**Identificación y trazabilidad de Atributos de Calidad 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

Septiembre 2010

Índice de contenidos

[Capítulo I - Introducción 8](#_Toc273625479)

[I.1. Contexto 9](#_Toc273625480)

[I.2. Problemática 9](#_Toc273625481)

[I.3. Solución Propuesta 10](#_Toc273625482)

[I.4. Organización del trabajo 11](#_Toc273625483)

[Capítulo II - Conceptos relacionados 12](#_Toc273625484)

[II.1. Arquitectura de Software 12](#_Toc273625485)

[II.1.1. Definición de Arquitectura de Software 12](#_Toc273625486)

[II.1.2. Ejemplo de una arquitectura 13](#_Toc273625487)

[II.2. Atributos de calidad 16](#_Toc273625488)

[II.2.1. Principales atributos de calidad 17](#_Toc273625489)

[II.2.1.1. Disponibilidad 17](#_Toc273625490)

[II.2.1.2. Performance 17](#_Toc273625491)

[II.2.1.3. Seguridad 18](#_Toc273625492)

[II.2.1.4. Usabilidad 18](#_Toc273625493)

[II.2.1.5. Modificabilidad 19](#_Toc273625494)

[II.2.1.6. Testeabilidad 19](#_Toc273625495)

[II.2.2. Atributos de calidad en una arquitectura 19](#_Toc273625496)

[II.3. Aspectos tempranos y atributos de calidad 20](#_Toc273625497)

[II.4. Especificación de requerimientos 21](#_Toc273625498)

[II.4.1. Casos de uso 22](#_Toc273625499)

[II.4.2. Escenarios de calidad 23](#_Toc273625500)

[Capítulo III - Trabajos Relacionados 26](#_Toc273625501)

[III.1. Enfoques y técnicas existentes 26](#_Toc273625502)

[III.1.1. Métodos de elicitación 26](#_Toc273625503)

[III.1.1.1. Elicitación de requerimientos con casos de uso 26](#_Toc273625504)

[III.1.1.2. Atributos de calidad “crosscutting” en la ingeniera de requerimientos 27](#_Toc273625505)

[III.1.2. Métodos de detección semiautomáticos 28](#_Toc273625506)

[III.1.2.1. Clasificador NFR 29](#_Toc273625507)

[III.1.2.2. Técnicas de clasificación para requerimietnos informales 30](#_Toc273625508)

[III.1.2.3. Identificación de QARs en especificaciones textuales. Enfoque semi-supervisado de aprendizaje. 31](#_Toc273625509)

[III.2. Conclusiones 34](#_Toc273625510)

[Capítulo IV - Identificación de atributos de calidad 35](#_Toc273625511)

[IV.1. Enfoque propuesto 35](#_Toc273625512)

[IV.2. Tokens Generation 37](#_Toc273625513)

[IV.2.1. Input Processor 38](#_Toc273625514)

[IV.2.2. Tokens Filter 40](#_Toc273625515)

[IV.2.2.1. Filtro Lower Case 42](#_Toc273625516)

[IV.2.2.2. Filtro Stop Words 42](#_Toc273625517)

[IV.2.2.3. Filtro Stemming 43](#_Toc273625518)

[IV.2.2.4. Filtro Ocurrencias 44](#_Toc273625519)

[IV.2.2.5. Filtro Pesos 45](#_Toc273625520)

[IV.3. Token Analysis 46](#_Toc273625521)

[IV.3.1. QAT Identification 47](#_Toc273625522)

[IV.3.1.1. Maps Identification 48](#_Toc273625523)

[IV.3.1.1.1. Descripción algorítmica 48](#_Toc273625524)

[IV.3.1.1.2. Descripción matemática 50](#_Toc273625525)

[IV.3.1.1.3. Ejemplo 51](#_Toc273625526)

[V.3.1.2 Maps Combination 53](#_Toc273625527)

[IV.3.2. Word-QA Association 54](#_Toc273625528)

[IV.3.2.1. Ontología 55](#_Toc273625529)

[IV.3.2.1.1. Componentes de una ontología 56](#_Toc273625530)

[IV.3.2.1.2. Ontología definida 56](#_Toc273625531)

[IV.3.2.2. Word-Instance Association. 58](#_Toc273625532)

[IV.3.2.3. Instance-QA Map Association 60](#_Toc273625533)

[Capítulo V - Evaluación de la técnica propuesta 61](#_Toc273625534)

[V.1. Métricas propuestas 61](#_Toc273625535)

[V.1.1. Falsos y verdaderos, positivos y negativos 63](#_Toc273625536)

[V.1.2. Recall, Precision y F-Measure 63](#_Toc273625537)

[V.2. Casos de estudio 64](#_Toc273625538)

[V.2.1. HWS (Health Watcher System) 64](#_Toc273625539)

[V.2.1.1. Casos de Uso 65](#_Toc273625540)

[V.2.1.2. Aspectos tempranos identificados 65](#_Toc273625541)

[V.2.1.3. Análisis con la técnica propuesta 68](#_Toc273625542)

[V.2.1.4. Resumen de resultados del caso de estudio HWS 74](#_Toc273625543)

[V.2.1.5. Métricas 78](#_Toc273625544)

[V.2.1.6. Tiempo de ejecución 80](#_Toc273625545)

[ANEXO I 81](#_Toc273625546)

[Definición de “mapa” 81](#_Toc273625547)

[Suma de dos mapas 81](#_Toc273625548)

[División de un mapa por un número real 81](#_Toc273625549)

[Multiplicación de un mapa por un número real 81](#_Toc273625550)

[ANEXO II 82](#_Toc273625551)

[Bibliografía 95](#_Toc273625552)

Índice de Figuras

[Figura II.1 Arquitectura propuesta para el problema de KWIC 13](#_Toc273625448)

[Figura II.2 Escenario de calidad 22](#_Toc273625449)

[Figura II.3 Conjunto de valores para escenarios generales del atributo disponibilidad 23](#_Toc273625450)

[Figura II.4 Escenario concreto del atributo disponibiliad 24](#_Toc273625451)

[Figura III.1Creación de un modelo basado en la experiencia 26](#_Toc273625452)

[Figura III.2 Modelo de requerimientos para atributos de calidad 27](#_Toc273625453)

[Figura III.3 Fragmento del temaplate propuesto para atributos de calidad 27](#_Toc273625454)

[Figura III.4 Proceso de clasificación de NFRs (QARs) 28](#_Toc273625455)

[Figura III.5 Área de aplicación del sistema propuesto 29](#_Toc273625456)

[Figura III.6. Semi-suppervised approach 31](#_Toc273625457)

[Figura III.7 EM algorithm 32](#_Toc273625458)

[Figura IV.1 Diagrama de actividades principales 35](#_Toc273625459)

[Figura IV.2 Etapa 1 – Tokens Generation 37](#_Toc273625460)

[Figura IV.3 Actividad Input Processor 38](#_Toc273625461)

[Figura IV.4 Actividad Tokens Filter 40](#_Toc273625462)

[Figura IV.5. Etapa Token Analysis 45](#_Toc273625463)

[Figura IV.6. Actividad QAT Identification 46](#_Toc273625464)

[Figura IV.7 Sub-actividad Maps Identification 47](#_Toc273625465)

[Figura IV.8. Sub-actividad Maps Combination 52](#_Toc273625466)

[Figura IV.9. Sub-actividad Word-QA Association 54](#_Toc273625467)

[Figura IV.10. Ontología definida 56](file:///C:\Documents%20and%20Settings\Administrador\Escritorio\QATESIS\Capitulos%20Finales\Tesis_V2.docx#_Toc273625468)

[Figura IV.11 Una única instancia se corresponde con la palabra "user" 57](#_Toc273625469)

[Figura IV.12 Dos instancias, de diferente tipo, se corresponden con la palabra "system" 58](#_Toc273625470)

[Figura V.1 - Captura “Data Formating” 68](#_Toc273625471)

[Figura V.2 - Captura "Persistency" 69](#_Toc273625472)

[Figura V.3 - Captura "Consistency" 70](#_Toc273625473)

[Figura V.4 - Captura "Distribution" 71](#_Toc273625474)

[Figura V.5 - Captura "Security" 72](#_Toc273625475)

[Figura V.6 - Captura "Error Handling" 73](#_Toc273625476)

[Figura V.7 Arquitectura propuesta por Zhang y otros 74](#_Toc273625477)

[Figura V.8 Arquitectura propuesta por Pinto y otros 75](#_Toc273625478)

Índice de Tablas

[Tabla IV‑1: Fracción de use case 40](#_Toc273625426)

[Tabla IV‑2: Fracción de early aspect 40](#_Toc273625427)

[Tabla IV‑3: Tokens generados luego del análisis de la entrada 40](#_Toc273625428)

[Tabla IV‑4: Estado de los tokens luego del filtro LowerCase 42](#_Toc273625429)

[Tabla IV‑5: Lista de tokens pos filtro Stop Words 43](#_Toc273625430)

[Tabla IV‑6: Estado de los tokens luego del filtro Stemming 44](#_Toc273625431)

[Tabla IV‑7: Lista de tokens luego de aplicar el filtro Ocurrencias 44](#_Toc273625432)

[Tabla IV‑8: Ponderación de las secciones de los casos de uso 45](#_Toc273625433)

[Tabla IV‑9: Estado final de los tokens, luego del filtro Pesos 45](#_Toc273625434)

[Tabla IV‑10 Lista de dos tokens 51](#_Toc273625435)

[Tabla V‑1 Aspectos tempranos para HWS, detectados por la herramienta Aspect Extractor Tool, junto con los casos de uso que éstos atraviesan 68](#_Toc273625436)

[Tabla V‑2 Resultados para el aspecto "Data Formatting" 68](#_Toc273625437)

[Tabla V‑3 Resultados para el aspecto "Persistency" 69](#_Toc273625438)

[Tabla V‑4 Resultados para el aspecto "Consistency" 70](#_Toc273625439)

[Tabla V‑5 Resultados para el aspecto "Distribution" 71](#_Toc273625440)

[Tabla V‑6 Resultados para el aspecto "Security" 72](#_Toc273625441)

[Tabla V‑7 Resultados para el aspecto "Error Hadling" 73](#_Toc273625442)

[Tabla V‑8 Resumen de los resultados obtenidos para cada aspecto temprano 75](#_Toc273625443)

[Tabla V‑9 QARs del sistema HWS 79](#_Toc273625444)

[Tabla V‑10 Valores de QVP. QFP, QFN y QV para el caso de estudio del HWS 79](#_Toc273625445)

[Tabla V‑11 Valores de Precision, Recall y F-Measure para el caso de estudio HWS 80](#_Toc273625446)

[Tabla V‑12 Tiempos de ejecución para el análisis de cada uno de los aspectos 80](#_Toc273625447)

# Introducción

La calidad de software se define como el grado en el cual el software posee una combinación deseada de atributos, tales como: performance, disponibilidad, portabilidad, seguridad, usabilidad, etc. [1]. Estos atributos son requerimientos adicionales del sistema [11] que hacen referencia a características o restricciones que éste debe satisfacer, y complementan los requerimientos funcionales del sistema. Estas características o atributos se conocen con el nombre de *atributos de calidad* [9].

En términos generales, Bass y otros autores [10] establece 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, escabilidad, etc.).

Estos atributos de calidad son especificados a través de requerimientos de atributos de calidad *(quality-attribute requirements)*, en adelante denominados QARs. Los QARs juegan un rol crítico en el desarrollo de un sistema y guían el diseño arquitectónico. Por lo tanto, deben ser considerados y especificados en etapas tempranas del desarrollo [3].

Existen evidencias de la relación entre QARs correctamente especificados y el éxito de un sistema [4]. Por ejemplo, las consecuencias de una especificación equivocada de performance pueden variar considerablemente, incluyendo: relaciones dañadas con el cliente, fallas en el negocio, pérdida de ingresos, aumento del costo del proyecto debido a los costos de sumar recursos adicionales para resolver la falla, reducción de la competencia y, potencialmente, contribuir al fracaso del proyecto. En este contexto, los QARs son comúnmente señalados como un factor clave de éxito en la construcción de software de calidad, estableciendo un enfoque sistemático y pragmático para introducir calidad en los sistemas de software [5].

Un concepto relacionado con los QARs es el de *aspectos tempranos* (*early aspects*). Un aspecto temprano es un *concern* que atraviesa el diseño de un sistema [8], 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 [3]. En consecuencia, el descubrimiento de *early aspects* puede proporcionar pistas para identificar QARs.

Los casos de uso son una de las técnicas más comunes para especificar los requerimientos de un sistema [4]. Un caso de uso es una tarea, que describe el comportamiento del sistema, originada a partir de una acción de un usuario final, que tiene un resultado [13].

## Contexto

Han sido escritos varios trabajos significantes en las comunidades de ingeniería de requerimientos y de diseño arquitectónico (diferentes workshops realizados en las conferencias ECOOP [5] 1996, 2000, 2001; OOPSLA [6] 1999, 2000; ICSE [7] 2001, etc.) sobre la separación de concerns (por ejemplo, viewpoints, casos de uso, goals y modelos de análisis de trade-offs arquitectónicos). Sin embargo, los enfoques propuestos no puntualizan explícitamente en los concerns crosscutting. El trabajo sobre aspectos tempranos, por lo tanto, complementa estos enfoques proporcionando una manera sistemática de manejar dichos concerns.

Recientemente un gran esfuerzo de investigación está siendo llevado a cabo en el área de Desarrollo de Software Orientado a Aspectos [2, 3] para identificar y modelar aspectos desde las etapas más tempranas del ciclo de vida. Generalmente, se plasman en enfoques, técnicas y/o herramientas semi-automatizadas que permiten la identificación de los concerns crosscutting, extrayéndolos de los documentos de requerimientos. Estas tienen diferentes objetivos, y utilizan diferentes técnicas para realizar su propósito. Algunos ejemplos son *Theme/Doc* [19-22], que expone las relaciones entre los comportamientos del sistema; *EA-Miner* [28-30, 59-62], que permite identificar los aspectos que conciernen al sistema de manera rápida realizando un análisis sensible al contexto de la documentación provista; *IR for Identifying Crosscutting Concerns in Requirements Specifications* [58], que se basa en la aplicación de técnicas de *Recuperación de Información* para explorar las especificaciones de requerimientos en busca de algún concern en particular; entre otros.

## Problemática

Mientras que los requerimientos funcionales han recibido considerable atención en la literatura, los QARs, de una arquitectura de software han recibido un menor interés [4].

Dado que los QARs juegan un rol crítico en el diseño de la arquitectura, la detección temprana de estos atributos de calidad es crucial para tomarlos en consideración desde las primeras decisiones de diseño. A pesar de que parece una tarea sencilla, la identificación y priorización de atributos de calidad puede resultar dificultosa para un analista e insumir mucho tiempo. En la práctica, la mayoría de QARs  están, de alguna forma, “ocultos” entre los requerimientos que especifican la funcionalidad y por ello tienden a ser ignorados [6]. Tal es así que, a diferencia de los requerimientos funcionales, los requerimientos no funcionales (QARs) son muy difíciles de elicitar [7].

Las especificaciones de requerimientos tienden a ser organizadas en base a la funcionalidad, con los QARs dispersos en varios documentos. Esto puede llevar a una serie de problemas importantes como conflictos no detectados, soluciones arquitectónicas que no toman en cuenta las limitaciones críticas de calidad, y desarrollo de productos que no llegan a satisfacer las necesidades reales de los stakeholders.[3].

Por esta razón, el desarrollo de herramientas que asistan a analistas a identificar atributos de calidad en especificaciones de requerimientos (por ej., casos de uso), y permitan establecer trazabilidad entre dichos atributos de calidad y los requerimientos y posteriores decisiones de diseño arquitectónico son de gran utilidad.

## Solución Propuesta

Durante la captura de requerimientos difícilmente se logren identificar claramente aquellos atributos de calidad que guiarán el desarrollo del sistema. Sin embargo, existe la posibilidad de realizar un análisis sobre los documentos generados en esta etapa, para identificar aquellos QARs que aparecen de forma implícita dentro de los documentos.

En este sentido, la herramienta Aspect Extractor Tool [12] es un enfoque semi-automático para la identificación de aspectos tempranos en especificaciones de requerimientos. Aspect Extractor Tool utiliza técnicas de minería de aspectos (aspect mining). La herramienta toma la especificación de casos de uso como entrada, y devuelve una especificación de casos de uso extendida, que incluye aspectos candidatos reconocidos y sus relaciones de trazabilidad con los casos de uso.

El objetivo de este trabajo es extender la funcionalidad de Aspect Extractor Tool, agregando información de atributos de calidad a los *early aspects* descubiertos y a los casos de uso involucrados en dichos *early aspects*. Para ello, se utilizará la noción de Quality Attribute Theme (QAT), que básicamente está formado por un subconjunto de la especificación de casos de uso y un *early aspect*, que en forma conjunta hacen referencia a un atributo de calidad. Como resultado, se busca producir una lista priorizada de QATs que resuma los principales *concerns* y atributos de calidad de una especificación de requerimientos.

De esta forma, al analista contará con información relevante desde las primeras etapas del desarrollo del sistema, reconociendo los atributos de calidad del mismo, minimizando las consecuencias de no conocerlos a tiempo y posibilitando una mejor toma de decisiones las siguientes etapas de desarrollo.

## Organización del trabajo

En el Capítulo II se presentará una explicación los conceptos claves del Desarrollo de Sistemas de Software, enfatizando principalmente en Arquitecturas de Sistemas y en Atributos de Calidad. Adicionalmente, se explicarán conceptos que serán necesarios durante el desarrollo de la técnica propuesta.

En el se llevará a cabo el análisis y la comparación de trabajos de investigación actuales en el campo de la identificación de QARs en especificaciones de requerimientos.

En el se presentará la técnica propuesta para la identificación de atributos de calidad en especificaciones de requerimientos.

En el se describirán varios casos de estudio, los cuales serán analizados por la herramienta utilizando tanto la técnica propuesta.

En el se presentarán las conclusiones del trabajo y se discutirán las posibles mejoras de esta propuesta como también las próximas direcciones de las investigaciones de identificación de aspectos desde etapas tempranas.

# Conceptos relacionados

En este capítulo se presentarán los conceptos que se utilizarán a lo largo del presente trabajo. 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 mayormente en requerimientos funcionales. 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 funcionales eran suficientes para construir un diseño satisfactorio.

En este contexto, la *Arquitectura de Software* ha emergido como un eslabón crucial en el proceso de diseño, especialmente para el desarrollo exitoso de sistemas de software complejos [1].

La visión arquitectónica de un sistema es abstracta, concentrándose en el comportamiento e interacción de componentes, prestando poca atención a detalles de implementación, algoritmos o representación de datos. La arquitectura de software constituye 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. 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. [2].

Una definición reconocida es la de Clements [3]:

*La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, el comportamiento 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 [4]:

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

Por lo tanto, a pesar de la gran cantidad 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, compuesta por elementos de software, junto con sus propiedades y relaciones entre ellos. Estas cuestiones estructurales se vinculan con el diseño, pues la arquitectura de software es una forma de diseño de software que se manifiesta tempranamente en el proceso de creación de un sistema. 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.

La primera implicancia 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. 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 poseen más de una estructura y que ninguna de estas puede adjudicarse ser “la arquitectura” del sistema.

La tercera implicancia es que la definición implica que todo sistema de software tiene una arquitectura, porque todo sistema puede ser descripto como un conjunto de componentes y relaciones entre estos componentes.

Finalmente, la cuarta implicancia es que el comportamiento de cada componente es parte de la arquitectura en tanto y en cuanto 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.

### Ejemplo de una arquitectura

A continuación se muestra un ejemplo 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 [5]:

*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 debe 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 basa en el estilo de “Pipe and Filters”[6]. 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. Idealmente los filtros 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 propuesta, 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. [7]. Estos atributos son requerimientos adicionales del sistema [8] 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” [9].

La arquitectura de un sistema de software es determinante para que éste posea una combinación deseada o requerida de atributos de calidad. La arquitectura 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 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 mediante la evaluación de su arquitectura.

En términos generales, Bass y otros autores [4] 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 (por ejemplo disponibilidad, performance, seguridad, etc.)
* No observables en tiempo de ejecución: aquellos atributos que se establecen durante el desarrollo del sistema (por ejemplo 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 [4]. 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.

En general, es en la funcionalidad en donde se pone el foco de atención durante las distintas etapas de desarrollo, dejando a los atributos de calidad, con mucha suerte, en un segundo plano. De esta manera, la mayoría de los enfoques tienen en cuenta a los atributos de calidad separadamente de la funcionalidad, dejando su integración para las etapas finales del ciclo de desarrollo [9]. Sin embargo, los atributos de calidad deben ser considerados en todas las etapas del proceso, ya que ninguno es completamente dependiente de una sola etapa [4].

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 [10].

En este trabajo se denominará QAR (Quality Attribute Requirement) a un requerimiento de un atributo de calidad [11]. 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 describen algunos atributos de calidad que se mencionarán luego a lo largo del trabajo. Estos son disponibilidad (availability), 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 (por ejemplo, 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 tiene que ver con 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 la performance de un sistema mediante 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.

#### 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 sin cambios.

La regla anterior tiene excepciones. Un solo componente que es excesivamente grande y complejo, puede ser más costoso de modificar que cinco componentes más simples. Sin embargo, también es fácil imaginar un cambio global, simple y sistemático: por ejemplo, modificar el valor de una constante que aparece en todos lados. De esta manera, se considera como principio general que los cambios localizados son los mejores. [4]

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 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 costo 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 mencionados.

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 sencillo ubicar 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. Como los filtros no tienen estado, 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 un atributo dado tiene un impacto negativo sobre otro atributo.

## Aspectos tempranos y atributos de calidad

Se define un “concern” como “cualquier asunto de interés en un sistema de software” [12]. 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 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 [12].

En el contexto de la Ingeniería de Requerimientos (*Requirements Engineering*, RE) los concerns de interés central son los requerimientos. Un requerimiento es un tipo especial de concern. Una especificación de requerimientos bien escrita está caracterizada por el hecho de que cada declaración de requerimientos no mezcla varios requerimientos sino que representa exactamente un requerimiento, por ejemplo, solamente un concern.

Sin embargo, esto no siempre sucede en la práctica, donde ocurre que ciertos concerns se encuentran presentes en más de un requerimiento. Los requerimientos que cortan transversalmente a otros son denominados *requerimientos crosscutting* (crosscutting requirements) [13] o aspectos tempranos (early aspects). 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.

La Ingeniería de Requerimientos orientada a Aspectos (Aspect-oriented requirement Engineering, AORE) se enfoca en la identificación, especificación y representación de los aspectos tempranos. Ejemplos de estos últimos incluyen a concerns que se identifican con persistencia, distribución, seguridad o restricciones de tiempo real.

Muchos aspectos tempranos se corresponden con QARs de alto nivel como seguridad, performance, portabilidad y usabilidad [15]. De hecho, el descubrimiento tanto de aspectos tempranos como de atributos de calidad tiene una significativa importancia a la hora de diseñar la arquitectura. Además de que la correcta identificación de aspectos tempranos es significativa no sólo para propósitos de diseño arquitectónico, sino para que puedan ser evaluados y modelados en la etapa de diseño y codificación del sistema, evitando tener que ser “minados” y refactorizados en etapas posteriores.

En consecuencia, dada la similitud entre aspectos tempranos y atributos de calidad, el descubrimiento de los primeros puede proporcionar pistas para identificar QARs. De esta manera, dado un aspecto temprano pude ser posible realizar un razonamiento acerca del potencial atributo de calidad al que éste hace referencia.

## 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].

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 ó, en caso contrario, los QARs no son especificados de una manera formal mediante escenarios de calidad, sino que son expresados de una manera informal utilizando lenguaje natural.

De hecho, en un estudio realizado [15], en base a 15 especificaciones de requerimientos, se manifestó una 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 sobreentendidos y aceptados por todos los stakeholders.

En general, los documentos son organizados en términos de funcionalidad, poniendo muy poco énfasis en los QARs. De esta manera, éstos quedan diseminados o esparcidos en varios documentos. Como se mencionó antes, ésto conlleva a que los atributos de calidad no sean tenidos en cuenta en las primeras etapas del desarrollo, originando 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 no es una tarea sencilla y puede consumir gran cantidad de tiempo y esfuerzo. En general, esta identificación es realizada “ad-hoc”, analizando varios documentos que describen al sistema. Como los QARs pueden estar esparcidos entre muchos de estos documentos, se corre el peligro de malinterpretarlos fácilmente o, directamente, ignorarlos por completo.

### Casos de uso

En ingeniería del software, un caso de uso es una técnica para la captura de requisitos 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.

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.

### Escenarios de calidad

Los escenarios de calidad son una forma de representar, de una manera formal, los QARs de un sistema [10]. Un escenario consiste de seis partes [4] (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 distingue entre escenarios generales y concretos de calidad. Escenarios son aquellos que son independientes del sistema y pueden, potencialmente, pertenecer a cualquier sistema. Los escenarios concretos 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. Se muestran sus seis partes, 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. La información contenida en el conjunto de escenarios indica qué atributos de calidad deben estar presentes en la arquitectura de un sistema de software.

En general, las definiciones de 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.

# Trabajos Relacionados

En este capítulo se realizará una reseña de técnicas (o enfoques) existentes para interiorizar y comprender las ideas exploradas por los investigadores. Éste no pretende ser un estudio exhaustivo acerca del conjunto de técnicas actuales, sino más bien mencionar las que sean más representativas de las líneas seguidas.

Finalmente, se hará una comparación entre estos estudios y la motivación de la metodología seguida en la técnica propuesta en este trabajo.

## Enfoques y técnicas existentes

En general, el problema de identificación y/o clasificación de QARs es abordado por los investigadores desde dos perspectivas distintas [15]. La primera se basa en métodos de elicitación, proveyendo soporte para la identificación de QARs a medida que estos son descriptos, negociados o explicados con los stakeholders. La segunda incluye métodos de detección para la extracción semi-automática o manual de QARs a partir de una variedad de documentos existentes.

.

### Métodos de elicitación

Los métodos de elicitación basan su enfoque en la provisión de cuestionarios, checklists o plantillas para indagar a los stakeholders acerca de los atributos de calidad involucrados. Los QARs son identificados a medida que los documentos de especificaciones de requerimientos están siendo creados. Este hecho brinda la posibilidad de que los QARs puedan ser negociados con los stakeholders en las etapas tempranas de desarrollo.

#### Elicitación de requerimientos con casos de uso

En Dörr y otros [18] se presenta un enfoque tendiente a lograr un mínimo, completo y focalizado conjunto de medibles y trazables QARs. “Mínimo” se refiere a que solo los QARs relevantes son detectados para no restringir a la arquitectura prematuramente. “Completo” indica que todos los QARs de los stakeholders son identificados. “Focalizado” se refiere a que es claro el impacto de los QARs en la solución final, soportando ambigüedad en el sentido del estándar IEEE 830 [16]. “Medibles” significan que una métrica es dada para poder verificar que el sistema final satisface el QAR detectado. Por último, “trazable” se refiere a que el enfoque propuesto también provee la lógica efectuada por la cual el QAR en cuestión fue identificado.



Figura III.1Creación de un modelo basado en la experiencia

Para lograr estos objetivos la solución parte de un metamodelo de referencia propuesto. Este incluye los conceptos de atributo de calidad, QAR, sistema, tarea, entre otros. A partir de este modelo una organización puede instanciarlo, total o parcialmente, en un modelo concreto siguiendo sus políticas propias y experiencia de desarrollo de proyectos.

La Figura III.1 muestra lo afirmado anteriormente, en donde un modelo de referencia es “instanciado” en un modelo basado en la experiencia. A partir de este modelo la tarea “Derive Facilities” crea los checklists y templates especiales para identificar QARs. En esto caso, éstos estarán asociados con el atributo de calidad para el cual el metamodelo fue creado.

Finalmente, este conjunto de documentos creados en este proceso se utilizan para identificar QARs en la etapa de elicitación.

#### Atributos de calidad “crosscutting” en la ingeniera de requerimientos

En el trabajo propuesto por Moreira y otros [19] se pone el foco en los atributos de calidad que atraviesan distintas funcionalidades descriptas en los requerimientos. Es por ello que se propone un modelo para identificar y especificar QARs que atraviesen los requerimientos, incluyendo su integración semántica en la descripción funcional.

El modelo de proceso propuesto es compatible con UML y está compuesto por tres actividades principales: *identificar* (identify), *especificar* (specify) e *integrar* (integrate). La primera actividad consiste en identificar todos los requerimientos de un sistema y seleccionar, de éstos, los atributos de la calidad relevantes al dominio de aplicación y a los stakeholders. La segunda actividad se divide en dos partes principales: (1) especificar los requisitos funcionales utilizando un enfoque basado en casos de uso, (2) describir los atributos de calidad utilizando plantillas especiales e identificar aquellos que atraviesan los requerimientos funcionales (es decir, los atributos de calidad transversales). La tercera actividad propone una serie de modelos para representar la integración de los atributos de calidad transversales y requerimientos. La Figura III.2 muestra las principales actividades del modelo.



Figura III.2 Modelo de requerimientos para atributos de calidad

A su vez, el modelo seleccionado para especificar atributos de calidad está influenciado por los enfoques de Chung y otros [20] y Malan y Bredmeyer [21]. En la Figura III.3 se puede ver un fragmento del mismo.



Figura III.3 Fragmento del temaplate propuesto para atributos de calidad

### Métodos de detección semiautomáticos

Otros enfoques propuestos consisten en desarrollar herramientas semi-automáticas que “minen” QARs a partir de una variedad de documentos creados en la etapa de requerimientos. En estos casos se suelen utilizar técnicas de recuperación de la información (Information Retrieval, IR) o procesamiento natural del lenguaje (Natural Language Process, NLP) [22].

#### Clasificador NFR

En [15] el problema de detección y clasificación de QARs es abordado desde una perspectiva de clasificación supervisada. Este enfoque utiliza métodos de recuperación de la información para encontrar e identificar QARs. El método supone que los diferentes tipos de QARs se caracterizan por el uso de palabras claves, relativamente distintas, a las que denominan "términos indicadores”. Cuando los términos indicadores se aprenden de un tipo específico de QAR, pueden ser utilizados para detectar requerimientos, oraciones, o frases relacionadas con ese tipo. El proceso, que se representa en la Figura III.1, incluye las tres fases principales de *Training (Entrenamiento)*, *Classification* (*clasificación)*, y *Application* (*aplicación)*.



Figura III.4 Proceso de clasificación de NFRs (QARs)

Durante la fase de entrenamiento los términos indicadores son extraídos de especificaciones de requerimientos existentes, en las que los QARs han sido manualmente categorizados por tipo. Esto lo realiza la actividad “Indicator miner”, formando un conjunto de “Indicator Terms”. Estos términos luego se utilizan durante la fase de clasificación para detectar y clasificar otros QARs, mediante la actividad “NFR Classifier”. Finalmente durante la fase de aplicación, los requerimientos clasificados se utilizan para apoyar actividades más avanzadas de la etapa de desarrollo, como la negociación o diseño arquitectónico. En esta fase, las actividades “Architectural Design”, “Aspect Oriented Design Tools”, etc. son externas a la técnica propuesta y muestran algunos ejemplos que pueden tomar como entrada los NFRs clasificados. Una fase adicional, marcada en el diagrama como I*terative Training Phase* refina los términos indicadores previamente clasificados al proveer feedback indicando la correctitud de éstos resultados.

#### Técnicas de clasificación para requerimietnos informales

En el trabajo realizado por Park et al [23] se propone un sistema de soporte para el analista que permite la clasificación de requerimientos bajo distintas categorías.

Los proyectos de gran escala deben lidiar con un gran número de requerimientos, generalmente provenientes de fuentes distribuidas. Estos requerimientos en general son escritos en lenguaje natural, sin una adecuada organización o formalidad. Es por ello que se hace necesaria una correcta clasificación de los mismos, para su adecuado manejo en las etapas tempranas de desarrollo.

La herramienta propuesta en este trabajo presenta una técnica automática de clasificación de requerimientos en distintas categorías, utilizando técnicas de procesamiento del lenguaje natural (Natural Language Process, NLP). Estás categorías pueden ser, por ejemplo, costo, prioridad, tiempo de desarrollo, criticidad, atributo de calidad o cualquier otra en la que se haya entrenado a la misma.



Figura III.5 Área de aplicación del sistema propuesto

En la Figura III.2 se muestra el área de aplicación del sistema propuesto. Varios requerimientos, potencialmente provenientes de distintas fuentes, son ingresados al sistema. Éstos forman un conjunto de requerimientos desorganizados. Este grupo es justamente la entrada de la técnica propuesta que, luego de su aplicación, devuelve un conjunto de requerimientos correctamente organizados según algún criterio y listos para un análisis posterior.

El proceso seguido para la clasificación de requerimientos consta de tres etapas: *pre-procesamiento*, *construcción de un conjunto de “oraciones-centrales” para el entrenamiento de cada categoría* y *aprendizaje del clasificador*.

* Pre-procesamiento: En esta etapa se extraen las sentencias de un grupo de requerimientos preseleccionados. Por cada sentencia se sepan las palabras y se realiza un etiquetado POS [24].
* *Construcción de un conjunto de “oraciones-centrales” para el entrenamiento de cada categoría:* La técnica propuesta se basa en el hecho de que un analista puede realizar una lista de *topic-word*. Éstas son palabras más representativas de cada categoría o tópico por el que se quiere clasificar los requerimientos. Mediante esta lista de *topic words* se crea una lista de *key words*. Estas *key words* son palabras, provenientes de los requerimientos utilizados en el pre-procesamiento, altamente relacionadas con las *topic words.* Estos dos conjuntos de palabras son utilizados para obtener un conjunto de *oraciones centrales* para cada categoría. Este conjunto está formado por las oraciones más representativas de cada categoría y contienen una o más *key words* o *topic words.*
* *Aprendizaje del clasificador:* Las oraciones centrales obtenidas en el paso anterior sirven como base para el entrenamiento de un clasificador bayesiano.

#### Identificación de QARs en especificaciones textuales. Enfoque semi-supervisado de aprendizaje.

El enfoque propuesto en [22] es de aprendizajesemi-supervisado. En el mismo, un clasificador reconoce automáticamente diferentes tipos de QARs ocultos en un conjunto de documentos (cada uno describe un requerimiento para el sistema de forma textual) y presenta los resultados al analista para su inspección. El objetivo de la técnica es la integración de un sistema de recomendación para asistir a los analistas de requerimientos y diseñadores de software en el proceso de diseño arquitectónico. La Figura III.6 muestra una visión general del enfoque propuesto.



Figura III.6. Semi-suppervised approach

En principio, durante la etapa de *Learning (Aprendizaje)*, algunos requerimientos clasificados son utilizados en conjunto con los no clasificados para entrenar el clasificador utilizando un algoritmo semi-supervisado. En esta etapa se encuentran actividades principales: *Pre-processing* y *Semi-Supervisd Learning*.

Durante el pre-procesamiento, actividad *Pre-proccessing,* se utiliza uno de los métodos más comunes en el campo de recuperación de información (IR) para obtener representaciones de documentos efectivos, el modelo de espacio vectorial (VSM). En este modelo, cada documento es identificado por una característica en el espacio vectorial, en el cual cada dimensión corresponde a un término distinto con un valor numérico o peso indicando su importancia. Se realizan varios pasos de pre procesamiento para transformar los requerimientos textuales en vectores conforme a modelo de espacio vectorial. Luego, se llevan a cabo algunos pasos de normalización comunes (modificar letras capitales, dígitos, guiones y marcas de puntuación), limpieza de stop-words y stemming, para estructurar los documentos.

En la actividad de *Clasificación semi-supervisada* (*EM)* el algoritmo seleccionado (por ejemplo, naive Bayes, k-NN, etc.) utiliza algunos ejemplos clasificados de cada clase para generar una función de clasificación o hipótesis. El algoritmo de Maximización Expectativa (EM) es una clase común de algoritmo iterativo para estimar con máxima probabilidad en este tipo de problemas. Consiste de dos etapas, la etapa de *Expectation* y la de *Maximization*. Básicamente, la primera etapa, rellena los datos faltantes basándose en la estimación actual de los parámetros. La segunda etapa estima nuevamente los parámetros maximizar la probabilidad. En la observa Figura III.7 se muestra un pseudocódigo del algoritmo utilizado.



Figura III.7 EM algorithm

## Conclusiones

Se han inspeccionado distintas técnicas para la identificación y clasificación de QARs en etapas tempranas del desarrollo. La mayoría de estos enfoques utilizan estrategias de elicitación de requerimientos o la utilización de herramientas semi-automáticas, frecuentemente basadas en técnicas de NLP o IR.

En general, ninguna de las propuestas utilizando información aportada por aspectos tempranos para la identificación de QARs.

En los métodos de elicitación es muy difícil detectar los aspectos tempranos del sistema como para que estos puedan, en esta misma etapa, ser utilizados para la creación de templates, checklists o cuestionarios.

En las herramientas semi-automáticas analizadas no se utiliza información de los aspectos tempranos para identificar QARs. Algunas, como la propuesta de Cleland-Huang, poseen una técnica que identifica indistintamente aspectos tempranos y QARs, dada la similitud entre ambos aunque no se utiliza información aportada por los primeros para “descubrir” a los últimos.

En la actualidad existen herramientas semi-automáticas que “minan” aspectos tempranos a partir de una variedad de documentos creados en las etapas tempranas de desarrollo. Es por ello que se ve la necesidad de investigar si, a partir de la información que aportan estos aspectos tempranos detectados, es posible descubrir eficientemente los QARs del sistema.

# Identificación de atributos de calidad

En este capítulo se describe en detalle 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.

La misma parte de un número de aspectos tempranos previamente detectados con una herramienta externa. Cada uno de estos aspectos atraviesa o relacionan un subconjunto del total de casos de uso del sistema. A partir del análisis de las palabras de un aspecto temprano, y de los casos de uso que este relaciona, se intenta identificar el atributo de calidad al que aquellos dos hacen referencia.

## Enfoque propuesto

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 IV.1. Este proceso utiliza como entrada una lista de casos de uso y un early aspect, 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 aspect) 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 un early aspect. Adicionalmente, un QAT contiene un atributo de calidad al que esos casos de uso y el early aspect hacen referencia.

Para soportar dicho proceso se ha implementado una herramienta como una extensión - plugin - del entorno de desarrollo integrado (IDE) Eclipse [25]. La herramienta, permite al analista seleccionar como entrada el early aspect 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 analista podrá contar con la información acerca de los atributos de calidad identificados, además de tener la posibilidad de visualizar la trazabilidad del proceso completo. Esto último se refiere a la posibilidad que se le presenta al analista de observar como cada token, extraído de la entrada, influye en la formación del QAT.

## 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 el early aspect que los relaciona. 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, denominado *Input Processor*. En segundo lugar se requiere un filtrado sobre los mismos para identificar qué información es relevante para la próxima etapa, *Tokens Filter*. 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 llevarla a una representación interna uniforme. Este procesamiento es realizado a través de la separación de la información de las entidades de entrada (especificaciones de los use cases y del early aspect) 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 IV.1. Tomando ésta como entrada, el resultado de aplicar el procesamiento descripto anteriormente generaría como resultado la lista tokens mostrada en la Tabla IV‑2.

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

Tabla IV‑1: Fracción de use case

|  |  |
| --- | --- |
| **Nombre** | Distribution |
| **<verbo, objeto directo>** | <sent, server> , <sending, data> |

Tabla IV‑2: Fracción de early aspect

|  |  |
| --- | --- |
| **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> |
| Distribution | <id,9>, <tipo, early aspect>,<sección, name> |
| sent | <id,10>, <tipo, early aspect>,<sección, pair> |
| server | <id,11>, <tipo, early aspect >,<sección, pair> |
| sending | <id,12>, <tipo, early aspect >,<sección, pair> |
| data | <id,13>, <tipo, early aspect >,<sección, pair> |

Tabla IV‑3: 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 use cases y del early aspect.

El objetivo de esta actividad es filtrar la información proveniente de la etapa anterior, *Tokens Generation*, eliminando información irrelevante y dejando sólo aquella que es útil al análisis. Además, los filtros enriquecen cada token con diferentes atributos (por ejemplo peso, ocurrencias, etc.) que serán concluyentes para la etapa de análisis.



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 [6] 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 definio 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. De esta forma, se reconocería que, por ejemplo, “server” y “Server” son la misma palabra. En caso de que se desee que la herramienta sea sensible a mayúsculas/minúsculas bastaría con desconectar este filtro de la secuencia de procesamiento.

Continuando con los tokens generados en la Tabla IV‑2, sólo se modifican los caracteres en mayúscula de las palabras de los tokens. En la Tabla IV‑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> |
| distribution | <id,9>, <tipo, early aspect>,<sección, name> |
| sent | <id,10>, <tipo, early aspect>,<sección, pair> |
| server | <id,11>, <tipo, early aspect >,<sección, pair> |
| sending | <id,12>, <tipo, early aspect >,<sección, pair> |
| data | <id,13>, <tipo, early aspect >,<sección, pair> |

Tabla IV‑4: 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. Este es uno de los puntos de configuración que presenta la herramienta.

La Tabla IV‑4 muestra el estado de los tokens de la Tabla IV‑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> |
| distribution | <id,9>, <tipo, early aspect>,<sección, name> |
| sent | <id,10>, <tipo, early aspect>,<sección, pair> |
| server | <id,11>, <tipo, early aspect >,<sección, pair> |
| sending | <id,12>, <tipo, early aspect >,<sección, pair> |
| data | <id,13>, <tipo, early aspect >,<sección, pair> |

Tabla IV‑5: 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, el algoritmo de Porter [30].

Continuando con el ejemplo observado en la Tabla IV‑4, se modifican las palabras de los tokens, llevando cada una de ellas a su raíz. En la Tabla IV‑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> |
| distribution | <id,9>, <tipo, early aspect>,<sección, name> |
| sent | <id,10>, <tipo, early aspect>,<sección, pair> |
| server | <id,11>, <tipo, early aspect >,<sección, pair> |
| sending | <id,12>, <tipo, early aspect >,<sección, pair> |
| data | <id,13>, <tipo, early aspect >,<sección, pair> |

Tabla IV‑6: 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 IV‑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> |
| distribution | <id,9>, <tipo, early aspect>,<sección, name> |
| sent | <id,10>, <tipo, early aspect>,<sección, pair> |
| server | <id,11>, <tipo, early aspect >,<sección, pair> |
| sending | <id,12>, <tipo, early aspect >,<sección, pair> |
| data | <id,13>, <tipo, early aspect >,<sección, pair> |

Tabla IV‑7: 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 las precondiciones.

Hemos realizado un estudio 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 IV‑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 IV‑7, la lista de tokens final quedaría tal como se muestra en la Tabla IV‑8.

|  |  |
| --- | --- |
| **Sección** | **Peso** |
| <Nombre>, <Descripción>,<Prioridad>,<Actor> | 1 |
| <Flujo Básico> | 2 |
| <Flujo Alternativo>,<Trigger><Requerimientos especiales>,<Precondiciones>,<Postcondiciones> | 3 |

Tabla IV‑8: 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 IV‑9: Estado final de los tokens, luego del filtro Pesos

Para el early aspect no es necesario aplicar el filtro de peso debido a que se considera que todas las palabras dentro del early aspect 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 *Token Analysis* 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 ponderado según su importancia en el contexto y se ha dejado constancia del número de ocurrencias.



Figura IV.5. Etapa Token Analysis

La etapa *Token Analysis* (Figura IV.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 *Token Generation* (Figura IV.6.). Como se mencionó anteriormente, la entrada de esta actividad son dos listas de tokens: una extraída de los use cases y otra del early aspect. Cada uno de estos tokens posee diferentes atributos, como sección, peso, ocurrencias, etc. que serán determinantes en la identificación del QAT.



Figura IV.6. Actividad QAT Identification

La actividad QAT Identification 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 IV.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 del early aspect.



Figura IV.7 Sub-actividad Maps Identification

A continuación, se muestra el algoritmo que sigue ésta 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 de entrada (use cases list o early aspect list). 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 del early aspect 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 del early aspect, 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 IV‑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 “minute”, 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> |
| minute | <id,2>, <tipo, use case >,<sección, name>, <ocurrencias,2>, <peso,3> |

Tabla IV‑10 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 Availability. 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>; <* Availability*,0>; <Performance,0>).*

*cantidad\_de\_palabras = 0;*

En este punto comienza la iteración sobre los tokens 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>; <* Availability*,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 “Availability” 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>;<* Availability*,0.0>;<Performance,0.75>).*

*cantidad\_de\_palabras = 1;*

*cantidad\_de\_palabras = 0;*

Distinto es el caso del segundo token, que posee la palabra “minute”, 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 Availability 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>;<* Availability*,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>;<* Availability*,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 Availability y un 32 % con Performance.

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

#### 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 use cases y otro a partir de la lista de tokens extraídos del early aspect. 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 IV.8. 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 analista 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 del early aspect (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>; < Availability,0.25>; <Performance,0.50>) es el relacionado con la lista de los casos de uso y el mapa B= (<Modificability,0.10>; <Availability,0.30>; <Performance,0.60>) es el relacionado con el early aspect. 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>; < Availability,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 del early aspect. 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 del early aspect. Entonces el mapa resultante, totalMap, es igual a

en donde k es un número 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 de calidad 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.”* [31]*.*

Un dominio es un área de temática específica o de conocimiento, tal como medicina, fabricación de herramientas, bienes inmuebles, reparación automovilística, gestión financiera, etc. [32]. 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 [33]. 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‑10 se muestra un diagrama con la ontología 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” [4] y explicadas en el Capítulo II.

Esta ontología, definida para el enfoque propuesto, representa el dominio de atributos y escenarios de calidad sobre el cual se está investigando. La técnica propuesta se basa en esta ontología para consultar el grado de relación de las palabras de cada token con los atributos de calidad.

Se asume que las instancias de la ontología están creadas por un experto en el dominio. De esta manera, lo que la ontología incluye es la realidad del domino que se desea modelar, permitiendo un entendimiento común de todos los conceptos.



Figura .. Ontología definida

Como se puede apreciar en la Figura IV‑10, 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 Scenario 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.11 Una única instancia se corresponde con la palabra "user"

La Figura IV‑11 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 IV‑12).



Figura IV.12 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 el caso de que haya un empate se elige, indistintamente, cualquiera de las instancias.

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 “isSpecificOf”. 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 " Availability " 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 " Availability " y 5% con "Modificability". En este caso, esta sub-actividad relacionaría a la palabra “latency” con <Performance 0.75, Availability 0.2, Modificability 0.05>.

# Evaluación de la técnica propuesta

En este capítulo se realiza un análisis de la herramienta desarrollada siguiendo la técnica propuesta en este trabajo para la identificación de atributos de calidad. Esta evaluación será llevada a cabo con X casos de estudios reales, los cuales serán explicados brevemente y se proveerá su especificación, como así también los aspectos tempranos detectados.

## Métricas propuestas

Antes de entrar en el análisis de los casos de estudio en particular, en esta sección se realiza un análisis cuantitativo que permite establecer métricas de los resultados obtenidos a través de la herramienta. Para realizar dicho análisis, se utilizaron 2 tipos de métricas: el lapso de tiempo para efectuar la identificación, y aquellas especificas extraídas de la rama de *Recuperación de Información*. Estas últimas se pueden entender desde dos puntos de vistas: como un escenario de recuperación de información o como una tarea de clasificación estadística. Estos son análogos, y ambos serán explicados para cada métrica.

Las métricas utilizadas para evaluar el desempeño de los algoritmos son *Precision*, *Recall* y *F-Measure* [REF]. *Precision* puede ser interpretada como una métrica de exactitud o fidelidad, mientras que *Recall* es una métrica de completitud. *F-Measure* es una combinación de ambas.

Desde el punto de vista de la recuperación de información: una *Precision* perfecta se puntuaría con 1.0, y significa que cada resultado recuperado por una búsqueda fue relevante (pero no dice nada acerca de si todos los documentos relevantes fueron recuperados). En contraste, un *Recall* perfecto se puntuaría con 1.0, y significa que todos los documentos relevantes fueron recuperados por la búsqueda (pero no dice nada acerca de cuantos documentos irrelevantes también fueron recuperados). Para realizar el análisis, se utilizan los siguientes conceptos: *Relevant* *Documents*, el cual representa la cantidad de documentos relevantes que hay sobre una búsqueda dada, y *Documents* *Retrieved*, que representa aquellos documentos que fueron recuperados en una búsqueda.

Análogamente, desde el punto de vista de la clasificación estadística: se definen los siguientes conceptos: *True Positives*, que representa el número de ítems etiquetados como pertenecientes a la clase que efectivamente pertenecen a esa clase; *True Negatives*, que representa el número de ítems no etiquetados como pertenecientes a la clase que efectivamente no pertenecen a esa clase; *False Positives*, que representa el número de ítems etiquetados como pertenecientes a la clase que efectivamente no pertenecen a esa clase; y *False Negatives*, que representa el número de ítems no etiquetados como pertenecientes a la clase que efectivamente pertenecen a esa clase. En este contexto, *Precision* de una clase es el número de *True Positives* (por ejemplo, el numero de ítems correctamente etiquetados como pertenecientes a la clase) dividido por el número total de elementos etiquetados como pertenecientes a la clase (por ejemplo, la suma de *True Positives* y *False Positives*, que son los ítems incorrectamente etiquetados como pertenecientes a la clase). *Recall* en este contexto es definido como el número de *True Positives* dividido el número total de elementos que actualmente pertenecen a la clase (por ejemplo, la suma de *True Positives* y *False Negatives*, que son aquellos ítems que no fueron etiquetados como pertenecientes a esa clase pero debieron haberlo estado).

Frecuentemente, hay una relación inversa entre *Precision* y *Recall*, donde es posible incrementar una de ellas con el costo de reducir la otra, y viceversa. Usualmente, y debido a esta relación, estas dos métricas no son discutidas aisladamente. En cambio, ambos valores para una métrica son comparados a un nivel fijado de la otra métrica (por ejemplo, *Precision* a un nivel de 0.75 de *Recall*), o son combinadas en una medida única, como *F-Measure*, que representa la media armónica ponderada de *Precision* y *Recall*.

### Falsos y verdaderos, positivos y negativos

Para nuestro análisis, especificaremos las siguientes definiciones, basándonos en las establecidas previamente:

|  |  |
| --- | --- |
| *QA Verdaderos Positivos* (QVP) | Son aquellos QARs **identificados** por la herramienta, que **son realmente** QARs del sistema. |
| *QA Verdaderos Negativos*(QVN) | Son aquellos QARs **no identificados** por la herramienta, que **no son realmente** QARs del sistema. |
| *QA Falsos Positivos* (QFP) | Son aquellos QARs **identificados** por la herramienta, que **no son realmente** QARs del sistema. |
| *QA Falsos Negativos* (QFN) | Son aquellos QARs **no identificados** por la herramienta, que **son realmente** QARs del sistema. |
| *QA Verdaderos* (AV) | Son QARs que **son realmente** atributos de calidad del sistema. |

Se podría decir que los QARs en los casos de estudio se corresponden con la definición de la siguiente manera: los QVP serian “los QARs correctos” y los QFP serian los “QARs incorrectos y mal etiquetados”. La cantidad total de los QV se estableció realizando inspecciones manuales de las especificaciones. También en algunos casos se analizaron arquitecturas propuestas para así extraer de las mismas los atributos de calidad que los diseñadores tuvieron en cuenta a la hora de modelarlas. Estos valores se determinaron para cada caso de estudio. Además, los QVN no son considerados porque no tiene sentido en este tipo de evaluación.

### Recall, Precision y F-Measure

Utilizando la nomenclatura utilizada anteriormente, las formulas para calcular estas métricas se reescriben de esta manera:

La formula de *F-Measure* se mantiene como se definió anteriormente. Se debe resaltar que según nuestra opinión, en el caso particular de la identificación de atributos de calidad presentada en este trabajo, creemos que el *Precision* es la métrica más importante de las tres. En general, en los trabajos relacionados se pone el foco principalmente en *Recall*, ya que los costos generados por la omisión de identificar un QA pueden llegar a ser muy costosos. Sin embargo, la técnica propuesta por nosotros basa su identificación de QARs en aspectos previamente detectados, por lo que esto ya de por sí es un factor limitante.

Para ejemplificar lo mencionado anteriormente supóngase, por ejemplo, que un sistema dado cuenta con 10 QARs. Luego de un primer análisis realizado con una herramienta de identificación de aspectos tempranos, se detectan 5 aspectos tempranos. En el mejor de los escenarios, suponiendo que la técnica propuesta relacione correctamente los aspectos con los atributos de calidad y que además esos conformen QARs del sistema, se obtendría un *recall* igual a 0.5 y una precisión igual a 1. Es decir, es muy difícil alcanzar el máximo recall ya que en ese caso todos los QARs del sistema se deberían relacionar con un aspecto dado.

## Casos de estudio

Se estudiará individualmente cada uno de los casos de estudio que fueron seleccionados para la evaluación. Por cada uno de ellos se atravesarán varias instancias.

En primera instancia se explicará brevemente el sistema en cuestión, además de mostrar los casos de uso que conforman sus requerimientos.

En segunda instancia se indicarán los aspectos tempranos detectados utilizando la herramienta Aspect Extractor Tool. También se mostrará, por cada aspecto temprano, la lista de casos de uso que éste atraviesa.

En tercera instancia se realizará una serie de pruebas. Primeramente se correrá la técnica propuesta solamente sobre los casos de uso del sistema, sin ningún aspecto temprano presente. Luego se correrá la herramienta sobre cada aspecto temprano, junto con los casos de uso que éste relaciona. El objetivo es corroborar que los aspectos tempranos efectivamente brindan información sobre la identificación de QARs.

Finalmente, en última instancia se realizará un resumen y análisis de los resultados obtenidos en cada caso.

### HWS (Health Watcher System)

El objetivo del sistema consiste en recoger y gestionar las quejas y notificaciones relacionadas con la salud pública. El sistema también se utiliza para notificar a las personas información importante con respecto al Sistema de Salud.

El sistema de Salud Vigía (Healt Watcher System) también debe intercambiar información con el Sistema de Vigilancia Sanitaria (Sanitary Surveillance System, SSVS). Inicialmente, este cambio supondrá la consulta de los certificados sanitarios. Posteriormente, cuando el SSVS tenga el control de demanda de módulos desplegados, las quejas del SSVS serán intercambiadas entre los dos sistemas.

Con el desarrollo del sistema de salud de vigía, el Sistema Público de Salud mejoraría considerablemente:

• El control de quejas (registro y notificaciones).

• La calidad del servicio en relación a la diseminación de la información; por ejemplo: las campañas de vacunación, prevención de enfermedades, guías de salud, obtener certificados de nacimiento y defunción y detalles de las aplicaciones para licencias sanitarias.

Un ciudadano puede tener acceso al sistema a través de Internet o marcando 1520, y hacer su denuncia o solicitar información sobre los servicios de salud. En el caso de que se haya hecho una queja, esta se registrará en el sistema y dirigida por un departamento específico. Este departamento será capaz de manejar la demanda en una manera apropiada y devolver una respuesta luego del que reclamo haya sido tratado. Esta respuesta se registrará en el sistema y estará disponible para ser consultada.

El sistema será para uso público, distribuido en terminales estratégicamente ubicadas, en las cuales el ciudadano será capaz de registrar quejas o requerir información.

#### Casos de Uso

El presente caso de estudio está formado por nueves casos de uso. Debido a la extensión de los mismos, estos se pueden consultar en el Anexo II.

#### Aspectos tempranos identificados

Como ya se ha mencionado anteriormente se utiliza la herramienta Aspect Extract Tool para la identificación de aspectos tempranos del sistema. Ésta toma como entrada los casos de uso del mismo. Al ser una técnica semi-automática los resultados de la misma deben ser analizados por un analista experto en el dominio.

A continuación se muestran los seis aspectos tempranos detectados, junto con los pares verbo-objeto, o en algunos casos sólo verbos, que los conforman. Cabe mencionar que los nombres de los aspectos tempranos son arbitrarios, indicados por el analista experto en el dominio.

**DATA FORMATING**: (perform, queries); (presented, user); (presented, user); (presented, user); (presented, user); (presented, user); (presented, employee); (presented, employee); (presented, employee); (presented, employee); (representing, unit); (representing, specialty); (representing, complaint); (representing, type); (creates, instance); (generates, assigns); (alters, data); (being, system)

**PERSISTENCY**: (stored); (stored); (stored, state); (store, information); (storing, complaint); (storing, complaint); (storing, data); (have, state); (have, HELP); (has, access); (retrieves, list); (retrieves, list); (retrieves, list); (retrieves, list); (retrieves, list); (retrieves, list); (retrieved, ); (retrieve, entry); (retrieve, information); (retrieve, type); (retrieving, data); (retrieves, details); (retrieving, details); (retrieving, complaints); (retrieves, data); (determine, type); (updated ); (updated); (updated ); (updated); (update); (informing, user); (described, login); (changed ); (queried ); (queried ); (queried ); (left, state); (left, state); (left, state); (updated, system); (updated, system); (updated, system); (saves, complaint); (saved, system); (saves, data); (saved ); (stores, information); (stores, information);

**CONSISTENCY**: (ensures, information); (ensures, information); (ensures, information); (ensures, information); (ensures, information); (ensures, data); (ensures, data); (ensures, data); (ensures, data); (ensures, data); (ensures, data); (ensured); (ensured); (assured); (assured); (confirms, operation); (confirms, update);

**DISTRIBUTION**: (sent, server); (sent, server); (sent, server); (sent, server); (sent, server); (sent, server); (sent, server); (sent, server); (sent, server); (sent, server); (sent, server); (retrieved); (transmitted, server); (sending, data);

**SECURITY**: (validates, password); (validate, employee); (validated, system);

**ERROR HANDLING**: (ocurrs); (ocurrs); (ocurrs); (ocurrs); (ocurrs); (dealt, complaint); (verifies, data); (verifies, data);

Cada uno de estos aspectos tempranos identificados “atraviesa” un subconjunto del conjunto total de casos de uso. La herramienta Aspect Extractor Tool también brinda esa información que se resume a continuación.

|  |  |  |
| --- | --- | --- |
| ASPECTO TEAMPRANO | CASOS DE USO ATRAVESADOS POR EL ASPECTO | |
| Data Formatting | | * Query information * Complaint specification * Login * Update complaint * Change logged employee |
| Persistency | | * Query information * Update complaint * Change logged employee |
| Consistency | | * Query information * Complaint specification * Register new employee * Update health unit * Change logged employee |
| Distribution | | * Query information * Complaint specification * Login * Update complaint * Register new employee * Update health unit * Change logged employee |
| Security | | * Login |
| Error handling | | * Query information * Complaint specification * Login * Register Tables * Update complaint * Register new employee * Update employee * Update health unit * Change logged employee |

Tabla V‑1 Aspectos tempranos para HWS, detectados por la herramienta Aspect Extractor Tool, junto con los casos de uso que éstos atraviesan

#### Análisis con la técnica propuesta

A continuación se muestran los resultados obtenidos con la técnica propuesta para cada uno de los aspectos tempranos.

Para el filtro de pesos se fijan los valores mostrados en Tabla IV‑7. También se muestran los resultados para distintos valores de k. Se debe recordar que un valor de k igual a cero sólo toma en cuenta el mapa de tokens extraídos de los aspectos tempranos y un valor de k igual a 1 sólo toma en cuenta el mapa extraído a partir de los casos de uso. Un k igual a 0.5 hace una combinación de los dos mapas, multiplicando cada mapa por 0.5 y luego sumando ambos.

**Data Formatting**

|  |  |  |  |
| --- | --- | --- | --- |
|  | K=0 | K=0.5 | K=1 |
| Security | 0.19 | 0.24 | 0.28 |
| Availability | 0.0 | 0.04 | 0.07 |
| Testability | 0.08 | 0.09 | 0.11 |
| Modifiability | 0.0 | 0.05 | 0.09 |
| Usability | **0.69** | **0.50** | **0.31** |
| Performace | 0.04 | 0.09 | 0.14 |

Tabla V‑2 Resultados para el aspecto "Data Formatting"

Para el aspecto temprano “Data Formatting” gana ampliamente Usabilidad. Éste está bien identificado ya que, debido a los pares (verbo, objeto) que forman al aspecto, pareciera el atributo que debería estar relacionado.

A continuación (Figura V.1) se observa una captura de la herramienta luego de realizar el análisis para el early aspect *Data Formating.*

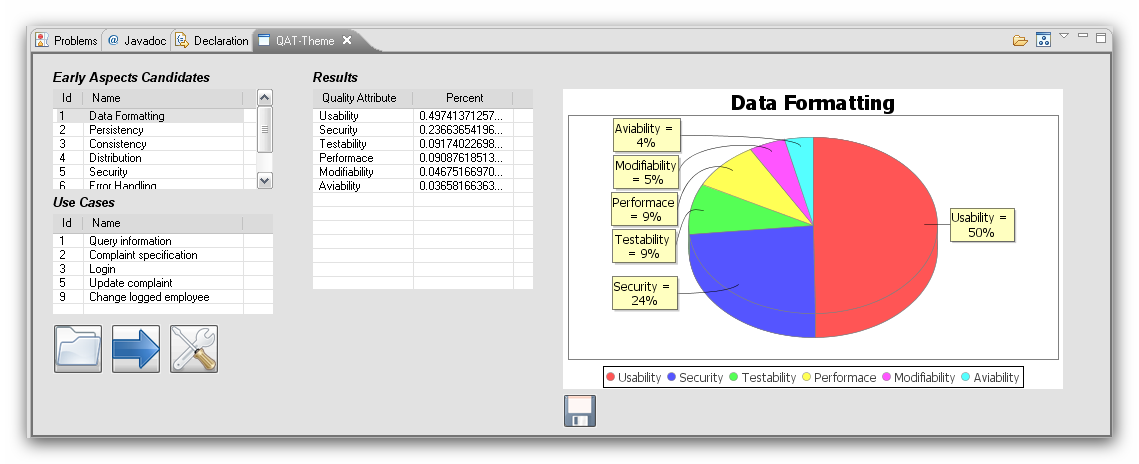
****

Figura . - Captura “Data Formating”

**Persistency**

|  |  |  |  |
| --- | --- | --- | --- |
|  | K=0 | K=0.5 | K=1 |
| Security | **0.39** | **0.33** | 0.28 |
| Availability | 0.29 | 0.05 | 0.07 |
| Testability | 0.07 | 0.09 | 0.11 |
| Modifiability | 0.26 | 0.18 | 0.09 |
| Usability | 0.1 | 0.20 | **0.31** |
| Performace | 0.16 | 0.15 | 0.14 |

Tabla V‑3 Resultados para el aspecto "Persistency"

Para el aspecto Data Formatting gana ligeramente Seguridad. Es interesante como Disponibilidad y Modificabilidad tienen valores bastante altos analizando solamente los aspectos tempranos, pero bajan bastante cuando se tiene en cuenta el mapa de los casos de uso. Esto podría indicar que analizando el aspecto temprano se tiene un trade­offs entre varios atributos, o no se corresponde claramente con algún atributo predominante.

En la Figura V.2 se muestra una captura de la herramienta para el early aspect *Persistency.*

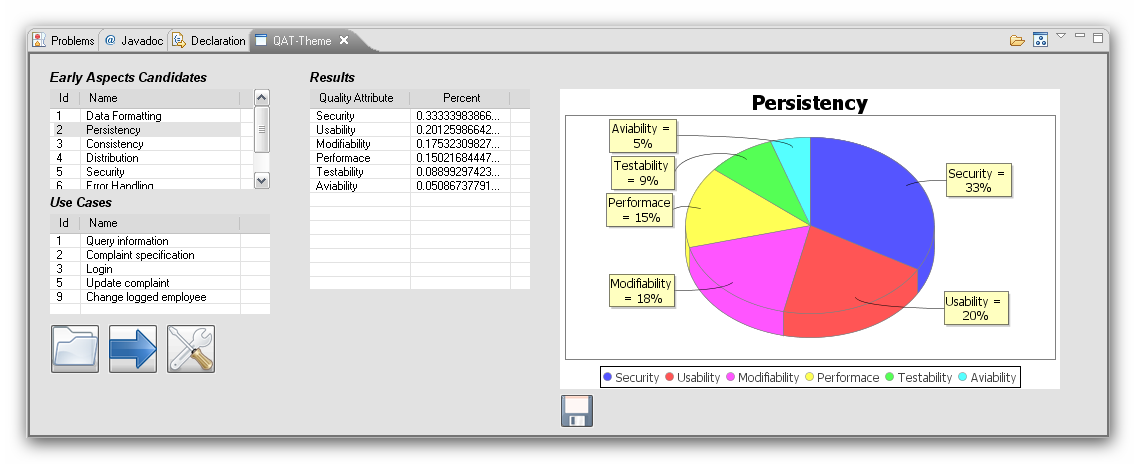


Figura . - Captura "Persistency"

**Consistency**

|  |  |  |  |
| --- | --- | --- | --- |
|  | K=0 | K=0,5 | K=1 |
| Security | **0.59** | **0.43** | **0.27** |
| Availability | 0.02 | 0.04 | 0.06 |
| Testability | 0.0 | 0.07 | 0.14 |
| Modifiability | 0.04 | 0.08 | 0.13 |
| Usability | 0.05 | 0.15 | 0.26 |
| Performace | 0.30 | 0.22 | 0.14 |

Tabla V‑4 Resultados para el aspecto "Consistency"

Para el aspecto temprano “Consistency” gana claramente Seguridad, aunque con k igual a 0 y a 0.5, performance arroja valores significativos, mostrando cierto tipo de relación con el aspecto.

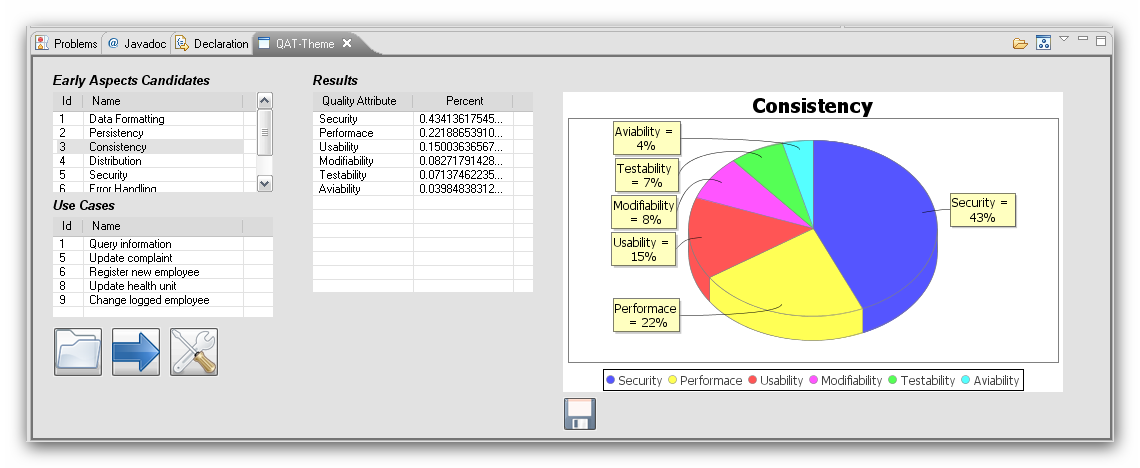
A continuación (Figura V.3) se observa una captura del plugin luego de analizar el early aspect *Consistency.*

Figura . - Captura "Consistency"

**Distribution**

|  |  |  |  |
| --- | --- | --- | --- |
|  | K=0 | K=0.5 | K=1 |
| Security | 0.05 | 0.17 | **0.28** |
| Availability | **0.87** | **0.47** | 0.06 |
| Testability | 0.0 | 0.06 | 0.12 |
| Modifiability | 0.0 | 0.05 | 0.11 |
| Usability | 0.0 | 0.14 | **0.28** |
| Performace | 0.08 | 0.11 | 0.15 |

Tabla V‑5 Resultados para el aspecto "Distribution"

Para el aspecto temprano “Distribution”, availability gana ampliamente para un valor de k=1 y 0.5. Finalmente, para un valor de k=1 los valores de security y usability empatan.

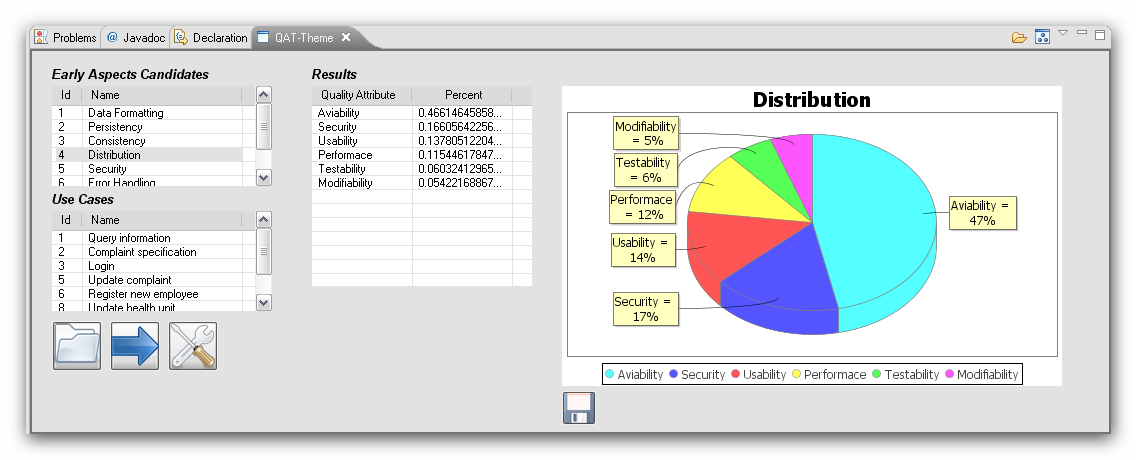
En la Figura V.4 se observa una captura del plugin luego de analizar el early aspect *Distribution.*****

Figura . - Captura "Distribution"

**Security**

|  |  |  |  |
| --- | --- | --- | --- |
|  | K=0 | K=0.5 | K=1 |
| Security | **1.0** | **0.69** | **0.38** |
| Availability | 0.0 | 0.04 | 0.09 |
| Testability | 0.0 | 0.01 | 0.03 |
| Modifiability | 0.0 | 0.01 | 0.01 |
| Usability | 0.0 | 0.17 | 0.35 |
| Performace | 0.0 | 0.07 | 0.15 |

Tabla V‑6 Resultados para el aspecto "Security"

Para el aspecto “Security” el atributo de calidad de Seguridad posee altos valores. Esta relación pareciera correcta, dado las palabras que conforman al aspecto.

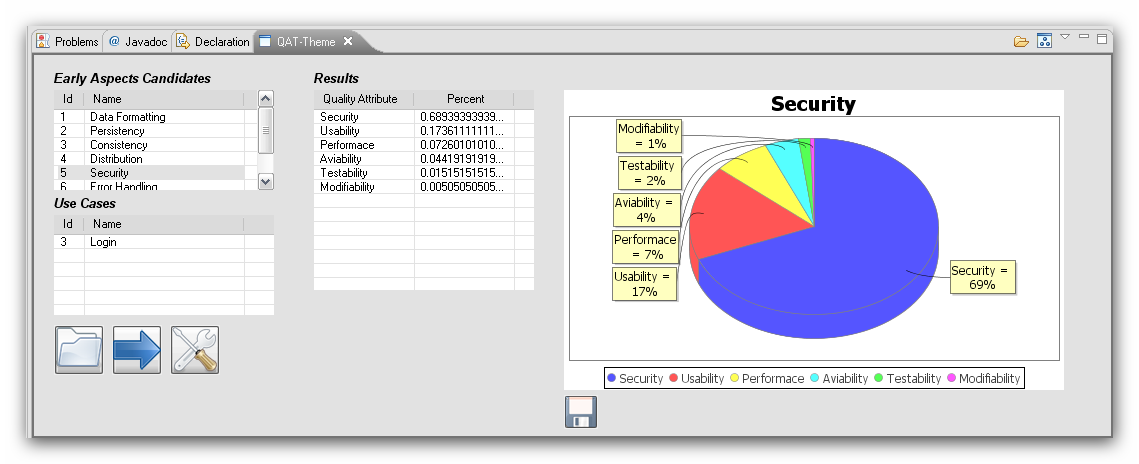
En la Figura V.5 se observa una captura del plugin luego del analisis del early aspect *Security.* 

Figura . - Captura "Security"

**Error handling**

|  |  |  |  |
| --- | --- | --- | --- |
|  | K=0 | K=0.5 | K=1 |
| Security | **0.7** | **0.54** | **0.37** |
| Availability | 0.0 | 0.04 | 0.09 |
| Testability | 0.0 | 0.02 | 0.03 |
| Modifiability | 0.0 | 0.01 | 0.01 |
| Usability | 0.2 | 0.27 | 0.35 |
| Performace | 0.1 | 0.12 | 0.15 |

Tabla V‑7 Resultados para el aspecto "Error Hadling"

Para el aspecto “Error handling” también el atributo de calidad de Security arroja los mayores valores para los distintos valores de k. Cabe mencionar que las palabras que conforman este aspecto no son muy descriptivas, por lo que la relación con el atributo de calidad es confusa, sin aportar demasiada información.

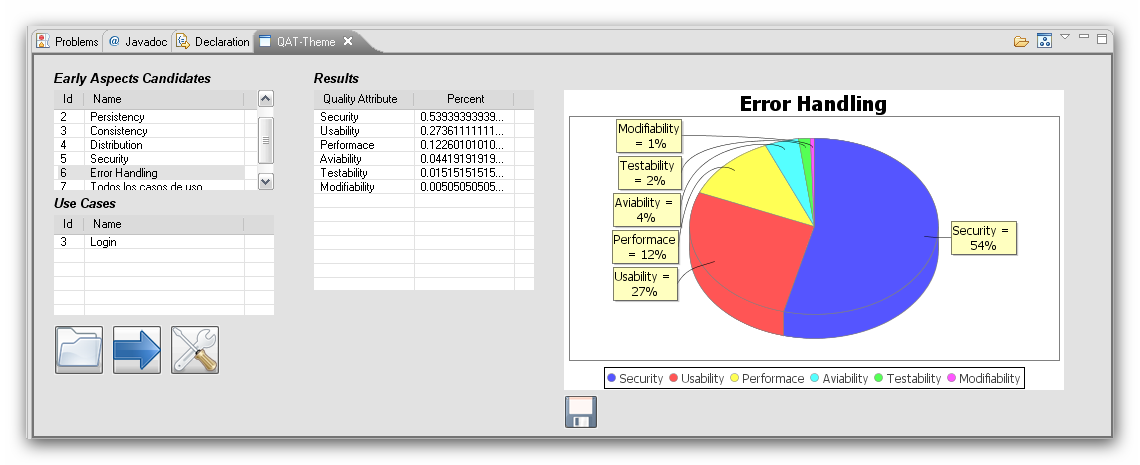
A continuación (Figura V.6) se observa el resultado de la herramienta luego de analizar el aspecto *Error Handling.* 

Figura V.6 - Captura "Error Handling"

#### Resumen de resultados del caso de estudio HWS

La a Tabla V‑8 muestra la correctitud de la relación entre los aspectos tempranos y los atributos de calidad identificados. Para los atributos de calidad incorrectamente relacionados (Consistency y Error Handling) se indica entre paréntesis el atributo de calidad al que ese aspecto temprano, junto con los casos de uso que relaciona, debería haber hecho referencia. Esto último no implica que ese sea efectivamente un QAR del sistema, sino que es un razonamiento acerca de a qué atributo de calidad debería haber apuntado, teniendo en cuenta las palabras que formaban el aspecto.

A su vez, se analizó que con un k=0,5 se arrojaron los mejores resultados, por lo que para cada aspecto temprano se toma el atributo de calidad alcanzado con ese valor.

|  |  |  |  |
| --- | --- | --- | --- |
| Aspecto | Atributo de Calidad | ¿Correcto? | Comentarios |
| Data Formatting | Usability | Si | (1) |
| Persistency | Security | Si | (2) |
| Consistency | Security | No (Avialability) | (3) |
| Distribution | Availability | Si | (4) |
| Security | Security | Si | (5) |
| Error Handling | Security | No (Avialability) | (6) |

Tabla V‑8 Resumen de los resultados obtenidos para cada aspecto temprano

Para evaluar la correctitud, o no, de la relación entre los aspectos tempranos y los atributos de calidad detectados se utilizaron los trabajos de Pinto y otros [34] y de Zhang y otros [35]. En ambos trabajos se proponen diseños arquitectónicos basados en aspectos (*aspectual architectures*) para el sistema HW. Basándose en las soluciones arquitectónicas propuestas, se pueden evaluar los atributos de calidad atacados, así como las tácticas adoptadas, y por lo tanto verificar si son correctos los atributos de calidad identificados a partir de cada aspecto.

En la Figura V.7 y la Figura V.8 se muestran dos vistas de las arquitecturas propuestas, de los trabajos mencionados, para el HWS.



Figura V.7 Arquitectura propuesta por Zhang y otros



Figura V.8 Arquitectura propuesta por Pinto y otros

A continuación se muestran los comentarios de los razonamientos para los datos mostrados en la Tabla V‑8. Es decir, la justificación del por qué se considera que esos aspectos están correctamente relacionados, o no, con el atributo de calidad y de esta manera formar un QAR del sistema. Cabe aclarar que el hecho de que algunos módulos de las arquitecturas se nombren igual que los aspectos no significa que obligatoriamente el aspecto temprano detectado se corresponda con ese módulo.

* (1) En la arquitectura mostrada en la Figura V.7, el modulo *Data Formatting* es invocado cada vez que la información es mostrada al usuario. Este módulo sería el que implemente la funcionalidad del aspecto Data Formatting, y por eso se presume que es correcta la relación con el atributo de usabilidad.
* (2) En la arquitectura mostrada en la Figura V.7, el modulo *Data Store* pareciera estar provisto con métodos de seguridad. En palabras de la autora:

*“In the Data Store component all the data about complaints, employees, health units, specialties and so on is saved. This component has a provided interface to set and get information from the HW System, and two provided interfaces to provide services to the Enrollment ’base’ component and to the Authentication ’aspectual’ component. Notice that this means that the Authentication component […] which plays the role of an ’aspectual’ component in its interaction with the UserGUI component, is playing the role of a ’base’ component in this interaction with the Data Store component.”*

Al proveer información al módulo *Authentication*, se asume que ésta (nombre de usuario, contraseñas, etc.) debe poseer mecanismos de seguridad, como por ejemplo encriptación, por lo que no está mal la relación con Security (aunque es cierto que quizás otro atributos puedan complementar a éste).

* (3) Similar al aspecto Data Formatting pero al revés. En este caso la información es chequeada cuando entra al sistema*.* Según Pinto,

*“The Consistency ’aspectual’ component is applied, during the interaction among the UserGUI and the HW System components, every time the HW system has to check the consistency of the received data”.*

Es por ello que disponibilidad parecería ser el atributo adecuado.

* (4) El módulo “*Distribution*” Figura V.7 es donde se ve la necesidad de elegir alguna táctica para satisfacer disponibilidad. Es en el módulo “*Replication*” Figura V.7 donde este problema se ataca. Pinto afirma:

*“In order to provide Avialability, the Replication component stores a copy of the data the HWS system sends….”*

En el trabajo de Haimei Zhang, Figura V.7, se destaca el siguiente concern:

*“Dependability and Availability: A serve or a processor can fail, usually in various ways, and must be built reliable using internal redundancy so that the register service remains available”*

Entonces, en este trabajo se detecta el atributo de calidad en ese contexto. Justamente el aspecto tiene pares de verbo-objeto como (sent, server), (transmitted, server), (sending, data), por lo que se esta refiriendo justamente a eso.

* (5) La arquitectura incluye un módulo “*Authentication*”. La función del mismo es “*interceptar la comunicación entre el empleado y el sistema HWS para chequear las credenciales del usuario cuando la operación lo requiera”.* Teniendo en cuenta los pares del aspecto temprano y el atributo de calidad relacionado con este aspecto, se puede afirmar que lo detectado es correcto.
* (6) En la Figura V.8 se especifica un modulo “*Error Handling*”. Básicamente, su funcionamiento es:

“*Concretely, there are different exceptional situations in the HW system that require different solutions, such as communication problems, data storing and retrieving problems, invalid data problems and data consistency problems. All these problem categories can be represented as provided interfaces of the ErrorHandling*”.

No parece muy clara la relación con Seguridad, aunque las palabras del aspecto temprano no aportan información significante. Pareciera ser más clara la relación con disponibilidad, ya que este modulo quizás debería implementar tácticas de “detección de fallas” (*Fault Detection*).

#### Métricas

La Tabla V‑9 muestra los QARs de referencia tenidos en cuenta para el sistema HWS. Ésta se armó teniendo en cuenta las especificaciones de requerimientos del sistema HWS detalladas en [36]. A su vez, se tuvieron en cuenta los trabajos de [34] y de [35]. En ambos análisis, se proponen diseños arquitectónicos orientados a aspectos para el Health Watcher System, como ya se mencionó anteriormente.

|  |  |
| --- | --- |
| Nombre | Descripción |
| Usability | The system should have an easy to use GUI, as any person who has access to the internet should be able to use the system. |
| Usability | The system should have an on-line HELP to be consulted by any person that uses it. |
| Availability | The system should be available 24 hours a day, 7 days a week. The nature of the system not being a critical system, the system might stay off until any fault is fixed. |
| Performance | The system must be capable to handle 20 simultaneous users. |
| Performance | The response time must not exceed 5 seconds. |
| Security | The system should use a security protocol when sending data over the internet. |
| Distribution | The system should be capable of running on separate machines. |
| Availability | A serve or a processor can fail, usually in various ways, and must be built reliable using internal redundancy so that the register service remains available. |
| Security | To have access to the complaint registration features, access must be allowed by the access control sub-system. (Authentication) |
| Scalability | HW system would be used by citizens from different area, and it must be designed to cater for variations |
| Persistence/ Avialability | Data is usually constructed and shared by citizen distributed in different locations. Therefore a major issue is the preserve the consistency of data in the concurrency and failure. |
| Security | The system should store confidential information using some kind of encryption |

Tabla V‑9 QARs del sistema HWS

Al tomar en cuenta las especificaciones de requerimientos de [36] se agregan QARs de referencia que no van a estar, necesariamente, distribuidos u ocultos en los casos de uso. Un requerimiento de utilización de un lenguaje de programación particular, por ejemplo, conformaría un QAR que no sería posible detectar a partir de los casos de uso, y por ende, en la técnica propuesta en este trabajo. A pesar de ello, esos QARs se toman como de referencia, con la posterior consecuencia de impactar negativamente en la métrica de *recall*.

A partir de los QARs de referencia mostrados y de los QARs identificados por la herramienta, se obtienen los valores de QVP, QFP, QFN y QV mostrados en la Tabla V‑9, para este caso de estudio.

|  |  |
| --- | --- |
| Caso de Estudio | HWS |
| QVP | 4 |
| QFP | 2 |
| QFN | 8 |
| QV | 12 |

Tabla V‑10 Valores de QVP. QFP, QFN y QV para el caso de estudio del HWS

Finalmente, teniendo en cuenta los valores de la Tabla V‑10, se obtienen los siguientes resultados de Precision, Recall y F-Measure.

Los resultados de las métricas se resumen en la Tabla V‑11.

|  |  |
| --- | --- |
| Caso de Estudio | HWS |
| Precision | 0.67 |
| Recall | 0.33 |
| F-Measure | 0.44 |

Tabla V‑11 Valores de Precision, Recall y F-Measure para el caso de estudio HWS

#### 

#### Tiempo de ejecución

De manera de tener una noción de la velocidad de procesamiento de la identificación por parte de la técnica propuesta, en Tabla V‑12 se muestran los lapsos de tiempo necesarios para efectuar el análisis de identificación de atributos de calidad (medidos en milisegundos) para cada conjunto <aspecto temprano, casos de usos relacionados>.

|  |  |
| --- | --- |
| Aspecto temprano | Tiempo de ejecución (en milisegundos) |
| Data Formatting | 1108 |
| Persistency | 1359 |
| Consistency | 1131 |
| Distribution | 1146 |
| Security | 761 |
| Error Handling | 696 |
| Promedio | 1033,5 |

Tabla V‑12 Tiempos de ejecución para el análisis de cada uno de los aspectos

En general, se tienen tiempos de ejecución bajos, en donde no se superan los 1.5 segundos en ningún caso. El máximo valor registrado es cuando se analizó el aspecto Persistency, con un tiempo de ejecución de 1359 milisegundos. El menor valor registrado se encontró en el aspecto Error Handling, con un valor de 696 milisegundos. El promedio de análisis de todos los aspectos tempranos arroja un valor de 1033,5 milisegundos.

# ANEXO I

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 números 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.

# ANEXO II

|  |  |
| --- | --- |
| Nombre | [FR01] Query information |
| Descripción | This use case allows a citizen to perform queries.  **Query Health Guide**  The citizen might query:   * Which health units take care of a specific specialty? * What are the specialties of a particular health unit?     **Query Specialty Information**  The citizen might query:   * Information about a complaint made by a citizen: * Complaint details. * Situation (OPENED, SUSPENDED, or CLOSED). * Technical analysis. * Analysis date. * Employee that made the analysis.      * Information about diseases: * Description. * Symptoms. * Duration. |
| Actor | * Citizen * Employee |
| Disparador | - |
| Flujo Básico | 1. The citizen chooses the type of query  1.1 In the case of query on specialties grouped by health units, the system retrieves the list of health units stored.  1.1.1 The system retrieves the details of each health unit such as its description and specialties.  1.1.2 The list of health units is presented to the user on their local display.  1.2 In the case of a query on health units grouped by specialties, the system retrieves the list of registered specialties.  1.2.1 The system retrieves the details of each specialty such as its unique identifier and name.  1.2.2 The list of specialties is presented to the user on their local display.  1.3 In the case of a query on diseases, the system retrieves the list of diseases.  1.3.1 The system retrieves the details of each disease type such as its unique identifier and name.  1.3.2 The list of disease is presented to the user on their local display.  2. The citizen provides the data for the query 2.1 In the case of a query on specialties grouped by health units; the citizen selects the health unit to be queried.  2.1.1 A unique identifier representing the selected health unit is sent to the server.  2.1.2 The system ensures the health unit information is consistent.  2.1.3 The unique identifier is used by the system to search the repository for the selected health unit.  2.1.4 The details of the selected health unit are retrieved including its specialties.  2.1.5 The specialties for the selected health unit are returned to the user.  2.2 In the case of a query on health units grouped by specialties, the citizen selects the specialty to be queried.  2.2.1 A unique identifier representing the selected specialty is sent to the server.  2.2.2 The system ensures the health unit information is consistent.  2.2.3 The unique identifier is used to retrieve the list of health units which are associated with the selected specialty.  2.2.4 The details of the health units and specialties are retrieved.  2.2.5 The retrieved health units are returned to the user.  2.3 In the case of a query on complaints, the citizen provides the complaint code.  2.3.1 The unique identifier representing the complaint to be retrieved is sent to the server.  2.3.2 The system ensures the complaint information is consistent.  2.3.3 The unique identifier is used to retrieve the complaint entry.  2.3.4 The system must determine the complaint type as to retrieve the appropriate information.  2.3.4.1 If the complaint is a special complaint the complainer’s age, education level and occupation are retrieved (in addition to the standard complaint information).  2.3.4.2 If the complaint is a food complaint the meal which was consumed, the number of people who ate the meal, the number of sick people, etc. are retrieved (in addition to the standard complaint information).  2.3.4.3 If the complaint is an animal complaint the animal species and the number of animals affected (in addition to the standard complaint information).  2.3.5 The complaint with all the appropriate information is returned to the user.  2.4 In the case of a query on diseases, the citizen selects the disease to be queried.  2.4.1 The unique identifier representing the disease type to be retrieved is sent to the server.  2.4.2 The system ensures the disease type information is consistent.  2.4.3 The unique identifier is used to retrieve the disease type to query.  2.4.4 The symptoms for the selected disease type are retrieved.  2.4.5 The complete disease information is returned to the user.  3. The query results are formatted and presented to the user on their local display. |
| Flujos Alternativos | 1.x and 2.x: A communication problem occurs.  1. Raise an error message  1.x.1 and 2.x.4: A problem occurs retrieving the complaint data.  1. The system retrieves the available information.  2. Raise an error message  2.3.3: An invalid complaint code is entered.  1. Raise an error message informing the user the complaint does not exist.  2.x.2: Consistent data cannot be assured.  1. The system abandons the data retrieval.  2. Raise an error message. |
| Precondiciones | The data to be queried must be registered on the system |
| Poscondiciones | The query result to the citizen |
| Requerimientos especiales | - |
| Prioridad | Important |

|  |  |
| --- | --- |
| Nombre | [FR02] Complaint Specification |
| Descripción | This use case allows a citizen to register complaints. Complaints can be:  **Animal Complaint – DVA**  • Sick animals.  • Infestations (rodents, scorpions, bats, etc.)  • Diseases related to mosquitoes (dengue, filariose).  • Animal maltreatment.  **Food Complaint - DVISA**  • Cases where there is a suspicion infected food being eaten.  **Special Complaint - DVISA**  • Cases related to several reasons, which are not mentioned above (restaurants with hygiene problems, leaking sewerage, suspicious water transporting trucks, etc.).    The three kinds of complaints have the following information in common:   * Complaint data: description (mandatory) and observations (optional); * Complainer data: name, street, complement, district, city, state/province, zip code, telephone number and e-mail. All these fields are optional. * Complaint state (mandatory), which might be: OPENED, SUSPENDED or CLOSED. When a complaint is first registers its state must be OPENED. * The system must register the complaint registration date.   In addition to the above data, each complaint type has its own specific data,  including:  **Animal Complaint – DVA**  • Type of animal (mandatory), amount of animals (mandatory), date problem was observed (mandatory).  • Problem location data: street, complement, district, city, state/province, zip code and telephone number. All of these fields are optional.    Food Complaint - DVISA  • Victim's name (mandatory).  • Victim's data: street, complement, district, city (or closest one), state/province, zip code and telephone number. All of these fields are optional.  • Amount of people who ate the food, amount of sick people, amount of people who were sent to a hospital and amount of deceased people.  All mandatory.  • Location where the patients were treated, suspicious meal. All optional.    **Special Complaint - DVISA**  • Age (mandatory), academic qualifications (optional), occupation (optional).  • Street, complement, district, city, state/province, zip code and telephone number of the closest location to the complaint location. All optional. |
| Actor | * Citizen * Employee |
| Disparador |  |
| Flujo Básico | 1. The citizen selects the kind of complaint.  2. The system shows the specific screen for each type of complaint.  3. The system registers the kind, date and time of the complaints.  4. The citizen provides the complaint specific data.  5. The system saves the complaint.  5.1. The information entered by the user is sent to the server.  5.2. The system parses the data entered by the user.  5.3. The system creates a new instance of the appropriate complaint type.  5.4 The system generates a unique identifier and assigns this to the new complaint.  5.5. The complainers address is parsed and saved.  5.6. The common complaint information is parsed and stored with the  OPENED state.  5.7. The specific complaint data is then extracted and stored accordingly.  5.8. The system ensures the data is left in a consistent state.  6. The unique identifier is returned and presented to the user on their local display. |
| Flujos Alternativos | 5.1: A communication problem occurs.  1. Raise an error message.  5.2: Invalid data is entered by the user.  1. Raise an error message.  5.5-5.7: A problem occurs storing the complaint.  1. The complaint entry is rolled-back.  2. Raise an error message.  5.8: Data consistency cannot be ensured.  1. The complaint entry is rolled-back.  2. Raise an error message. |
| Precondiciones | None |
| Poscondiciones | The complaint saved on the system |
| Requerimientos especiales |  |
| Prioridad | Essential |

|  |  |
| --- | --- |
| Nombre | [FR10] Login |
| Descripción | This use case allows an employee to have access to restricted operations on the Health-Watcher System |
| Actor | * Employee |
| Disparador |  |
| Flujo Básico | 1. The employee provides the login and password.  2. The login and password are sent to the server.  3. The system retrieves the employee details using the login as a unique identifier.  4. The system validates the entered password.  5. The result of the login attempt is presented to the employee on their local display. |
| Flujos Alternativos | 2: A communication error occurs.  1. Raise an error message.  3: A problem occurs retrieving the employee details.  1. Raise an error message.  4: The system cannot validate the employee.  1. Raise an error message. |
| Precondiciones | None |
| Poscondiciones | Password validated by the system |
| Requerimientos especiales |  |
| Prioridad | Essential |

|  |  |
| --- | --- |
| Nombre | [FR11] Register Tables |
| Descripción | This use case allows the registration of system tables. The following operations are possible: insert, update, delete, search and print.  The available tables include:  • Health unit (unit code, unit description).  • Specialty (code and description).  • Health unit / Specialty (health unit and specialty).  • Employee (login, name and password).  • Type of disease (code, name, description, symptom and duration).  • Symptom (code and description).  • Type of disease / Symptom (type of disease and symptom). |
| Actor | * Employee |
| Disparador |  |
| Flujo Básico | 1. The employee chooses the option to register (insert/update) in one of the tables.  2. The employee enters the data.  3. The system saves the data |
| Flujos Alternativos |  |
| Precondiciones | Verified employee |
| Poscondiciones | Updated data on the tables |
| Requerimientos especiales |  |
| Prioridad | Essential |

|  |  |
| --- | --- |
| Nombre | [FR12] Update complaint |
| Descripción | This use case allows the state of a complaint to be updated. |
| Actor | * Employee |
| Disparador |  |
| Flujo Básico | 1. The employee selects the update complaint option.  2. The system retrieves the list of all registered complaints.  2.1. The complaint list is populated with general and complaint type specific data.  3. The list of complaints is returned to the employee.  4. The complaints are formatted and presented to the employee on their local display.  5. The employee selects the complaint they wish to update.  6. The complaint unique identifier is sent to the server.  7. The system ensures the complaint data is consistent.  8. The system retrieves the complaint entry.  9. The complaint is returned to the employee.  10. The complaint is formatted and presented to the employee on their local display.  11. The employee enters the conclusion.  12. The conclusion is sent to the server.  13. The complaint status is set to close; the date the conclusion was entered is set in addition to the employee who dealt with the complaint.  14. The system ensures the complaint is left in a consistent state.  15. The complaint information is updated to store the new information. |
| Flujos Alternativos | 2 and 8: An error occurs retrieving the registered complaints.  2. Raise an error message.  7 and 14: Data consistency cannot be ensured.  1. The complaint changes are rolled-back.  2. Raise an error message.  3, 6, 9, and 12: A communication error occurs.  1. Raise an error message.  15: An error occurs storing the updated complaint.  1. The complaint changes are rolled back.  2. Raise an error message. |
| Precondiciones | The complaint must be registered and have the OPENED state.  Verified employee. |
| Poscondiciones | Complaint updated and with state CLOSED. |
| Requerimientos especiales |  |
| Prioridad | Essential |

|  |  |
| --- | --- |
| Nombre | [FR13] Register new employee |
| Descripción | This use case allows new employees to be registered on the system. |
| Actor | * Employee |
| Disparador |  |
| Flujo Básico | 1. The employee selects the insert employee option.  2. The employee provides the following information about the new employee:   * Name * Login ID * Password (with second password field for confirmation).   3. The employee confirms the operation.  4. The entered data is transmitted to the server.  5. The system verifies the entered data.  6. The system ensures employee data is consistent.  7. The system saves the new employee's data. |
| Flujos Alternativos | 2: Incomplete data entered.  1. Show a message informing the employee of the missing/incorrect data.  4: A communication error occurs.  1. Raise an error message.  5: The employee is already entered.  1. Inform the employee that the new employee is already entered.  2. Abandon the entry.  6: Data Consistency cannot be ensured.  1. The employee entry is rolled-back.  2. The employee is informed the employee cannot be inserted.  7: An error occurs storing the new employee’s details.  1. The employee entry is rolled-back.  2. Raise an error message. |
| Precondiciones | Verified employee. |
| Poscondiciones | New employee registered on the system |
| Requerimientos especiales |  |
| Prioridad | Essential |

|  |  |
| --- | --- |
| Nombre | [FR14] Update employee |
| Descripción | This use case allows of the employee's data to be updated on the system. |
| Actor | * Employee |
| Disparador |  |
| Flujo Básico | 1. The employee chooses the update employee option.  2. The employee provides the data to be updated:   * Name * New password (with second password field for confirmation) * Current password   3. The employee confirms the update.  4. The entered data is sent to the server.  5. The system verifies the entered data.  6. The system ensures the employee data is consistent.  7. The system stores the updated employee information. |
| Flujos Alternativos | On step 3, in case the name or the current password is missing/invalid, an error message should be showed. |
| Precondiciones | Verified employee. |
| Poscondiciones | Employee's data updated on the system |
| Requerimientos especiales |  |
| Prioridad | Essential |

|  |  |
| --- | --- |
| Nombre | [FR15] Update health unit |
| Descripción | This use case allows the health unit's data to be updated. |
| Actor | * Employee |
| Disparador |  |
| Flujo Básico | 1. The employee chooses the update health unit option.  2. The system retrieves the list of all health units.  3. The list of health units is returned to the employee.  4. The list of health units is formatted and displayed on the employee’s local display.  5. The employee selects the health unit to be updated.  6. The unique identifier for the selected health unit is sent to the server.  7. The system ensures the health unit data is consistent.  8. The system retrieves the data for the selected health unit.  9. The data retrieved is returned to the employee.  10. The health unit data is formatted and presented on the employee’s local display.  11. The employee alters the necessary data.  12. The updated information is sent to the server.  13. The system ensures the health unit data is left in a consistent state.  14. The system stores the updated health unit information. |
| Flujos Alternativos | 2, 8: A problem occurs retrieving the health unit information.  1. Raise an error message.  3, 6, 9, 12: A communication problem occurs.  1. Raise an error message.  7 and 13: Data consistency cannot be assured.  1. Any health unit updates are rolled-back.  2. Raise an error message.  14: A problem occurs storing the updated health unit data.  1. Any health unit updates are rolled-back.  2. Raise an error message. |
| Precondiciones | Verified employee. |
| Poscondiciones | Health unit's data updated on the system. |
| Requerimientos especiales |  |
| Prioridad | Essential |

|  |  |
| --- | --- |
| Nombre | [FR16] Change logged employee |
| Descripción | This use case allows the currently logged employee to be changed. |
| Actor | * Employee |
| Disparador |  |
| Flujo Básico | 1. The employee chooses the change logged employee option.  2. The system shows the login screen, and from this point on, the flow will follow the one described in [Login.FR10]. |
| Flujos Alternativos |  |
| Precondiciones | Verified employee. |
| Poscondiciones | • First employee signed out and new employee logged-in. |
| Requerimientos especiales |  |
| Prioridad | Essential |

# Bibliografía

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

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

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

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

5. **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.

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

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

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

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

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

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

12. *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.

13. **Sampaio, A.** EA-Miner: a Tool for Automating Aspect-Oriented Requirements Identification. *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering.* 2005.

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. **Dörr, J., et al.** Eliciting Efficiency Requirements with Use Cases. *9th International Workshop on Requirements Engineering. Foundation for Software Quality, REFSQ '03. Pre-Proceedings.* 2003. Vol. 3, pp. 23-32.

19. **Moreira, Ana, Araujo, Joao and Brito, Isabel.** Crosscutting quality attributes for requirements engineering. *Proceedings of the 14th international conference on Software engineering and knowledge engineering.* 2002. Vol. 27, pp. 167-174 .

20. **Chung, L., Mylopoulos, J. and Nixon, B.** Representing and using nonfunctional requirements: a process-oriented approach. Junio 1992. Vol. 18, pp. 483 - 497. 0098-5589.

21. **Malan, M. and Bredemeyer, D.** Definining Non-Functional Requirements. *http://www.bredemeyer.com/papers.htm.*

22. **Casamayor, Agustin, Godoy, Daniela and Campo, Marcelo.** Identification of non-functional requirements in textual specifications:A semi-supervised learning approach. *Information and Software Technology.* 2010. Vol. 52, pp. 436-445.

23. **Park, Sooyong, et al.** Using classification techniques for informal requirements. *Information and Software Technolog.* 2007. Vol. 49, pp. 1128-1140.

24. **Brill, E.** Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. *Computational Linguistics.* Diciembre 1995. Vol. 7, pp. 543 -565.

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. **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.

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

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

34. **Pinto, Monica, Gámez, Nadia and Fuentes, Lidia.** Towards the architectural definition of the Health Watcher system with AO-ADL. *Proceedings of the Early Aspects at ICSE: Workshops in Aspect-Oriented Requirements Engineering and Architecture Design.* 2007.

35. **Haimei, Zhang and Kerong, Ben.** Architectural Design of the Health Watch System with an Integrated Aspect­. *201O International Conference On Computer Design And Applications.* 2010.

36. **Massoni, Tiago, Soares, Sérgio and Borba, Paulo.** Healt Watcher Requirements Document V\_2. 2006.

37. **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 nombre y 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)