Modelación de Requisitos, Aspectos y Calidad de Software*

(Modeling Requirements, Aspects and Software Quality)

Rafael J. Caldera
Universidad de Oriente - Núcleo de Sucre, Prog. Lic. en Informática, Cumaná, Venezuela, 6101
Universidad Central de Venezuela, Centro ISYS, Escuela de Computación, Caracas, Venezuela, 1041-A
rjcaldera@sucre.udo.edu.ve

Isi S. Castillo
Universidad Nacional Experimental Sur del Lago, Lab. de Informática, Santa Bárbara, Venezuela, 5448
Universidad Central de Venezuela, Centro ISYS, Escuela de Computación, Caracas, Venezuela, 1041-A

castilloi@cantv.net

Francisca Losavio
Universidad Central de Venezuela, Centro ISYS, Escuela de Computación
Caracas, Venezuela, 1041-A
flosav@cantv.net

Alfredo Matteo
Universidad Central de Venezuela, Centro ISYS, Escuela de Computación
Caracas, Venezuela, 1041-A
almatteo@cantv.net

Resumen

El desarrollo de software orientado a aspectos ("Aspect-Oriented Software Development", AOSD), representa un nuevo paradigma de ingeniería de software basado en los conceptos de la Programación Orientada a Aspectos. Bajo este contexto, la investigación se centra en el tratamiento temprano de las incumbencias transversales o concerns transversales en combinación con los procesos clásicos de ingeniería de requisitos y diseño arquitectónico, donde las propiedades de calidad son relevantes. A pesar del reciente y creciente interés de esta línea de investigación, no es evidente la existencia de una visión compartida y homogénea que sirva de referencia para el razonamiento acerca de los términos utilizados. El objetivo de este trabajo es proponer un marco conceptual que permita representar y relacionar los principales términos del AOSD, asociados al proceso de ingeniería de requisitos y a la calidad de software, con el fin de establecer algunas bases para un mejor entendimiento y consenso en el manejo de un vocabulario común en la disciplina emergente de la Ingeniería de Requisitos Orientada a Aspectos.

Palabras Claves: aspectos, concerns, concerns trasversales, ingeniería de requisitos, Calidad de software.

Abstract

Aspect Oriented Software Development (AOSD), based on Aspect Oriented Programming, is part of the post-object paradigm of software engineering. The early treatment of requirements elicitation and specification, combining quality properties and crosscutting concerns, particularly at classical requirements engineering and architectural design stages, are open research subjects. In spite of the recent and increasing interest, a shared and homogenous vision of the terminology involved is still missing. The goal of this work is to model the main concepts used in AOSD using a standard notation, relating them with requirements engineering and software quality, setting the basis for a better understanding and consensus towards a common vocabulary for the emerging Aspect Oriented Requirements Engineering discipline.

Keywords: aspects, concerns, crosscutting concerns, requirements engineering, software quality.

* Esta investigación ha sido financiada por el Programa Alma Mater de la Oficina de Planificación del Sector Universitario (OPSU).

1. INTRODUCCIÓN

El proceso de desarrollo de software ha experimentado una continua evolución desde sus inicios hasta nuestros días en la búsqueda de nuevas perspectivas o paradigmas que permitan, entre otras cosas, mejorar el proceso y la calidad de los productos resultantes. La comprensión de los requisitos y la importancia de su correcta y precisa especificación, se incrementa drásticamente con la magnitud y complejidad de los sistemas de software, particularmente si estos tienen que responder a exigencias que van mas allá del alcance de sus funcionalidades principales, tales como interoperabilidad, adaptabilidad, disponibilidad, seguridad, entre otros. Estas exigencias deben ser consideradas oportunamente por los profesionales del software, en el desarrollo de sus aplicaciones y deben dirigir la atención de la comunidad de investigadores hacia la disciplina de la ingeniería de requisitos, en particular hacia los requisitos no funcionales, los cuales tienen un impacto directo en el cumplimiento de la funcionalidad principal o servicio ofrecido por los componentes de software [4]. Estos requisitos no funcionales o puntos de interés asociados generalmente a las propiedades de calidad del software y que en muchos casos se diseminan y entrecruzan a la funcionalidad principal, dificultando su entendimiento, mantenimiento y reutilización, son reconocidos generalmente como *incumbencias transversales* o *concerns transversales* ("crosscutting concerns"*)*, dentro del emergente paradigma del Desarrollo de Software Orientado a Aspectos ("Aspect-Oriented Software Development", AOSD) [1]. El estudio y tratamiento temprano de los *concerns transversales* en las primeras etapas del proceso de desarrollo de software se corresponde a la corriente reconocida en la comunidad como Ingeniería de Requisitos y Diseño Arquitectónico Orientado a Aspectos [8].

Actualmente existen por un lado un número de trabajos significativos en la identificación y tratamiento de los requisitos con énfasis en el tratamiento de las propiedades de calidad [4][16][21] [2][20], y por otro sobre la *separación de concerns* en las disciplinas de ingeniería de requisitos y diseño arquitectónico de parte de la comunidad del AOSD [3][8][9][19]; sin embargo, estas propuestas no se centran explícitamente en los *concerns transversales* [8]. Hasta ahora, en los aportes presentados por la comunidad de investigadores se observa muchas veces entremezclado el concepto *aspecto* y el de *concern*, siendo estos términos en muchos casos utilizados como sinónimos. Adicionalmente enmarcado en la disciplina emergente de la Ingeniería de Requisitos Orientada a Aspectos ("Aspect-Oriented Requirements Engineering", AORE), es común la utilización e intercambio de los términos *requisitos* y *concern*; así como la asociación de los posibles *concerns transversales* a las *propiedades de calidad* del software tales como: disponibilidad, fiabilidad, respuesta en tiempo, entre otras. Esta falta de homogeneidad y consenso dificulta el entendimiento y comprensión adecuada del paradigma.

Ante esta diversidad de posiciones y criterios, resulta importante resaltar que la caracterización de un dominio a través del modelado es la base para un entendimiento claro y común de sus elementos conceptuales, razón por la cual, el objetivo principal de este trabajo es establecer a través de un modelo, la correspondencia entre los principales términos del AOSD y su vinculación en la identificación de requisitos y la calidad del software, con el fin de sentar las bases para el establecimiento de un vocabulario común que sirva de soporte a la emergente disciplina de la AORE. Un adecuado manejo de los términos y relaciones asociados a este contexto, permite por un parte la identificación y separación temprana de los *concerns*, distinguiéndolos de manera clara y específica; por la otra, libera parcialmente a los programadores de dicha responsabilidad, y obliga a los ingenieros de software y diseñadores a producir sistemas con diseños limpios, para así finalmente generar código de alta consistencia.

Además de la introducción, este artículo esta estructurado de la siguiente manera: la sección 2 presenta los términos principales de los aportes del AOSD, la ingeniería de requisitos y calidad del Software. La sección 3 presenta y describe el modelo conceptual del dominio precisando los términos y relaciones principales para una ingeniería de requisitos de calidad orientada a aspectos. Finalmente en la sección 4, se presentan las conclusiones y directrices para los trabajos futuros.

2. TERMINOLOGÍA

Son diversas las definiciones y conceptos básicos relativos al enfoque orientado a aspectos, muchas son propias de la Programación Orientada a Aspectos ("Aspect-Oriented Programming", AOP) y extendidas al AOSD, a continuación se presenta un resumen de los términos comúnmente definidos en la comunidad de investigadores:

Un *concern* se define como una propiedad, o punto de interés de un sistema [9][10]. La IEEE en [11] define los *concerns* para un sistema como aquellos intereses que pertenecen al desarrollo del sistema y su operación, o demás aspectos que son críticos; o por el contrario, importantes para uno o varios *stakeholders* o participantes del proyecto de desarrollo de software. Los *concerns* incluyen consideraciones del sistema tales como, confiabilidad, seguridad, distribución, eficiencia, entre otros. En [15] se define un *concern* como un requisito de un sistema o consideración que debe ser direccionada con el fin de satisfacer una meta del sistema. El concepto de *concern* aparece frecuentemente asociado a incumbencia, competencia, área de interés, entre otros; para efectos de esta investigación se mantiene su sintaxis en inglés con el fin unificar términos.

Los *concerns transversales* ("crosscutting concerns"), son los concerns que se extienden a lo largo de múltiples módulos de los sistemas (autenticación, persistencia de data, seguridad transaccional, chequeo de errores, etc.) [15]. En [18] se definen como aquellas propiedades que se reparten a lo largo de todo el código de una aplicación, son conceptos que no pueden encapsularse dentro de una unidad funcional, debido a que atraviesan todo el sistema.

La separación de concerns ("Separation of Concerns", SOC) es un término utilizado desde hace muchos años; fue primero introducido por Dijkstra [6], como un principio para encapsular características en entidades separadas con la finalidad de ubicar en un lugar particular sus cambios y tratarlas una por una en el tiempo. En forma general, la SOC ha sido definida como la habilidad de identificar, encapsular y manipular partes del software (concerns) que son relevantes para un propósito particular; la SOC permite reducir la complejidad del software y mejorar la entendibilidad y trazabilidad a lo largo de un proceso de desarrollo, minimizando el impacto del cambio que promueve la evolución [22].

Un *aspecto* se define como una unidad modular diseñada para implementar un concern, pueden contener en su definición código y las correspondientes instrucciones de donde, cuando y como invocarlo [9]. Los aspectos son unidades modulares de implementaciones comunes, definidos por medio de declaraciones de tipo aspecto, las cuales tienen una forma similar a las declaraciones de las clases en el enfoque orientado a objetos. En [14] Kiczales define un aspecto de programa o código como la unidad modular que aparece en otras unidades modulares del programa, similar a una clase pero con un nivel de abstracción mayor; Kiczales también define un aspecto como la estructura que encapsula un concern transversal.

El AOSD representa un nuevo paradigma de ingeniería de software para dar soporte a la SOC en el proceso de desarrollo de software [5]; esta basado en los conceptos de la AOP [14]; apareciendo como una evolución natural en un intento por aplicar el concepto de aspectos a lo largo de todo el ciclo de desarrollo de software. En el AOSD, se extienden las técnicas tradicionales de desarrollo, tal es el caso de la orientación a objetos; permitiendo a los desarrolladores encapsular en módulos separados aquellas propiedades o *concerns* que normalmente atraviesan diversos componentes de un sistema. [1].

La idea central del AOSD, es dar solución al problema de los *concerns transversales*, a través de la aplicación de métodos y técnicas en todas las fases del ciclo de vida de desarrollo de software desde requisitos y diseño, hasta la implementación. El objetivo es desarrollar procesos, métodos, lenguajes, y mecanismos de abstracción, que permitan la sistemática identificación, modularización, representación y composición de los *concerns transversales* [1].

Inicialmente las líneas y trabajos investigación, se centraban en los lenguajes de programación y en la fase de implementación, en la medida que se desarrollaba el paradigma, las investigaciones comenzaron a atacar desde diversos puntos de vista, la adopción de este enfoque a través del todo el ciclo de vida de desarrollo de software. Bajo este contexto, se han agrupado las investigaciones y aplicaciones orientadas al tratamiento de los concerns transversales desde las primeras etapas del ciclo de vida de desarrollo de software, identificadas en la comunidad de investigadores como Ingeniería de Requisitos Orientada a Aspectos ("Aspect Oriented Requirements Software", AORE) y/o Early Aspects (Ingeniería de Requisitos y Diseño Arquitectónico Orientado a Aspectos).

La AORE [9], incluye técnicas como toda ingeniería de requisitos para modelar explícitamente aspectos o concerns de un sistema de software; proporcionando nuevas abstracciones y mecanismos para modularizar y componer tales concerns. Algunos enfoques de la AORE alcanzan la separación de concerns a través de la separación de los requisitos funcionales y no funcionales. En [9] se presentan a los requisitos no-funcionales como restricciones que afectan diversos componentes de un sistema, los cuales están asociados con la calidad de servicio (Ejemplo: usabilidad, seguridad) razón por la cual son bien vistos como buenos candidatos para los Aspectos.

En el proceso de ingeniería de requisitos clásico, los requisitos son definidos durante las primeras etapas del ciclo de desarrollo, representan especificaciones de cómo el sistema será implementado [20]. Los requisitos son descripciones de cómo el sistema debe comportarse, o de sus propiedades y atributos, pueden ser también restricciones sobre el proceso de desarrollo. Los requisitos incluyen tres niveles: requisitos del negocio (reglas de negocio), requisitos de usuarios y requisitos funcionales; adicionalmente cada sistema tiene requisitos no funcionales asociados [21]. En [19] un requisito es definido como un tipo especial de concern; en general una especificación de requisitos bien escrita se caracterizaría por expresar cada requisito en una oración simple y clara (representando exactamente un concern), evitando la descripción cruzada de otros requisitos. En [9] se reconoce de manera explícita que al menos algunos de los requisitos de los sistemas representan aspectos que entrelazan tanto a los requisitos como a los artefactos producidos a lo largo del ciclo de vida del desarrollo de software. De manera general los requisitos de software se pueden clasificar en dos grandes grupos: requisitos funcionales y requisitos no funcionales [20][21]. Los requisitos funcionales especifican la funcionalidad del software ha ser incorporada por los desarrolladores en el producto, para garantizar que sean cumplidas las tareas de sus usuarios, además deben satisfacer los requisitos del negocio y permiten describir lo que el desarrollador necesita implementar. Los requisitos no funcionales especifican condiciones que un sistema debe satisfacer, controlan la especificación, implementación y ejecución de los requisitos funcionales. Tanto los requisitos funcionales como los no funcionales son utilizados en el diseño de productos de software que implementan la funcionalidad requerida dentro de los objetivos de calidad deseados y las restricciones impuestas.

En [4] se presenta una propuesta en donde se extiende el proceso de ingeniería de requisitos clásico a través de un modelo de clasificación de requisitos, el cual considera de manera explícita, los requisitos no funcionales asociados a propiedades de calidad1 (ej: desempeño, seguridad, tolerancia a fallas, etc.). Estas propiedades o características de calidad se agrupan bajo un modelo de calidad el cual proporciona un marco de trabajo para especificar los requisitos de calidad y evaluar la calidad del producto de software.

A continuación se propone un modelo conceptual para este contexto, estableciendo la correspondencia entre los principales términos antes descritos.

3. MODELO CONCEPTUAL

Esta sección, presenta el modelo conceptual, expresado en UML [13] propuesto para el razonamiento sobre los principales conceptos inherentes al dominio de una ingeniería de requisitos de calidad orientada a aspectos. En el modelo (ver figura 1) los elementos *requisito de software*, *concern y propiedades de calidad*, representan roles de gran importancia para la comprensión y debida relación de la disciplina de requisitos, con el paradigma del AOSD y la calidad del software.

De manera general, los requisitos son descripciones de cómo el sistema debe comportarse, de las propiedades o atributos de un sistema o de las restricciones sobre el proceso de desarrollo [20]. Para efectos de este trabajo los *requisitos de software* se asumen han sido previamente especificados y derivados de los requisitos del sistema e impuestos por las *reglas de negocio*. En las reglas de negocio se incluyen *políticas* (corporativas/gubernamentales), *restricciones de proceso* (estándares industriales/dominio), restricciones tecnológicas (estándares tecnológicos/capacidad). Esto implica que dependen del *dominio de aplicación* del software. Los *requisitos de software* se clasifican en dos grandes grupos [20][21]: los *requisitos funcionales* que capturan el comportamiento deseado del sistema y pueden ser expresados como servicios o funciones y los *requisitos no funcionales* que especifican las condiciones que debe satisfacer el sistema y restringen, condicionan o controlan la ejecución de los *requisitos funcionales*. Los *requisitos no funcionales* se asocian a propiedades cuantificables de calidad (*propiedad de calidad*) tales como: fiabilidad, disponibilidad, respuesta en tiempo entre otros [4]. Un *modelo de calidad* [7][12][17] es usado para expresar las *propiedades de calidad* que caracterizan el *dominio de aplicación*. Una *propiedad de calidad* puede originar una *funcionalidad implícita* impuesta o no por una *regla de negocio*. La *funcionalidad implícita* no compete al usuario final y no ha sido explícitamente indicada, pero debe ser adicionada a fin de garantizar el logro de la funcionalidad global de la aplicación. Una *funcionalidad implícita* responde a un *requisito no funcional* y por lo general es identificada por los miembros del equipo de desarrollo [4][1][16]. Una *propiedad de calidad* derivada de un *requisito funcional, no funcional* o de una *funcionalidad implícita* puede ser un potencial *concern transversal*, objeto de ser tratado a través del paradigma del AOSD [1]. En el contexto de

[1] Se entiende por "calidad" el conjunto de rasgos y características que dan a un producto o servicio su habilidad para satisfacer necesidades definidas o implícitas.

este paradigma, un *concern transversal* puede ser un *concern funcional* o *no funcional* que se entrecruza entre dos o mas *concerns funcionales* afectando a distintas partes de la aplicación y que de implementarse, producirían enmarañamiento y dispersión en su especificación; por tanto, un *concern funcional* o *no funcional* puede ser un *concern transversal* (representados a través de la especialización en el modelo).

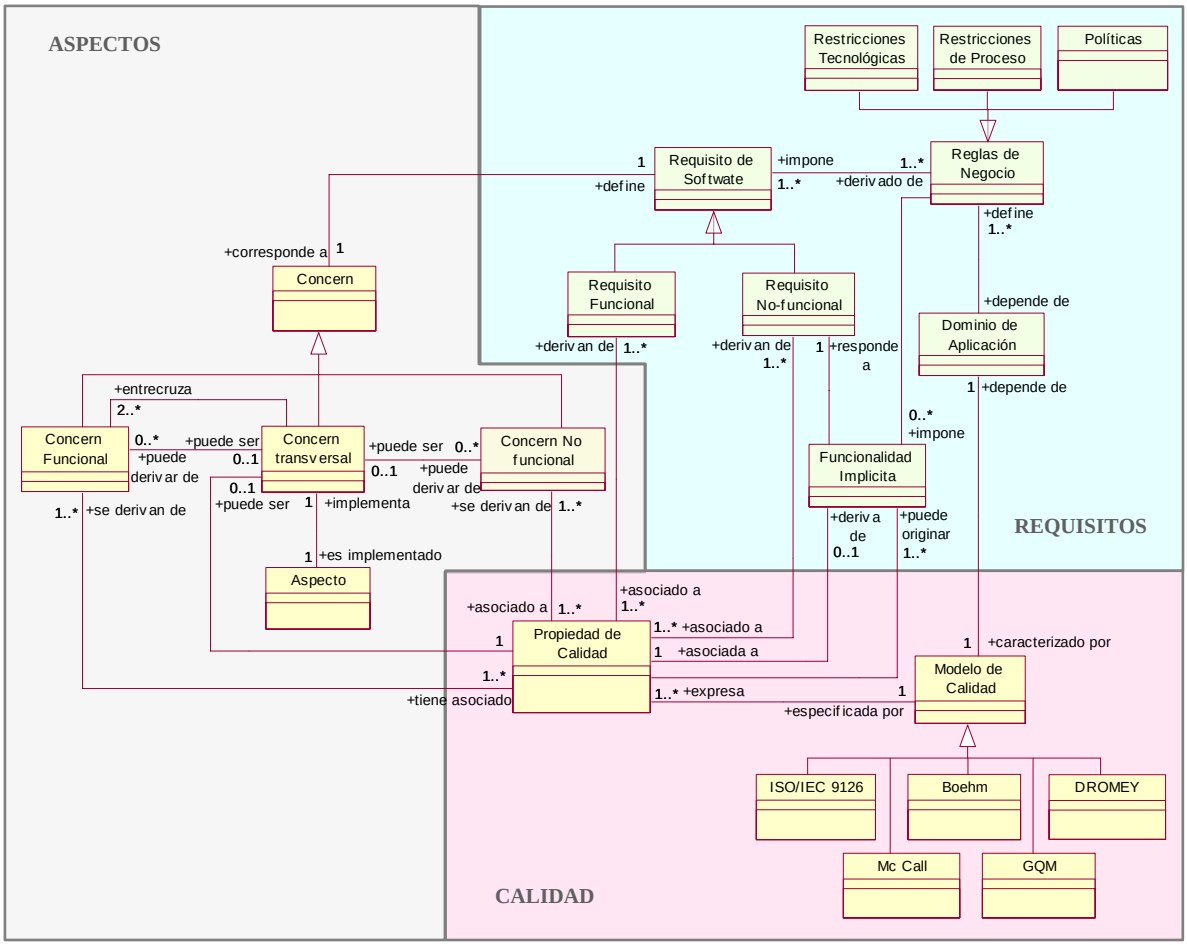

Figura 1: Vista Conceptual de Requisitos, Aspectos y Calidad de Software

Un *concern* corresponde a un *requisito de software*, del cual es necesario ocuparse para resolver un problema inherente al software [15]; por tanto, puede asignársele igual estructura, comportamiento y tratamiento dentro de las disciplinas de ingeniería de requisitos y diseño arquitectónico. Algunos *concerns*, tal como los *requisitos* se asocian a las funciones o servicios específicos que debe realizar la aplicación, y se especifican como *concerns funcionales;* los cuales tienen una o más *propiedades de calidad* asociadas para cumplir con su funcionalidad. Otros *concerns*, denominados *concerns no funcionales* describen las condiciones o restricciones que la aplicación debe satisfacer, pudiendo afectar la puesta en práctica de los *concerns funcionales*. Los *concerns no funcionales* surgen de las necesidades de los stakeholders, debido a las políticas de la organización, a la necesidad de interoperabilidad con otros sistemas (software/hardware) o a factores externos como los reglamentos de seguridad, las políticas de privacidad, entre otros. Los *concerns no funcionales* al igual que los *requisitos no funcionales*, se asocian a una o más *propiedades de calidad* cuantificables y han sido el objeto principal de estudio dentro del AOSD por ser considerados como potenciales *concerns transversales*. El mecanismo propuesto por el AOSD para resolver los *concern transversales* son los *aspectos*. Un *aspecto* es una unidad modular diseñada para implementar o encapsular un *concern transversal*, es una abstracción que a través de un mecanismo de composición presenta una alternativa para tratar adecuadamente un *concern transversal*.

4. CONCLUSIONES

Este trabajo define un marco común para el razonamiento, entendimiento y manejo de los conceptos asociados al dominio del paradigma del desarrollo de software orientado a aspectos. Se presenta un modelo conceptual expresado en UML, que asocia los elementos clásicos de la ingeniería de requisitos y de la calidad del software con el AOSD. Este modelo constituye una herramienta útil para dar soporte a la emergente disciplina de la ingeniería de requisitos orientada a aspectos, permitiendo la especificación de los requisitos de una aplicación usando la terminología de la orientación a aspectos y una terminología estándar de calidad. Como trabajos futuros, esta investigación se dirige hacia la definición de un proceso de ingeniería de requisitos de calidad orientada a aspectos. Adicionalmente, se trabaja hacia la caracterización de los sistemas fiables basada en el paradigma del AOSD, para tratar las propiedades de calidad asociadas a este tipo de sistemas.

Referencias

[1] AOSD Home Page: http://www.aosd.net

[2] Bass L., Klein M. y Bachmann. Quality Attribute Design Primitives and Attribute Driven Design Method. 4th International Workshop on Product Family Engineering. Bilbao, Spain, 3-5. 2001.

[3] Brito I. y Moreira, A. Integrating the NFR framework in a RE Model. In Early Aspects 2004: Aspects-Oriented Requirements Engineering and Architecture Design Workshop (AOSD). Lancaster. 2004

[4] Chirinos L., Losavio F. y Matteo A. Identifying Quality-based Requirements. Information Systems Management (ISYM). Auerbach Publications, 21(1), 15-21, 2004.

[5] Daves T. Reflective Software Engineering – From MOPS to AOSD. Journal of Objet Technology, 1(4). 2002.

[6] Dijsktra E. A Discipline of Programming. Englewood Cliffs, NJ. Prentice Hall. 1976.

[7] Dromey, R., "Cornering the Chimera", IEEE Software, vol. 13, Nº 1, pp. 33-34, 1996.

[8] Early Aspect Home Page: http://www.early-aspects.net

[9] Filman R., Elrad, T., Clarke, S. y Aksit, M. Aspect-Oriented Software Development. Addison Wesley, Boston. 2005

[10] Gutiérrez J., Villadiego D., Escalona M. y Mejías M. Aplicación de la Programación Orientada a Aspectos en el Diseño e Implementación de Pruebas Funcionales. DSOA'2004, IX Jornadas de Ingeniería de Software y Bases de Datos. Málaga. 2004.

[11] IEEE. Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Std. 1471-2000.

[12] ISO/IEC: FCD 9126-1. Information Technology - Software Engineering Product Quality. Part 1: Quality Model. 2001.

[13] Jacobson I., Booch G y Rumbaugh J. El Lenguaje de Modelado Unificado. Segunda Edición. Madrid: Addison Wesley. 2000.

[14] Kiczales G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M. y Irwin, J. Aspect-Oriented Programming. ECCOP'97 Object-Oriented Programming, 11th European Conference, M. Aksit y S. matsouka, Eds. LNCS 1241, 220-242. 1997.

[15] Laddad R. AspectJ IN ACTION. Practical Aspect-Oriented Programming. Manning Publications. 2003

[16] Losavio F., Chirinos L., Levy N. y Randane-Cherif A. Quality Characteristics for Software Architecture. Journal of Objetc Technology, 2(2), 133-150. 2003.

[17] McCall J.A., Richards P.K., y Walters G.T. Factors in Software Quality.Vols 1,2,3. ADLA-049-015/055 SPRINGFIELD. Va: National Technical Information Service, 1977.

[18] Reina, A., Torres, J., Toro, M., Álvarez, J. y Nieto, J. Una experiencia Práctica reutilizando Aspectos. Actas del Taller de Trabajo de Desarrollo de Software Orientado a Aspectos DSOA'03. Alicante. 2003.

[19] Rosenhainer L. Identifying Crosscutting Concerns in Requirements Specifications. In Early Aspects 2004: Aspects-Oriented Requirements Engineering and Architecture Design Workshop (AOSD). Lancaster. 2004.

[20] Sommerville I y Sawyer, P. Requirements Engineering. A Good Practice Guide. John Wiley and Sons, New Cork 1997.

[21] Wiegers K. Software Requirements: Practical techniques for gathering and managing requirements throughout the product development cycle. Microsoft Press, Washington, USA, pp 12-14. 2003.

Disponible en: http://www.idt.mdh.se/phd/courses/sa/IEEE.pdf

[22] Workshop on Multi-Dimensional Separation of Concerns, International Conference on Software Engineering, ICSE 2000.

Disponible en: www.research.ibm.com/hyperspace/workshops/icse2000