Institute") presentó en 1991 la primera versión de CMM. CMM es un modelo de referencia que proporciona a las organizaciones de desarrollo de software una guía de cómo controlar sus procesos de desarrollo y mantenimiento software y cómo obtener una cultura de ingeniería del software.

CMM proporciona 5 niveles de madurez de una organización en el desarrollo de procesos software con el fin de que:

- Cada organización se evalúe usando como modelo de referencia estos niveles.
- Una vez detectado el nivel en el que se está, se sepan los pasos a seguir para conseguir llegar al siguiente nivel. El modelo es incremental.

A medida que una organización vaya madurando, el proceso software estará más definido dentro de la organización y la implantación del mismo estará más arraigada. El paso de un nivel a otro suele llevar aproximadamente dos años.

Se definen los siguientes cinco niveles de madurez:

- Inicial: el proceso software se caracteriza porque se hace como se puede, y a veces resulta caótico. El éxito del proceso depende de la competencia de la persona individual, no de la organización.
- Reproducible: existen disciplinas básicas de gestión del proceso software. Los procesos se definen, se documentan, se realiza una planificación y ciertos controles software.
- Definido: tanto la gestión del proceso software como las actividades de ingeniería se documentan, se estandarizan y están integradas dentro de la organización.
- Gestionado: se realizan medidas detalladas del proceso software y se obtienen productos de calidad.
- Optimización: se utiliza una mejora continua del procesos usando realimentación del mismo y los nuevos avances y tecnología.

Comparando ISO 9001 y la CMM parece ser que, más o menos, la certificación ISO 9001 es equivalente a estar en los niveles 2 ó 3 de la CMM, pero con la ventaja de que CMM es un modelo incremental más a largo plazo y con mejores resultados.

SPICE 98 "Software Process Improvement and Capability dEtermination" oISO/IEC TR 15504-1998 "Information Technology-Software Process Assessment"

En 1993, el comité ISO/IEC puso en marcha el proyecto SPICE con los objetivos de:

- Ayudar al proyecto de estandarización, en su fase preparatoria, para desarrollar los borradores iniciales de trabajo.
- Realizar las pruebas de usuario para obtener datos de experiencias reales.
- Dar a conocer el estándar y fomentar su aceptación.

En la actualidad, SPICE 98 (ISO/IEC TR15504) se encuentra en fase de informe técnico y consta de 9 secciones. Es un estándar emergente en toda la comunidad internacional. Incorpora las "buenas prácticas" para un correcto desarrollo del software para conseguir una mejora de la calidad.

Como modelo de referencia presenta 29 actividades que intervienen en el desarrollo del software que se pueden agrupar en 5 categorías:

- **Relación cliente-proveedor:** Actividades que afectan directamente al cliente: envío del producto al cliente, soporte del producto, asegurar la operación correcta del producto y su uso. El cliente es quien solicita el desarrollo o modificación del software y el proveedor es quien lo construye.

- **Ingeniería del software:** Conjunto de actividades de especificación, implementación y mantenimiento del sistema y su documentación.

- **Gestión:** Actividades relacionadas con la gestión del proyecto, como coordinación y gestión de recursos.

- **Soporte:** Conjunto de actividades que posibilitan las consecución de los buenos resultados al resto de procesos, ayudan y dan soporte para construir software con eficacia y eficiencia.

- **Organizativos:** Actividades que establecen las metas del negocio, y proporcionan la infraestructura para conseguirlas.

SPICE ayuda y da soporte para que tanto en el desarrollo y evaluación como en la implantación de productos software se tengan en cuenta tanto el nivel de capacitación como la utilización de la evaluación para la mejora del proceso. Estos objetivos pueden verse en la figura 1, tomada de Molina (1999).

Otras normas también conocidas en el campo de la calidad del software son:

Def 00-55 "Requirements for Safety Related Software in Defense Equipment"

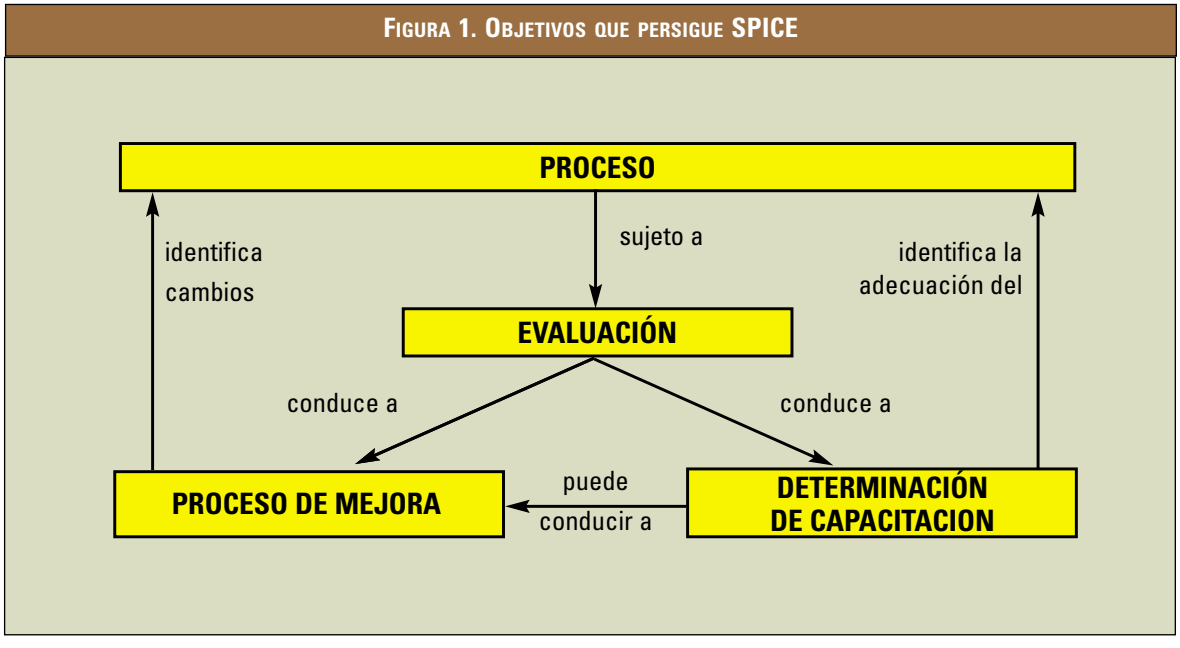
Según Ould, (1990) en esta norma se indican cuáles deben ser los requisitos de calidad para cada una de las fases del ciclo de vida del desarrollo software. Además, tiene indicaciones de seguridad, por lo que se puede decir que es una norma de seguridad, pero aplicable a la calidad.

IEEE/EIA 12207 "Software Lifecycle Processes" (IEEE/EIA, 1998)

Originalmente esta norma fue la ISO/IEC 12207, un estándar de alto nivel que sirvió como marco para que cada organización comprobara su propio estándar.

Posteriormente, el grupo grupo SESC de trabajo del IEEE ("IEEE Software Engineering Standards Committee") (Moore, 1999) realizó el esfuerzo de unificar e integrar su colección de estándares de ingeniería del software. Como resultado de sus estudios, se publicó en 1999 una edición con cuatro volúmenes de estándares de dicho co-

FIGURA 1. OBJETIVOS QUE PERSIGUE SPICE



mité, creándose el IEEE/EIA 12207 "Software Lifecycle Processes" como punto de entrada al resto de los estándares de la colección de estándares del IEEE.

En Ferguson & Sheard, (1998) se muestra una comparativa entre CMM y la IEEE/EIA 12207 de manera que se concluye que para cumplir este estándar es necesario estar casi al nivel 3 de CMM. En dicho artículo, aparece también un cuadro de las relaciones entre los distintos estándares y modelos de referencia actuales.

ESA PSS-05-0 es el estándar de la agencia espacial europea para ingeniería del software

Se presenta el ciclo de vida del software y tiene asociado un árbol de estándares de cada fase (Jones & Mazza, 1997). Se puede entender como una implantación práctica de los requisitos de la ISO 9001.

Es interesante también tener en cuenta que muchos de los estándares han sido realizados en EEUU. En Europa, se reconocen oficialmente tres organizaciones europeas de estándares y normas: CEN ("The European Committee for Standardization", www.cenorm.be), CENELEC ("The European Committee for Electrotechnical Standardization", www.cenelec.be) y la ETSE ("The European Telecommunications Standards Institute", www.etsi.org). Por tanto, hay que tener en cuenta los estándares que proponen estas organizaciones (en muchos casos, adaptaciones de los del IEEE) a la hora de seleccionar los estándares a utilizar para un proyecto.

Métricas

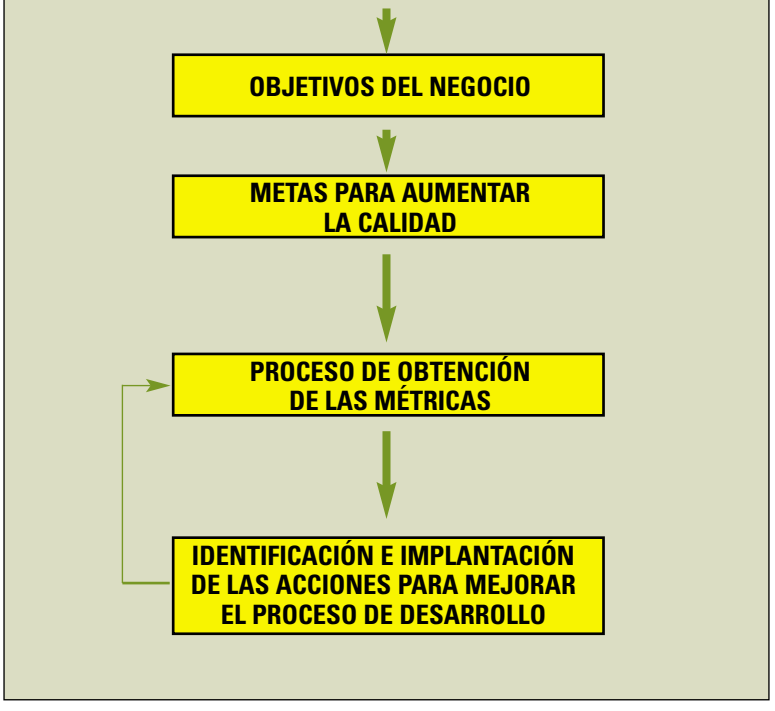
Las métricas de calidad aportan una indicación de cómo se ajusta el software a los requisitos implícitos y explícitos del cliente. A veces, cuando se intentan obtener medidas precisas de la calidad del software se acaba frustrado por la naturaleza subjetiva de esta actividad. Para resolver este problema, se buscan medidas cuantitativas de la calidad del software, para poder llevar a cabo un análisis objetivo. Pero no es posible medir la calidad del software de forma exacta, ya que cada medida es parcialmente imperfecta. Las medidas de calidad siempre son indirectas, ya que no se mide directamente la calidad sino algunas de sus manifestaciones. El factor que lo complica es la relación precisa entre la variable que es medida y la calidad del software.

En la figura 2, Möller (Möller & Paulish, 1993) muestra la importancia de las métricas para conseguir una mejora en la calidad del software.

El término "métrica" se usa frecuentemente para indicar un conjunto de mediciones tomadas en un proceso particular. En el estándar EEE, (1998) se define la métrica de calidad del software como:

"Una función cuyas entradas son datos del software y cuya

FIGURA 2. ENFOQUE DE LAS MÉTRICAS DEL SOFTWARE



salida es un único valor numérico que puede ser interpretado como el grado que posee el software de un atributo dado que afecta a la calidad."

Las métricas de ingeniería del software se usan para caracterizar:

- Productos de ingeniería del software; por ejemplo, diseños, código fuente, casos de prueba…
- Procesos de ingeniería del software; por ejemplo, actividades del análisis, diseño, codificación…
- Tareas realizadas por personas; por ejemplo, la eficiencia de una persona que realiza las pruebas, o la productividad de un diseñador individual.

Si se usan de manera adecuada, las métricas de ingeniería del software deben permitir:

- Definir cuantitativamente el grado de éxito y/o fracaso para un producto, un proceso o las tareas realizadas por una persona.
- Identificar y cuantificar las mejoras, su ausencia o la degradación del producto, del proceso o de las tareas realizadas por las personas.
- Hacer entendibles y útiles las decisiones técnicas y las de gestión.
- Identificar tendencias.
- Realizar estimaciones cuantificables y con sentido.

Los trabajos preliminares de la aplicación de métodos cuantitativos al desarrollo del software se establecieron en 1970 (Möller & Paulish, 1993). Existían cuatro tendencias tecnológicas principales que fueron las que evolucionaron en las métricas de hoy en día:

1. Medidas de la complejidad del código. A mediados de los 70 existía una significativa actividad investigadora para desarrollar medidas de la complejidad del código. Estas métricas eran fáciles de calcular a partir del código del producto. Algunos ejemplos de ellas son la medida de complejidad ciclomática de McCabe y la ciencia del software de Halstead.

2. Estimación de los costes del proyecto software. Estas técnicas fueron desarrolladas a mediados de los 70 para estimar el esfuerzo y la planificación necesarios para desarrollar un producto software, basado en la estimación del número de líneas de código necesarias para implementarlo u otros factores. Algunos ejemplos de estas técnicas incluyen el modelo SLIM de Larry Putnam y el modelo COCOMO de Barry Boehm.

3. Garantía de calidad del software. Las técnicas relacionadas con la garantía del software mejoraron significativamente a finales de los 70 y principios de los 80. Especialmente interesante para los métodos cuantitativos fue la acumulación de datos de fallos durante las distintas fases del ciclo de vida del software.

4. Proceso de desarrollo software. A medida que los proyectos de software se fueron haciendo más grandes y más complejos, se fue viendo la necesidad de controlar el proceso de desarrollo. Este proceso incluía la definición del ciclo de vida de desarrollo en varias fases secuenciales y ponía más énfasis en la gestión del proyecto software con mejores controles de los recursos.

En la actualidad existen multitud de métricas. Cada organización deberá decidir cuáles son sus metas y, en virtud de ellas, decidir qué métricas emplear.

En la tabla 3, presentada en Horch (1996), se enumeran algunas de las métricas más conocidas relacionadas con unas metas determinadas. Las siglas SQS de la tabla significan "Software Quality System", sistema de calidad del software. Las siglas STR[2] quieren decir "Software Trouble Report", informe de los problemas software. En este libro, se utilizan las métricas como una relación entre dos medidas.

[2] Un informe de problemas software (STR) está abierto si se ha detectado un problema, creando dicho informe, pero todavía no se ha resuelto. Estará cerrado si el problema ya ha sido abierto.

Tabla 3. Metas y Métricas	
Metas del SQS	**Métricas aplicables**
Mejora de la gestión de defectos	Cambios del coste de calidad / Planificación de la implantación del SQS Coste del software rechazado / Coste total del proyecto Coste de la corrección de defectos / Coste de la detección de defectos Densidad de defectos / Fase del ciclo de vida Defectos encontrados en las revisiones / Defectos encontrados en las pruebas Defectos detectados por el usuario / Defectos detectados por el desarrollador STR's cerrados / Total de STR's abiertos STR's que quedan abiertos / STR's cerrados STR's abiertos y cerrados / Periodo de tiempo Tiempo medio entre fallos Fiabilidad del producto software Llamadas a la ayuda / Producto software Requisitos cambiados / Requisitos totales
Mejora de los requisitos	Requisitos implementados / Requisitos totales Errores de los requisitos / Errores totales
Mejora en la detección de defectos	Pruebas realizadas con éxito / Total de pruebas planificadas Defectos encontrados en las revisiones / Defectos encontrados en las pruebas Densidad de defectos / Producto software Defectos detectados por el usuario / Defectos detectados por el desarrollador
Mejora de la productividad del desarrollador	Miles de líneas de código o puntos función / Mes de una persona Planificación o presupuesto real / Estimación Desembolso del presupuesto / Estado de la planificación Tiempo medio para reparar un defecto Defectos incorrectamente corregidos / Defectos totales Defectos del producto software / Complejidad del producto software
Mejora de las técnicas de estimación	Planificación o presupuesto real / Estimación Tiempo medio para reparar un defecto Desembolso del presupuesto / Estado de la planificación
Incremento del procesamiento del centro de datos	Correcciones realizadas de manera errónea / Correcciones totales Tiempo medio para reparar un defecto Defectos detectados por el usuario / Defectos detectados por el desarrollador

La referencia clásica en cuanto a métricas sigue siendo el libro de Fenton & Pfleeger, (1996), punto de partida de todos los estudios que se realizan hoy en día. Con el paradigma de orientación a objetos se ha comprobado que muchas de las métricas utilizadas anteriormente no son adecuadas, de manera que en la actualidad se están realizando diversas investigaciones en el campo de las métricas en la orientación a objetos (Genero, Piattini, & Calero, 2000; Meyer, 1998).

Aspectos discutibles de conceptos sobre calidad

Aunque las técnicas estudiadas en los apartados previos son de gran ayuda para conseguir un producto de calidad, hay que ser realistas y no confiar en ellas ciegamente. En este apartado, se van a mostrar ciertas matizaciones en este sentido.

Voas (Voas, 1999) presenta en su artículo ocho mitos que existen en cuanto a algunas técnicas que se utilizan para conseguir mejorar la calidad en el desarrollo del software. El autor reconoce que aunque estas ocho actividades utilizadas con moderación y en combinación pueden proporcionar un camino para producir software de calidad, insiste en que no hay que pensar en ellas como la panacea. Algunos de los puntos de vista mostrados en el artículo son discutibles, pero resulta interesante tenerlos en cuenta:

Mito 1. Madurez y mejora del proceso: medir la madurez del proceso en una organización es equivalente a medir la calidad del software de la organización. El hecho de que una organización de desarrollo software tenga una buena puntuación no significa que el software que produzca sea tan bueno como la propia organización (Hunter & Rae, 1993).

Mito 2. Métodos formales: los métodos formales son una técnica cada vez más utilizada, pero en algunos entornos se tratan como si fueran la respuesta a una mejora del proceso para resolver problemas de seguridad y fiabilidad. Los métodos formales son únicamente un proceso de desarrollo riguroso para demostrar formalmente que el software conserva algunas propiedades deseadas. De esta manera, se consigue eliminar ambigüedades, inconsistencias y comportamientos lógicamente incorrectos que pudieran existir en las definiciones del sistema. Pero son técnicas con ciertas limitaciones (Pfleeger & Hatton, 1997) ya que son complicados de implementar, caros y no son infalibles. De hecho suelen aplicarse únicamente a una pequeña parte del software.

Mito 3. Lenguajes y diseño orientado a objetos: el mito indica que cambiando el lenguaje o el paradigma de diseño, los problemas que no se han podido resolver usando los lenguajes o las estrategias de diseño existentes se podrán resolver. Lo cierto es que muchos de los problemas que existen no son relativos al lenguaje, por lo que nuevos lenguajes tampoco podrán resolverlos (Hatton, 1998).

El problema es que los lenguajes de programación cada vez son más complicados y, en muchas ocasiones, se utilizan sin tener un conocimiento profundo del funcionamiento de las nuevas características que contienen. Por ello, se podría argumentar que el uso de estos lenguajes dificulta más la realización de un software de calidad que con los lenguajes clásicos, de los que existe un mayor conocimiento. Además, la abstracción que se utiliza en los modernos paradigmas de diseño hace que las pruebas sean más difíciles de realizar (Binder, 1995).

Mito 4. Métricas y medidas: la información numérica acerca del proceso de desarrollo y del código generado da una idea de lo bueno o no que es un software.

Para un observador externo, un código es bueno si realiza la función deseada en la manera deseada. No es una medida de cómo el código está estructurado o qué tal se entiende el código. Las métricas estructurales no proporcionan una medida de la semántica, por lo que no pueden indicar si un código es bueno o no. Aunque sí es cierto que la métricas son medidas indirectas de propiedades no medibles (no se puede medir la facilidad de

mantenimiento de un código, pero lo que sí es cierto es que cuantas menos líneas de código tenga, más fácil será en principio). Para que las métricas fueran más útiles, deberían repercutir en el proceso de desarrollo de los posteriores códigos, con el fin de ir mejorando el proceso. Por tanto, se puede decir que las métricas sirven como guía, pero no se pueden tomar como una receta absoluta de cómo conseguir la calidad del software.

Mito 5. Estándares de software: existe la idea de que cuando se sigue un estándar, hay que "olvidar" el sentido común. Pero los estándares tienen algunos problemas (en ocasiones, para cuando se publican ya carecen de relevancia, algunos son más un impedimento para la competencia que un abogado de la calidad, tienen beneficios añadidos que no son cuantificables...) de manera que lo adecuado es, si se tiene opción, estudiar la situación específica del proyecto y optar por el estándar que más se adecue al problema, adaptando el estándar y no adoptándolo.

Mito 6. Pruebas: se ha demostrado que en la fase de prueba se detectan muchos errores, pero no se puede pensar que las personas que realizan las pruebas son capaces de detectar todos los problemas. Por tanto, no se puede dejar todo el peso de la calidad a esta fase.

Mito 7. Herramientas CASE: una de las funciones de estas herramientas es la realización de una especificación utilizando un lenguaje gráfico o esquemático que puede ayudar a crear un software de mayor calidad (mediante la generación automática de código, ya que el diseñador comete menos errores) pero siempre y cuando los gráficos sean correctos y se haya modelado adecuadamente la solución al problema. Por tanto, son útiles, pero no infalibles.

Mito 8. Gestión de calidad total: se tiene la idea de que cuando se piensa mucho en la calidad de un producto, esta calidad acaba penetrando en el producto. Y en procesos de fabricación es cierto, pero no en una actividad como el desarrollo de software que es un proceso creativo en cierta medida.

Por otro lado, como apunta Pressman (Pressman, 1995), aunque pocos profesionales pondrían en duda la necesidad de calidad del software, muchos no están dispuestos a establecer funciones de garantía de calidad de software formales. Las razones son:

(1) Los responsables del desarrollo se resisten a hacer frente a los costes extra inmediatos.

(2) Los profesionales creen que ya están haciendo todo lo que hay que hacer.

(3) Nadie sabe dónde situar esa función dentro de la organización.

(4) Todos quieren evitar el "papeleo" que la garantía de calidad del software tiende a introducir en el proceso de ingeniería del software.

La realidad mostrada tanto en Dutta et al. (1999) como en Holt (1997) apunta al hecho de que hoy en día todavía no se utilizan apenas las técnicas para garantizar la calidad del software, aunque la tendencia es creciente.

Dutta pone de relieve en los estudios realizados en 20 países europeos sobre 397 grupos de ingeniería del software que, de media, sólo un 51% de estos grupos adoptan buenas prácticas en el desarrollo del software. Por países, los que adoptan en mayor medida estas prácticas son Francia e Inglaterra y los que menos las adoptan son España, Bélgica y Suecia. Por sectores de negocio, donde se adoptan en mayor medida es en los sectores de aviación, espacio, telecomunicaciones y finanzas y, en menor medida, el sector de consultas e ingeniería mecánica.

Conclusiones

El software que se realiza en la actualidad es cada vez más complejo y una pieza clave en una gran cantidad de empresas. Cualquier tarea que consiga garantizar su éxito debe ser muy tenida en cuenta, como es el caso de las actividades que garantizan la calidad del software.

En este artículo se han presentado someramente ciertas actividades que ayudan a conseguir un software de mayor calidad. Desgraciadamente, los estudios realizados hoy en día en diversos entornos industriales en los que se realiza software, muestran que la obtención de un software de calidad está muy lejos de conseguirse, debido a que todavía no se han asumido como necesarias muchas de las técnicas para obtenerlo.

Bibliografía

Alarcón Rodríguez, M. I. (1999). Certificación: ¿Objetivo o consecuencia? Novática, 137(Enero-Febrero), 9-11.

Binder, R. V. (1995). Trends in Testing OO Software. IEEE Computer(October), 68-69.

Dutta, S.; Lee, M. & Van Wassenhove, L. (1999). Software Engineering in Europe: A study of best practices. IEEE Software, May/June, 82-90.

Fenton, N. E. & Pfleeger, S. L. (1996). Software Metrics: A Rigorous and Practic Approach. (Second ed.) London: International Thomson Computer Press.

Ferguson, J. & Sheard, S. (1998). Leveraging your CMM efforts for IEEE/EIA 12207. IEEE Software, 23-38.

Genero, M.; Piattini, M. & Calero, C. (2000). Métricas para diagramas de clases expresados en UML. Novática, Ene/Feb.

Hatton, L. (1995). Safer C. Developing Software for high-integrity and Safety-critical systems. (First ed.) London: McGrawHill International Series in Software Engineering.

Hatton, L. (1998). Does OO Sync with How We Think? IEEE Software, May/Jun, 46-54.

Holt, J. (1997). Current practice in software engineering: a survey. Computing & Control Engineering Journal(August), 167-172.

Horch, J. W. (1996). Practical Guide to Software Quality Management: Artech House Publishers.

Hunter, R. B.; & Rae, A. K. (1993). Software Quality Management. In I. C. S. P. Tutorial (Ed.), Software Engineering: A European Perspective (pp. 409-408): IEEE Press.

IEEE. (1998). IEEE Std. 1061-1998: Standard for a Software Quality Metrics Methodology : IEEE.

IEEE/EIA. (1998). ISO/IEC 12207, International Standard - Information Technology - Software Life Cycle Processes.

ISO/IEC. (1991). ISO/IEC 9126: Software enginnering - Product quality. .

Jones, M. & Mazza, C. (1997). 1977-1997: Twenty Years of Software Engineering Standarisation in ESA (ESA. Bulletin nº 90). Darmstadt, Germany: Ground System Engineering Department, European Space Operations (ESOC), ESA.

Meyer, B. (1998). The Role of Object-Oriented Metrics. IEEE Computer(November), 123-125.

Molina, A. (1999). La calidad integrada en la producción de servicios SPICE como modelo de referencia. Novática(Enero/Febrero), 34-36.

Möller, K. H. & Paulish, D. J. (1993). Software Metrics: IEEE Press y Champman & Hall.

Moore, J. W. (1999). An Integrated Collection of Software Engineering Standards. IEEE Software, November/December, 1999, 51-57.

Ould, M. A. (1990). Software Development under Def Stan00-55: a guide. Information and Software Technology, 32(3, April), 170-175.

Pfleeger, S. L. Fenton, N. E. & Page, S. (1994). Evaluating Software Engineering Standars. IEEE Computer, 27-9(September), 71-79.

Pfleeger, S. L. & Hatton, L. (1997). Investigating the influence of Formal Methods". Computer, 30(2, Feb.), 33-43.

Pressman, R. S. (1995). Ingeniería del Software: un enfoque práctico (Software Engineering. A Practitioner's Approach, Trans.). (Cuarta ed.) McGraw-Hill / Interamérica de España, S.A.

Thayer, R. H. (1993). European software engineering standards compendium. In I. C. S. P. Tutorial (Ed.), Software Engineering: A European Perspective (pp. 649-657): IEEE Press.

Voas, J. M. (1999). Software Quality's Eight Greatest Miths. IEEE Software, October, 118-120.