EVALUACIÓN DE LA CALIDAD DE LA REPRESENTACIÓN DE ESPECIFICACIONES DE SOFTWARE ORIENTADAS A OBJETOS

Clifton E. Clunie B.
Universidad Tecnológica de Panamá
FISC- Facultad de Ingeniería de Sistemas Computacionales

RESUMEN

El uso de la tecnología de orientación a objetos está en un proceso de expansión y está siendo utilizada en aplicaciones complejas y críticas. Como consecuencia de este comportamiento, la urgencia por la calidad se convierte en un aspecto fundamental.

Este artículo define atributos de calidad y procedimientos de evaluación de software desarrollado utilizando el paradigma de orientación a objetos, considerando las fases de especificación de requisitos y diseño. La organización y discusión de estos atributos se realiza de acuerdo a un método de evaluación de la calidad de software anteriormente propuesto.

Palabras claves : calidad de software, orientación a objetos, especificaciones orientadas a objetos, atributos de calidad de software orientado a objetos, ingeniería de software orientada a objetos

ABSTRACT

The scale of object-oriented technology has changed. Object-oriented technology use is expanding to large complex-critical applications being built by teams of developers with diverse skills. The urgency for quality assurance is heightened.

This work define quality attributes and evaluation processes for object-oriented software development considering the requeriments and design phases. The attributes organization and discussion are fulfilled according to a software quality evaluation method previously proposed.

Keywords: software quality, object oriented, object oriented specifications, quality attributes for object oriented software, object oriented software engineering.

1. Introducción

La Ingeniería del Software está realizando grandes esfuerzos, con el objetivo de ofrecer nuevas técnicas y métodos de desarrollo para alcanzar mayor calidad en los productos resultantes del proceso y disminuir los costos de desarrollo y mantenimiento de los sistemas.

La baja calidad y productividad en el proceso de desarrollo del software se atribuye, principalmente, al aumento constante del tamaño y la complejidad del software, al incremento de la demanda por nuevos productos, al carácter dinámico e iterativo a lo largo de todo el ciclo de vida, a la falta de procedimientos normalizados de evaluación de la calidad y a la falta de métodos y herramientas adecuadas para construir nuevas aplicaciones [1].

Las nuevas aplicaciones que surgen son diversas en complejidad y dominios de aplicación. Tratar de representar y modelar el comportamiento de tales aplicaciones empleando métodos y lenguajes de programación convencional, muchas veces, no es simple ni eficiente. Esto se debe, en parte, a la dificultad para abstraer a partir de

un modelo de la realidad, un modelo computacional.

Resulta evidente que el aumento de la complejidad es un acelerador paralelo para el uso del paradigma porque ofrece el mejor camino en el tratamiento de la complejidad de los sistemas [2]. Además, la tecnología basada en objetos ofrece subsidios que atienden a la naturaleza iterativa y dinámica del proceso de desarrollo.

Estas ventajas hacen que existan pocas dudas de que el paradigma de orientación a objetos será utilizado de modo generalizado para el desarrollo de pequeñas y grandes aplicaciones de software, independiente del dominio de aplicación. Prueba de esta tendencia es la aparición de propuestas de procesos de desarrollo y notaciones orientado a objetos, como es el caso del Proceso de Desarrollo Unificado [3] y el uso ya extendido de la notación UML (*Unified Modeling Language*), que se define como un "*lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software orientado a objetos*" [4].

Este artículo está organizado en cuatro secciones. Iniciamos las secciones 2 y 3 con una discusión sobre el desarrollo de especificaciones, definiendo un proceso de construcción y evaluación de especificaciones orientadas a objetos. En la sección 4, discutimos en detalle un conjunto de atributos de calidad para evaluar la confiabilidad de la representación de especificaciones orientadas a objetos. La organización de la discusión es realizada de acuerdo a un método de evaluación de la calidad. Finalmente, en la sección 5 son presentadas las conclusiones del artículo destacando sus contribuciones.

2. Generación y Evaluación de una Especificación Orientada a Objetos

La especificación describe como el software debe responder a los requisitos del problema y constituye la base sobre la cual se realiza todo el proceso de desarrollo. La generación de una especificación completa y con calidad es esencial para el éxito del esfuerzo de desarrollo de software. Consecuentemente, el proceso de generación de la especificación involucra las etapas de levantamiento de requisitos, construcción, evaluación y corrección de la especificación (Figura 1).

En la etapa de identificación de requisitos es realizado el proceso de extraer los requisitos. Este proceso tiene por objetivo hacer explícitos los deseos, las intenciones y necesidades de los usuarios con relación al software que va ser construido. En la etapa de construcción de la especificación es realizada la formalización del conocimiento obtenido en la etapa anterior, a través de una correcta representación, organización y documentación, utilizando un método de desarrollo orientado a objetos. En esta etapa son producidos modelos conceptuales, que describen estática y dinámicamente aspectos sobre el problema. En la etapa de evaluación de la especificación es utilizado el Plan de Control de la Calidad y el Manual para Control de la Calidad de Especificaciones Orientadas a Objetos. En este manual están definidos todos los criterios que serán evaluados. A través de la utilización del *scripting language* es posible capturar, a partir de los diagramas, algunos valores cuantitativos de criterios. En la etapa de corrección de la especificación, son realizadas las modificaciones necesarias para corregir los errores identificados en la evaluación.

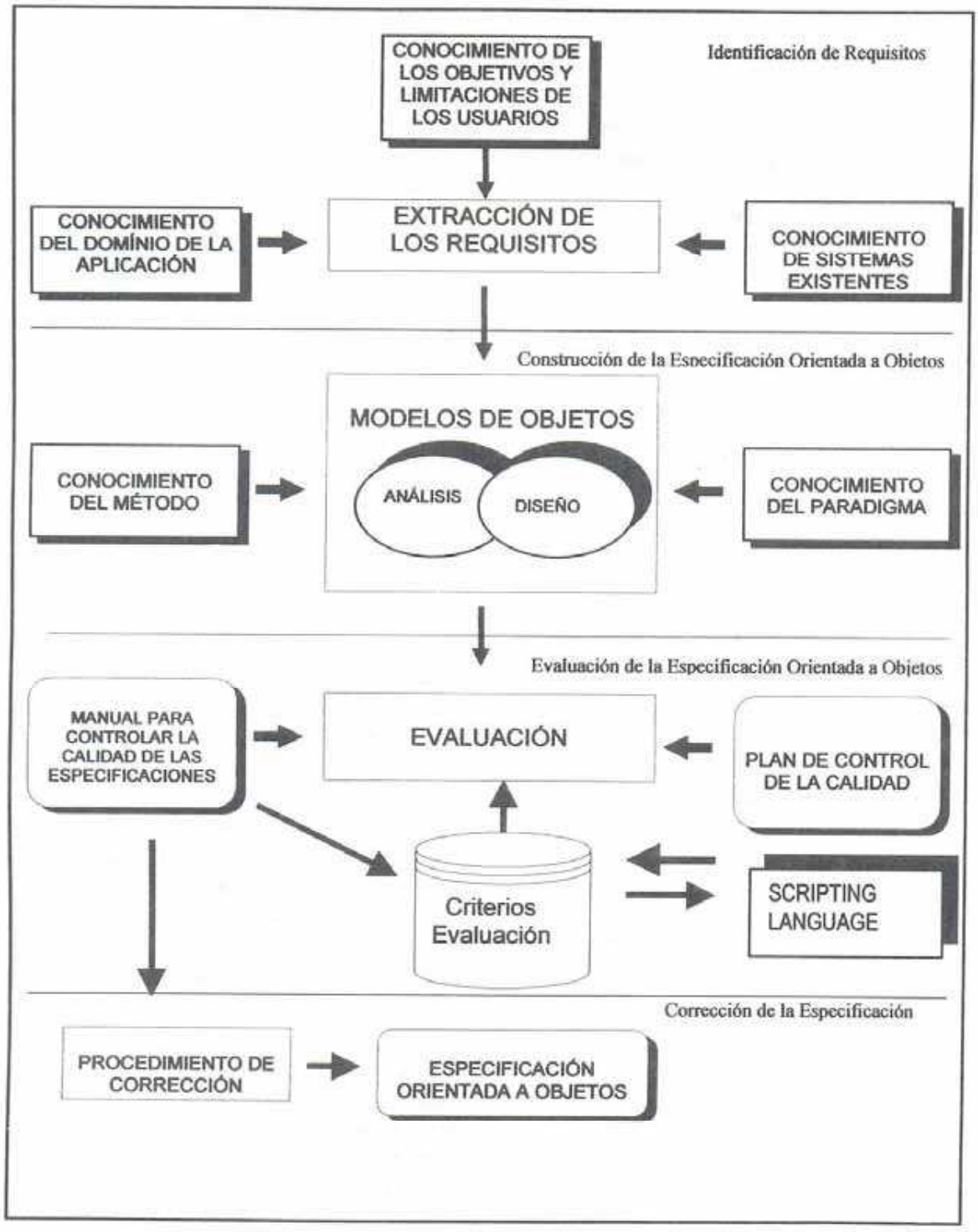

Figura 1: Proceso de Generación/Evaluación de Especificaciones Orientadas a Objetos

3. Desarrollo de Especificaciones Orientadas a Objetos

Los métodos de desarrollo orientado a objetos establecen un conjunto de actividades que deben ser realizadas en las fases de análisis y diseño. Sin embargo, la construcción de la especificación del sistema se realiza simultáneamente en diferentes dimensiones. La experiencia ha demostrado que en realidad existen algunas diferencias que pueden ser observadas en la generación de la especificación del sistema, si objetivos específicos son definidos para cada fase de desarrollo.

Basado en esto, varios aspectos pueden ser comentados con relación al desarrollo de una especificación orientada a objetos y las fases de análisis y diseño.

3.1 Con relación a los Niveles de Abstracción de las Clases

La identificación y documentación de las principales clases del dominio de la aplicación son realizadas a través de la revisión y discusión del documento de descripción del sistema (*casos de uso)* y de la interacción constante con los especialistas del dominio. Los criterios de selección de las clases, durante la fase de análisis, deben permitir identificar abstracciones que representen entidades del mundo real en el contexto del dominio del problema.

3.2 Con relación a las Jerarquía de las Clases

En la fase de análisis, son identificados las relaciones de tipo generalización – especialización con el objetivo de facilitar el entendimiento del modelo, observando las diferencias y semejanzas entre las clases formando una jerarquía. Sin embargo, en la fase de diseño, estas relaciones – *que incluye herencia simple* y *herencia múltiple* – son tratadas considerando su influencia sobre las decisiones de implementación, pruebas, mantenimiento y reutilización.

3.3 Con relación a las Instancias de las Clases

Cuando especificamos una clase en el modelo de análisis, asumimos de manera natural que la clase tenga instancias múltiples. Sin embargo, en la fase de diseño, es crítico resaltar y especificar condiciones para crear y destruir instancias de una clase. En esta fase, es necesario detallar algoritmos de creación y destrucción de las clases considerando sus asociaciones.

3.4 Con relación al Nivel de Detalle de las Operaciones

En la especificación del modelo de análisis es común no especificar detalles de las operaciones en las clases. En ese instante, lo que interesa, es entender el papel que juega la clase en el contexto del dominio del problema y no invertir tiempo en detalles innecesarios relativos a una especificación detallada de sus operaciones. Las operaciones de diseño, sin embargo, deben ser especificadas en detalle, utilizando un lenguaje de diseño o un lenguaje de programación que será utilizado en la implementación.

3.5 Con relación a los Atributos en las Clases

En la fase de análisis estamos preocupados en representar, de la mejor manera posible, a través de un subconjunto de clases, el dominio de la aplicación. Los atributos identificados en esta etapa, generalmente, están incompletos y no representan propiedades de las clases relacionadas a implementación. En la fase de diseño, los atributos son adicionados para caracterizar mejor las clases e implementar las relaciones entre ellas.

3.6 Con relación al Agrupamiento de Clases

El agrupamiento de las clases, se realiza para mejorar el entendimiento del modelo, ofreciendo una organización de clases en una perspectiva superior, mostrando claramente los sub-dominios de la aplicación. En la fase de diseño, son representadas las clases de implementación, aumentando el nivel de detalle de las operaciones y especificando los atributos de implementación. El modelo crece en tamaño y cantidad de informaciones, permitiendo efectuar agrupamientos de clases de acuerdo a criterios diferentes.

4. Características de Calidad de Especificaciones Orientadas a Objetos

A pesar de que la comunidad de ingeniería del software cuenta con una propuesta de estandarización, UML (*Unified Modeling Language)* avalada por la OMG (*Object Management Group)*, donde se integra lo mejor de las propuestas de métodos de desarrollo orientado a objetos, el proceso de desarrollo orientado a objetos aún no ha alcanzado su mayoría de edad.

En esta sección definimos un conjunto de atributos de calidad que deben ser considerados para evaluar especificaciones de un modo general y para el caso particular de especificaciones orientadas a objetos. Estos atributos de calidad fueron definidos tomando como referencias trabajos de calidad de especificaciones de software descritos en [5, 6, 7, 8, 9, 10], de la literatura técnica sobre calidad en orientación a objetos [11, 12, 13, 1, 14, 15] y [16, 17, 18, 2] de los métodos de desarrollo de software orientados a objetos. La organización y discusión de los atributos se realiza de acuerdo al Modelo de Evaluación de la Calidad, discutido en [19 y 20] que describimos a continuación (Figura 2):

- ***Objetivos de Calidad:*** son propiedades generales que el producto debe poseer;

- ***Factores de Calidad del Producto***: determinan la calidad del producto desde el punto de vista de los diferentes usuarios (usuario final, mantenedor, etc.);

- ***Criterios:*** son atributos primitivos posibles de ser evaluados;

- ***Procesos de Evaluación*:** determinan el proceso y los instrumentos que serán utilizados para medir el grado de presencia, en el producto, de un determinado criterio;

- ***Medidas***: indican el grado de presencia, en el producto, de un determinado criterio;

- ***Medidas Agregadas*:** son el resultado de la agregación de las medidas obtenidas en la evaluación de acuerdo con los criterios que cuantifican los factores;

- ***Funciones Fuzzy:*** relaciona y cuantifica los atributos de calidad primitivos o agregados, a través de términos lingüísticos.

Los objetivos de calidad son alcanzados a través de los factores de calidad, que pueden estar compuestos por otros factores y son evaluados a través de criterios. Los criterios definen los atributos de calidad para los factores. Medidas son valores resultantes de la evaluación de producto de acuerdo a un criterio específico. Objetivos y factores no son directamente mensurables y solo pueden ser evaluados a través de criterios. Un criterio es un atributo primitivo, o sea, un atributo independiente de todos los otros atributos. Ningún criterio aislado es una descripción completa de un factor o sub-factor determinado. Así como, ningún factor define completamente un objetivo.

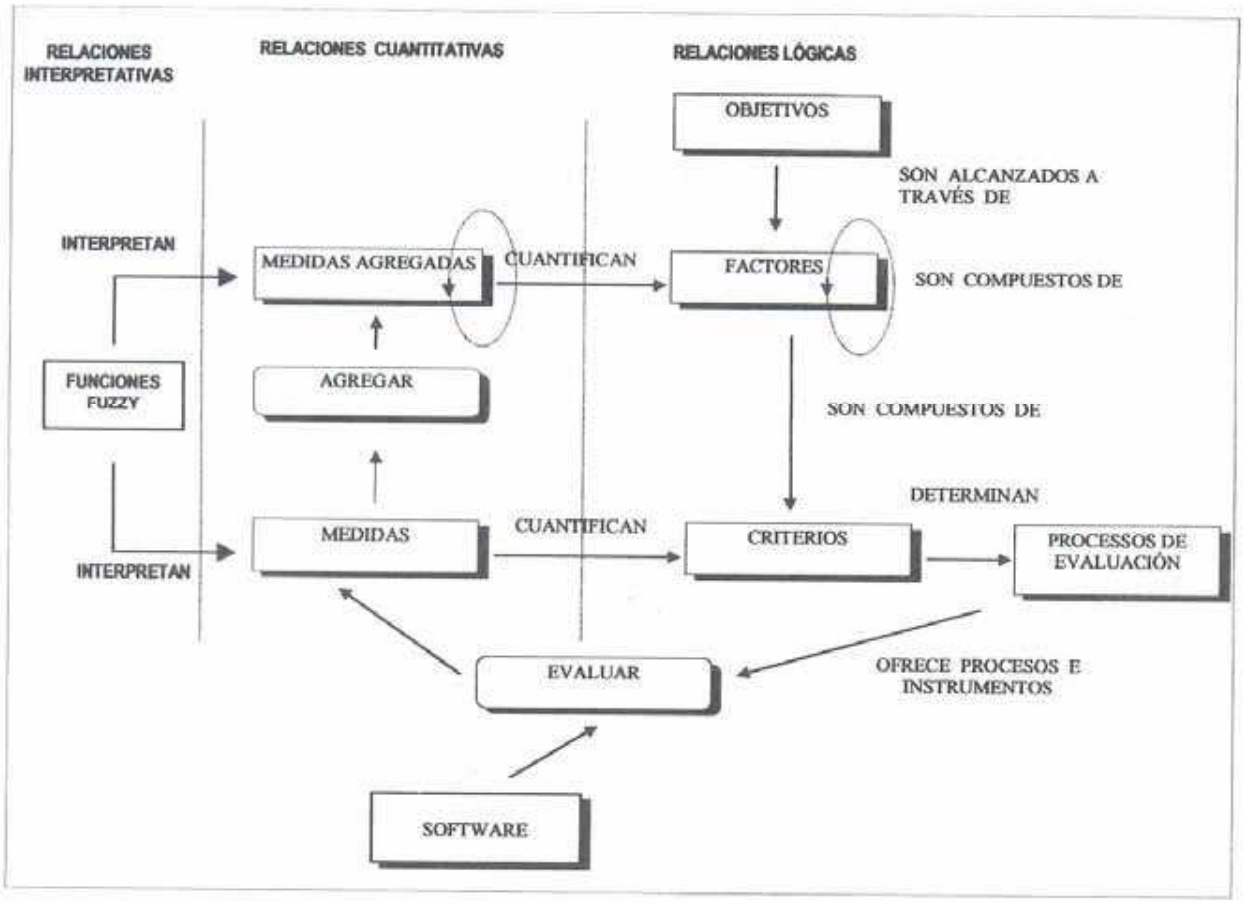

Figura 2: Modelo de Evaluación de la Calidad del Software.

Productos de software son desarrollados para atender a determinadas necesidades de sus usuarios. Después de que son colocados en operación, se espera que tengan una vida útil, larga y productiva. Para que esto pueda ser concretizado, deben ser alcanzados los siguientes objetivos de calidad: ***Confiabilidad de la Representación,*** ***Confiabilidad Conceptual*** y ***Utilidad.***

4.1 Objetivo: Confiabilidad de la Representación

El objetivo ***Confiabilidad de la Representación*** se refiere a las características que hacen la especificación fiable para sus usuarios, considerando los aspectos relativos a su forma y que hacen posible su comprensión y manipulación, considerando que la especificación evoluciona gradualmente a lo largo del desarrollo y debe ser modificada durante la vida útil del producto cuando se realizan acciones de mantenimiento. Este objetivo se realiza a través de los factores de calidad Comunicabilidad y Manipulabilidad .

4.1.1 Factor: Comunicabilidad – *Conjunto de atributos de calidad que evalúan la capacidad de la especificación de comunicar su contenido.*

Especificaciones orientadas a objetos incluyen modelos de clases y objetos, donde son representadas y documentadas las entidades que caracterizan el dominio de la aplicación y resuelven el problema. Estos modelos de clases y objetos sirven a diversos

ATRIBUTOS DE CALIDAD		Especificaciones en General	Modelo Análisis O. O.	Modelo Diseño O.O.
Factor: COMUNICABILIDAD				
	Corrección en el Uso del Método			
	Corrección de la Notación	✖		
	Corrección Sintáctica	✖		
	Corrección Semántica	✖		
	Corrección en el Uso del Formato de Documentación	✖		
	Uniformidad de Terminologia			
	Uniformidad de Términos	✖		
	Uniformidad de Notación	✖		
	Uniformidad con relación al Nivel Abstracción			
	Uniformidad de Detalle de la Documentación	✖		
	Independencia de Detalles de Diseño	✖		
	Uniformidad de Abstracción de las Clases		✖	✖
	Uniformidad de Abstracción de los Atributos		✖	✖
	Uniformidad de Abstracción de las Operaciones		✖	✖
	Uniformidad de Abstracción de los *Clusters*		✖	✖
	Modularidad de la Documentación			
	Cohesión de las Informaciones	✖		
	Acoplamiento entre las Secciones	✖		
	Estructura de la Documentación	✖		
	Corrección de la Arquitectura			
	Nivel de Fatorización de la Clase		✖	✖
	Nivel de Profundidad de la Jerarquía de la Clase		✖	✖
	Cohesión Estructural de la Clase			✖
	Cohesión Estructural del *Cluster*			✖
	Acoplamiento de Relación entre Clases		✖	✖
	Acoplamiento de Relación entre *Clusters*			✖
	Acoplamiento de Interacción entre Clases			✖
	Acoplamiento de Interacción entre *Clusters*			✖
	Tamaño			
	Tamaño de la Especificación	✖		
	Tamaño de la Clase		✖	✖
	Tamaño de la Operación		✖	✖
	Tamaño del *Cluster*			✖
	Tamaño de la Interface			✖
	Simplicidad			
	Simplicidad de la Operación			✖
	Simplicidad del Atributo			✖

Tabla 1: Características de Calidad del Objetivo Confiabilidad de la Representación.

objetivos: proporcionan una base de concordancia entre el cliente y el equipo de desarrollo, son la base para la implementación y permiten evaluar si los requisitos descritos están siendo implementados.

El factor comunicabilidad es evaluado a través de los sub-factores ***Corrección en el uso del Método, Uniformidad de la Terminología, Uniformidad en el Nivel de Abstracción, Modularidad de la Documentación, Corrección de la Arquitectura, Tamaño y Simplicidad*** (Tabla 1).

A continuación discutimos en detalle los sub-factores de calidad relacionados a modelos de clases, ***Corrección de la Arquitectura, Tamaño y Simplicidad.*** Atributos de calidad que corresponden a los otros sub-factores pueden ser encontrados en detalle en [21].

4.1.1.1 SubFactor: Corrección de la Arquitectura – *Conjunto de atributos de calidad que evalúan la corrección del modelo considerando la disposición, composición y relación entre sus componentes.*

Uno de los objetivos de construir modelos de clases y objetos es proporcionar una adecuada disposición, composición y relaciones entre sus componentes, considerando que el modelo resultante sea de fácil entendimiento, evaluación, implementación y mantenimiento y con posibilidades de reutilización en otros contextos similares. Para alcanzar estos objetivos, es necesario realizar una evaluación de los componentes del modelo, observando una perspectiva local, al nivel de clases y objetos y una perspectiva de conjunto, al nivel de *clusters*. Al evaluar este sub-factor, debemos considerar los siguientes criterios: ***Nivel de Factorización de la Clase, Nivel de Profundidad de la Jerarquía de la Clase, Cohesión Estructural de la Clase, Cohesión Estructural del Clusters, Acoplamiento de Relación entre Clases, Acoplamiento de Interacción entre Clases, Acoplamiento de Relación entre Clusters, Acoplamiento de Interacción entre Clusters.***

- **Criterio: Nivel de Factorización de la Clase - NFC**

El nivel de factorización de la clase es entendido como la cantidad de subclases inmediatas de una clase ancestral C_i. El valor puede ser obtenido a partir de:

> *NFC* (C_i) = *Número de clases inmediatas de la clase* C_i

Si la clase presenta una gran cantidad de descendientes inmediatos, es una indicación de cuantas clases de especialización van a heredar los atributos y operaciones de la clase ancestral. Aunque esto sea considerado positivo, bajo la óptica de reutilización, una clase con muchos descendientes podrá requerir de un mayor esfuerzo durante el proceso de pruebas de las operaciones de las clases involucradas, porque las operaciones de la clase ancestral necesitan ser nuevamente probados en las clases descendientes.

- **Criterio: Nivel de Profundidad de la Jerarquía de la Clase - NPJ**

Una estructura de generalización - especialización representa una jerarquía de clases formadas por varios niveles. A través de un análisis de la jerarquía, es posible identificar en que medida las clases ancestrales afectan a las demás clases, observándose la profundidad de las clases en la jerarquía y el número de operaciones heredadas. Estas informaciones indican el grado de complejidad de la estructura.

Sean C_1, ..., C_n las clases y ***NPC(Ci)*** el nivel de la clase en la jerarquía, el nivel de profundidad ***NPJ*** es dado por:

$$NPJ = \frac{\sum_{i=1}^{n} NPC(C_i)}{n}$$

El aumento de los niveles de profundidad de la jerarquía implica en un esfuerzo mayor para la comprensión y mantenimiento de las clases. Esto se debe, a que la herencia afecta la localizabilidad y el principio de encapsulamiento.

- **Criterio: Cohesión Estructural de la Clase - CEC**

Si una clase tiene diferentes operaciones realizando funciones diferentes sobre el mismo conjunto de argumentos, entonces la clase tiene alta cohesión. Para realizar esta verificación, es posible determinar el grado de similaridad[1] entre las operaciones especificados en la clase, observando el conjunto de argumentos que participan en ellas. Operaciones que trabajan con los mismos argumentos pueden ser una indicación de que los servicios están relacionados.

Sea la clase C_i, con n operaciones $O1, ..., On$ y {A} el conjunto de argumentos declarado en la operación Oi, onde $1 \leq i \leq n$. Entonces, existen N conjuntos {A1},...,{An}, tal que M (número de conjuntos disjuntos) es generado a partir de la intersección entre los conjuntos de argumentos Ai $\cap$ Aj. La ***Cohesión Estructural (CEC)*** puede ser calculado por la relación entre el número de conjuntos disjuntos M y el conjunto total de argumentos N.

$$CEC(C_i) = (1 - M/N)$$

[1] Según [22] en [15], la similaridad entre dos entidades es determinada a partir de la comparación de sus propiedades.

Un valor alto de ***CEC*** es una indicación de que existe una fuerte relación entre las operaciones especificadas en la clase. De lo contrario, puede indicar que la clase deba ser subdividida en dos o más clases, pues probablemente realiza operaciones no relacionadas, haciendo que la clase sea más compleja aumentando las posibilidades de errores.

- **Criterio: Cohesión Estructural del Cluster - CCL**

La cohesión de un conjunto de clases puede ser calculada a través de la media del número de relaciones internas entre las clases. Sea CL un *cluster* y r el número de relaciones que son internas al *cluster*. Sea n el número de clases en el *cluster*. Entonces, el indicador de la ***Cohesión Estructural del Cluster (CCL)*** puede ser calculado de la siguiente manera:

$$CCL(CLi) = \frac{(r + 1)}{n}$$

(El valor de 1 en la ecuación evita que $CCL = 0$, cuando $n=1$)

Un valor alto de ***CCL*** es una indicación de que existe una fuerte relación entre las clases, de lo contrario, el objetivo definido para el *cluster* debe ser reformulado.

- **Criterio: Acoplamiento de Relación entre Clases - ACR**

El ***Acoplamiento de Relación*** es definido por el número de relaciones - *dependencia estática* - que una determinada clase tiene con otras clases en el modelo, excluyendo las relaciones por herencia. Sea C_i una clase del modelo y r la indicación del número de relaciones con otras clases. Entonces ***ACR*** es dada por:

$$ACR(Ci) = \sum_{i=1}^{n} r_i$$

Un modelo de clases con un alto ***Acoplamiento de Relación*** puede tornar el modelo difícil de entender y llevar a errores durante el desarrollo. Un cambio, o una falla, en una clase puede tener un efecto de propagación en otras clases en el modelo. Esto hace difícil la actividad de mantenimiento y la realización de las pruebas.

- **Criterio: Acoplamiento de Relación entre *Clusters* - ARCL**

Un *cluster*, además de estar compuesto por un conjunto de clases relacionadas, deben tener dependencia mínima con relación a otros *clusters*. El acoplamiento entre *clusters* es definido por el número de relaciones entre clases del *cluster* y clases que pertenecen a *clusters* diferentes. Sea CL_i ,..,CL_n un conjunto de *clusters* y r la indicación del número de relaciones entre clases que pertenecen a *clusters* diferentes. Entonces, ***ARCL*** es dado por:

$$ARCL(CLi) = \sum_{i=1}^{n} r_i$$

Un valor pequeño para ***ARCL*** es deseable, en la medida en que las relaciones entre las clases quedan confinadas a los *cluster*, sin atravesar sus fronteras. Eso permite que el conjunto de clases pueda ser implementado, probado y reutilizado de modo casi independiente, sin afectar los demás *clusters*.

- **Criterio: Acoplamiento de Interacción entre Clases - ACI**

Una clase tiene ***Acoplamiento de Interacción*** con otra clase si una de ellas actúa sobre la otra a través de mensajes, o sea, si sus operaciones utilizan las operaciones o atributos de otra clase. A partir de las operaciones especificadas en la clase, es posible determinar el grado de interacción que una clase tiene con otras clases. Sea ***Ci*** una clase con ***S1, . . . ,Sn*** los servicios declarados y ***Ri*** el conjunto de operaciones llamados por ***Si*** = {***Ri j***}, entonces el indicador que mide el grado de comunicación de la clase es el siguiente [15, 2]:

$$ACI(Ci) = Si \cup \{Rij\}$$

Un valor alto de ***ACI*** es una indicación de que la clase se comunica vía mensaje con una gran cantidad de otras clases, lo que puede significar una alta complejidad de la clase. A través de este indicador es posible determinar la complejidad de las pruebas en varias partes del modelo.

- **Criterio: Acoplamiento de Interacción entre Clusters - AICL**

Un *cluster* tiene *Acoplamiento de Interacción* si existe comunicación vía mensajes entre clases del *cluster* y clases que pertenecen a *clusters* diferentes. Sea CL_i un *cluster* formado por las clases C_1,... ,C_n y ***ACI*** (C_i) el acoplamiento de interacción de la clase C_i. El indicador del grado de ***Acoplamiento de Interacción entre Clusters*** es calculado a partir de:

$$AICL(CL_i) = \frac{\sum_{i=1}^{n} ACI(C_i)}{n}$$

Un valor bajo para ***AICL*** es deseable, pues indica que la mayoría de las interacciones entre las clases se realizan dentro de los límites del *cluster*. Esto permite que el conjunto de clases pueda ser implementado, reutilizado y probado con mayor facilidad. Para el caso específico de las pruebas de *clusters*, estos se tornan más complejos, pues deben ser planeados casos de prueba considerando las clases que están dentro y fuera del *cluster*.

4.1.1.2 Subfator: Tamaño - *conjunto de atributos de calidad que evalúa si la especificación contiene la cantidad mínima de componentes sin comprometer su calidad.*

Al considerar el sub-factor *tamaño*, para el caso de especificaciones orientadas a objetos, deben ser considerados, los siguientes criterios: ***Tamaño de la Clase, Tamaño de la Operación, Tamaño del Cluster*** y ***Tamaño de las Interfaces.*** A seguir presentamos una discusión sobre los mismos.

- **Criterio: Tamaño de la Clase - TC**

El tamaño adecuado para una clase varía dependiendo de la complejidad de la aplicación y de su responsabilidad en el contexto del problema. El *Tamaño de la Clase* es definido como la cantidad de atributos y operaciones declaradas en la clase. Sea la clase C_i con *NA* atributos y *NO* operaciones. El tamaño de la clase puede ser calculado a partir de:

$$TC(C_i) = NA + NO$$

Una clase con un valor de *TC* muy grande - *comparado con valores para otras clases* -, puede ser una indicación de la necesidad de dividir en clases menores. Si es necesario, se debe preservar la cohesión de las operaciones que serán definidas en ellas.

Desde el punto de vista de la reutilización, el tamaño de la clase objetos es un dilema. Por un lado, si se considera el costo/beneficio, una clase con una gran cantidad de atributos y operaciones es deseable, en la medida en que sus especializaciones puedan heredar y utilizar una gran cantidad de atributos y operaciones.

- **Criterio: Tamaño de la Operación -TO**

El *Tamaño de la Operación* puede ser obtenido a partir del número de líneas de pseudocódigo declaradas en las operaciones. Sea $O_1 \ldots O_n$ las operaciones definidas en la clase C_i. El tamaño de la operación puede ser obtenido a partir de:

$$TS(O_i) = \text{número de líneas de pseudocódigo de la operación } O_i$$

Podemos identificar algunas consecuencias no deseables, de la existencia de *TO* muy grandes: a) son más difíciles de comprender; b) Pueden ser un indicador de que el código que va ser generado es mas orientado a funciones que a orientado a objetos c) Son probablemente, más complejos y mas orientado a aplicaciones específicas, minimizando sus posibilidades de reutilización.

- **Criterio: Tamaño del Cluster - TCL**

El ***Tamaño del Cluster (TCL)*** es entendido como la cantidad de clases definidas en el cluster. Sea CL_i un cluster compuesto por clases-objetos $C_1, \ldots, C_n$. Su tamaño puede ser obtenido a través de:

$$TCL(CL_i) = \text{número de clases en el cluster } CL_i$$

A través del empleo de un mecanismo de identificación incremental de clases, es posible tener un mejor control, minimizar los errores y aumentar la productividad. Valores exactos de cantidad de clases para un determinado *cluster*, todavía no son establecidos. Sin embargo, se recomienda que un *cluster* adecuado, contiene en media cinco a veinte clases.

- **Criterio: Tamaño de la Interface - TI**

Una forma de construir diseños de alta calidad, con clases reutilizables, es a través del refinamiento de los protocolos estándar. Esto puede ser realizado a través de la reducción del número de argumentos que participan en los mensajes, dividiendo un

mensaje en varios. Esto aumenta el número de mensajes con argumentos similares. Por consiguiente, el ***Tamaño de las Interfaces (TI)*** es entendido como la cantidad de argumentos definidos en un determinado mensaje. Sea M_i un mensaje compuesto por los argumentos A_1 ,... A_n . Su tamaño puede ser calculado a través de:

$TI(M_i)$ = *número de argumentos del mensaje* M_i

Según [23], un buen número para ***TI*** es próximo a seis. Por otro lado, [24] recomienda que el protocolo de mensajes debe ser lo más simple posible y contener pocos parámetros, el autor sugiere que tenga en media, tres parámetros.

4.1.1.3 Subfator: Simplicidade - *conjunto de atributos de calidad que evalúan el grado de complejidad de las operaciones y atributos definidos en la clase.*

Las especificaciones sufrirán inevitablemente mantenimiento, debido a que frecuentemente encontramos fallas en las especificaciones. Con el objetivo de minimizar los impactos causados por modificaciones y/o posibles extensiones en la especificación, la simplicidad debe ser una característica a ser considerada durante la generación de clases. Clases con atributos y servicios complejos son más difíciles de entender, implementar, reutilizar y probar.

Al considerar el subfator *Simplicidad*, debemos considerar los criterios: ***Simplicidad de las Operaciones*** y ***Simplicidad de los Atributos.***

- **Criterio: Simplicidad de Operación - SIO**

El criterio ***Simplicidad de las Operaciones*** evalúa el grado de simplicidad de cada clase a través del análisis de cada una de sus operaciones. De esta manera, se determina el esfuerzo necesario para desarrollar, probar y mantener la clase. *Sea Ci* una clase, con n operaciones O_1 , O_2 , ..., O_n y sean s_1 , s_2 , ..., s_n valores de 0 - 1 asociados al nivel de simplicidad de cada operación , según la Tabla 2.

Nivel de Simplicidad de la Operación	Significado
0	Operaciones con instrucciones CASE, IF's , Ciclos
0.25	Operaciones con instrucciones IF's, Ciclos
0.50	Operaciones con instrucciones IF's
0.75	Operaciones con instrucciones Ciclos
1	Operaciones sin instrucciones IF's, CASE, Ciclos

Tabla 2: Escala de valores para el criterio simplicidad de la operación considerando su estructura de control.

El valor de la ***Simplicidad de la Operación*** de la clase es dada por:

$$SIO(C_i) = \sum_{i=1}^{n} s_i \ / \ \mathrm{n}$$

Un valor pequeño de ***SIO,*** puede significar en un gran impacto en las subclases, debido a la herencia de operaciones. Mientras mayor sea el valor de ***SIO,*** probablemente, más fácil será la reutilización , mantenimiento y prueba de la clase, pues se torna menos compleja. Se espera, que el valor de ***SIO*** sea lo mayor posible.

- **Criterio: Simplicidad de Atributos - SIA**

El criterio ***Simplicidad de Atributos*** evalúa el grado de simplicidad de los atributos definidos en cada clase. De esta manera es

conocido el esfuerzo necesario para desarrollar y mantener la clase. Sea C_i una clase, con n atributos $A_1, A_2, ..., A_n$, y sean $a_1, a_2, ..., a_n$ los valores de 0 - 1 asociados al grado de simplicidad de cada atributo, según la Tabla 3.

Nivel de Simplicidad De los Atributos	Significado
0	Listas/Vectores
0.25	Referencias
0.50	Registro/Estructura
0.75	Carácter
1	Entero, Real, Booleano

Tabla 3: Escala de valores para el criterio simplicidad de atributos considerando su estructura.

El valor de la simplicidad de atributos de la clase es dada por:

$$SIA(C_i) = \sum_{i=1}^{n} a_i \ / \ n$$

Un alto valor para ***SIA*** es un indicador de la existencia de una clase menos compleja, y probablemente mas fácil de ser entendida, modificada, extendida y reutilizada.

5. Conclusión

En este artículo fue discutido un conjunto de características de calidad propuesto para evaluar la confiabilidad de la representación de especificaciones orientadas a objetos, en lo que respecta al modelo de clases y su documentación asociada. Estos atributos fueron definidos a partir de trabajos sobre calidad de especificaciones, de la literatura técnica sobre calidad en orientación a objetos y del uso de métodos de desarrollo orientado a objetos. La organización y discusión de los atributos es realizada de acuerdo a un método de evaluación de la calidad de software.

6. Referencias Bibliográficas

[1] (Basili et al. 1996). V. Basili , L. Briand, W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators", En ***IEEE Transactions on Software Engineering***, vol 22, no.10, octubre, pp. 1-29.

[2] (Henderson-Sellers 1996). B. Henderson-Sellers, ***Object-Oriented Metrics - Measures of Complexity***, Prentice-Hall, Inc.

[3] (Jacobson et al. 1998). I. Jacobson, ***Unified Modeling Process***, Addison Wesley.

[4] (Larman 1998). Craig Larman, ***Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design,*** Prentice-Hall, Inc.

[5] (Clunie 1987). C. E. Clunie, ***Verificação e Validação de Software na Fase de Especificação de Requisitos***, Tesis de Maestría, COPPE/UFRJ.

[6] (Clunie et al. 1995a). C. E. Clunie, C. Werner, A. R. Rocha, "Metrics of Quality and their Relation with the Object-Oriented Modeling", En ***Proceedings of the 5th.International Symposium on Systems Research, Informatics and Cybernetics*** - *ISAS*, Baden-Baden, Alemania.

[7] (Clunie et al. 1995b). C. E. Clunie, R. M. Araujo, C. M. Werner, A. R. Rocha, G. H. Travassos, "Una Experiencia en el Desarrollo de un Sistema Orientado a

Objetos", En ***Proceedings of the II International Congress on Information Engineering - ICIE' 95***, Buenos Aires, Argentina.

[8] (Clunie et al. 1996). C. E. Clunie, C. M. Werner, A. R. Rocha, "How to Evaluate the Quality of Object-Oriented Specification", En ***Proceedings of the 6th. International Conference of Software Quality***, Canada, outubro, pp. 283-293.

[9] (Clunie e Rocha 1987), C. E. Clunie, A. R. Rocha, "Verificação e Validação de Software na Fase de Especificação de Requisitos" En ***XIII Conferencia Latinoamericana de Informática***, Bogotá - Colombia.

[10] (Davis 1995). A. Davis, "Object-Oriented Requirements to Object-Oriented Design: An Easy Transition?", En ***Journal Systems and Software***, volumen 30, julio- agosto, pp. 151-159.

[11] (Love 1993). T. Love, ***Object Lessons - Lessons Learned in Object-Oriented Development Projects***, SIGS Books, New York.

[12] (Kolewe 1993). R. Kolewe, "Metrics in Object-Oriented Design and Programming", En ***Software Development***, octubre, pp. 53-62.

[13] (Abreu e Carapuça 1994). F. B. Abreu, R. Carapuça , "Candidate Metrics for Object-Oriented Software within a Taxonomy Framework", En ***Journal of Systems and Software***, volumen 26, no.1, pp. 87- 96.

[14] (Coad e Yourdon 1994). Coad P., Yourdon E., ***Object-Oriented Analysis***, Yourdon Press, Englewood Cliff, New Jersey.

[15] (Chidamber y Kemerer 1994). S. Chidamber , C. Kemerer, "A Metrics Suite for Object-Oriented Design", En ***IEEE Transactions on Software Engineering***, 7(5), pp. 510-518.

[16] (Chen y Lu 1994). J-Y Chen, J-F Lu , "A New Metric for Object-Oriented Design", Em ***Information and Software Technology***, volumen 35, no.4, abril, pp. 232 - 239.

[17] (Stiliglic et al. 1995). B. Stiliglic, M. Hericko, R. Ivan, "How to Evaluate Object-Oriented Software Development?", Em ***ACM SIGPLAN Notices***, volume 30, no. 5, mayo, pp. 3-10.

[18] (Lorenz e Kidd 1994). M. Lorenz, J. Kidd, ***Object-Oriented Software Metrics***, PTR Prentice Hall, Englewood Cliff, New Jersey.

[19] (Rocha 1983). A. R. Rocha, ***Um Modelo para Avaliação da Qualidade de Especificações***, Tesis de Doctorado, PUC-RJ.

[20] (Belchior 1997). A. D. Belchior, ***Avaliação da Qualidade de Software utilizando Lógica Fuzzy***, Tesis de Doctorado, COPPE/UFRJ.

[21] (Clunie 1997). C. E. , ***Manual para el Control de Calidad de Especificaciones Orientadas a Objetos,*** Informe Técnico - No.1, UTP.

[22] (Bunge 1977). M. Bunge, ***Treatise on Basic Philosophy: Ontology I: The Furniture of the World***; Boston Riedel.

[23] (Jonhson y Foote 1998)., ***"Designing Reusable Classes"***; En Journal of Object Oriented Programming, vol. 1, No.2.

[24] (Yourdon, Edward 1994)., ***"Object - Oriented Design"***; Yourdon Press., Englewood Cliffs, NJ.