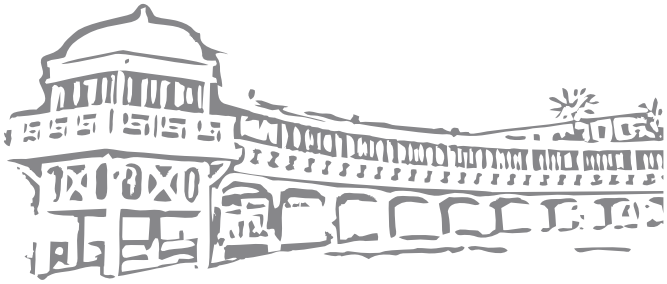

Jornadas Sistedes 2014

Cádiz, del 16 al 19 de septiembre

JISBD PROLE JCIS DC

XIX Jornadas de Ingeniería del Software y Bases de Datos

ACTAS

XIX Jornadas de Ingeniería del Software y Bases de Datos

Editores:

Javier Tuya
Mercedes Ruiz
Nuria Hurtado

Actas de las XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD)

Cádiz, 17 al 19 de septiembre 2014

Editores: Javier Tuya, Mercedes Ruiz y Nuria Hurtado

ISBN – 10: 84-697-1152-0
ISBN – 13: 978-84-697-1152-1

Prólogo

Las XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD), se celebran en Cádiz del 16 al 19 de septiembre de 2014. JISBD forma parte de las Jornadas SISTEDES, organizadas por la Sociedad de Ingeniería del Software y Tecnologías de Desarrollo de Software. Además de JISBD, las Jornadas SISTEDES agrupan a las Jornadas sobre Programación y Lenguajes (PROLE) y las Jornadas sobre Ciencia en Ingeniería de Servicios (JCIS), con las que la comunidad JISBD mantiene una estrecha relación.

JISBD se han consolidado, gracias al esfuerzo de nuestra comunidad, como un foro de intercambio y promoción de conocimiento en el área de Ingeniería del Software y Bases de Datos en el ámbito español, portugués y latinoamericano. En JISBD 2014, como en ocasiones anteriores, se han presentado cuatro tipos de contribuciones: artículos regulares, artículos de investigación emergente, artículos relevantes ya publicados y demostraciones de herramientas. Este año se cuenta con 50 contribuciones (14 regulares, 15 emergentes, 12 ya publicados y 9 demos), con un ratio de aceptación del 87% en regulares y 88% en emergentes.

Estas contribuciones se han repartido en las seis sesiones temáticas habituales de JISBD:

- Sesión 1: Bases de Datos, Almacenes de Datos, Minería de Datos, Recuperación de la Información.
- Sesión 2: Ingeniería Web, Interfaces de Usuario, Sistemas Colaborativos, Computación Ubicua.
- Sesión 3: Apoyo a la Decisión en Ingeniería del Software, Proceso Software y Metodologías.
- Sesión 4: Calidad, Pruebas y Requisitos.
- Sesión 5: Desarrollo de Software Dirigido por Modelos.
- Sesión 6: Líneas de Producto, Componentes y Arquitecturas Software.

Además, este año se han incorporado dos importantes novedades:

- Un taller de emprendimiento en las TIC que cuenta con la participación de varios casos de éxito de emprendedores así como la presentación de iniciativas de apoyo y financiación.

- Un Doctoral Consortium para que los doctorandos en el ámbito de SISTEDES presenten su proyecto de tesis doctoral y reciban realimentación por parte de expertos en un comité de evaluación

Se cuenta asimismo con dos conferencias invitadas: “Software Testing and/or Software Monitoring: Differences and Commonalities” ofrecida por Antonia Bertolino, Directora de Investigación del Italian National Research Council (CNR) en Pisa y “Automatic Identification of Service Candidates from Business Process Models” ofrecida por Jan Mendling, Director de Institute for Information Business en WU Vienna.

Me gustaría destacar y agradecer especialmente la importante contribución María José Escalona y Pepe Riquelme, responsables de las nuevas actividades que se han incorporado (el taller de emprendimiento y el Doctoral Consortium) que abordaron gustosamente esta nueva

tarea con gran determinación. También destacar el papel de los miembros del Comité de Programa, cuyas revisiones dejan entrever el importante esfuerzo que han realizado para enriquecer los trabajos y promover la discusión científica. Finalmente, quiero subrayar el papel de los Coordinadores de Sesiones, en los que he delegado parte del trabajo, para hacer de éstas un foro interactivo de intercambio y generación de conocimiento, así como la labor realizada por los Comentaristas de los artículos en las distintas sesiones.

Pero JISBD 2014 no habrían dado su fruto sin el trabajo realizado por todos los miembros del Comité Organizador de la Universidad de Cádiz, presidido por Mercedes Ruiz y el apoyo de los miembros del Comité Permanente, a los que tengo que agradecer que hayan depositado en mí su confianza permitiéndome actuar como Presidente del Comité de Programa de JISBD 2014. Adicionalmente, tengo que mencionar la estrecha colaboración mantenida con los Presidentes de los Comités de Programa de los dos años anteriores, Ana Moreno y Antonio Ruiz, cuyos inestimables consejos han sido vitales.

No puedo terminar sin manifestar mi gratitud a los autores de los distintos trabajos, así como a todos los asistentes a las Jornadas, por contribuir al éxito y a la consolidación de las mismas.

Cádiz, Septiembre de 2014

Javier Tuya

Presidente del Comité de Programa de JISBD 2014

Prólogo de la organización

La décima novena edición de las Jornadas de Ingeniería del Software y Bases de Datos (JISBD) se celebra en 2014 en la ciudad de Cádiz, en el ámbito de las Jornadas Sistedes 2014. Como organizadores, agradecemos al Comité Permanente la confianza depositada en nosotros y confiamos en que nuestro trabajo les permita a todos disfrutar de unas Jornadas enriquecedoras tanto desde el punto científico como personal.

Queremos agradecer a los miembros del comité de programa de JISBD su disponibilidad y trabajo efectuado durante las revisiones y la confección del programa científico del evento, así como a los coordinadores de las diferentes sesiones su atención a los detalles particulares de cada una de ellas. Es imprescindible manifestar la excelente labor del presidente del comité de programa, Javier Tuya, que, con su implicación en todos los detalles relacionados con las Jornadas, nos ha facilitado en gran medida nuestra tarea.

También es necesario agradecer a las entidades colaboradoras su contribución en la celebración del evento. En tiempos como los actuales resulta, si cabe, aún más importante este agradecimiento. Agradecemos a la Universidad de Cádiz, a la Escuela Superior de Ingeniería y al Departamento de Ingeniería Informática las facilidades que nos han dado para el uso de las instalaciones y recursos. Queremos reconocer la importante contribución de los grupos de investigación de Mejora del Proceso Software y Métodos Formales y UCASE de Ingeniería del Software, porque han contribuido con lo más preciado que tienen que son sus miembros. De igual manera, queremos agradecer al Excmo. Ayuntamiento de Cádiz y a su Delegación de Turismo su atenta colaboración y atención en la organización de algunas de las actividades sociales del programa de las Jornadas. De igual manera, agradecemos a las empresas Renfe e Iberia su colaboración a la hora de poner a disposición de los asistentes los descuentos para el desplazamiento a Cádiz.

Nuestra gratitud a Luis Iribarne y a Juan Manuel Vara por su disponibilidad, sus valiosos consejos y experiencia compartida sobre la organización de las dos ediciones anteriores.

Finalmente, nuestro agradecimiento más particular y especial debe ir dedicado a todos aquellos que han colaborado en los trabajos de organización de las Jornadas. Agradecemos profundamente el tiempo, las energías, la implicación, el compromiso y el trabajo de todos los que han participado en la preparación y celebración de las Jornadas.

Os deseamos a todos una feliz estancia en Cádiz y confiamos en que las Jornadas constituyan un excelente foro de intercambio de conocimientos, experiencias y reflexión.

Cádiz, septiembre de 2014

Mercedes Ruiz

Presidenta del Comité Organizador
de las Jornadas Sistedes 2014

Comité Ejecutivo

PRESIDENTE DEL COMITÉ DE PROGRAMA
Javier Tuya (Universidad de Oviedo)

COORDINADORES DE SESIONES TEMÁTICAS
Trabajos regulares, emergentes y ya publicados

Sesión 1. Bases de Datos, Almacenes de Datos, Minería de Datos, Recuperación de la Información
Sergio Ilarri (Universidad de Zaragoza)
José Francisco Aldana (Universidad de Málaga)

Sesión 2. Ingeniería Web, Interfaces de Usuario, Sistemas Colaborativos, Computación Ubicua
Elena Navarro (Universidad de Castilla-La Mancha)
Roberto Rodríguez Echeverría (Universidad de Extremadura)

Sesión 3. Apoyo a la Decisión en Ingeniería del Software, Proceso Software y Metodologías
Mercedes Ruiz (Universidad de Cádiz)
Agustín Yagüe (Univ. Politécnica de Madrid)

Sesión 4. Calidad, Pruebas y Requisitos
Carme Quer (Universidad Politécnica de Cataluña)
Claudio de la Riva (Universidad de Oviedo)

Sesión 5. Desarrollo de Software Dirigido por Modelos
Orlando Ávila (Open Canarias)
José Raúl Romero (Universidad de Córdoba)

Sesión 6. Líneas de Producto, Componentes y Arquitecturas Software
Juan Manuel Murillo Rodríguez (Universidad de Extremadura)
David Benavides (Universidad de Sevilla)

Demos
Macario Polo (Universidad de Castilla-La Mancha)
José Antonio Cruz Lemus (Universidad de Castilla-La Mancha)

Emprendimiento en las TIC
María José Escalona (Universidad de Sevilla)

Comité de Programa

Silvia Abrahão (Univ. Polit. Valencia)
Jesús Aguilar (Univ. Pablo Olavide)
Pedro Pablo Alarcón (Univ. Polit. Madrid)
Maria José Aramburu (Univ. Jaume I)
Joao Araujo (Univ. Nova Lisboa)
José Luis Arjona (Univ. Huelva)
David Benavides (Univ. Sevilla)
Rafael Berlanga (Univ. Jaume I)
Raquel Blanco (Univ. Oviedo)
Verónica Bollati (Univ. Rey Juan Carlos)
Artur Boronat (Univ. Leicester)
Nieves Brisaboa (Univ. A Coruña)
Jordi Cabot (École des Mines de Nantes)
Paloma Cáceres (Univ. Rey Juan Carlos)
Cristina Cachero (Univ. Alicante)
Javier Cámara (Carnegie Mellon Univ.)
Carlos Canal (Univ. Málaga)
José Hilario Canós (Univ. Polit. Valencia)
Rafael Capilla (Univ. Rey Juan Carlos)
Pepe Carsí (Univ. Polit. Valencia)
José María Cavero (Univ. Rey Juan Carlos)
Pedro J. Clemente (Univ. Extremadura)
Rafael Corchuelo (Univ. Sevilla)
Dolors Costal (Univ. Polit. Catalunya)
Yania Crespo (Univ. Valladolid)
Carlos Cuesta (Univ. Rey Juan Carlos)
Valeria de Castro (Univ. Rey Juan Carlos)
Pablo de la Fuente (Univ. Valladolid)
Óscar Díaz (Univ. País Vasco)
Óscar Dieste (Univ. Polit. Madrid)
Javier Dolado (Univ. País Vasco)
M. José Escalona (Univ. Sevilla)
Joao Falcao e Cunha (Univ. Porto)
Carles Farré (Univ. Polit. Catalunya)
Manuel Fernández Bertoa (Univ. Málaga)
José Luis Fernández-Aleman (Univ. Murcia)
Eduardo Fernández-Medina (Univ. Castilla-La Mancha)
Juan Garbajosa (Univ. Polit. Madrid)
Félix García (Univ. Castilla-La Mancha)
Jesús García Molina (Univ. Murcia)
Irene Garrigós (Univ. Alicante)
Marcela Genero (Univ. Castilla-La Mancha)
Pascual González (Univ. Castilla-La Mancha)
Alfredo Goñi (Univ. País Vasco)

Miguel Goulao (Univ. Nova de Lisboa)
Esther Guerra (Univ. Autónoma de Madrid)
Francisco Gutiérrez-Vela (Univ. Granada)
Juan Hernández (Univ. Extremadura)
Arantza Illarramendi (Univ. País Vasco)
Emilio Insfrán (Univ. Polit. Valencia)
Arantza Irastorza (Univ. País Vasco)
Luis Iribarne (Univ. Almeria)
Jon Iturrioz (Univ. País Vasco)
Javier Jaén (Univ. Polit. Valencia)
Natalia Juristo (Univ. Polit. Madrid)
Marcos López (Univ. Rey Juan Carlos)
Lidia López (Univ. Polit. Catalunya)
Miguel R. Luaces (Univ. A Coruña)
María Esperanza Manso (Univ. Valladolid)
José Norberto Mazón (Univ. Alicante)
Santiago Meliá (Univ. Alicante)
Ana Moreno (Univ. Polit. Madrid)
Juan José Moreno (Univ. Polit. Madrid)
Manuel Núñez (Univ. Complutense de Madrid)
Patricia Paderewski (Univ. Granada)
Ignacio Panach (Univ. Valencia)
Óscar Pastor (Univ. Polit. Valencia)
Óscar Pedreira (Univ. A Coruña)
Vicente Pelechano (Univ. Polit. Valencia)
Jenifer Pérez (Univ. Polit. Madrid)
Mario Piattini (Univ. Castilla-La Mancha)
Macario Polo (Univ. Castilla-La Mancha)
Antonio Polo (Univ. Extremadura)
Juan Carlos Preciado (Univ. Extremadura)
Isidro Ramos (Univ. Polit. Valencia)
José Riquelme (Univ. Sevilla)
Antonio Rito (Univ. Tec. Lisboa)
José Luis Roda (Univ. La Laguna)
Daniel Rodríguez (Univ. Alcalá)
José Raül Romero (Univ. Córdoba)
Francisco Ruiz (Univ. Castilla-La Mancha)
Antonio Ruiz (Univ. Sevilla)
Roberto Ruiz (Universidad Pablo Olavide)
Goiuria Sagardui (Univ. Mondragón)
José Samos (Univ. Granada)
Fernando Sánchez (Univ. Extremadura)
Pedro Sánchez (Univ. Polit. Cartagena)
Juan Sánchez (Univ. Polit. Valencia)
Maribel Sánchez-Segura (Univ. Carlos III)
Ismael Sanz (Univ. Jaume I)
Sergio Segura (Univ. Sevilla)

Almudena Sierra-Alonso (Univ. Rey Juan Carlos)
María José Suárez-Cabal (Univ. Oviedo)
Ernest Teniente (Univ. Polit. Catalunya)
Miguel Toro (Univ. Sevilla)
Ambrosio Toval (Univ. Murcia)
Salvador Trujillo (IKERLAN)
Juan Carlos Trujillo (Univ. Alicante)
Antonio Vallecillo (Univ. Málaga)
Juan Manuel Vara (Univ. Rey Juan Carlos)
Sira Vegas (Univ. Polit. Madrid)
Belén Vela (Univ. Rey Juan Carlos)
Cristina Vicente-Chicote (Univ. Extremadura)

Comité de Organización

PRESIDENTA DEL COMITÉ ORGANIZADOR

Mercedes Ruiz Carreira

MIEMBROS DEL COMITÉ ORGANIZADOR

Juan Boubeta Puig
Mª del Carmen de Castro Cabrera
Pedro Delgado Pérez
Juan Manuel Dodero Beardo
Antonio García Domínguez
Mª Teresa García Horcajadas
Lorena Gutiérrez Madroñal
Nuria Hurtado Rodríguez
José Luis Isla Montes
Inmaculada Medina Bulo
José Miguel Mota Macías
Manuel Palomo Duarte
Elena Orta Cuevas
Guadalupe Ortiz Bellot
Carlos Rioja del Río
Iván Ruiz Rube
Alberto Gabriel Salguero Hidalgo

Entidades Colaboradoras

Índice de contenidos

Keynotes

Sesión 1. Bases de Datos, Almacenes de Datos, Minería de Datos, Recuperación de la Información

Trabajos de investigación emergentes

Trabajos relevantes ya publicados

Sesión 2. Ingeniería Web, Interfaces de Usuario, Sistemas Colaborativos, Computación Ubicua

Trabajos regulares

Sesión 3. Apoyo a la Decisión en Ingeniería del Software, Proceso Software y Metodologías

Sesión 4. Calidad, Pruebas y Requisitos

Trabajos regulares

Trabajos de investigación emergentes

Trabajos relevantes ya publicados

Demostraciones de herramientas

Sesión 5. Desarrollo de Software Dirigido por Modelos

Trabajos regulares

Trabajos de investigación emergentes

Sesión 6. Líneas de Producto, Componentes y Arquitecturas software

Demostraciones de herramientas

Entorno para la Evaluación y Certificación de la Calidad del Producto Software

Moisés Rodríguez[1], Mario Piattini[2]

[1] Alarcos Quality Center, Universidad de Castilla-La Mancha, Ciudad Real, España.
moises.rodriguez@alarcosqualitycenter.com
[2] Instituto de Tecnologías y Sistemas de la Información, Universidad de Castilla-La Mancha, Ciudad Real, España.
mario.piattini@uclm.es

Resumen. La calidad del software está adquiriendo durante los últimos años una gran importancia, principalmente debido a que el software está presente en prácticamente todo lo que nos rodea. Para poder controlar dicha calidad se hace necesario llevar a cabo evaluaciones del software, que inicialmente se centraron en los procesos de desarrollo y que durante los últimos años se están enfocando más en la calidad del propio producto software. Para poder evaluar la calidad de un producto software se requiere de varios elementos entre los que se pueden destacar: un modelo de calidad, las métricas que se van a utilizar, el proceso de evaluación y las herramientas de soporte que automaticen todo lo posible el proceso. El presente artículo expone un entorno integrado que permite llevar a cabo la evaluación de la calidad del producto software, alineando todos los elementos anteriores a un estándar internacional como es la nueva familia de normas ISO/IEC 25000. Dicho entorno se ha validado además mediante su aplicación en un laboratorio de evaluación acreditado y la realización de varios proyectos piloto con una entidad de certificación, cuyo resultado ha sido la obtención de los primeros productos certificados por AENOR en mantenibilidad software a nivel internacional.

Palabras clave: modelo de calidad, métricas de calidad, ISO/IEC 25000, mantenibilidad, laboratorio de evaluación, certificación del producto software.

1 Introducción

Hoy en día el software está en prácticamente todo lo que nos rodea, no solo en los ordenadores, móviles o tablets, sino en los electrodomésticos, en los vehículos, en el sector de la medicina, en la banca y seguros, etc. Este aumento en la demanda de productos software ha dado lugar a un crecimiento de las empresas y departamentos encargados de su desarrollo, lo que se conocen como "factorías de software". Por otro lado, la falta de personal especializado para ciertas tareas del desarrollo software, así como la búsqueda de la reducción de costes han dado lugar a lo que se conoce como "outsorcing" del desarrollo software, de manera que las empresas externalizan todo o parte de las actividades de desarrollo software a otros departamentos o empresas. Sin

embargo, cuando se externalizan actividades de desarrollo software, también aumentan los riesgos y la falta de control sobre la calidad del software que la empresa contratada entrega, surgiendo la necesidad de evaluar y asegurar la calidad del software de dichas empresas desarrollan.

Aunque desde hace bastantes años la evaluación de la calidad del software es un campo de gran actividad tanto investigadora como en el sector industrial, la mayor parte del esfuerzo realizado se ha centrado en la calidad de los procesos, habiéndose desarrollado gran cantidad de modelos y estándares de referencia, evaluación y mejora de procesos software: ISO 90003, ISO 12207, ISO 15504, CMM, CMMI, IDEAL, SCAMPI, etc., en los que numerosas empresas de todo el mundo se han evaluado y/o certificado. Sin embargo, hay poca evidencia de que cumplir un modelo de procesos asegure la calidad del producto software desarrollado, y aunque la estandarización de los procesos garantiza la uniformidad en la salida de los mismos, podría llegar a darse el caso de que institucionalizara la creación de malos productos [1]. En este sentido, nosotros estamos de acuerdo con que las evaluaciones deberían basarse en evidencias directas del propio producto, y no solo en evidencias del proceso de desarrollo [2]. Por ello, es cada día mayor el número de organizaciones y empresas que se interesan, no solo por la calidad de los procesos que se siguen en el desarrollo de software, sino también por la calidad de los productos que desarrollan y/o adquieren, ya que una vez que el producto ha sido implantado en sus instalaciones se encuentran con graves problemas de calidad y complicaciones a la hora de corregirlo, adaptarlo o evolucionarlo.

En los últimos años se han elaborado también trabajos de investigación, normas y estándares, con el objetivo de crear modelos, procesos y herramientas de evaluación de la calidad del propio producto software, entre los que se pueden destacar los presentados en [3, 4, 5, 6]. Precisamente para dar respuesta a estas necesidades nace la nueva familia de normas ISO/IEC 25000 conocida como SQuaRE (*Software Product Quality Requirements and Evaluation*), que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto software, sustituyendo a las anteriores ISO/IEC 9126 e ISO/IEC 14598 y convirtiéndose así en el referente a seguir. La ISO/IEC 25000 se encuentra compuesta de varias partes o divisiones, entre las que podemos destacar la ISO/IEC 25040 [7] que define el proceso de evaluación de la calidad del producto software, y la ISO/IEC 25010 [8] que determina las características y subcaracterísticas de calidad que se pueden evaluar para un producto software (Fig. 1). Sin embargo, la división que define las métricas de calidad de la familia ISO/IEC 25000 todavía está pendiente de publicación, por lo que no existe un acuerdo respecto a los indicadores y umbrales que se deben considerar para poder determinar la calidad de un producto software de manera estandarizada.

Por otro lado, los modelos y normas anteriores relacionados con la evaluación del producto software, no tratan el proceso posterior de la certificación, que permita a las empresas superar una auditoría realizada por una entidad acreditada y obtener un certificado que refleje la calidad de su producto software. Por tanto la certificación de la calidad del producto software sigue siendo a día de hoy un área relativamente inmadura de la Ingeniería del Software, en la que todavía no existe un consenso definitivo.

Fig. 1. Modelo de calidad del producto software según la ISO/IEC 25010

Por este motivo y con el objetivo de conocer los trabajos existentes sobre certificación de la calidad del producto software, en 2012 se realizó una revisión sistemática [9] siguiendo la guía propuesta por Kitchenham en [10]. Como resultado se obtuvo un conjunto de 10 estudios primarios que cumplían con los requisitos de búsqueda: [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]. Entre las principales conclusiones obtenidas de esta revisión sistemática, se pueden destacar las siguientes:

- La mayoría de los estudios destacan la necesidad de que la certificación sea también extendida a las características del producto y no se centre sólo en el proceso.
- La mayoría de los estudios se basan en características de calidad extraídas de normas internacionales como la ISO/IEC 9126, utilizando métodos de evaluación también estándares como los de la ISO/IEC 14598. Sin embargo, ninguna de las propuestas ha adoptado el nuevo modelo y proceso propuestos por la familia ISO/IEC 25000.
- Aunque ya existen varias propuestas para certificar el producto software, la mayoría de ellas carece o tiene un número reducido de aplicaciones reales en la industria del software, lo que se considera vital a la hora de poder extender su aceptación a nivel internacional y poder contrastar los niveles obtenidos por diferentes productos software.
- La mayoría de los estudios utiliza indistintamente los conceptos de evaluación y certificación, lo que consideramos un error. Es necesario diferenciar entre el proceso de evaluación, realizado frente a un modelo de calidad, y el posterior proceso de certificación realizado por un organismo acreditado e independiente que asegure la validez del certificado emitido.
- Un denominador común de los estudios que presentan casos de aplicación práctica es la importancia que atribuyen a disponer de una herramienta que automatice las actividades. La mayoría de estos estudios carecen de dicha herramienta y se plantean como trabajo futuro abordar su construcción para que les permita medir, procesar los datos y analizar los resultados de una manera más rápida y sencilla.

Por todo lo anterior, durante los últimos años hemos trabajado en la construcción de un entorno para realizar evaluaciones de la calidad del producto, adoptando la familia de normas ISO/IEC 25000, que permita además a las empresas obtener una certificación del nivel de calidad alcanzado. Por ello, el objetivo del presente artículo es presentar las características principales de este entorno para la evaluación y certificación del producto software y demostrar su aplicación mediante un laboratorio acreditado, teniendo en cuenta que por cuestiones de espacio, hemos tenido que omitir ciertos detallalles del modelo, métricas y herramientas concretas utilizadas. Para ello, el resto del artículo se estructura de la siguiente manera: en el apartado 2 se presenta el modelo de calidad elaborado. En el apartado 3 se resumen las características del proceso de evaluación. Posteriormente en el apartado 4 se describe el entorno automatizado para la evaluación del producto. En el apartado 5 se resumen las principales características del laboratorio acreditado AQC Lab y su funcionamiento junto con AENOR para la evaluación y certificación del producto. Y finalmente, el apartado 6 presenta las conclusiones obtenidas con este trabajo y las líneas futuras de investigación.

2 Modelo de Calidad

El entorno de Evaluación descrito en el presente artículo se encuentra compuesto principalmente de tres elementos:

1. El **Modelo de Calidad**, que describe las características, subcaracterísticas, propiedades de calidad y métricas que se utilizan para determinar el valor de calidad de un producto software. En este elemento se centra el presente apartado.
2. El **Proceso de Evaluación**, que describe el conjunto de actividades y tareas que se realizan para llevar a cabo un proceso de evaluación del producto software.
3. El **Entorno Tecnológico**, formado por el conjunto de herramientas software que permiten automatizar la medición, aplicación de criterios de evaluación y visualización de los resultados.

De entre las características de calidad propuestas por la ISO/IEC 25010, inicialmente se decidió centrar el modelo de calidad en la característica de la mantenibilidad, entendida como el grado de efectividad y eficiencia con el que un producto puede ser modificado, debido principalmente a las siguientes razones:

- El mantenimiento supone una de las fases del ciclo de vida de desarrollo más costosa, sino la más, llegando a alcanzar el 60%.
- La mantenibilidad es una de las características más demandadas hoy en día por los clientes de software, que piden que el producto software que se les desarrolle pueda ser después mantenido por ellos mismos o incluso por un tercero.
- Las tareas de mantenimiento sobre productos con poca mantenibilidad tienen más probabilidad de introducir nuevos errores en el producto.
- Del estudio realizado en [9], se deduce que la mantenibilidad, junto con la funcionalidad, la usabilidad y la fiabilidad, es una de las características que más interesan desde el punto de vista de la certificación del producto.

El resto del presente artículo se centrará en esta característica, si bien el modelo de calidad ya se ha ampliado para la característica de funcionalidad y está en fase de completarse también para la usabilidad. Todo ello será publicado en la comunidad científica una vez se validen y se acrediten formalmente, igual que se ha hecho para la característica de la mantenibilidad.

2.1 Subcaracterísticas y propiedades de la mantenibilidad

El modelo de calidad definido para la mantenibilidad parte exactamente de las cinco subcaracterísticas de calidad definidas en la ISO/IEC 25010, que son:

- **Analizabilidad**. Se define como la facilidad para identificar las partes de un sistema que se deben modificar debido a deficiencias o fallos, o la capacidad de evaluar el impacto que puede provocar un cambio en el sistema.
- **Modularidad**. Se define como el grado, en el que un sistema se encuentra dividido en módulos de forma que el impacto que causa una modificación en un módulo sea mínimo para el resto.
- **Capacidad de ser Modificado**. Se define como el grado en el que se pueden realizar cambios en un producto software de forma efectiva y eficiente, sin introducir defectos ni degradar su rendimiento.
- **Capacidad de ser Reutilizado**. Se define como el grado en que un activo (módulo, paquete, clase, etc.) puede ser usado en más de un sistema o en la construcción de otros activos.
- **Capacidad de ser Probado**. Se define como la facilidad para establecer criterios de prueba para un sistema y realizar las pruebas que permitan comprobar que se cumplen esos criterios.

Sin embargo, la familia de normas ISO/IEC 25000 todavía no ha definido el conjunto de métricas e indicadores que afectan a cada una de estas subcaracterísticas, los umbrales para las mismas, ni las funciones de medición a aplicar para poder calcular el valor de calidad de cada una de ellas. Por ello, para completar este modelo de calidad y hacerlo operativo, se han identificado un conjunto de propiedades de calidad, que obtienen su valor a partir de métricas del código fuente, y se ha establecido la relación que existe con las subcaracterísticas anteriormente indicadas. El objetivo al identificar estas propiedades de calidad y métricas no ha sido que fuera el conjunto mayor posible, sino que fuera un grupo completo y sin lugar a controversias, basado en los estudios e investigaciones previas y aceptados por la comunidad científica. Estas propiedades de calidad son:

- **Incumplimiento de Reglas**. Esta propiedad se basa en el incumplimiento de reglas de programación y optimización de código establecidas por los estándares de programación de cada lenguaje y que se obtienen por varias herramientas de comprobación de reglas. Para cada subcaracterística de calidad se comprueba un conjunto de reglas relacionadas.
- **Documentación del Código**. La documentación de código hace referencia a los comentarios que existen en el código, los cuales se utilizan para explicar la funcio-

nalidad del código. Los comentarios influyen en la Analizabilidad, ya que los comentarios bien expresados ayudan a entender qué es lo que hace el código. Por esta misma razón, los comentarios también influyen en la capacidad para reutilizar el software: conocer qué es lo que hace un módulo ayuda a reutilizarlo en otros sistemas.

- **Complejidad**. La propiedad Complejidad se relaciona con la dificultad para implementar, probar, entender, modificar o mantener un programa. Como esta definición muestra, la Complejidad está relacionada con las subcaracterísticas: Analizabilidad, Capacidad de ser Modificado y Capacidad de ser Probado del modelo de calidad. En aplicaciones con complejidad elevada, las tareas de mantenimiento requieren más esfuerzo y, por tanto, son más costosas.
- **Estructuración**. La estructuración de un sistema es la división que se hace del mismo en componentes más pequeños. La calidad en el diseño de un sistema está estrechamente relacionada con esta propiedad: la estructuración correcta de un sistema en cualquiera de sus niveles (subsistemas, paquetes, clases, etc.) facilita su desarrollo y mantenimiento. Una correcta estructuración del sistema: facilita su comprensión y la identificación de elementos que se deben modificar (Analizabilidad), minimiza el impacto de los cambios que haya que realizar en el sistema (Modularidad), facilita la reutilización de únicamente aquellos elementos que sean necesarios, pudiendo utilizar toda la funcionalidad que se espera de ellos y no más o menos (Capacidad de ser Reutilizado) y facilita la elaboración de pruebas permitiendo que éstas se centren en comprobar el funcionamiento correcto de porciones concretas del sistema (Capacidad de ser Probado).
- **Tamaño de Métodos**. El tamaño de un sistema afecta directamente a la mantenibilidad del mismo, ya que, intuitivamente, un sistema más grande requiere más esfuerzo de mantenimiento. Se considera más adecuado evaluar el tamaño de sus elementos a un nivel más bajo, como por ejemplo a nivel de métodos, que evaluar directamente el tamaño global del sistema, ya que una modificación normalmente no afecta a todo el sistema, sino a un conjunto de sus elementos.
- **Código Duplicado**. Esta propiedad hace referencia a fragmentos de código que se encuentran repetidos en diferentes lugares del sistema. El código duplicado hace más difícil la realización de modificaciones en una aplicación software, ya que solucionar un defecto o introducir una mejora en dicho código requiere realizar las modificaciones en todas las partes del sistema en las que aparezca dicho código duplicado. El Código Duplicado afecta a las subcaracterísticas Analizabilidad y Capacidad de Ser Modificado, ya que el código duplicado dificulta la identificación del código que se debe modificar y dificulta la realización de modificaciones que sean efectivas. También afecta a la Subcaracterística Capacidad de Ser Probado, ya que se deben definir varios casos de prueba diferentes (aunque similares en forma) para comprobar el correcto funcionamiento del código duplicado en sus distintas instancias.
- **Acoplamiento**. El acoplamiento indica el grado de interdependencia entre las unidades de software (módulos, funciones, clases, bibliotecas, etc.). De forma general, cuanto más bajo sea el acoplamiento en una aplicación software, mejor se considera su diseño, ya que el bajo acoplamiento permite mejorar la mantenibilidad (los

cambios en una unidad no afectan al resto de unidades si no hay acoplamiento) y aumentar la reutilización de las unidades de software. El acoplamiento también dificulta la realización de pruebas, siendo más fácil probar elementos que dependen de pocos elementos externos [21].

- **Balance Inestabilidad - Abstracción**. La inestabilidad indica la relación entre el acoplamiento de salida y el acoplamiento total de un paquete y da una idea de la resistencia al cambio del mismo. La abstracción indica la relación entre clases abstractas (e interfaces) y el número de clases de un paquete. Debe existir un balance entre ambas, de forma que los paquetes deben ser abstractos y estables o concretos e inestables.
- **Ciclos**. La propiedad Ciclos hace referencia a la existencia de ciclos de dependencia entre los paquetes del sistema. El Principio de Dependencias Acíclicas [22] indica que no deben existir ciclos en la estructura de dependencias de un sistema. La existencia de Ciclos en un sistema afecta negativamente a su mantenibilidad dificultando la realización de modificaciones, ya que un cambio en un paquete repercute a todos los paquetes en el ciclo de dependencias [21]. Además, dificulta la reutilización del código y la realización de pruebas [22], ya que si se quiere reutilizar un paquete o realizar pruebas sobre alguna de sus clases es necesario tener presentes todos los paquetes involucrados en el ciclo.
- **Cohesión**. El concepto de cohesión [23] indica el grado de relación que existe entre los elementos internos de un módulo. Una clase tiene cohesión baja cuando realiza varias funciones no relacionadas. Así, las funcionalidades que proporciona mediante sus métodos tienen poco en común y para realizar sus actividades utiliza conjuntos de datos que no están relacionados. La Analizabilidad, la Modularidad y, en general, la Mantenibilidad en sistemas cuyas clases tienen una cohesión baja se ven perjudicadas, ya que estos sistemas son más difíciles de comprender, proporcionan funcionalidades que no son necesarias y su mala modularidad hace que los cambios en los requisitos afecten a varios módulos.

Tabla 1. Relación entre Propiedades y Subcaracterísticas de Mantenibilidad

Propiedades / Subcaracterísticas	Analizabilidad	Modularidad	Capacidad de ser Modificado	Capacidad de ser Reutilizado	Capacidad de ser Probado
Incumplimiento de Reglas	X	X	X	X	X
Documentación del Código	X			X	
Complejidad	X		X		X
Balance Inestabilidad-Abstracción / Acoplamiento		X	X	X	X
Ciclos		X	X	X	X
Cohesión	X	X			
Estructuración de Paquetes	X	X		X	X
Estructuración de Clases	X	X			X
Tamaño de Métodos	X	X			
Código Duplicado	X		X		X

En la Tabla 1 se resumen las relaciones existentes entre las subcaracterísticas y las propiedades de calidad. La relación entre una propiedad y una subcaracterística refleja que la primera influye en gran medida en el valor de calidad de la segunda. Que en dicha tabla no se indique relación entre una propiedad y una subcaracterística concretas no significa que la primera no influya en la segunda, sino que dicha influencia es menos relevante que la de otras propiedades.

2.2 Utilización del Modelo de Calidad

El valor de Mantenibilidad de un producto se obtiene a partir de los valores de calidad de sus subcaracterísticas. Para obtener la evaluación de las subcaracterísticas es necesario obtener el valor de las propiedades de calidad (vistas en el apartado anterior), que a su vez se evaluarán a través de los valores obtenidos para las métricas de las que dependen. Estas métricas son la parte base de la evaluación y su cálculo se realiza directamente a partir del código fuente del producto software que se está evaluando. En la Fig. 2 se puede observar la jerarquía de evaluación que se sigue en el modelo de calidad definido para determinar el valor de una característica de calidad.

La evaluación se realiza siguiendo un proceso *bottom-up*, de manera que primero se calculan las métricas de bajo nivel utilizando herramientas que procesan el código fuente del producto software bajo evaluación. Como vemos a la izquierda de la Fig. 2, cada una de estas métricas tiene un rango propio: número de clases, número de líneas de código duplicado, valor de la complejidad ciclomática de cada método, etc. A partir de los valores obtenidos en las métricas, se escala en la jerarquía al siguiente nivel para poder obtener un valor de calidad que asignarle a la propiedad que aplique. En este caso, el valor de las propiedades se ha normalizado entre 0 y 100 utilizando una función por perfiles (igual que se hará para las subcaracterísticas y la característica de calidad), de manera que todas las propiedades tengan la misma escala, siendo 0 el valor más bajo de calidad que puede tener una propiedad y 100 el valor más alto.

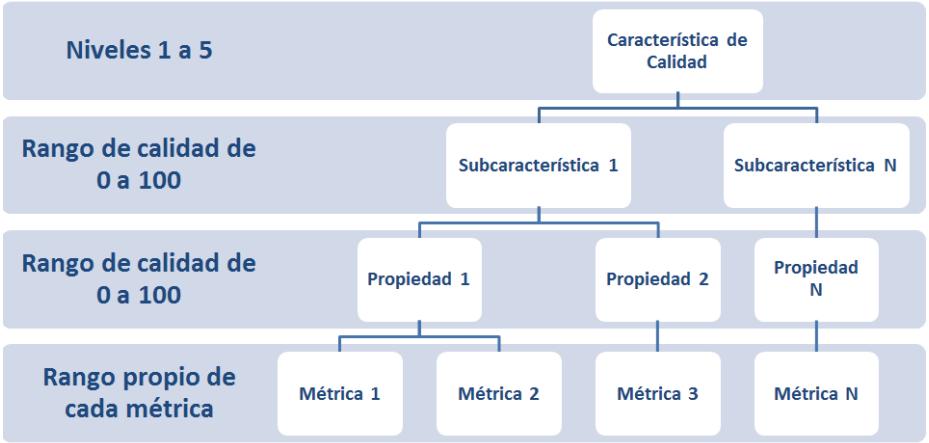

Fig. 2. Jerarquía y elementos que influyen en la evaluación del producto software

Una vez se dispone del valor de todas las propiedades, se pasa a calcular el valor de cada una de las subcaracterísticas de calidad. Cada una de estas subcaracterísticas estará influenciada por una o varias de las propiedades para las que previamente se ha calculado su valor. De nuevo, como se observa a la izquierda de la Fig. 2, el valor de las subcaracterísticas ha sido normalizado en un rango de 0 a 100, siendo igualmente el 0 el valor más bajo de calidad que puede obtener una subcaracterística y el 100 el valor más alto. Una vez se ha calculado el valor de todas la subcaracterísticas, estamos en disposición de calcular el nivel de calidad de la característica, en este caso mantenibilidad. Para ello, se aplica una función de evaluación sobre el conjunto de valores de 0 a 100 que presentan el grupo de subcaracterísticas y se obtiene un valor definitivo para la característica de calidad. En este caso, como se observa a la izquierda de la Fig. 2, el valor de calidad de las características se ha normalizado entre 1 y 5, en lugar de entre 0 y 100. La decisión de realizar esta clasificación por niveles se debe precisamente a cómo se encuentra hoy en día la cultura de la calidad del software, donde modelos anteriores para calidad de procesos ampliamente extendidos (CMMI, ISO/IEC 15504, etc.), siguen dicha clasificación por niveles de 1 a 5. La correspondencia de estos niveles de calidad con los valores normalizados de 0 a 100 siguen los rangos establecidos en la Tabla 2. Como se puede observar, un producto con una mantenibilidad en nivel 1 habrá obtenido un valor de calidad entre 0 y 25, por lo que se podrá considerar como deficiente. Una mantenibilidad en nivel 3, habrá obtenido un valor de calidad entre 50 y 75, por lo que se podrá considerar buena. Y una mantenibilidad en el nivel 5 habrá obtenido un valor de calidad entre 95 y 100, por lo que se podrá considerar como excelente.

Tabla 2. Niveles de Calidad para la Mantenibilidad del Producto Software

Nivel	Valor de Calidad	Descripción
1	0-25	Calidad Deficiente
2	25-50	Calidad Insuficiente
3	50-75	Calidad Buena
4	75-95	Calidad Muy Buena
5	95-100	Calidad Excelente

Aunque queda fuera del alcance del presente artículo, se debe indicar que la influencia que tienen las métricas y propiedades seleccionadas sobre la mantenibilidad ha sido validada. Como indica la norma ISO/IEC 25020 [24], uno de los métodos para demostrar la validez de una medición consiste en comprobar si existe correlación entre la característica de calidad y los valores medidos. Así, para realizar la validación se ha utilizado un método empírico basado en esta definición. En concreto, se ha realizado un caso de estudio en el que se han evaluado 30 proyectos reales y se ha comprobado que los valores de calidad proporcionados por el método están correlacionados con un indicador de esfuerzo de mantenimiento, como es el tiempo medio empleado en la resolución de defectos encontrados en fase de mantenimiento.

3 Proceso de Evaluación

Para la elaboración de este proceso de evaluación se tomaron como referencia las experiencias previas que se tenían en la construcción de metodologías de evaluación, como la expuesta en [25], pero principalmente la norma ISO/IEC 25040 [7], que describe el modelo de referencia para la evaluación del producto, formado por entradas, salidas, restricciones, recursos y actividades y sus tareas, estas dos últimas son:

1. Establecer los requisitos de la evaluación:
   (a) Establecer el propósito de la evaluación.
   (b) Obtener los requisitos de calidad del producto.
   (c) Identificar las partes del producto software que se van a evaluar.
   (d) Definir el rigor de la evaluación.
2. Especificar la evaluación:
   (a) Seleccionar las métricas de calidad (módulo de evaluación).
   (b) Definir los criterios de decisión para las métricas.
   (c) Establecer los criterios de decisión para la evaluación.
3. Diseñar la evaluación:
   (a) Planificar las actividades de la evaluación.
4. Ejecutar la evaluación:
   (a) Realizar las mediciones.
   (b) Aplicar los criterios de decisión a las medidas de calidad.
   (c) Aplicar los criterios de decisión a la evaluación.
5. Concluir la evaluación:
   (a) Revisar los resultados de la evaluación.
   (b) Crear el informe de evaluación.
   (c) Revisar la calidad de la evaluación.
   (d) Realizar la disposición o tratamiento de los datos

El proceso de evaluación construido adopta estas actividades y tareas, realizando una instanciación de las entradas, salidas, recursos y roles concretos que intervienen en nuestro proceso. En la Fig. 3 se expone un ejemplo para una de las actividades:

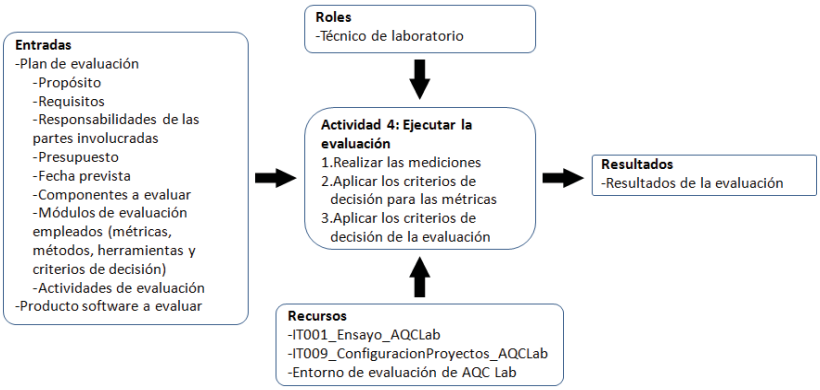

Fig. 3. Diagrama de la actividad "Ejecutar la Evaluación"

4 Entorno Tecnológico

El tercer elemento es el entorno tecnológico, para cuya construcción se partió de las experiencias previas que se tenían en la construcción de cuadros de mando relacionados con la calidad del producto [26]. El entorno construido permite automatizar hasta en un 90% las evaluaciones y está formado por tres niveles diferenciados:

1. **Herramientas de medición**. Constituyen el primer nivel y su misión es analizar el código fuente y generar archivos (normalmente XML) con los datos sobre métricas base y/o incumplimientos de reglas de programación para lenguajes como Java, .NET, PHP, C++, ObjetiveC, Android y Groovy. La ventaja de este nivel es que es fácilmente ampliable, añadiendo nuevas herramientas que permitan medir nuevos lenguajes de programación o calcular métricas para otras características de calidad.
2. **Sistema de evaluación**. Supone el nivel intermedio del entorno y su objetivo es analizar el conjunto de archivos generados por el nivel inferior y aplicar los criterios de evaluación del modelo de calidad, obteniendo como resultado los valores para las propiedades, subcaracterísticas y características de calidad.
3. **Entorno de visualización**. Representa el nivel superior del entorno y permite presentar de manera comprensible la información obtenida tras la evaluación del producto software. Además de mostrar los valores de calidad para propiedades, subcaracterísticas y características de calidad, este entorno permite la obtención de históricos para varias versiones de un producto, comparativas entre diferentes productos (Fig. 4) e incluso la generación de informes predefinidos.

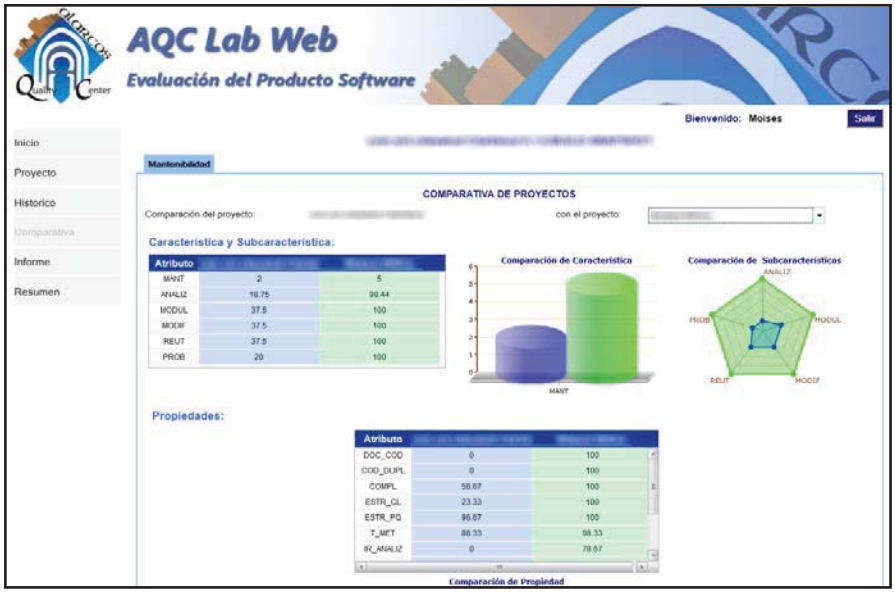

Fig. 4. Entorno de Visualización de los Resultados de Evaluación

5 Laboratorio de Evaluación

En paralelo a la última fase de construcción del entorno expuesto en los apartados anteriores y con el objetivo de obtener un reconocimiento a la validez de las evaluaciones realizadas mediante el mismo, se decidió elaborar toda la infraestructura de gestión necesaria para la creación de un laboratorio acreditado, siguiendo además para ello prácticas del desarrollo ágil como se expone en [27]. El resultado fue que en 2012 AQC Lab conseguía la acreditación de ENAC (Entidad Nacional de Acreditación) en la norma ISO/IEC 17025, como el primer laboratorio para la evaluación de la calidad de aplicaciones software bajo la familia de normas ISO/IEC 25000 (Fig. 5). La acreditación de acuerdo a la Norma ISO/IEC 17025 confirma la competencia técnica del laboratorio y garantiza la fiabilidad en los resultados de los ensayos realizados. Para ello, ENAC realiza una minuciosa evaluación de todos los factores que contribuyen a que los laboratorios obtengan resultados fiables:

- Son organizaciones que cuentan con personal cualificado y con experiencia.
- Disponen del equipamiento y de las infraestructuras necesarias.
- Aplican métodos y procedimientos de trabajo validados y apropiados.
- Emplean técnicas de evaluación de la calidad de los resultados.
- Emitiendo informes de evaluación claros y precisos.
- Cuentan con un sistema de aseguramiento de la calidad para gestionar su actividad.

Una vez alcanzada la acreditación del laboratorio, se estableció un proceso de trabajo con AENOR (Asociación Española de Normalización y Certificación) que permitiera que los productos software una vez se hubieran evaluado y obtenido un nivel adecuado de calidad, pudieran también conseguir un certificado. Como resultado AENOR creó un procedimiento que, a partir del informe del laboratorio acreditado y tras una auditoría, permitía certificar la calidad del producto software bajo estudio. Finalmente, a lo largo de 2013 se realizó un proyecto piloto de evaluación y certificación de los primeros productos software. Dicho proyecto permitió a tres empresas españolas evaluar, mejorar y finalmente certificar la calidad de sus productos software en base a la ISO/IEC 25000, tal y como se presenta en [28]. Entre los beneficios obtenidos, dichas empresas destacaron haber reducido hasta en un 40% el esfuerzo en las tareas de mantenimiento y hasta en un 30% el tamaño de sus productos.

Fig. 5. Marca de acreditación ILAC-ENAC con número de laboratorio AQC Lab

6 Conclusiones y Trabajo Futuro

La calidad del producto software es una de las principales preocupaciones tanto para las empresas desarrolladoras, como para los organismos que los adquieren. El presente artículo ha presentado una solución, formada por un modelo, un proceso y un entorno tecnológico, que permiten realizar la evaluación del producto software y que han sido validados y posteriormente acreditados por ENAC, consiguiendo además que los productos software puedan obtener un certificado de calidad emitido por AENOR. La experiencia del proyecto piloto nos ha permitido por un lado conocer cómo se encuentra la calidad real del producto software que desarrollan las empresas de nuestro país, que inicialmente obtuvo unos niveles bajos, pero que con el esfuerzo de refactorización les permitió obtener el certificado. Y por otro lado, detectar la necesidad real expresada por estas empresas, de difundir la posibilidad e importancia de evaluar la calidad de los productos software.

Como trabajo futuro destacamos la ampliación del entorno para la evaluación del resto de características de la familia ISO/IEC 25000, así como la inclusión de nuevas herramientas de medición que permitan analizar productos software desarrollados en lenguajes como Cobol, Ruby o PL/SQL, puesto que aunque las métricas son por norma general independientes del lenguaje, las herramientas de medición del primer nivel comentado en el entorno de evaluación, sí que lo son.

Agradecimientos. Esta investigación forma parte del proyecto ECU: Evaluación y Certificación del Producto Software (Consejería de Empleo y Economía y FEDER, 1313CALT0056) y el proyecto GEODAS-BC (Ministerio de Economía y Competitividad y FEDER, TIN2012-37493-C03-01).

Referencias

1. Kitchenham, B.,Pfleeger, S.L., *Software Quality: The Elusive Target.* IEEE Software, 1996. **20**(1): p. 12-21.
2. Maibaum, T.,Wassyng, A., *A Product-Focused Approach to Software Certification.* Computer, 2008. **41**(2): p. 91-93.
3. Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., Macleod, G.J., Merrit, M.J., *Characteristics of Software Quality*. 1978: North-Holland.
4. ISO, *ISO/IEC 9126, Software Product Evaluation–Quality Characteristics and Guidelines for their Use*. 2001, International Organization for Standardization.
5. ISO, *ISO/IEC 14598-5:1998 - Information technology -- Software product evaluation -- Part 5: Process for evaluators.* 1998, International Organization for Standardization: Ginebra.
6. Heitlager, I., Kuipers, T., Visser, J., *A Practical Model for Measuring Maintainability*, in *Quality of Information and Communications Technology, 2007. QUATIC 2007.* 2007. p. 30-39.
7. ISO, *ISO/IEC 25040 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Evaluation process*. 2011: Ginebra, Suiza.
8. ISO, *ISO/IEC 25010, Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. 2011: Ginebra, Suiza.

9. Rodríguez, M.,Piattini, M., *Systematic review of software product certification*, in *CISTI 2012: 7th Iberian Conference on Information Systems and Technologies*. 2012: Madrid. p. 631-636.
10. Kitchenham, B., *Guideline for performing Systematic Literature Reviews in Software Engineering. Version 2.3*. 2007, University of Keele (Software Engineering Group, School of Computer Science and Mathematics) and Durham (Department of Computer Science).
11. Heck, P., Klabbers, M., van Eekelen, M., *A software product certification model*. Software Quality Journal, 2009. **18**(1): p. 37-55.
12. Carvalho, F., Meira, S.R.L., Freitas, B., Eulino, J., *Embedded software component quality and certification.* Conference Proceedings of the EUROMICRO, 2009: p. 420-427.
13. Burger, E.,Reussner, R., *Performance certification of software components.* Electronic Notes in Theoretical Computer Science, 2011. **279**(2): p. 33-41.
14. Serebrenik, A., Mishra, A., Delissen, T., Klabbers, M., *Requirements certification for offshoring using LSPCM*. Proceedings - 7th International Conference on the Quality of Information and Communications Technology, QUATIC 2010, 2010: p. 177-182.
15. Yahaya, J.H., Deraman, A., Hamdan, A.R., *SCfM_PROD: A software product certification model*. 2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications, ICTTA, 2008.
16. Yahaya, J., Deraman, A., Hamdan, A.R., *Continuously ensuring quality through software product certification: A case study*. 2010 International Conference on Information Society, i-Society 2010, 2010: p. 183-188.
17. Hatcliff, J., Heimdahl, M., Lawford, M., Maibaum, T., Wassyng, A., Wurden, F., *A Software Certification Consortium and its Top 9 Hurdles.* Electronic Notes in Theoretical Computer Science, 2009. **238**(4): p. 11-17.
18. Morris, J., Lee, G., Parker, K., Bundell, G.A., Lam, C.P., *Software component certification.* Computer, 2001. **34**(9): p. 30-36.
19. Baggen, R., Correia, J.P., Schill, K., Visser, J., *Standardized code quality benchmarking for improving software maintainability.* Software Quality Journal, 2012. **20**(2): p. 287-307
20. Alvaro, A., De Almeida, E.S., Meira, S.L., *Towards a software component certification framework.* Proceedings - International Conference on Quality Software, 2007: p. 298-303.
21. Spinellis, D., *Code Quality: The Open Source Perspective*. 1 ed. Effective Software Development Series. 2006: Addison-Wesley Professional.
22. Martin, R.C., *Granularity*, in *C++ Report*. 1996. p. 57–62.
23. DeMarco, T., *Structured Analysis and System Specification.* 1979: Yourdon Press Computing Series.
24. ISO, *ISO/IEC 25020 Software and system engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Measurement reference model and guide*, in *International Organization for Standardization*. 2005: Ginebra, Suiza.
25. Rodríguez, M., Genero, M., Torre, D., Blasco, B., Piattini, M., *A Methodology for Continuous Quality Assessment of Software Artefacts*, in *The 10th International Conference on Quality Software (QSIC 2010)*. 2010: Zhangjiajie (China). p. 254-261.
26. Rodríguez, M., Garzás, J., Piattini, M., *Software Product Quality: An Open Source Automated Measurement Environment*, in *11th International Conference on Quality Engineering in Software Technology (CONQUEST 08)*. 2008: Postdam (Alemania).
27. Verdugo, J., Rodríguez, M., Piattinni, M., *Using Agile Methods to Implement a Laboratory for Software Product Quality Evaluation*, in *15th International Conference on Agile Software Development (XP 2014)*. 2014: Roma (Italia).
28. Rodríguez, M., Fernández, C.M., Piattini, M., *ISO/IEC 25000 Calidad del Producto Software.* AENOR. Revista de la Normalización y la Certificación, 2013(288): p. 30-35.