**Identificación y priorización de Quality Attributes en requerimientos**

Tesis entregada para el grado de

**Ingeniero de Sistemas**

en la Facultad de Ciencias Exactas



Por

**Francisco Bertoni**

**Sebastián Villanueva**

Bajo la supervisión de la

Dra. Claudia Marcos

Dr. Andrés Díaz Pace

**Universidad Nacional del Centro**

**de la Provincia de Buenos Aires**

Tandil, Argentina

Agosto 2010

Tabla de contenidos

[Capítulo I - Introducción 4](#_Toc271701327)

[Capítulo II - Dominio 5](#_Toc271701328)

[II.1. Arquitectura de Software 5](#_Toc271701329)

[II.1.1. Definición de Arquitectura de Software 5](#_Toc271701330)

[II.1.2. Representación de una arquitectura 7](#_Toc271701331)

[II.1.3. Ejemplo de una arquitectura 7](#_Toc271701332)

[II.2. Atributos de calidad 9](#_Toc271701333)

[II.2.1. Principales atributos de calidad 10](#_Toc271701334)

[II.2.1.1. Disponibilidad 10](#_Toc271701335)

[II.2.1.2. Performance 11](#_Toc271701336)

[II.2.1.3. Seguridad 11](#_Toc271701337)

[II.2.1.4. Usabilidad 11](#_Toc271701338)

[II.2.1.5. Modificabilidad 12](#_Toc271701339)

[II.2.1.6. Testeabilidad 12](#_Toc271701340)

[II.2.2. Atributos de calidad en una arquitectura 13](#_Toc271701341)

[II.3. Escenarios de calidad 13](#_Toc271701342)

[II.4. Aspectos tempranos y atributos de calidad 16](#_Toc271701343)

[II.5. Especificación de requerimientos 16](#_Toc271701344)

[II.5.1. Casos de uso 17](#_Toc271701345)

[II.5.2. Especificación de un requerimiento mediante casos de uso (o Aplicación de Casos de Uso) 17](#_Toc271701346)

[Capítulo III - Trabajos Relacionados 19](#_Toc271701347)

[Capítulo IV - Identificación de atributos de calidad 20](#_Toc271701348)

[IV.1. Técnica propuesta 20](#_Toc271701349)

[IV.2. Tokens Generation 21](#_Toc271701350)

[IV.2.1. Input Processor 22](#_Toc271701351)

[IV.2.2. Tokens Filter 24](#_Toc271701352)

[IV.2.2.1. Filtro Lower Case 26](#_Toc271701353)

[IV.2.2.2. Filtro Stop Words 26](#_Toc271701354)

[IV.2.2.3. Filtro Stemming 27](#_Toc271701355)

[IV.2.2.4. Filtro Ocurrencias 27](#_Toc271701356)

[IV.2.2.5. Filtro Pesos 28](#_Toc271701357)

[IV.3. Token Analysis 30](#_Toc271701358)

[IV.3.1. QAT Identification 30](#_Toc271701359)

[IV.3.1.1. Maps Identification 31](#_Toc271701360)

[IV.3.1.1.1. Descripción algorítmica 32](#_Toc271701361)

[IV.3.1.1.2. Descripción matemática 34](#_Toc271701362)

[IV.3.1.1.3. Ejemplo 35](#_Toc271701363)

[V.3.1.2 Maps Combination 38](#_Toc271701364)

[IV.3.2. Word-QA Association 39](#_Toc271701365)

[IV.3.2.1. Ontología 40](#_Toc271701366)

[IV.3.2.1.1. Componentes de una ontología 40](#_Toc271701367)

[IV.3.2.1.2. Ontología definida 41](#_Toc271701368)

[IV.3.2.2. Word-Instance Association. 43](#_Toc271701369)

[IV.3.2.3. Instance-QA Map Association 44](#_Toc271701370)

[ANEXO I 46](#_Toc271701371)

[ANEXO II 47](#_Toc271701372)

[Definición de “mapa” 47](#_Toc271701373)

[Suma de dos mapas 47](#_Toc271701374)

[División de un mapa por un número real 47](#_Toc271701375)

[Multiplicación de un mapa por un número real 47](#_Toc271701376)

[Bibliografía 48](#_Toc271701377)

# Introducción

*Acá iría el capítulo 1*

# Dominio

En este capítulo se presentará el dominio junto con los conceptos que se utilizarán a lo largo del presente trabajo. De esta forma, se introducirán los conceptos de “Arquitectura de software”, “Atributo de calidad”, “Escenario” y “Early Aspect” (Aspecto Temprano). Además, se introducirán algunas técnicas de especificaciones de requerimientos, centrándose en el concepto de “Caso de Usos” y su relación con los aspectos tempranos y atributos de calidad.

## Arquitectura de Software

Durante mucho tiempo los diseñadores de software han construido sistemas basados exclusivamente en requerimientos técnicos. De esta manera, el diseñador debía basar su diseño en estos requerimientos, para luego crear el sistema a partir del mismo. A pesar del feedback que podría haber entre el diseñador y el analista, existía implícitamente la idea que los requerimientos técnicos eran suficientes para construir un diseño satisfactorio.

Sin embargo, la *Arquitectura de Software* ha emergido como un eslabón crucial en el proceso de diseño, siendo un concepto fundacional para el desarrollo exitoso de largos y complejos sistemas de software. [1]

La vista arquitectónica de un sistema es abstracta, concentrándose en el comportamiento e interacción de “cajas negras”, prestando casi nula atención a detalles de implementación, algoritmos o representación de datos. La arquitectura de software es desarrollada como el primer paso hacia el diseño de un sistema que tiene un conjunto de propiedades deseadas.

### Definición de Arquitectura de Software

En la actualidad no hay ninguna definición de arquitectura de software que esté unánimemente respaldada por la totalidad de la comunidad. El número de definiciones circulantes, que aumenta con el tiempo, alcanza un orden de tres dígitos, amenazando llegar a cuatro. De hecho, existen grandes compilaciones de definiciones alternativas o contrapuestas, como la colección que se encuentra en el SEI (Software Engeenerning Institute), en donde cualquier persona puede agregar una [2]. En general, las definiciones entremezclan el trabajo dinámico de estipulación de la arquitectura dentro del proceso de ingeniería o el diseño (su lugar en el ciclo de vida), la configuración o topología estática de sistemas de software contemplada desde un elevado nivel de abstracción y la caracterización de la disciplina que se ocupa de uno de esos dos asuntos, o de ambos. [3].

Una definición reconocida es la de Clements [4]:

*La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema.*

Sin embargo se debe mencionar que, algunos años después, el mismo Clements, junto con Bass, Kazman y Northrop, definen a la arquitectura de software de la siguiente forma [5]:

*La arquitectura de software de un programa o sistema computarizado es la estructura o estructuras del sistema, que involucra elementos de software, las propiedades externas visibles de esos elementos.*

De esta manera, a pesar de la abundancia de definiciones del campo de la Arquitectura de Software, existe en general acuerdo de que ella se refiere a la estructura a grandes rasgos del sistema, estructura consistente en elementos, junto con sus propiedades y relaciones entre ellos. Estas cuestiones estructurales se vinculan con el diseño, pues la Arquitectura de Software es, después de todo, una forma de diseño de software que se manifiesta tempranamente en el proceso de creación de un sistema; pero este diseño ocurre a un nivel más abstracto que el de los algoritmos y las estructuras de datos. En este trabajo se tomará como referencia la última definición mostrada. Por ello vale la pena profundizar en algunas implicaciones de tal definición.

Lo primero es que la arquitectura define *elementos de software*. Es decir que la arquitectura incorpora información acerca de cómo esos elementos se relacionan entre ellos. Esto último implica que la arquitectura omite cierta información acerca de elementos que no pertenecen a la interacción. De esta manera se puede ver a la arquitectura como una abstracción de un sistema que suprime ciertos detalles que no afectan a cómo los elementos son usados, se relacionan o interactúan con otros elementos.

La segunda implicancia es que la definición deja bien en claro que los sistemas pueden comprometer a más de una estructura y que ninguna de estas puede adjudicarse ser “la arquitectura” del sistema.

Lo tercero es que la definición implica que todo sistema de software tiene una arquitectura de software, porque todo sistema puede ser descripto como un conjunto de componentes y relaciones entre estos componentes. En el caso más trivial, el sistema completo es un único elemento – si interés y probablemente sin utilidad alguna, pero, sin embargo, siendo una arquitectura –.

Finalmente, lo cuarto es que el comportamiento de cada componente es parte de la arquitectura en tanto y en cuento este comportamiento pueda ser observado o discernido desde el punto de vista de otro elemento. Ese comportamiento es lo que permite a los elementos interactuar entre ellos, lo que es parte de la arquitectura.

### Representación de una arquitectura

Una arquitectura puede ser representada de varias maneras. Cada una de estas maneras en que se representa la arquitectura, o una parte de ella, se denomina “vista”.

Dada la complejidad y el gran tamaño de los sistemas actuales es muy difícil obtener, de una sola vez, una representación arquitectónica completa del sistema en su totalidad. Es por ello que se restringe la atención o el análisis, en un momento dado, a un número pequeño de estructuras de un sistema de software.

De esta manera, una *vista* se define como una representación de un conjunto coherente de elementos arquitectónicos, en la manera en que fueron escrito para, y leídos por, los “stakeholders”. Consiste en una representación de un conjunto de elementos y las relaciones entre los mismos. [5].

Distintas vistas resaltan diferentes aspectos o propiedades de un sistema. De esta manera todas las vistas son necesarias para poder describirlo adecuadamente. Entre las más comunes se encuentran la vista de módulos, la de componentes y conectores o la de deployment.

En la primera vista los elementos de la arquitectura se representan por módulos, o unidades de implementación, y se describe la estructura estática del mismo, sin hacer mucho énfasis en cómo los módulos de software se manifiestan durante la ejecución. En la vista de componentes y conectores los elementos son componentes en ejecución y los conectores son las comunicaciones e interacciones entre los componentes. Finalmente, en la vista de deployment se muestra cómo se relacionan los elementos de software con los elementos en ambientes externos, en donde el software es ejecutado.

### Ejemplo de una arquitectura

A continuación se muestra un ejemplo de una porción de una arquitectura simple, utilizando una vista de componentes y conectores. El mismo es un ejemplo de una arquitectura de software para el problema de KWIC (Key Word in Contex).

En su artículo de 1972, Parnas propuso el siguiente problema [6]:

*El KWIC [Kew Word in Context] acepta un conjunto ordenado de líneas, donde cada línea es un conjunto ordenado de palabras, y cada palabra es un conjunto ordenado de caracteres. Toda línea sebe ser "desplazada circularmente'' varias veces, quitando la primera palabra y añadiéndola al final de la línea. El sistema de KWIC tiene como salida un listado, ordenado alfabéticamente, de todos los posibles desplazamientos circulares de todas las líneas.*

En la se muestra una arquitectura propuesta como solución para el problema citado anteriormente. La misma se describe en una vista de componentes y conectores, cuya solución se basa en el estilo de “Pipe and Filters”[7]. En este esquema, a los elementos se los denomina filtros (filters), mientras que a los conectores se los denominan tuberías (pipes).

Cada filtro realiza una tarea específica, procesando los datos de entrada y emitiéndolos por su salida. El control es distribuido: cada filtro puede procesar los datos apenas le llegan, independientemente de lo que estén haciendo los otros filtros. Éstos no comparten entre sí variables o estados internos. Cada filtro tiene una entrada, por donde le llegan datos, y una salida donde escribe datos producidos. La comunicación entre los filtros es restringida, sólo se realiza a través de las tuberías. Éstas conectan la salida de los filtros con las entradas de otros, transmitiendo los datos de una punta a la otra.



Figura II.1 Arquitectura propuesta para el problema de KWIC

En la arquitectura mostrada se muestran cuatros filtros: “Input”, “Circular Shifter”, “Alphabetizer” y “Output”.

El filtro “Input” parsea la entrada y escribe las líneas parseadas en su salida. Una tubería conecta la salida de este filtro con la entrada del filtro “Circular Shifter”. De esta forma, las líneas parseadas por el filtro “Input” sirven como entrada del segundo filtro. Éste procesa las líneas de entrada produciendo desplazamientos circulares sobre esas líneas. Las nuevas líneas desplazadas son escritas en la salida del filtro, en donde nuevamente se encuentra una tubería que conecta la salida del filtro con el filtro “Alphabetizer”. De esta manera, las líneas producidas por el filtro “Circular Shifter” sirven como entrada para el filtro “Alphabetizer”. Éste ordena las líneas alfabéticamente, escribiéndolas en su salida. Análogamente, una tubería conecta este filtro al filtro “Output”, que mediante otra tubería escribe los resultados finales en la salida estándar.

## Atributos de calidad

La calidad de software se define como el grado en el cual éste posee una combinación deseada de atributos, tales como: Performance, Disponibilidad, Portabilidad, Seguridad, Usabilidad, etc. [8]. Estos atributos son requerimientos adicionales del sistema [9] que hacen referencia a características o restricciones que éste debe satisfacer, y complementan los requerimientos funcionales del mismo. Estas características o atributos se conocen con el nombre de “atributos de calidad” [10].

La arquitectura de un sistema de software es determinante para que éste posea una combinación deseada o requerida de atributos de calidad. Ella es la que exhibe o inhibe los atributos de calidad de un sistema, pues una arquitectura representa las decisiones más tempranas del diseño, que impactarán de manera decisiva sobre todas las etapas de desarrollo posteriores. De esta manera es posible hacer predicciones sobre cuáles atributos de calidad, y cuáles no, poseerá el sistema de software con únicamente evaluando a la arquitectura.

En términos generales, Bass y otros autores [5] establecen una clasificación de los atributos de calidad en dos categorías:

* Observables en tiempo de ejecución: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución (disponibilidad, performance, seguridad, etc.)
* No observables en tiempo de ejecución: aquellos atributos que se establecen durante el desarrollo del sistema (modificabilidad, reusabilidad, portabilidad, escalabilidad, etc.).

La “funcionalidad” es la habilidad de un sistema para realizar el trabajo para el que fue pensado. Los atributos de calidad de un sistema son independientes de la funcionalidad del mismo [5]. Si esto no fuera así, una funcionalidad seleccionada dictaría los niveles de seguridad, modificabilidad, disponibilidad, etc. de un sistema. Esto no significa que cualquier nivel de cualquier atributo de calidad puede ser alcanzado para cualquier funcionalidad. Manipular gráficos complejos o realizar complejos cálculos matemáticos, por ejemplo, ciertamente tienen un impacto negativo sobre la performance. Sin embargo, lo que sí es posible es, bajo determinada funcionalidad, tomar ciertas decisiones que determinen un nivel relativo de calidad.

A pesar de estar muy relacionada con los atributos de calidad, en general es en la funcionalidad en donde se pone el foco de atención durante las distintas etapas de desarrollo, dejando a estos últimos, con mucha suerte, en un segundo plano. De esta manera la mayoría de los enfoques lidian con los atributos de calidad separadamente de la funcionalidad, dejando su integración para las etapas finales del ciclo de desarrollo [10]. Sin embargo, los atributos de calidad deben ser considerados en todas las etapas del proceso. Ningún atributo de calidad es enteramente dependiente de la etapa de diseño o de la implementación, o de cualquier otra [5].

Particularmente, la arquitectura es fundamental para la realización de varios atributos de calidad de un sistema. Es la primera etapa del ciclo de desarrollo en donde las estructuras deben ser diseñadas satisfaciendo a los atributos requeridos, y en donde estos pueden ser incluso testeados. Es por esta razón que el tener un conocimiento exacto de los atributos de calidad de un sistema en las etapas tempranas de desarrollo es fundamental para lograr un sistema exitoso [11].

Los sistemas no son generalmente rediseñados porque sean funcionalmente deficientes - de hecho los modulo reemplazados son, en general, funcionalmente idénticos - sino porque son difíciles de mantener, difíciles de escalar, demasiado lentos o se han visto comprometidos por hackers.

En este trabajo se denominará como un QAR (Quality Attribute Requirement) a un requerimiento de un atributo de calidad [12]. En la bibliografía los QARs también son conocidos como “Non functional requirements”, o NFRs. Sin embargo se adopta como convención la primera denominación.

### Principales atributos de calidad

A continuación se mostrará una descripción de algunos atributos de calidad que se mencionarán luego a lo largo del trabajo. Estos son disponibilidad (aviability), performance (performance), seguridad (security), usabilidad (usability), modificabilidad (modifiability) y testeabilidad (testeability).

#### Disponibilidad

La disponibilidad se relaciona con las fallas o interrupciones que pueden ocurrir en un sistema y sus consecuencias asociadas. Una falla del sistema ocurre cuando el mismo no puede brindar un servicio consistente con sus especificaciones. Dicha falla puede ser observada por usuarios del sistema - ya sean humanos u otros sistemas.

Algunas de las áreas concernientes son identificar cómo la falla del sistema es detectada, cuán frecuentemente ésta puede ocurrir, qué sucede cuando la falla ocurre, cuánto tiempo el sistema está fuera de operaciones luego de que ésta haya ocurrido, cuántas fallas pueden ser prevenidas o que clases de notificaciones se deben producir cuando una falla ocurre.

De esta manera, la disponibilidad está fuertemente relacionada con la confiabilidad, es decir, con la habilidad del sistema de mantenerse operativo todo el tiempo.

#### Performance

Performance se relaciona con el tiempo. Eventos (interrupciones, mensajes, pedidos de usuarios o simplemente el paso del tiempo) ocurren, y el sistema debe responder a ellos. Hay una gran variedad de caracterizaciones de eventos que arriban y sus respuestas, pero básicamente la performance concierne a cuánto tiempo le lleva al sistema responder cuando un evento ocurre.

La performance a nivel arquitectónico se puede entender, modelar y analizar, observando: la distribución y la tasa de llegada de los servicios de pedidos, los tiempos de procesamiento, el tamaño de las colas, y la latencia. Se puede simular a la performance de un sistema por la construcción de un modelo de encolado estocástico, basado en escenarios de carga de trabajo anticipado.

#### Seguridad

La seguridad es una medida de la capacidad del sistema para resistir el uso no autorizado sin dejar de ofrecer sus servicios a los usuarios legítimos. Un intento de violación de la seguridad se denomina “ataque” y puede adoptar diversas formas. Puede ser un intento no autorizado para acceder a datos o servicios o para modificar datos, o podría ser el intento de negar servicios a los usuarios legítimos.

Los ataques pueden ser de una amplia variedad, como el robo de dinero por transferencia electrónica, modificación de datos sensibles, robo de números de tarjetas de crédito, destrucción de los archivos en los sistemas informáticos o ataques de denegación de servicio ocasionados por gusanos o virus.

La seguridad puede ser alcanzada como son un sistema que ofrezca confidencialidad, integridad, fiabilidad, disponibilidad o auditoría, entre otros.

#### Usabilidad

La usabilidad concierne a la facilidad que se le brinda al usuario para completar una tarea deseada y al soporte de usuario que el sistema provee. Puede ser descompuesto en las siguientes áreas:

* Aprender las características del sistema.
* Usar el sistema eficientemente.
* Minimizar el impacto los errores.
* Adaptar el sistema a las necesidades del usuario.
* Aumentar la confianza y satisfacción del usuario.

#### Modificabilidad

La modificabilidad de un sistema es la habilidad para hacer cambios en forma rápida con un costo razonable. En general esta capacidad deriva directamente de la arquitectura: este atributo es, casi por completo, función de la localización de cada cambio. Hacer un cambio disperso en el sistema es, frecuentemente, más costoso que hacer un cambio en un solo componente y que los restantes permanezcan iguales.

A la regla anterior hay excepciones. Un solo componente que es excesivamente largo y complejo, puede ser más costoso de modificar que cinco más simples. Sin embargo también es fácil imaginar un cambio global, simple y sistemático: modificar el valor de una constante que aparece en todos lados, por ejemplo. De esta manera se considera como principio general que los cambios locales son los mejores. [5]

Dado que la arquitectura define los componentes y las responsabilidades de cada uno, también define las circunstancias bajo las cuales cada componente tendrá que cambiar. Una arquitectura clasifica todos los posibles cambios en tres categorías, de acuerdo a si el cambio se realizará sobre un solo componente, sobre más de un componente, o algo más drástico como el cambio del estilo de la arquitectura.

#### Testeabilidad

La testeabilidad del software se refiere a la facilidad en la cual éste puede ser sujeto a pruebas (generalmente basadas en la ejecución) para demostrar sus defectos. Al menos el 40% del coste de desarrollo de sistemas bien diseñados es absorbido por las pruebas. Si el arquitecto de software puede reducir este coste, la recompensa es significativa.

En particular, la capacidad de prueba se refiere a la probabilidad, en el supuesto de que el software tiene al menos un defecto, de que se produzca una falla en la ejecución de la siguiente prueba.

Para que un sistema sea testeable, debe ser posible controlar el estado y la entrada de cada componente interno y después observar su salida.

### Atributos de calidad en una arquitectura

Siguiendo con el ejemplo de la arquitectura mostrada en la Figura II‑1 se puede analizar en qué medida este diseño impacta en los atributos de calidad principales.

En primera medida se aprecia que ese diseño posee un alto nivel de modificabilidad. Cada filtro posee una funcionalidad específica, por lo que a la hora de realizar una modificación a la funcionalidad actual es sencilla el lugar indicado. Al ser cada módulo independiente del otro, el cambio en uno no afecta al resto. También es sencillo agregar, modificar o eliminar módulos sin afectar al resto de sistema.

El hecho de que dos o más módulos puedan realizar su tarea simultáneamente tiene un impacto positivo sobre la performance, al agregar concurrencia.

Sin embargo, como atributos de calidad en los cuales la arquitectura impacta negativamente se debe mencionar que el diseño dificulta la interacción con el usuario, resultando en un decremento del atributo de Usabilidad. También, al no tener, los filtros, estados internos, se debe transmitir toda la información de un filtro a otro. Si es un volumen grande de información, se produce un alto overhead, que impacta negativamente sobre la performance.

Como se ve, ningún atributo de calidad se logra en aislamiento. Generalmente, satisfacer uno de ellos tiene un impacto negativo sobre el otro.

## Escenarios de calidad

Los escenarios de calidad son una forma de representar, de una manera formal, los QARs de un sistema [11].

Un escenario consiste de seis partes [5] (Figura II.2) :



Figura II.2 Escenario de calidad

1. Fuente del estímulo (Concrete Source). Alguna entidad (un humano, un sistema de computación u otro actor) que genera un estímulo.
2. Estímulo (Concrete Stimulus). Una condición que necesita ser considerada cuando arriba al sistema.
3. Ambiente (Concrete Enviroment). El estímulo ocurre bajo ciertas condiciones. El sistema puede estar sobrecargado o puede estar ejecutándose normalmente cuando el estímulo ocurre, o alguna otra condición puede ser verdadera.
4. Artefacto (Concrete Artifact). Algún artefacto es estimulado. Este puede ser el sistema completo o alguna parte de él.
5. Respuesta (Concrete Response). La respuesta es una actividad llevada a cabo luego del arribo del estímulo.
6. Medida de respuesta (Concrete Response Measure). Cuando la respuesta ocurre, ésta debería ser medida de alguna manera así el requerimiento puede ser testeado.

Se distinguen como escenarios generales de calidad (escenarios generales) a aquellos que son independientes del sistema y pueden, potencialmente, pertenecer a cualquier sistema; de aquellos denominados escenarios concretos de calidad (escenarios concretos), que son específicos de un sistema en particular que están bajo consideración. La caracterización de atributos de calidad puede ser hecha como una colección de escenarios generales, sin embargo, para traducir la caracterización de los atributos a requerimientos para un sistema específico, los escenarios generales deben transformarse en específicos.

Un escenario general para el atributo de calidad de disponibilidad, por ejemplo, se describe en la Figura II‑3. Sus seis partes se muestran, indicando el conjunto de valores que pueden tomar. El escenario general se forma al elegir, para cada parte, uno de estos posibles valores.



Figura II.3 Conjunto de valores para escenarios generales del atributo disponibilidad

En la Figura II‑4 se muestra un ejemplo de un escenario concreto de disponibilidad, derivado del escenario general mostrado en Figura II‑3. Esto se logra haciendo a cada parte del escenario general específica del sistema en consideración. En este caso, el escenario representado es "An unanticipated external message is received by a process during normal operation. The process informs the operator of the receipt of the message and continues to operate with no downtime."



Figura II.4 Escenario concreto del atributo disponibiliad

Una colección de escenarios concretos puede ser usada como los QARs de un sistema. Cada escenario es suficientemente concreto para ser significativo para un arquitecto, y los detalles de la respuesta son suficientemente significativos para soportar tests convalidando que los requerimientos hayan sido cumplimentados.

En general, las definiciones provistas por un atributo de calidad no son operacionales. No tiene significado decir que un sistema será, por ejemplo, modificable, ya que todos los sistemas son modificables respecto a un conjunto de cambios y no modificable con respecto a otro conjunto. Los escenarios brindan una solución a esta cuestión al hacer posible su testeo a un nivel arquitectural.

## Aspectos tempranos y atributos de calidad

Se define un “concern” como “cualquier asunto de interés en un sistema de software” [13]. De esta manera el desarrollo de software tiene que tratar con un gran número de concerns. Algunos de estos están relacionados al producto (el software) que debe ser creado, como la funcionalidad y la performance. Otros concerns están relacionados con el proceso de desarrollo en sí mismo, como los tiempos y costos del desarrollo.

Mientras que (por medio de técnicas convencionales de modularización u orientación a objetos) un tipo de concern puede ser fácilmente encapsulado dentro de artefactos como módulos, clases y operaciones a nivel de diseño o implementación, esto mismo no es posible para otro tipo de concern. Estos cortan transversalmente (crosscut) el diseño o implementación de unos cuantos o incluso muchos artefactos y por lo tanto son llamados crosscutting concerns [13].

De esta manera, se define a un “aspecto temprano” como un concern que atraviesa el diseño de un sistema [14], y se manifiesta generalmente en las especificaciones de requerimientos u otros documentos preliminares producidos en el análisis de requerimientos. Muchos aspectos tempranos se corresponden con QARs de alto nivel como seguridad, performance, portabilidad y usabilidad [15]. En consecuencia, el descubrimiento de aspectos tempranos puede proporcionar pistas para identificar QARs.

## Especificación de requerimientos

Tanto los requerimientos funcionales como los requerimientos de atributos de calidad son capturados en un documento denominado “Especificación de Requerimientos de Software” (Software Requirements Specification), o SRS [16].

De esta manera el SRS es una completa descripción del comportamiento de un sistema a ser desarrollado. Principalmente, está conformado por un conjunto de casos de uso, que especifican la funcionalidad del sistema y como éste responde a distintos estímulos externos. Idealmente, el SRS también incluye un una especificación de los requerimientos de atributos de calidad, como así también un prototipo del sistema a ser desarrollado. Sin embargo, esto último rara vez se cumple.

En un estudio realizado [15] en base a 15 especificaciones requerimientos se encontró una importante escasez de menciones a QARs. Esto puede indicar que los desarrolladores fallan a la hora de analizar la importancia de los QARs, o que falsamente asumen que éstos son ya entendidos y aceptados por todos los stakeholders.

En general los documentos son organizados en cuanto a la funcionalidad, poniendo muy poco, y casi nulo, énfasis en los QARs. De esta manera, éstos quedan atravesados o esparcidos en varios documentos. Como ya se mencionó antes, esto conlleva a que los atributos de calidad no sean tenidos en cuenta en las primeras etapas del desarrollo, originando problemas potenciales sin detectar, diseños arquitectónicos que no cumplen con los requerimientos solicitados y productos de software que no satisfacen las necesidades del cliente.

Más aún la identificación de los QARs es una tarea poco sencilla y muy complicada, que puede consumir una gran cantidad de tiempo y esfuerzo. En general ésta es realizada “ad-hoc” entre varios documentos que describen al sistema. Como los QARs pueden estar esparcidos entre muchos de estos documentos, se corre el peligro de fácilmente malinterpretarlos o, directamente, ignorarlos por completo.

### Casos de uso

En ingeniería del software, un caso de uso es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software [17]. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario ó con otro sistema para conseguir un objetivo específico. Normalmente, en los casos de usos se evita el empleo de términos técnicos, prefiriendo en su lugar un lenguaje más cercano al usuario final. En ocasiones, se utiliza a usuarios sin experiencia junto a los analistas para el desarrollo de casos de uso.

Los casos de uso se convirtieron en una de las prácticas más comunes para la captura de requisitos funcionales, especialmente con el desarrollo del paradigma de la programación orientada a objetos. También, como se mencionó anteriormente, representan la espina dorsal de los SRS y es en donde generalmente se pone el mayor énfasis, dejando en un segundo plano la especificación de los requerimientos de los atributos de calidad.

### Especificación de un requerimiento mediante casos de uso (o Aplicación de Casos de Uso)

Los casos de uso evitan típicamente los términos técnicos, prefiriendo la lengua del usuario final o del experto del campo del saber al que se va a aplicar. Los casos del uso son a menudo elaborados en colaboración por los analistas de requerimientos y los clientes. Cada caso de uso se centra en describir cómo alcanzar una única meta o tarea de negocio.

Desde una perspectiva tradicional de la ingeniería de software, un caso de uso describe una característica del sistema. Para la mayoría de proyectos de software, esto significa que quizás a veces es necesario especificar diez o centenares de casos de uso para definir completamente el nuevo sistema. El grado de la formalidad de un proyecto particular del software y de la etapa del proyecto influenciará el nivel del detalle requerido en cada caso de uso. Los casos de uso pretenden ser herramientas simples para describir el comportamiento del software o de los sistemas. Un caso del uso contiene una descripción textual de todas las maneras que los actores previstos podrían trabajar con el software o el sistema. Los casos de uso no describen ninguna funcionalidad interna (oculta al exterior) del sistema, ni explican cómo se implementará. Simplemente muestran los pasos que el actor sigue para realizar una tarea.

Un caso de uso debe:

* tener un nivel apropiado del detalle;
* describir una tarea del negocio que sirva a una meta de negocio;
* ser bastante sencillo como que un desarrollador lo elabore en un único lanzamiento.

# Trabajos Relacionados

*Acá iría el capítulo 3*

# Identificación de atributos de calidad

En este capítulo se describe en detalle el enfoque elegido para la técnica de identificación de atributos de calidad en especificaciones de requerimientos. Esta técnica ofrece una asistencia extra al analista en la identificación de los atributos de calidad del sistema en etapas tempranas del desarrollo y, en consecuencia, facilita el tratamiento eficaz de los mismos.

## Técnica propuesta

En este trabajo se ha definido un proceso por medio del cual es posible identificar el grado de relación que tiene un conjunto de early aspects (aspectos tempranos) y casos de uso con los atributos de calidad del sistema, tal como se muestra en la Figura V.1. Este proceso utiliza como entrada una lista de casos de uso y early aspects, y agrega a cada conjunto <early aspect, casos de uso relacionados> información del atributo de calidad dominante para el conjunto en cuestión.



Figura IV.1 Diagrama de actividades principales

Básicamente, el proceso está constituido por dos etapas. La primera etapa, denominada *Tokens Generation* (Generación de Tokens), realiza un procesamiento sobre los artefactos de entrada (use cases y early aspects) para extraer de los mismos la lista de tokens que los conforman y luego, mediante un filtrado, establecer cuáles son relevantes para la etapa de análisis.

En la segunda etapa, *Tokens Analysis* (Análisis de Tokens), se calcula el porcentaje de pertenencia de cada elemento de entrada con los diferentes atributos de calidad del sistema. Posteriormente, esa información es combinada para obtener el atributo de calidad dominante para el conjunto de entrada. De esta manera la salida de la técnica es un *Quality Attribute Theme* (QAT). El mismo está formado por un conjunto de casos de uso relacionados con uno o varios aspectos tempranos. Adicionalmente, un QAT contiene un atributo de calidad al que esos casos de uso y los aspectos tempranos hacen referencia.

Para soportar dicho proceso se ha implementado una herramienta como una extensión - plugin - del entorno de desarrollo integrado (IDE) Eclipse [25]. La misma, permite al desarrollador seleccionar como entrada los early aspects que desea analizar, junto a los casos de uso que los relacionan. A partir del resultado obtenido a través de la ejecución de la herramienta, el desarrollador podrá contar con la información acerca de los atributos de calidad identificados, además de tener la posibilidad de visualizar la trazabilidad del proceso completo.

## Tokens Generation

El conjunto de actividades correspondientes a esta etapa, tiene como finalidad llevar a cabo un análisis léxico y sintáctico sobre los textos definidos como entrada para la técnica propuesta. Esta entrada, es una lista de casos de uso y los early aspects detectados sobre los mismos. Existen herramientas que generan esta información, o sobre las cuales se puede adaptar la salida para este propósito. Una de las herramientas utilizada para el desarrollo de este enfoque fue Aspect Extractor Tool [26], la cual, a través de un análisis semántico de los casos de uso, identifica los early aspects ocultos en la especificación [27]. La salida producida por esta herramienta es la lista de early aspects detectados, los casos de uso analizados y las relaciones entre ellos.

La especificación textual de los casos de uso respeta el estándar establecido por Rational [28]. De igual modo, la especificación de los early aspects sigue el formato definido en Aspect Extractor Tool, el cuál es semejante al de los casos de uso. Con toda esta información, mediante la técnica propuesta, se confeccionan las listas de tokens que serán suministradas a la próxima etapa.



Figura IV.2 Etapa 1 – Tokens Generation

Para el análisis de la entrada, se han definido dos actividades secuenciales. En primer lugar, un procesamiento de los datos de entrada para representarlos en un formato común. Mientras que en segundo lugar se requiere un filtrado sobre los mismos para identificar qué información es relevante para la próxima etapa. En estas actividades, se logrará determinar un subconjunto del conjunto de palabras que forman la especificación de los casos de uso y la definición de los early aspects, las cuales serán relevantes para la identificación del atributo de calidad involucrado.

A continuación se describen en detalle cada una de las actividades que se desarrollan durante esta etapa.

### Input Processor

El objetivo de esta actividad es procesar la información de entrada para establecerla en una representación interna uniforme. Este procesamiento es realizado a través de la separación de la información de las entidades de entrada (especificaciones de casos de uso y early aspects) en tokens. Se define un token como una unidad básica de texto que puede ser enriquecida con diferentes pares <atributos, valor>, como por ejemplo: <peso: 1>, <ocurrencias: 4>, etc. Estos tokens representan al conjunto de palabras de entrada.



Figura IV.3 Actividad Input Processor

El resultado de este procesamiento es una lista de tokens para las palabras obtenidas de los casos de uso y otra lista de tokens para las palabras del aspecto temprano en cuestión. Al realizar la división de la entrada en tokens, se agregan los siguientes atributos para cada uno de ellos:

* Id: identificador único del token.
* Tipo: este atributo representa el tipo de documento del cual se extrajo el token (en este punto será early aspect o use case).
* Sección: se registra la sección a la cual pertenece el token. Para los casos de uso, será alguna de las secciones establecidas por el template de Rational (nombre, descripción, flujo básico, actor, etc.) mientras que para el early aspect será alguna de las partes que conforman un aspecto temprano según lo establecido en la herramienta Aspect Extractor Tool[[1]](#footnote-1).

Por ejemplo, considérese la fracción de la especificación de un caso de uso mostrada en la Tabla V.1. Tomando ésta como entrada, el resultado de aplicar el procesamiento descripto anteriormente generaría como resultado la lista tokens mostrada en la Tabla V.2.

|  |  |
| --- | --- |
| **Nombre** | Register User |
| **Descripción** | Record user information in the system |

Tabla V.1: Fracción de caso de uso

|  |  |
| --- | --- |
| **Palabra** | **Atributos** |
| Register | <id,1>, <tipo, use case >,<sección,name> |
| User | <id,2>, <tipo, use case >,<sección, name > |
| Record | <id,3>, <tipo, use case >,<sección,description> |
| user | <id,4>, <tipo, use case >,<sección, description> |
| information | <id,5>, <tipo, use case >,<sección, description> |
| in | <id,6>, <tipo, use case >,<sección, description> |
| the | <id,7>, <tipo, use case >,<sección, description> |
| system | <id,8>, <tipo, use case >,<sección, description> |

Tabla V‑2: Tokens generados luego del análisis de la entrada

### Tokens Filter

Como se mencionó en la sección anterior, el filtrado de los tokens es otra de las tareas que se realiza sobre la información de entrada. En este caso, se ejecutan una serie de filtros sobre los tokens de las listas de casos de uso y de early aspects.



Figura IV.4 Actividad Tokens Filter

Se considera que un filtro es una unidad de procesamiento que realiza una modificación (enriquece, refina o transforma) sobre los datos de entrada y los copia a la salida para que otro filtro trabaje sobre los mismos datos. Mediante los filtros se pueden realizar transformaciones independientes sobre el flujo de datos.

El patrón de arquitectura Pipes & Filters [7] provee una estructura para procesar flujos de datos. Cada paso de procesamiento se encapsula en un filtro y es independiente del resto. Los datos se transmiten usando los pipes entre filtros adyacentes y mediante la combinación de éstos últimos se pueden conseguir diferentes salidas.

En este caso, la entrada para los filtros es una lista de tokens, por lo que el filtro ejecutará acciones sobre los tokens de la lista (modificar la palabra, agregar atributos), para luego devolver la lista de tokens modificada.

Para esta actividad, hemos definido una serie de filtros que realizan transformaciones sobre los tokens de las listas. Los mismos fueron definidos de forma tal que puedan ser reutilizados y combinados para darle un formato común a cualquier otro texto que represente información relevante durante el proceso.

#### Filtro Lower Case

Este filtro es el encargado de pasar todos los caracteres de las palabras de los tokens a minúscula.

Continuando con los tokens generados en la Tabla V.2, sólo se modifican los caracteres en mayúscula de las palabras de los tokens. En la Tabla V.3 se observa la lista de tokens luego de la transformación aplicada por este filtro.

|  |  |
| --- | --- |
| **Palabra** | **Atributos** |
| register | <id,1>, <tipo, use case >,<sección,name> |
| user | <id,2>, <tipo, use case >,<sección, name > |
| record | <id,3>, <tipo, use case >,<sección,description> |
| user | <id,4>, <tipo, use case >,<sección, description> |
| information | <id,5>, <tipo, use case >,<sección, description> |
| in | <id,6>, <tipo, use case >,<sección, description> |
| the | <id,7>, <tipo, use case >,<sección, description> |
| system | <id,8>, <tipo, use case >,<sección, description> |

Tabla V‑3: Estado de los tokens luego del filtro LowerCase

#### Filtro Stop Words

Este filtro elimina las denominadas stop-words, que son palabras que, desde el punto de vista no lingüístico, no contienen información relevante. Algunas de las stop words más comunes son los artículos, las preposiciones, etc. (por ejemplo: a, an, in, the, he, she, them).

Por lo tanto, este filtro elimina de la lista de tokens aquellos en los cuales su palabra aparezca en la lista de stop-words dada. El analista/desarrollador tiene la posibilidad de incluir en esta lista las palabras, que para el dominio en el cuál este analizando, no representen información valiosa para el análisis. De igual modo, se puede modificar la lista quitando aquellas palabras que en el domino resulten relevantes y el desarrollador desee que sean procesadas.

La Tabla V.4 muestra el estado de los tokens de la Tabla V.3 luego de haberse aplicado este filtro.

|  |  |
| --- | --- |
| **Palabra** | **Atributos** |
| register | <id,1>, <tipo, use case >,<sección,name> |
| user | <id,2>, <tipo, use case >,<sección, name > |
| record | <id,3>, <tipo, use case >,<sección,description> |
| user | <id,4>, <tipo, use case >,<sección, description> |
| information | <id,5>, <tipo, use case >,<sección, description> |
| system | <id,8>, <tipo, use case >,<sección, description> |

Tabla V‑4: Lista de tokens pos filtro Stop Words

#### Filtro Stemming

Diferentes palabras pueden tener el mismo significado desde el punto de vista semántico, pero no tener exactamente la misma secuencia de caracteres. Un ejemplo de esto pueden ser las palabras aprender, aprenden y aprendió, que, desde el enfoque en este trabajo, tienen el mismo significado. Por ello se decidió utilizar la técnica stemming [29], que es el proceso de transformar una palabra en su raíz (stem). Para el ejemplo anterior, todas esas palabras estarían identificadas por su raíz, es decir, “aprend”.

Para realizar el stemming se utiliza uno de los algoritmos más populares y conocidos, el algoritmo de Porter [30].

Continuando con el ejemplo observado en la Tabla V.4, se modifican las palabras de los tokens, llevando cada una de ellas a su raíz. En la Tabla V.5 se muestra el estado de los tokens luego de que se aplicó el filtro de stemming.

|  |  |
| --- | --- |
| **Palabra** | **Atributos** |
| regist | <id,1>, <tipo, use case >,<sección,name> |
| user | <id,2>, <tipo, use case >,<sección, name > |
| record | <id,3>, <tipo, use case >,<sección,description> |
| user | <id,4>, <tipo, use case >,<sección, description> |
| inform | <id,5>, <tipo, use case >,<sección, description> |
| system | <id,8>, <tipo, use case >,<sección, description> |

Tabla V‑5: Estado de los tokens luego del filtro Stemming

#### Filtro Ocurrencias

Este filtro elimina tokens duplicados en la lista de entrada y enriquece cada token con el número de ocurrencias. Se agrega el siguiente atributo al token:

* Ocurrencias: número de ocurrencias del token en la lista. Para los tokens obtenidos de los casos de uso se considera un token duplicado en caso de que el token sea la misma palabra y además aparezca en la misma sección. Esta particularidad se debe a que luego los tokens son ponderados y debe respetarse la sección en la cual aparece a la hora de asignarle su peso. Por lo tanto, existirán tokens con la misma palabra pero diferente sección.

Igualmente, para continuar con el ejemplo, por simplicidad sólo se tendrá en cuenta la palabra del token para que se considere repetido. De esta forma, se observa en la Tabla V.6 el nuevo atributo que se agrega a cada token.

|  |  |
| --- | --- |
| **Palabra** | **Atributos** |
| regist | <id,1>, <tipo, use case >,<sección, name>, <ocurrencias, 1> |
| user | <id,2>, <tipo, use case >,<sección, name>, <ocurrencias, 2> |
| record | <id,3>, <tipo, use case >,<sección,description>, <ocurrencias, 1> |
| inform | <id,4>, <tipo, use case >,<sección, description>, <ocurrencias, 1> |
| system | <id,5>, <tipo, use case >,<sección, description>, <ocurrencias, 1> |

Tabla V‑6: Lista de tokens luego de aplicar el filtro Ocurrencias

#### Filtro Pesos

Este filtro es aplicado para enriquecer los tokens de la lista de tokens de casos de uso, asignando un peso a cada token. Este peso depende de la sección donde aparece la palabra. En la técnica propuesta las palabras tienen distinta importancia según la sección del caso de uso en la que aparezcan. Por ejemplo, se asigna un peso alto a una palabra perteneciente al nombre o descripción del caso de uso, mientras que se asigna un peso más bajo a una palabra perteneciente a un flujo alternativo.

Hemos realizado un estudio riguroso de la importancia que tiene cada una de las componentes del template que conforman los casos de uso para nuestra problemática, y por defecto se asignan los pesos mostrados en la Tabla V.7 de acuerdo a la importancia de los textos y palabras que conforman las diferentes secciones. Sin embargo, el desarrollador podría modificar este peso dependiendo de las características de la definición de sus casos de uso. La herramienta soporta esta parametrización a través de un archivo de configuración donde se establecen estos pesos.

Considerando los pesos por defecto mostrados en la Tabla V.7, la lista de tokens final quedaría tal como se muestra en la Tabla V.8.

|  |  |
| --- | --- |
| **Sección** | **Peso** |
| <Nombre> | 5 |
| <Descripción>,<Requerimientos especiales> | 4 |
| <Flujo Básico> | 3 |
| <Flujo Alternativo> | 2 |
| <Precondiciones>,<Postcondiciones>,<trigger>,<prioridad>,<actor> | 1 |

Tabla V‑7: Ponderación de las secciones de los casos de uso

|  |  |
| --- | --- |
| **Palabra** | **Atributos** |
| regist | <id,1>, <tipo, use case >,<sección, name>, <ocurrencias, 1>,<peso, 5> |
| user | <id,2>, <tipo, use case >,<sección, name>, <ocurrencias, 2>,<peso, 5> |
| record | <id,3>, <tipo, use case >,<sección,description>, <ocurrencias, 1>,<peso, 4> |
| inform | <id,4>, <tipo, use case >,<sección, description>, <ocurrencias, 1>,<peso, 4> |
| system | <id,5>, <tipo, use case >,<sección, description>, <ocurrencias, 1>,<peso, 4> |

Tabla V‑8: Estado final de los tokens, luego del filtro Pesos

Para los early aspects no es necesario aplicar el filtro de peso debido a que se considera que todas las palabras dentro del aspecto temprano tienen la misma importancia, mientras que el caso de uso está dividido en secciones, y se considera que hay secciones que deben ser ponderadas sobre otras.

## Token Analysis

En la etapa *Análisis de Tokens* se encuentran las tareas requeridas para identificar los atributos de calidad a partir de las listas de tokens extraídos de la información de entrada. En este punto, la entrada ha sido analizada y se han construido dos listas de tokens. Además, la palabra de cada token ha sido pasada a minúscula, llevada a su raíz (stem), se ha ponderizado según su importancia en el contexto y se ha dejado constancia del número de ocurrencias. De esta manera, a esta etapa llegan dos listas de tokens, una correspondiente a los casos de uso y otra correspondiente al early aspect.



Figura .. Etapa Token Analysis

La etapa *Token Analysis* (Figura V.5.) se compone de dos actividades principales *QAT Identification* y *Tokens-QA Association*. La actividad *QAT Identification* se nutre de la información suministrada por la actividad *Tokens-QA Association*, que a su vez se nutre de información de una ontología (Ontology) para su ejecución.

El objetivo de esta etapa es identificar el atributo de calidad al que las listas de entrada hacen referencia, para así, finalmente, formar el Quality Attribute Theme. Adicionalmente, en el proceso de identificación del atributo de calidad dominante se podrán visualizar posibles trade-offs con otros atributos de calidad.

### QAT Identification

Esta actividad tiene como objetivo asignar un conjunto de atributos de calidad, con un porcentaje asociado a cada uno, al conjunto de la entrada suministrada por la etapa *Generación de Tokens* (Figura V.6). Como se mencionó anteriormente, la entrada de esta actividad son dos listas de tokens: una extraída de los casos de uso y otra del aspecto temprano. Cada uno de estos tokens posee diferentes atributos, como sección, peso, ocurrencias, etc. que serán determinantes en la identificación del QAT.



Figura .. Actividad QAT Identification

La actividad “Identificar Quality Attribute Theme” está compuesta por dos sub-actividades: *Maps Identification* y *Maps Combination*. La primera calcula un mapa a partir de cada una de las listas de entrada, mientras que la segunda combina estos dos mapas en un mapa final. Ambas sub-actividades se detallan a continuación.

Para poder llevar a cabo esta actividad es necesario obtener información mediante la invocación a la actividad *Tokens-QA Association (*Figura V.5*)*. Esta última actividad relaciona un token con un conjunto de pares <atributo de calidad, porcentaje>, indicando, para los distintos atributos de calidad, el porcentaje de asociación de ese token con el mismo. A este conjunto de pares se lo ha denominado con el término “mapa” y además se han definido las operaciones de *suma de dos mapas*, *división de un mapa por un número real* y *multiplicación de un mapa por un número real. [[2]](#footnote-2)*

#### Maps Identification

Esta sub-actividad tiene como objetivo calcular un mapa de atributos de calidad y porcentajes para cada una de las listas de entrada (Use Case Map y Early Aspect Map). Estas listas son la lista de tokens extraídos de los casos de uso y la lista de tokens extraídos de los aspectos tempranos.



Figura . Sub-actividad Maps Identification

A continuación se muestra el algoritmo que sigue esta sub-actividad para calcular el mapa a partir de los tokens extraídos de una lista. Luego se muestra el mismo razonamiento, pero expresado en forma matemática y definiendo ecuaciones formales. Finalmente, se presenta un ejemplo del funcionamiento de esta sub-actividad.

##### Descripción algorítmica

La idea subyacente de esta sub-actividad es realizar una iteración sobre cada uno de los tokens de la lista. Por cada uno de los mismos se calcula el porcentaje de asociación del atributo “palabra” con los distintos atributos de calidad. Los resultados parciales de cada token se van acumulando y, cuando finalmente concluye la iteración, se calcula el promedio entre de todos los tokens.

Para un mejor entendimiento, a continuación se muestra un fragmento de pseudocódigo correspondiente a la sub-actividad “Maps Identification”.

1: Mapa <QA,porcentaje> porcentaje\_total\_map = crear mapa con todos los atributos de calidad e igualar porcentajes a 0;

2: cantidad\_de\_palabras = 0;

3: por c/ token de la lista de tokens

4: *inicio loop*

5: palabra = token.palabra

6: Mapa<QA,porcentaje> porcentaje\_palabra\_map = word-qa association(palabra);

7: n = token.ocurrencias \* token.peso;

8: porcentaje\_palabra\_map = porcentaje\_palabra\_map \* n;

9: cantidad\_de\_palabras = cantidad\_de\_palabras + n

10: porcentaje\_total\_map = porcentaje\_total\_map + porcentaje\_palabra\_map

11: *final loop*

12: porcentaje\_total\_map = porcentaje\_total / cantidad\_de\_palabras

En la primera línea, se declara un mapa de pares <atributo de calidad, porcentaje>. A este mapa se le agregan como pares todos los nombres de los atributos de calidad con los que se trabajará (performance, disponibilidad, modificabilidad, etc.), con los porcentajes de cada uno inicializados en 0.

En la línea 2, se declara la variable *cantidad\_de\_palabras* en 0. Esta variable cuenta la cantidad de palabras analizadas de los tokens de la lista. Esto último es cierto siempre y cuando el número de ocurrencias de un token y su peso sean igual a 1. En caso contrario, esta variable aumenta en la magnitud del producto entre el número de ocurrencias del token y el peso del mismo (valor de la variable n). Esta información se extrae de los atributos “peso” y “ocurrencias” pertenecientes a cada token.

En la línea 3, se inicializa una iteración por cada uno de los tokens de una lista de tokens. Por cada uno de éstos se recupera el atributo “palabra”, como se muestra en la línea 5, y se la asocia un conjunto de pares <atributo de calidad, porcentaje>, como se observa en la línea 6. Como se mencionó anteriormente, esta asociación entre la “palabra” del token y un conjunto de atributos de calidad es realizada por la tarea “Word-QA Association”, en adelante denominada *WQA*, que toma como entrada un token (aunque solamente utiliza la palabra del mismo).

En la línea 7 se declara la variable n, que es un número que representa el producto entre los atributos “peso” y “ocurrencias” que posee el token analizado en esa iteración.

En la línea 8 cada porcentaje de cada atributo de calidad del mapa al que se le asocia una palabra se multiplica por el valor de n, que, como se mencionó anteriormente, representa el número de ocurrencias de ese token y el peso del mismo. Se supone que el número de ocurrencias es mayor o igual a uno (si no fuera así el token no aparecería en la lista). El peso también se supone mayor a cero, aunque un peso igual a cero no afectaría al algoritmo, ya que en este caso se interpretaría como que el token se encontraba en una sección que no debe ser tenida en cuenta para el análisis.

Los porcentajes de cada palabra, para cada atributo de calidad, se suman y almacenan en el mapa *porcentaje\_total*.

La línea 9 indica que la cantidad de palabras analizadas se va almacenando en la variable *cantidad\_de\_palabras.* Ya que cada valor del mapa es multiplicado n veces, la cantidad de palabras no aumenta necesariamente en una unidad por cada iteración, sino que aumenta n unidades. Esto se reproduce en la línea 9.

En la línea 10 se suman los resultados parciales de porcentaje\_palabra\_map a los totales acumulados en porcentaje\_total\_map.

Una vez terminadas las iteraciones, en la línea 12 se divide cada porcentaje de cada atributo de calidad del mapa *porcentaje\_total\_map* por el valor de la variable cantidad\_de\_palabras.

##### Descripción matemática

La metodología de la sub-actividad *Maps Identification* también puede ser detallada matemáticamente. De esta manera, sea UCL[] un arreglo compuesto por tokens extraídos, por ejemplo, de los casos de uso.

Como se ha mencionado, cada token tiene varias propiedades, y en las siguientes ecuaciones se denomina como *token*.palabra al conjunto de caracteres del token que representan la palabra, *token*.ocurrencias a la propiedad que indica el número de ocurrencias de ese token y *token*.peso al valor o peso que se le asigna al mismo.

Además, sea i un número entero, UCL[i] se refiere al token en la posición i del arreglo UCL. Por último, se define como “WQA” las siglas que denominan a la tarea “Word-QA Association”. Entonces se calcula el mapa porcentaje\_total\_map de la siguiente manera:

Esta ecuación muestra lo mismo que el fragmento de pseudocódigo explicado en la subsección anterior. Por cada palabra de cada token se invoca la actividad WQA y los resultados obtenidos son multiplicados por el peso y las ocurrencias del token. A su vez el producto entre estos dos valores se va sumando para cada token, como se muestra en el dividendo de la ecuación.

Para la lista de tokens extraídos de los aspectos tempranos se modifica ligeramente la ecuación anterior. En este caso, el peso del token siempre es igual a 1, ya que, como se explico anteriormente, el peso de los tokens sólo se aplica a los extraídos de los casos de uso y depende de la sección en donde se encuentren. De esta manera, para el caso de la lista de tokens extraídos de los aspectos tempranos, la ecuación anterior se puede ver como:

##### Ejemplo

A continuación se verá un ejemplo de lo explicado anteriormente. En el mismo se supone una lista de sólo dos tokens extraídos de los casos de uso (Tabla V‑9 ). El primero es un token con la palabra “fast”, tiene un peso de 1 y un número de ocurrencias también de 1. El segundo token de la lista posee la palabra “second”, con un valor de 2 para el número de ocurrencias y un valor de 3 para el peso.

|  |  |
| --- | --- |
| Palabra | Atributos |
| Fast | <id,1>, <tipo, use case>,<sección,description>,<ocurrencias,1>,<peso,1> |
| Second | <id,2>, <tipo, use case >,<sección, name>, <ocurrencias,2>, <peso,3> |

Tabla V‑9 Lista de dos tokens

A fin de que se aprecie mejor el ejemplo, las palabras de los tokens aparecen sin la transformación generada por el filtro de stemming.

Supóngase que se tiene sólo los atributos de calidad Modifiability, Performance y Aviability. Considerando el pseudocódigo presentado en la subsección V.3.1.1.1, la línea 1 inicializa un mapa con estos tres atributos en los pares, cada uno con valores en 0. En la línea 2 se inicializa el contador de palabras en 0. De esta manera, después de las inicializaciones, las variables quedan en el siguiente estado:

*porcentaje\_total\_map = (<Modifiability,0>; <Aviability,0>; <Performance,0>).*

*cantidad\_de\_palabras = 0;*

En este punto comienza la iteración sobre los tokens de de lista. El primer token contiene la palabra “fast”. Supóngase que la actividad *WQA* relaciona, para esa palabra, el siguiente mapa:

*WQA(“fast”) = (<Modifiability,0.25>; <Aviability,0.0>; <Performance,0.75>)*

Este resultado significa que la actividad asigna a la palabra “fast” una relación de 0.25 o 25% con el atributo de calidad “Modifiability”, un 0% con “Aviability” y un 75% con el atributo “Performance”.

La línea 7 indica que la variable n es igual al producto entre el peso del token multiplicado por el número de ocurrencias. Por lo que el valor de n es igual a 1 en la primera iteración.

Los valores de *porcentaje\_total\_map* no varían luego del producto de este por n, y la variable cantidad\_de\_palabras aumenta en 1, ya que este es el valor de n.

De esta manera, el estado de las variables luego de la primera iteración es el siguiente:

*porcentaje\_total\_map = (<Modifiability,0.25>;<Aviability,0.0>;<Performance,0.75>).*

*cantidad\_de\_palabras = 1;*

*cantidad\_de\_palabras = 0;*

Distinto es el caso del segundo token, que posee la palabra “second”, con un peso de 3 y un número de ocurrencias igual a 2.

Supóngase que en este caso el mapa asociado para la palabra posee un valor de 0.25 para Modifiability, 0.50 para Aviability y 0.25 para Performance. Estos valores deben ser multiplicados por n=6 (ya que este es el valor del producto entre el peso y el número de ocurrencias). Además la cantidad de palabras también aumenta 6 unidades durante la segunda iteración. De esta manera, el estado de las variables luego de la segunda iteración es el siguiente:

*porcentaje\_total\_map = (<Modifiability,1.75>;<Aviability,3.0>;<Performance,2.25>).*

*cantidad\_de\_palabras = 7;*

*cantidad\_de\_palabras = 0;*

Luego de esta iteración no hay más tokens, por los que los valores del mapa *porcentaje\_total\_map* deben ser divididos por *cantidad\_de\_palabras*, dando como resultado el siguiente mapa:

*porcentaje\_total\_map = (<Modifiability,0.25>;<Aviability,0.43>;<Performance,0.32>)*

La interpretación de este resultado es que la lista de tokens se relaciona en un 25% con Modifiability, un 43% con Aviability y un 32 % con Performance.

Como se puede notar, el peso y el numero de ocurrencias del segundo token, que en su conjunto son seis veces mayores que el primero, hacen inclinar la balanza hacia Aviability, que es el atributo de calidad con el que más se relaciona la palabra “second”.

#### V.3.1.2 Maps Combination

En la sub-actividad precedente se calcularon dos mapas, uno proveniente de la lista de tokens extraídos de los casos de uso y otro a partir de la lista de tokens extraídos de los aspectos tempranos. De esta forma, la función de esta sub-actividad es combinar estos dos mapas para formar el mapa resultante que será parte del QAT.



Figura .. Sub-actividad Maps Combination

La solución propuesta en este trabajo combina los resultados parciales de ambos mapas a partir de un factor k. Este factor es un número real entre 0 y 1 e indica la importancia que se le debe dar a cada uno de los mapas de entrada. A su vez, la herramienta desarrollada en este trabajo permite que el desarrollador sea capaz de ingresar este valor, según sus preferencias.

Cada valor de cada atributo de calidad del mapa de la lista de tokens del caso de uso (use cases map) se multiplica por k, mientras que cada valor de la lista del mapa de tokens de los aspectos tempranos (early aspect map) se multiplica por 1-k. Luego, para cada atributo de calidad de cada mapa, se suman los valores y se almacenan en el mapa resultante.

Supóngase, por ejemplo, que el mapa A= (<Modificability,0.25>; <Aviability,0.25>; <Performance,0.50>) es el relacionado con la lista de los casos de uso y el mapa B= (<Modificability,0.10>; <Aviability,0.30>; <Performance,0.60>) es el relacionado con los aspectos tempranos. Eligiendo un valor de k=0.4 se tendría que multiplicar cada valor de A por 0.4 y cada valor de B por 1 - 0.4 = 0.6, para luego sumar, para cada atributo de calidad, los valores de ambas listas. Estos daría como resultado un mapa C=(<Modificability,0.16>; <Aviability,0.28>; <Performance,0.56>).

Nótese que eligiendo un k=1 el mapa resultante es el mismo que el mapa que se relaciona con los casos de uso (en el caso anterior el mapa resultante, C, sería igual al A). Sin embargo, con un k=0, el mapa resultante es el mismo que el mapa identificado a partir de la lista de tokens de los aspectos tempranos. Un valor de k=0,5 implica que cada mapa aporta la misma proporción al mapa resultante.

Para describir está sub-actividad de forma matemática supóngase a useCaseMap como el mapa calculado a partir de los casos de uso y earlyAspectMap como el mapa calculado a partir de los aspectos tempranos. Entonces el mapa resultante, totalMap, es igual a

en donde k es un numero real entre 0 y 1.

### Word-QA Association

Esta actividad relaciona una secuencia de caracteres que representa una palabra con un mapa de atributos de calidad y porcentajes. Cada porcentaje indica el grado de asociación o pertenencia que tiene esa palabra con cada atributo de calidad.



Figura .. Sub-actividad Word-QA Association

Básicamente, esta actividad consta de dos sub-actividades. La primera es averiguar a qué parte del escenario corresponde la palabra. Es decir, descubrir si corresponde a una “Fuente de estímulo”, a un “Estímulo” o a alguna otra. Este comportamiento esta agrupado en la sub-actividad *Word-Instance Association*. Una vez reconocida la instancia a la que se corresponde la palabra, se averigua cuántos y cuáles escenarios se relacionan con esa instancia. A partir de esto último, y de calcular los porcentajes en que ese grupo de escenarios se relacionan con los atributos de calidad, se obtiene un mapa de atributos de calidad y porcentajes. Este comportamiento se agrupa en la sub-actividad *Instance-QA Map Association*.

Ambas sub-actividades consultan las instancias de una ontología definida para el dominio en cuestión. Esta ontología se considera que ha sido modelada por un experto. Es por ello que a continuación ésta es detallada primero, para luego describir las sub-actividades que componen a esta actividad.

#### Ontología

La literatura contiene varias definiciones de ontologías, muchas veces contradictorias entre ellas. En este trabajo se acuerda en definir a una ontología de la siguiente manera:

*“Una ontología es un modelo de datos que describe conceptos (también llamados clases) en un dominio del discurso, propiedades de cada concepto que describen las diversas características y atributos del concepto, y restricciones sobre esas propiedades.”* [18]*.*

Un dominio es un área de temática específica o de conocimiento, tal como medicina, fabricación de herramientas, bienes inmuebles, reparación automovilística, gestión financiera, etc. [19]. Las ontologías incluyen definiciones de conceptos básicos del dominio, y las relaciones entre ellos. También codifican el conocimiento de un dominio específico y el conocimiento que extiende de los dominios.

Un dominio específico es la parte del mundo que se quiere modelar. Representa el significado aplicado a los términos usados en la construcción de la ontología. Una ontología es la descripción de los conceptos que forman parte del dominio según un punto de vista. Un sistema sólo conoce lo que puede representar en algún lenguaje, por lo tanto, todo lo que no se exprese en la ontología no será conocido para el sistema que use la ontología.

##### Componentes de una ontología

Las ontologías cuentan con los siguientes componentes que sirven para representar el conocimiento de algún dominio [20]. Los principales son:

1. Conceptos o Clases Generalmente el foco de cualquier ontología, las clases son las ideas básicas que se intentan formalizar. Por ejemplo, en una ontología de vinos, la clase “Vino” representa todos los vinos. Una clase también puede tener sub-clases que representan conceptos que son más específicos que las superclases. Por ejemplo, subclases de la clase “Vino” podrían ser “Tinto”, “Blanco”, “Rosado”, etc.
2. Instancias Las instancias son representaciones de objetos determinados de un concepto. Siguiendo el ejemplo anterior, vinos específicos son instancias de la clase “Vino”. Una botella de New Age que se vende en el supermercado es una instancia de la clase “Vino”.
3. Propiedades o Slots Los slots describen propiedades de clases o instancias. En el ejemplo anterior “New Age” es la marca del vino. Por lo tanto, esa instancia posee un slot (propiedad), que poseen todas las instancias de la clase “Vino”, que podría denominarse “marca”, y cuyo valor es la cadena de caracteres “New Age”. El valor de una propiedad puede ser una cadena de caracteres, un número, otra instancia, etc. En el caso de que fuera otra instancia se forma lo que se denomina una relación.

##### Ontología definida

En la Figura IV.1 se muestra un diagrama con la ontología definida, que servirá de soporte de las sub-actividades siguientes. Su construcción se basa principalmente en conceptos, definiciones, y relaciones extraídas del libro “Software Architecture in Practice” [5] y explicadas en el Capítulo II.



Como se puede apreciar en la ontología, en ella aparecen los conceptos de *Quality-Attribute Scenario*, que representa a los escenarios de calidad, y *Concrete Quality-Attribute Scenario* y *General Quality-Attribute Scenario* que representan a los escenarios concretos y generales, respectivamente.

Los atributos de calidad se representan mediante el concepto *Qualitty Attribute* y se relacionan con el concepto de *Quality-Attribute Scenario* mediante la propiedad *isSpecificOf*.

A su vez las partes de un escenario se ven reflejadas en los conceptos *Concrete Source*, *Concrete Stimulus*, *Concrete Enviroment*, *Concrete Artifact*, *Concrete Response* y *Concrete Response Measure*. Todos estos son conceptos hijos del concepto *Concrete Scenerio Part* y se relacionan con el concepto de *Quality-Attribute Scenario*.

#### Word-Instance Association.

En esta sub-actividad se busca identificar la correspondencia entre una palabra y alguna instancia de los conceptos de ConcreteSource (Fuente), ConcreteStimulus (Estímulo), ConcreteEnviroment (Ambiente), ConcreteArtifact (Artefacto), ConcreteResponse (Respuesta) y ConcreteResponseMeasure (Medida de Respuesta) de la ontología. Básicamente, a partir de una palabra dada se intenta responder a la pregunta, ¿la palabra es una “Fuente”, un “Estímulo”, un “Ambiente”, un “Artefacto”, una “Respuesta” o una “Medida de Respuesta”?

Esta tarea no es trivial. El caso ideal es que exista una sola instancia, de alguna de esas seis, que se corresponda con la palabra.



Figura IV.10 Una única instancia se corresponde con la palabra "user"

La Figura V.9 muestra lo afirmado anteriormente. En ella se observa en el recuadro superior una clase (“ConcreteSource”), mientras que en el recuadro inferior se representa la instancia (“Instancia\_1”). A su vez la instancia posee una propiedad (de tipo “cadena de caracteres”) que se denomina “concreteScenarioPartDescription” y cuyo valor, en este caso, es “user”. Uno o más escenario se pueden relacionar con esta instancia mediante la propiedad “concreteScenariohasSource”.

Se pretende identificar la palabra “user”. En este caso hay una sola instancia en la ontología, cuyo identificador único de instancia es “Instancia\_1”, de tipo ConcreteSource, que mediante la propiedad “concreteScenarioPartDescription” posee el valor “user”, por lo que se retorna esta instancia.

Distinto es el caso en el que la misma palabra pueda representar, simultáneamente, distintas partes de un escenario (Figura V.7).



Figura IV.11 Dos instancias, de diferente tipo, se corresponden con la palabra "system"

En este caso la palabra “system” podría ser un ConcreteSource o un ConcreteArtifact. Lo que lleva a la cuestión de decidir entre ambas opciones. Para ello se contabiliza con cuántos escenarios se relacionan cada una de las instancias. Será elegida como significado semántico de la palabra la instancia que se relacione con un mayor número de escenarios. En este último caso si la “instancia\_1” se relaciona con 3 escenarios y la “instancia\_2” con 6, se retorna la “instancia\_2”, tomándose como que la palabra “system” es un ConcreteArtifact.

En este punto cabe aclarar que no existen dos instancias que se relacionen con la misma palabra (mediante la propiedad “concreteScenraioPartDescription”) y que sea del mismo tipo (ConcreteScource, ConcreteArtifact, etc.) Es decir, por ejemplo, no puede haber dos instancias en las que ambas sean estímulos y además se relacionen con la palabra “developer”. Si esto ocurriera, se pasan todas las relaciones de una instancia a la otra, y se elimina la primera. Este es un proceso que se realiza al crear la ontología. Es por ello que en la descripción anterior, con devolver la instancia ya se asegura que es única para ese parte de escenario.

#### Instance-QA Map Association

Una vez que se ha identificado la instancia de alguna parte de un escenario que se corresponde con una palabra, se dispone a obtener los porcentajes de los distintos atributos de calidad que se relacionan con esa instancia. Los atributos de calidad, juntos con los porcentajes asociados, se agrupan en un mapa.

Para encontrar estos porcentajes, primero se recuperan todos los escenarios que se relacionan con la instancia encontrada en el paso anterior. Cada uno de estos escenarios se relaciona con un atributo de calidad mediante la propiedad “isSpecifOf”. De esta manera, para cada atributo de calidad, se devuelven los porcentajes en que estos escenarios se relacionan con cada uno de ellos.

Por ejemplo, supóngase que como entrada se obtuvo el término "latency". Después del paso anterior, se encuentra que esa palabra se corresponde con la “Instancia\_48” que es de tipo ResponseMeasure. A su vez 40 escenarios se relacionan con la “Instancia\_48” mediante la propiedad “scenarioHasResponseMeasure”. De esos 40 escenarios, 30 de ellos se relacionan con el atributo de calidad "Performance", 8 con "Aviability" y 2 con "Modificability". De esta manera, es posible concluir que ese concepto, “latency”, se relaciona en un 75% de certeza con "Performance", 20% con "Aviability" y 5% con "Modificability". En este caso, esta sub-actividad relacionaría a la palabra “latency” con <Performance 0.75, Aviability 0.2, Modificability 0.05>.

# ANEXO I

|  |  |
| --- | --- |
| Sección | Descripción |
| Nombre | Nombre del caso de uso. |
| Descripción | Breve descripción donde se describe el rol y propósito en un párrafo simple. |
| Flujo de Eventos | This use case starts when the actor does something. An actor always initiates use Cases. The use case should describe what the actor does and what the system does in response. It should be phrased in the form of a dialog between the actor and the system. The use case should describe what happens inside the system, but not how or why. If information is exchanged, be specific about what is passed back and forth. For example, it is not very illuminating to say that the Actor enters customer information. It is better to say the Actor enters the customer’s name and address. A Glossary of Terms is often useful to keep the complexity of the use case manageable - you may want to define things like customer information there to keep the use case from drowning in details. Simple alternatives may be presented within the text of the use case. If it only takes a few sentences to describe what happens when there is an alternative, do it directly within the Flow of Events section. If the alternative flows are more complex, use a separate section to describe it. For example, an Alternative Flow subsection explains how to describe more complex alternatives. A picture is sometimes worth a thousand words, though there is no substitute for clean, clear prose. If it improves clarity, feel free to paste graphical depictions of user interfaces, process flows or other figures into the use case. If a flow chart is useful to present a complex decision process, by all means use it! Similarly for state-dependent behavior, a state-transition diagram often clarifies the behavior of a system better than pages upon pages of text. Use the right presentation medium for your problem, but be wary of using terminology, notations or figures that your audience may not understand. Remember that your purpose is to clarify, not obscure. |
| Requerimientos Especiales | A special requirement is typically a non-functional requirement that is specific to a use case, but is not easily or naturally specified in the text of the use case’s event flow. Examples of special requirements include legal and regulatory requirements, application standards, and quality attributes of the system to be built including usability, reliability, performance or supportability requirements. Additionally, other requirements-such as operating systems and environments, compatibility requirements, and design constraints-should be captured in this section. |
| Precondiciones | A pre-condition of a use case is the state of the system that must be present prior to a use case being performed |
| Postcondiciones | A post-condition of a use case is a list of possible states the system can be in immediately after a use case has finished |
| Puntos de Extension | Extension points of the use case. |
| Relaciones | Definition of the location of the extension point in the flow of events. Definition of the location of the extension point in the flow of events. |

# ANEXO II

En este anexo se describe el concepto de mapa y sus operaciones de una manera formal y detallada.

### Definición de “mapa”

Se define un “mapa” como un conjunto de tuplas <K,V> , donde a K se lo denomina “clave” y a V se lo denomina “valor”. Un mapa no puede tener dos tuplas con claves idénticas. Es decir, dado un mapa M no existe ningún par de tuplas <K1, V>, <K2,V2> tal que K1=K2.

En los mapas definidos en este trabajo se utiliza como “claves” los nombres de los atributos de calidad y como “valores” números reales entre 0 y 1.

### Suma de dos mapas

Sean M1 y M2 dos mapas con igual número de tuplas, donde toda clave K que pertenece a alguna tupla de M1, también pertenece a alguna tupla de M2. Sea además cualquier valor de V perteneciente a M1 y M2 un número real. Definimos la suma M1+M2, como un mapa M3 en donde cada tupla que éste posee es el resultado de la suma de los valores de las tuplas de igual clave de M1 y M2.

Por ejemplo, si se tiene el mapa y el mapa se define siendo K1 y K2 claves; a, b, c y d números reales

### División de un mapa por un número real

Sea M un mapa compuesto por un conjunto de tuplas <K,V> donde K es la clave y V son números reales. Sea x un número real distinto de cero. La división de M por x da como resultado el mismo mapa, con todos los valores V divididos por ese número.

Por ejemplo, si entonces , siendo a, b y x numero reales; M un mapa; K1,K2 claves del mapa.

### Multiplicación de un mapa por un número real

Sea M un mapa compuesto por un conjunto de tuplas <K,V> donde K es la clave y V son números reales. Sea x un número real. La multiplicación de M por x da como resultado el mismo mapa, con todos los valores V multiplicados por ese número.

Por ejemplo, si , entonces , siendo a,b y x números reales; M un mapa; K1,K2 claves del mapa.

# Bibliografía

1. **Clements, et al.** A Practical Method for Documenting Software Architectures. *Carnegie Mellon University.* Pittsburgh, Pennsylvania, USA : s.n., 2002.

2. [Online] http://www.sei.cmu.edu/architecture/definitions.html.

3. **Reynoso, Carlos.** *Introducción a la Arquitectura de Software Version 1.0.* s.l. : Universidad de Buenos Aires, 2004.

4. *Survey of Architecture Description Languages.* **Clements, Paul.** 1996. Proceedings of the International Workshop on Software Specification and Design.

5. **Bass, Clements and Kazman.** *Software Architecture in practice.* s.l. : Addison-Wesley, 1998.

6. **Parnas, D. L.** On the criteria to be used in decomposing systems into modules. *Communications of the ACM.* Diciembre 1972. Vol. 15, pp. 1053-1058.

7. **Buschmann, et al.** *Pattern-Oriented Software Architecture: A System Of Patterns.* West Sus- : John Wiley & Sons, 1996.

8. *IEEE Standard 1061-1992. Standard for a Software Quality Metrics.* IEEE. 1992. Standard.

9. **Kazman, Clements and Kein.** *Evaluating Software Architectures. Methods and case studies.* s.l. : Addison Wesley, 2001.

10. **Barbacci, et al.** *Quality Attributes.* 1995. Technical Report. CMU/SEI-95-TR-021 ESC-TR-95-021.

11. **Bass, Clements and Kazman.** *Software Architecture in Practice, Second Edition.* s.l. : Addison-Wesley, 2003.

12. **Berenbach, Paulish and Kazmei.** *Software & Systems Requirements Engineering: in practice.* s.l. : McGraw Hill Professional, 2009.

13. *Identifying Crosscutting Concerns in Requirements Specifications.* **Rosenhainer, L.** Vancouver : s.n., 2004. Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design , held in Function with OOPSLA.

14. **Baniassan, et al.** Discovering early aspects. *IEEE Software.* 2001. Vol. 23, 1, pp. 61-70.

15. **Cleland-Huang, Jane, et al.** Automated classiﬁcation of non-functional requirements. Londres, UK : Springer-Verlag, Abril 2007. Vol. 12, 2, pp. 103-120.

16. IEEE 830: IEEE Recommended Practice for Software Requirements Specifications. 1998.

17. [Online] http://es.wikipedia.org/wiki/Caso\_de\_uso.

18. **Noy, Natalya and McGuinness, Deborah.** Ontology Development 101: A Guide to Creating Your First Ontology Escenario. *Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880.* Marzo 2001.

19. OWL Web Ontology Language - Use Cases and Requirements. *W3C Recommendation .* [Online] 2004. http://www.w3.org/TR/webont-req/.

20. **Gruber, T. R.** A translation approach to portable ontology specfications. Junio 1993. Vol. 5, 2, pp. 199-220.

21. OWL Web Ontology Language Overview. *W3C.* [Recommendation]. Febrero 10, 2004.

22. Extensible Markup Language (XML) 1.0 (Fifth Edition). *W3C Recommendation.* Noviembre 26, 2008. Disponible en http://www.w3.org/TR/2008/REC-xml-20081126/.

23. RDF Primer. *W3C Recommendation.* Febrero 10, 2004. Disponible en http://www.w3.org/TR/rdf-primer/.

24. **Connolly, Dan, et al.** DAM+OIL. *W3C Reference Description.* Diciembre 28, 2001.

25. Fundacion Eclipse. [Online] http://www.eclipse.org/.

26. *Identificacion Temprana de Aspectos.* **Haak, B., Pryor, A. and Marcos, C.** 2005, Revista SCC (Workshop in SE), Vol. 6.

27. *Early Aspect Identification from Use Cases using NLP and WSD Techniques.* **Rago, et al.** 2009. EA '09 Proceedings of the 15th workshop on Early aspects.

28. *UML Semantics version 1.1.* Rational Software Corporation. 1997.

29. [Online] http://es.wikipedia.org/wiki/Stemming.

30. Official home page for distribution of the Porter Stemming Algorithm. [Online] http://tartarus.org/~martin/PorterStemmer/.

31. **Shaw and Garlan.** *An introduction to software architecture.* CMU Software Engineering Institute Technical Report. 1994. CMU/SEI-94-TR-21, ESC-TR-94-21..

1. La herramienta Aspect Extractor Tool representa un early aspect mediante un conjunto de pares formados por un objeto directo y un verbo, o simplemente un verbo. [↑](#footnote-ref-1)
2. En el Anexo II se encuentra una definición formal del concepto “mapa” y de cada una de las operaciones mencionadas anteriormente, que se utilizarán a lo largo de este trabajo. [↑](#footnote-ref-2)