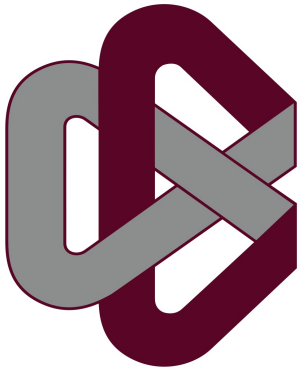


CIMAT

Centro de Investigación en Matemáticas, A.C.
Unidad Zacatecas

“Análisis de Patrones de Diseño Centrado en Atributos de Calidad”

Tesis
Para obtener el grado de

Maestro en Ingeniería de Software

Presenta
I.S.C. Ma. Del Rosario Marín P.

Directores
Dra. Perla Velasco Elizondo
M.S.I José Guadalupe Hernández Reveles

10 DE DICIEMBRE DE 2014

RESUMEN

En el contexto del diseño de la arquitectura de software los patrones de diseño son un concepto importante ya que, además de guiar la estructuración de un sistema, permiten promover atributos de calidad (ej. desempeño, disponibilidad, seguridad). Sin embargo, para diseñadores y arquitectos novatos la identificación y selección de patrones de diseño es frecuentemente costosa en tiempo y esfuerzo. En esta tesis se describe cómo usando técnicas de extracción de información y de representación del conocimiento es posible definir métodos y herramientas para analizar automáticamente textos de patrones de diseño escritos en lenguaje natural que permitan a diseñadores y arquitectos novatos a identificar patrones de forma menos costosa. Particularmente, en esta tesis se abunda sobre el caso del análisis automático de textos de patrones de diseño relacionados al atributo de calidad desempeño.

AGRADECIMIENTOS

A mi hijo, esposo, familia y amigos.

A mis directores de tesis Dra. Perla Velasco Elizondo y M.S.I José Guadalupe Hernández Reveles.

A CONACyT y COZCyT, por sus programas de becas, que soportan la formación de estudiantes e investigadores.

ÍNDICE

1 INTRODUCCIÓN

Este capítulo contiene una descripción general de este proyecto de tesis. Inicialmente se describe la importancia del diseño de arquitectura del software en este contexto. Posteriormente se describe la problemática a resolver, los objetivos, y finalmente la organización de este documento.

1.1 Contexto

Los sistemas de software permiten automatizar procesos o tareas específicas en muchos dominios de aplicación, reduciendo así el tiempo y esfuerzo que invierten las personas al realizar dichos procesos o tareas. Desde tiempo atrás se construyen sistemas de software observándose en los últimos años un aumento importante en su uso en muchos dominios de aplicación.

En términos generales, y con el propósito de promover su construcción sistemática, la construcción de un sistema de software se realiza atendiendo a un *ciclo de vida*. El ciclo de vida del desarrollo de software describe un conjunto de fases o etapas que permiten una evolución coherente y progresiva, independiente del enfoque de desarrollo que se utilice, del proceso de desarrollo de software. La Figura 1 ilustra las etapas, generalmente aceptadas, de este ciclo de vida. A continuación se describen de forma general estas etapas:

a) **Análisis**: Se enfoca en identificar el problema que se quiere resolver y la especificación de los comportamientos (ej. requerimientos de usuario o funcionales) y características (ej. requerimientos no funcionales) que debe satisfacer el sistema de software.
b) **Diseño**: Se enfoca en definir una solución, en términos de diversos modelos, para satisfacer los comportamientos y características especificados en la etapa de análisis.
c) **Implementación**: Se enfoca crear los programas que exhiban los comportamientos y características definidos en los modelos creados durante la etapa de diseño.
d) **Pruebas**: Se enfoca en evaluar que los programas construidos exhiban los comportamientos y características especificadas en los modelos creados durante la etapa de diseño y por ende satisfagan los requerimientos definidos en el análisis.
e) **Mantenimiento**: Se enfoca en atender todos los cambios que pueda requerir el sistema durante operación.

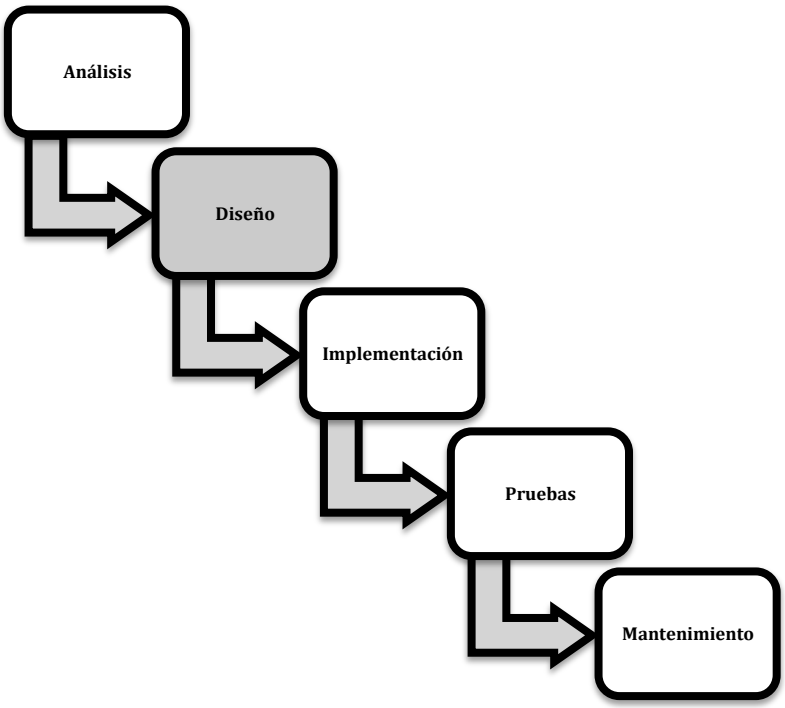


Figura 1. Ciclo de desarrollo de software.

En la Figura 1 se ha resaltado la fase de diseño debido a que el trabajo presentado en esta tesis tiene que ver con en esta etapa. Como ya mencionamos la etapa de diseño se enfoca en definir la solución, en términos de diversos modelos, para satisfacer los requerimientos en la etapa de análisis. El diseño puede entenderse entonces como una transformación de la información de los requerimientos a satisfacer a los modelos de solución que permiten satisfacerlos.

En la etapa de diseño se define la arquitectura del sistema de software, la cual es uno de los modelos más importantes de esta etapa. En las siguientes secciones definimos este modelo y los detalles relacionados a esta definición.

1.1.1 Arquitectura de Software

La *arquitectura de un sistema de software* es un modelo, que se genera en etapas tempranas de la fase de diseño, y está definido en términos de un conjunto de estructuras que comprenden componentes de software, las relaciones entre ellos, y las propiedades de ambos (componentes y relaciones) [1]. A continuación se describen los principales términos de esta definición:

Los *componentes* son los elementos que proveen comportamientos específicos. Por ejemplo, procesos o cálculos asociados a requerimientos de usuario o funcionales. Estos componentes pueden tomar diversas formas como por ejemplo módulos o subsistemas.

Las *relaciones* son las conexiones que comunican a los diversos componentes de la arquitectura. Existen distintos tipos de relaciones, por ejemplo “data-flow” o eventos.

Las *propiedades* son características que exhiben tanto los componentes como las relaciones. Generalmente estas características corresponden a requerimientos no funcionales como por ejemplo *atributos de calidad.* Ejemplos de atributos de calidad incluyen desempeño, seguridad, modificabilidad o disponibilidad.

Para ilustrar mejor los conceptos anteriores, en la Figura 2 se muestra la arquitectura de un compilador. Esta arquitectura está definida en términos componentes (cuadros) y las relaciones entre ellos (flechas).

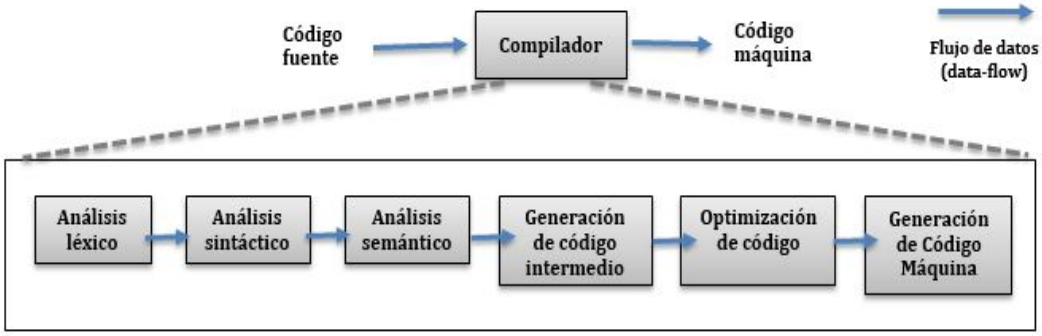


Figura 2. Arquitectura de un compilador.

Los componentes son módulos que realizan procesos específicos como por ejemplo análisis léxico que tiene que ver con de leer secuencias de caracteres de un código fuente, y dividirlo en tokens. También hay un componente de análisis sintáctico que revisa si los tokens corresponden a una gramática. Similarmente se tienen componentes para realizar el análisis semántico, generación de código intermedio, optimización de código y generación de código máquina, que son los procesos que se requieren en un compilador.

El tipo de relación que conecta a los componentes del ejemplo de la Figura 2, es de tipo “data-flow”. Esto significa que la ejecución de un componente depende del flujo o intercambio de datos que viene del componente anterior en la cadena de componentes en la arquitectura del compilador.

Un ejemplo de propiedad que se puede asociar a los componentes de esta arquitectura podría ser el desempeño, indicando que todos los componentes tienen un tiempo de ejecución no mayor a 200 milisegundos. En el caso de las relaciones se podría asociar la propiedad de seguridad, al requerirse un nivel de integridad particular en los datos que pasan de un componente a otro.

Como puede observarse la arquitectura es un modelo de alto nivel de abstracción y es importante porque representa la estructuración general de un sistema. La arquitectura guiará y restringirá el diseño detallado del sistema y eventualmente la implementación del mismo. La manera en que se estructura un sistema tiene un impacto directo sobre la capacidad de éste para satisfacer los requerimientos. Especialmente, los atributos de calidad (de los cuales hablaremos con mas detalle en el siguiente capítulo).

Las estructuras de la arquitectura se definen a partir de *decisiones de diseño* que toma un diseñador o *arquitecto de software*. Por simplicidad, en esta tesis nos referimos a esta persona como *arquitecto de software*. Las estructuras de la arquitectura, por lo general, se describen en representaciones gráficas denominadas vistas como la de la Figura 2.

1.2 Problemática

A raíz de la popularización del concepto de arquitectura de software se han generado diversos métodos para soportar el trabajo del arquitecto. En lo que se refiere a la creación de la arquitectura, existen varios métodos, por ejemplo: ADD [2] ACDM [3] o el de Rozanski y Woods [4]. En muchos de estos métodos las decisiones de diseño se articulan a partir de conceptos de diseño pre-existentes como *patrones* y *tácticas de diseño arquitectónico* (de los cuales hablaremos con mas detalle en el siguiente capítulo). De esta forma, el uso de estos métodos requiere que los arquitectos de software conozcan estos conceptos. Particularmente, que los entiendan con respecto a un conjunto de características. Por ejemplo, para el caso de los patrones de diseño, que son uno de los conceptos de diseño más sólidos y utilizados a la fecha en la práctica (existen muchos catálogos de patrones y muchos frameworks de desarrollo implementan varios de estos patrones), estos métodos asumen que, para la selección cierto patrón, los arquitectos conocen las respuestas a preguntas como:

a) ¿Cuáles son sus componentes principales? Por ejemplo, modelo, vista, controlador, cliente, servidor, etc.
b) ¿Cuáles son las configuraciones estructurales que permite? Por ejemplo, capas, estrella, peer to peer, etc.
c) ¿Cuál es el modelo computacional que utiliza? Por ejemplo, orientado a eventos, orientado a flujo de datos, etc.

d) ¿Cuáles son ejemplos comunes de su uso? Por ejemplo, sistemas transaccionales, sistemas de control, etc.
e) ¿Qué tecnologías de desarrollo lo implementan? Por ejemplo, Hibernate [5], Spring [6], etc.
f) ¿Qué atributos de calidad satisface o inhibe? Por ejemplo, tolerancia a fallos, rendimiento, etc.

En la práctica, sin embargo, esto no es así; o al menos no es así para *arquitectos novatos*. En la práctica pocos arquitectos de software han recibido una educación formal en arquitectura de software y muchos de ellos se han formado empíricamente a través de muchos años en la práctica. Entonces para usar alguno de los métodos antes mencionados, los diseñadores o arquitectos novatos tendrían que realizar primero la búsqueda de patrones, luego entenderlos considerando características como las mencionadas antes, y finalmente seleccionar los adecuados (dentro del contexto de un método de diseño) para definir la arquitectura del sistema en turno.

Estas actividades son generalmente costosas en tiempo y esfuerzo debido a los siguientes factores:

a) El número de patrones que existe actualmente
b) El surgimiento de nuevos patrones
c) La descripción heterogénea de los patrones
d) El idioma en el que están escritos los patrones
e) La falta de herramientas para soportar estas tareas

A continuación explicamos cada uno de estos factores.

a) El número de patrones que existe actualmente.

Actualmente existen muchos catálogos de patrones, por ejemplo, Pattern-Oriented Software Architecture [7], Design Patterns: Elements of Reusable Object-Oriented Software [8], SOA Design Patterns [9], Patterns of Enterprise Application Architecture [10] o Service Design Patterns [11]. Muchos de estos catálogos describen más de 100 patrones. Cada patrón en promedio está descrito en 2 páginas de texto. Existen estimaciones de que el tiempo de lectura de una página de texto puede ir desde 2 a 6 minutos, dependiendo si es material técnico o no [12]. Si tomamos un valor intermedio de 8 minutos por patrón, puesto que un patrón está normalmente descrito en 2 páginas, leer solo un libro de 100 patrones le tomaría a un arquitecto 13 horas.

b) El surgimiento de nuevos patrones.

Con el surgimiento de nuevos paradigmas de diseño y desarrollo de software, como por ejemplo SOA (Service Oriented Architecture), nuevos patrones arquitectónicos han sido definidos [9], [11]. De esta forma, el material que tendría

que leer un arquitecto novato en un tiempo determinado generalmente siempre es más que en un tiempo anterior.

c) La descripción heterogénea de los patrones.

Aunque existe un consenso en los elementos de información que se deben incluir en la descripción de un patrón (ej. nombre del patrón, problema que resuelve, contexto de aplicación, restricciones, etc.), la descripción de estos elementos es heterogénea pues no se hace atendiendo a un formato, plantilla o estilo de redacción estándar. Ya hemos mencionado que los atributos de calidad son características de calidad que los clientes esperan del sistema y que son determinantes para la selección de un patrón. Lamentablemente este tipo de información muchas veces no viene de forma explícita u homogénea en la descripción del patrón. Por ejemplo en diferentes catálogos de patrones podemos encontrar el uso de las palabras “rapidez”, “concurrencia”, “rendimiento” y “latencia” para indicar que el patrón promueve o inhibe un atributo de calidad relacionado al desempeño de un sistema.

d) El idioma en el que están escritos los patrones.

Ya hemos mencionado algunas métricas relacionadas a lectura de una página de texto. Sin embargo, estas métricas asumen que el texto está escrito en el idioma nativo del lector. Esta situación no aplicaría en nuestro contexto, México, puesto que las fuentes originales de los catálogos de los patrones están escritas en el idioma inglés. Consideramos que este es otro aspecto que podría incrementar el tiempo requerido para leer un patrón.

e) La falta de herramientas para soportar estas tareas.

Actualmente existen diversas herramientas que ayudan al arquitecto en la selección de patrones durante el diseño de arquitectura, por ejemplo [13], [14], [15]. Sin embargo, es muy importante resaltar que la selección de patrones se hace dentro del contexto de un repositorio de patrones pre-definido y en la mayoría de los casos estática. Esto es, el repositorio viene “de fábrica” con la herramienta y regularmente no existen facilidades para agregar nuevos patrones dentro de él. En los casos en los que se pueden agregar patrones, el análisis, clasificación y almacenamiento se hace de forma “manual”.

Este proyecto de tesis está orientado a minimizar algunos de estos factores.

1.3 Objetivos del Proyecto de Tesis

1.3.1 Objetivo General

El objetivo general de este proyecto es asistir a los arquitectos novatos en la identificación y selección de patrones durante el diseño de la arquitectura.

Lo patrones están escritos en lenguaje natural en el idioma inglés. La selección de patrones está determinada por la satisfacción de un atributo de calidad.

Por cuestiones de practicidad y alcance, en este trabajo se decidió trabajar con el atributo de *desempeño* debido a que es un atributo de calidad importante para prácticamente cualquier tipo de sistema y existen muchos patrones de diseño que promueven este atributo de calidad. El desempeño tiene que ver con el grado en el cual un sistema ejecuta sus operaciones dentro de restricciones de tiempo establecidas.

1.3.2 Objetivos Específicos

Considerando el objetivo general se tienen los siguientes objetivos específicos:

Definir un método y una herramienta de soporte para:

a) permitir el análisis automático de patrones de diseño arquitectónico, descritos en lenguaje natural en el idioma inglés, con el propósito de responder a la pregunta si un determinado patrón satisface o no el atributo de calidad desempeño.

b) lograr la escalabilidad del análisis con respecto a la heterogeneidad de la redacción del texto del patrón.

c) disminuir el tiempo que invierte un arquitecto novato en el análisis "manual" del texto de un patrón.

d) tener una exactitud en el resultado del análisis cercana a la que tendría un arquitecto experimentado.

1.4 Organización del Documento de Tesis

En el Capítulo 2 se cubren los conceptos necesarios para comprender el método de análisis propuesto. En el Capítulo 3 se describen algunos trabajos relacionados a este trabajo de tesis. En el Capítulo 4 se detalla el método de análisis propuesto. En el Capítulo 5 se describe la herramienta de soporte desarrollada. En el Capítulo 6 se describe la evaluación del método propuesto. El Capítulo 7 contiene un resumen de este trabajo y propuestas de trabajo futuro.

2 MARCO TEÓRICO

En este capítulo se explican los conceptos necesarios para comprender la solución propuesta al problema que se aborda en esta tesis. Específicamente, se describe los siguientes conceptos: *atributos de calidad, conceptos de diseño* haciendo especial énfasis en el de *patrón,* y finalmente *extracción de información* y *representación del conocimiento.* Para muchos de estos conceptos hemos conservado la redacción original en inglés pues, como le indicamos en el capítulo anterior, en este trabajo se analizarán descripciones en este idioma.

2.1 Atributos de Calidad

Como se explicó en el Capítulo 1, la arquitectura de software es un modelo importante porque representa la estructura de un sistema y la manera en que se estructura un sistema tiene un impacto directo sobre la capacidad de éste para satisfacer los requerimientos del sistema, particularmente los atributos de calidad.

Los atributos de calidad especifican características útiles para establecer criterios sobre la calidad de un sistema. Actualmente no existe un listado único de atributos de calidad relevantes para el diseño de un sistema pues esto depende generalmente del dominio de aplicación y los objetivos de negocio que se quieren alcanzar. Sin embargo, la Tabla 1 muestra un listado de atributos de calidad comúnmente utilizados.

Atributo de Calidad	Descripción
Desempeño (Performance)	Grado en el cual un sistema ejecuta sus operaciones dentro de restricciones de tiempo establecidas.
Eficiencia (Efficiency)	Grado en el cual un sistema hace uso adecuado de recursos como disco memoria o procesador.
Disponibilidad (Availability)	Se refiere a habilidad de un sistema para tolerar fallas y las consecuencias asociadas.
Seguridad (Security)	Denota la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.
Usabilidad (Usability)	Se refiere a qué tan sencillo les resulta a los usuarios realizar operaciones con el sistema.
Modificabilidad (Modifiability)	Grado en el cual es posible realizar cambios al sistema.
Capacidad de prueba (Testability)	Se refiere a la facilidad con la que el sistema, al ser sometido a una serie de pruebas, puede exhibir sus fallas.
Interoperabilidad (Interoperability)	Denota la capacidad de un sistema para trabajar con otros componentes y/o sistemas.

Tabla 1. Atributos de calidad comúnmente utilizados en el diseño de sistemas.

Es importante aclarar que en la definición del atributo de calidad desempeño muchas veces incluye el atributo definido en la Tabla 1 como Eficiencia. Sin embargo, esta interpretación no se considerará en este trabajo.

Existen propuestas formales sobre conjuntos de atributos de calidad de distintos autores, para los cuales el número, e incluso los nombres usados para denotar a cada atributo de calidad puede variar. Estas propuestas han sido difundidas

como *modelos de atributos de calidad*. En la siguiente sección mostramos algunos de los más representativos.

2.1.1 Modelos de Atributos de Calidad

En esta sección se muestran algunos de los modelos de calidad más representativos: *Dromey. McCall*, *FURPS* e *ISO/IEC 9126* [16] [17]. Aunque se incluyen otros atributos, nuestro objetivo principal es comunicar la manera en que en cada modelo se hace referencia al atributo de calidad *desempeño*.

2.1.1.1 Modelo de Dromey

Como se muestra en la Tabla 2, el *Modelo de Dromey* sugiere el uso de cuatro categorías que para organizar *propiedades del producto*: correctitud, internas, contextuales y descriptivas.

Propiedades del producto	Atributos de Calidad
Correctitud (Correctness)	Funcionalidad Confiabilidad
Internas (Internal)	Mantenibilidad **Eficiencia** Confiabilidad
Contextuales (Contextual)	Mantenibilidad Reusabilidad Portabilidad Confiabilidad
Descriptivas (Descriptive)	Mantenibilidad Reusabilidad Portabilidad Usabilidad

Tabla 2. Modelo de Dromey.

En este modelo el atributo de calidad *desempeño* se encuentra dentro de la categoría de *propiedades Internas* del producto con el término de *Eficiencia.*

2.1.1.2 Modelo de McCall

En el *Modelo de McCall* se utiliza el término *factor de calidad* para denotar las características de calidad principales que un producto debe exhibir. Cada factor tiene un conjunto de atributos denominados *criterios de calidad.*

Como se observa en la Tabla 3, el modelo de McCall, tiene 11 factores de calidad. En la tabla se puede ver que algunos de los criterios de calidad son compartidos por más de un factor.

Factor	Criterio de Calidad
1. Correctitud (correctness)	1. Rastreabilidad 2. Completitud 3. Consistencia
2. Confiabilidad (Reliability)	4. Consistencia 5. Exactitud 6. Tolerancia a fallas
3. Eficiencia (Efficiency)	**7. Eficiencia de ejecución** 8. Eficiencia de almacenamiento
4. Integridad (Integrity)	9. Control de acceso 10. Auditoría de acceso
5. Usabilidad (Usability)	11. Operabilidad 12. Comunicación 13. Entrenamiento
6. Mantenibilidad (Maintainability)	14. Simplicidad 15. Concreción
7. Capacidad de Prueba (Testability)	16. Simplicidad 17. Instrumentación 18. Auto-descriptividad 19. Modularidad
8. Flexibilidad (Flexibility)	20. Auto-descriptividad 21. Capacidad de expansión 22. Generalidad 23. Modularidad
9. Portabilidad (Portability)	24. Auto-descriptividad 25. Independencia del sistema 26. Independencia de máquina
10. Reusabilidad (Reusability)	27. Auto-descriptividad 28. Generalidad 29. Modularidad 30. Independencia del sistema 31. Independencia de máquina
11. Interoperabilidad (Interoperability)	32. Modularidad 33. Similitud de comunicación 34. Similitud de datos

Tabla 3. Modelo de McCall.

En este modelo se usa el factor Eficiencia y el criterio “Eficiencia de ejecución” para referirse a la noción que en este trabajo tenemos desempeño.

2.1.1.3 Modelo de FURPS

El modelo de McCall ha servido de base para modelos de calidad posteriores, y este es el caso del modelo FURPS. Como se ilustra en la Tabla 4, en este modelo se desarrollan un conjunto de factores de calidad de software, bajo el acrónimo de FURPS: funcionalidad (Functionality), usabilidad (Usability), confiabilidad (Reliability), desempeño (Performance) y capacidad de soporte (Supportability). Cada factor define un conjunto de atributos.

Factor de Calidad	Atributos
a) Funcionalidad (Functionality)	✓ Características y capacidades del programa ✓ Generalidad de las funciones ✓ Seguridad del sistema
b) Facilidad de uso (Interoperability)	✓ Factores humanos ✓ Factores estéticos ✓ Consistencia de la interfaz ✓ Documentación
c) Confiabilidad (Reliability)	✓ Frecuencia y severidad de las fallas ✓ Exactitud de las salidas ✓ Tiempo medio de fallos ✓ Capacidad de recuperación ante fallas ✓ Capacidad de predicción
d) Rendimiento (Efficiency)	✓ Velocidad del procesamiento ✓ **Tiempo de respuesta** ✓ Consumo de recursos ✓ **Rendimiento efectivo total** ✓ **Eficacia**
e) Capacidad de Soporte (Maintainability)	✓ Extensibilidad ✓ Adaptabilidad ✓ Capacidad de pruebas ✓ Capacidad de configuración ✓ Compatibilidad ✓ Requisitos de instalación

Tabla 4. Modelo de FURPS.

En el modelo de FURPS, el atributo de calidad desempeño se encuentra definido como factor de calidad con el nombre de Rendimiento y los atributos según la definición usada son: Tiempo de respuesta, Rendimiento efectivo total y Eficacia.

2.1.1.4 Modelo ISO/IEC 9126

El estándar ISO/IEC 9126 ha sido desarrollado en un intento de identificar los atributos clave de calidad para un producto de software. Este estándar es una simplificación del Modelo de McCall, e identifica seis características básicas de calidad que pueden estar presentes en cualquier producto de software. Como se observa en la Tabla 5, el estándar provee una descomposición de las características en subcaracterísticas.

Característica	Subcaracterística
– **Funcionalidad (Functionality)**	✓ Adecuación ✓ Exactitud ✓ Interoperabilidad ✓ Seguridad
– **Confiabilidad (Reliability)**	✓ Madurez ✓ Tolerancia a fallas ✓ Recuperabilidad
– **Usabilidad (Usability)**	✓ Entendibilidad ✓ Capacidad de aprendizaje ✓ Operabilidad
– **Eficiencia (Efficiency)**	✓ **Comportamiento en tiempo** ✓ Comportamiento de recursos
– **Mantenibilidad (Maintainability)**	✓ Analizabilidad ✓ Modificabilidad ✓ Estabilidad ✓ Capacidad de pruebas
– **Portabilidad (Portability)**	✓ Adaptabilidad ✓ Capacidad de instalación ✓ Capacidad de remplazamiento

Tabla 5. Modelo ISO/IEC 9126.

Para el modelo ISO/IEC 9126 el atributo de calidad desempeño se encuentra definido como una característica que lleva el nombre de *Eficiencia* y como subcarecterística C*omportamiento en tiempo*.

2.2 Métricas

En la sección anterior explicamos que los atributos de calidad denotan características para establecer la calidad de un sistema. Por ello, es importante que estas características puedan medirse de forma cuantitativa. Existe una gran cantidad de métricas que se pueden usar para este propósito. Algunas de las más clásicas, asociadas a los atributos de calidad descritos en la sección anterior, se muestran en la Tabla 6. En la tabla se resaltan las métricas definidas para el atributo de calidad desempeño.

Atributo de Calidad	Nombre Formal de la Métrica	Interpretación
Desempeño (Performance)	**Latencia (Latency)**	**Tiempo de respuesta por unidad de tiempo (minutos, segundos, etc.)**
	Carga (*Throughput*)	**Tiempo de respuesta operaciones, transacciones por unidad de tiempo.**
Eficiencia (Efficiency)	Utilización (Utilization)	Porcentaje de uso de recursos: • Uso de disco • Uso de memoria • Uso de procesador
Disponibilidad (Availability)	MTTF (Mean time to failure)	Tiempo promedio que el sistema ha dejado de funcionar.
	MTBF (Mean Time between Failures)	Tiempo promedio que excluye el tiempo invertido en la reparación, es decir solo se centra en medir el tiempo que el sistema esta disponible y en operación.
	Mean Time Between Replacements	Tiempo promedio entre dos eventos, donde se tienen que remplazar componentes del sistema. Ej. En una maquina un bulbo se tiene que remplazar después de cada cierta cantidad de horas en operación o por una falla.
	MTBF with Scheduled Replacements	Esta métrica es usada en las mismas condiciones de tiempos de remplazos. Aquí se describe el tiempo promedio de remplazo de dos fallas consecutivas.
	Disponibilidad No-Disponibilidad	Es el tiempo o bien porcentaje en el que un sistema se encuentra disponible o bien No-disponible.
Seguridad (Security)	n/a	Tiempo/Esfuerzo requerido parasortear medidas de seguridad con éxito o para reponerse de un ataque.
	n/a	Cantidad de recursos/servicios/datos disponibles después de un ataque.
	n/a	Probabilidad de detectar un ataque. Probabilidad de identificar el responsable de un ataque.
	n/a	Número de ataques resistidos/ o número datos que son vulnerables a un ataque

Tabla 6. Ejemplos de métricas para atributos de calidad.

Atributo de Calidad	Nombre Formal de la Métrica	Interpretación
Usabilidad (Usability)	n/a	Tiempo para ejecutar una acción en el sistema.
	n/a	Número de errores reportados/ Problemas resueltos.
	n/a	Razón porcentaje de operaciones exitosa.
	n/a	Nivel de satisfacción o confianza en el sistema.
Modificabilidad (Modifiability)	n/a	Tiempo requerido para realizar un cambio.
	n/a	Costo de realizar un cambio.
	n/a	Número de defectos introducidos al realizar un cambio.
Capacidad de prueba (Testability)	n/a	Tiempo/Esfuerzo para encontrar un defecto.
	n/a	Probabilidad para encontrar un defecto.
Interoperabilidad (Interoperability)	n/a	Porcentaje de información intercambiada correctamente.
	n/a	Porcentaje de información intercambiada incorrectamente.

Tabla 6. Ejemplos de métricas para atributos de calidad (continuación).

2.3 Trade-offs

Es importante tener en cuenta que durante el diseño de un sistema no puede lograrse la satisfacción todos los atributos de calidad al 100%. Satisfacer cierto atributo de calidad puede tener efectos positivos o negativos sobre otros atributos que, de alguna manera, también se desean satisfacer. En el contexto de diseño de sistemas a este fenómeno se le conoce como "*trade-off*". Un arquitecto de software, idealmente, debería conocer sobre "trade-offs".

En la Figura 3, se presentan algunos "trade-offs" entre atributos de calidad [18]. Específicamente se ilustra el impacto, [+] positivo o [-] negativo) que tendría promover un atributo de calidad (renglones) y las consecuencias asociadas a los otros atributos de calidad (columnas).

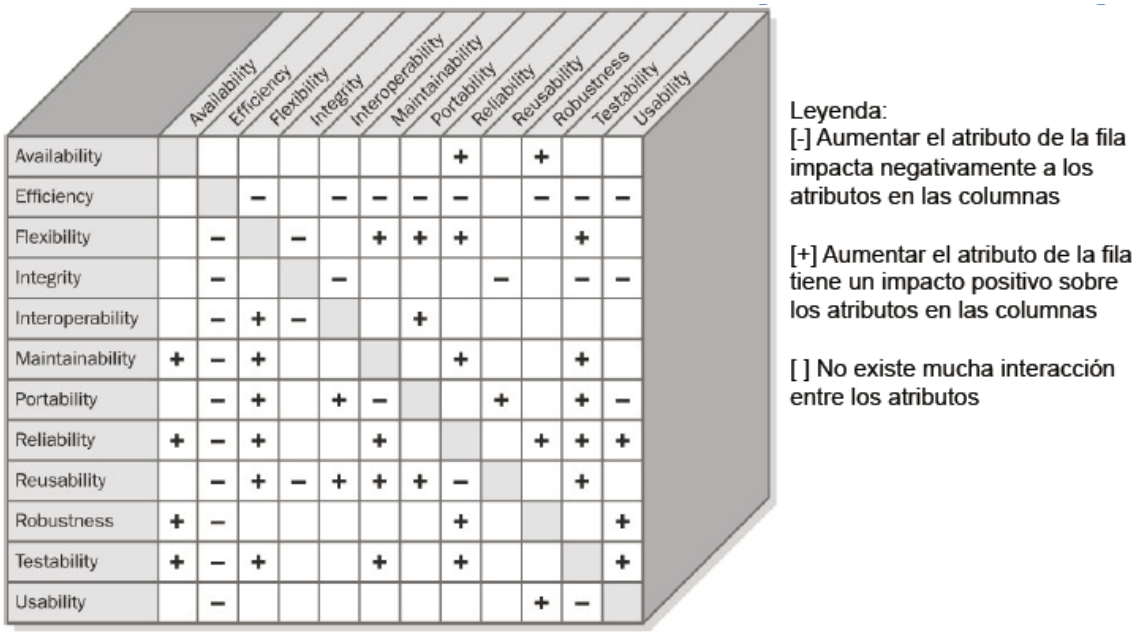


Figura 3. Trade-offs de atributos de calidad.

Aunque no se muestra en la figura anterior, se sabe que si tomamos decisiones de diseño para promover la seguridad esto impactará negativamente sobre el atributo desempeño debido al incremento de operaciones en el sistema por parte de los mecanismos de seguridad.

2.4 Conceptos de Diseño

Como explicamos en el Capítulo 1, la manera en que se estructura un sistema tiene un impacto directo sobre la capacidad de éste para satisfacer los atributos de calidad. También comentamos que para definir las estructuras de un sistema esto es, la arquitectura, es recomendable que el arquitecto tome decisiones de diseño que consideren conceptos de diseño pre-existentes. En esta sección explicamos los conceptos de diseño relevantes a este trabajo: patrones y tácticas de diseño arquitectónico. También se discute la relación de estos conceptos con los atributos de calidad.

2.4.1 Tácticas de Diseño Arquitectónico

Las tácticas de diseño arquitectónico son conceptos de diseño estrechamente ligados al concepto de atributo de calidad puesto que son estrategias permiten al arquitecto alcanzarlos. Esto es, las tácticas son técnicas probadas de las ciencias de la computación que permiten promover ciertos atributos de calidad durante el diseño de la arquitectura.

En [1] se describe un catálogo de tácticas de diseño. El catálogo muestra las tácticas de forma gráfica como una serie de árboles en cuya raíz se encuentra una categoría de atributo de calidad particular. Las tácticas se encuentran organizadas en estrategias. El catálogo considera tácticas para siete atributos de calidad: Desempeño, Disponibilidad, Seguridad, Modificabilidad, Usabilidad, Facilidad de Pruebas e Interoperabilidad. En la Figura 4 se muestran las tácticas para el atributo de calidad de desempeño y esta es la definición de la táctica Introducir Concurrencia (Introduce Concurrency):

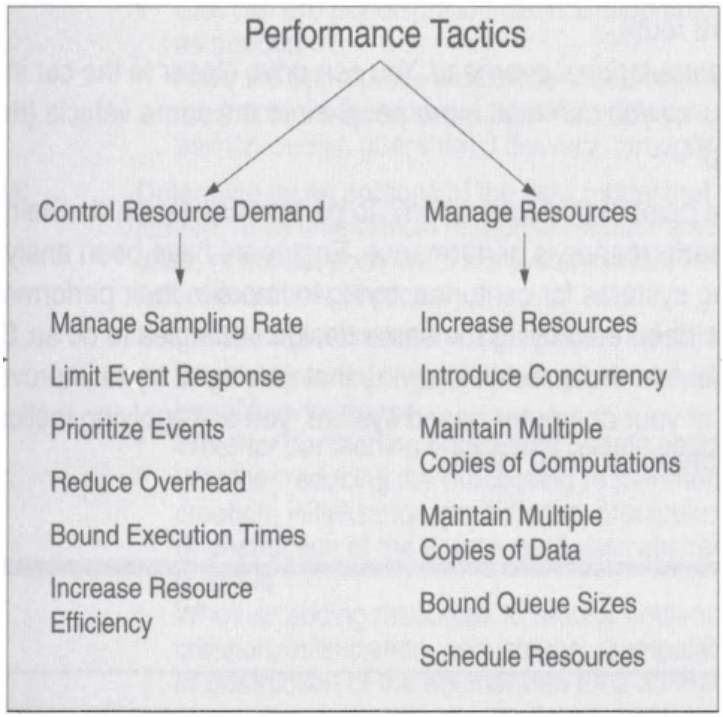


Figura 4. Tácticas de desempeño.

"If requests can be processed in parallel, the blocked time can be reduced. Concurrency can be introduced by processing different streams of events on different threads or by creating additional threads to process different sets of activities. Once concurrency has been introduced, scheduling policies can be used to achieve the goals you find desirable. Different scheduling policies may maximize fairness (all requests get equal time), throughput (shortest time to finish first), or other goals."

2.4.2 Patrones de Diseño Arquitectónico

En términos generales, los patrones de diseño de software son estructuras que representan soluciones genéricas a problemas recurrentes de diseño. De esta manera dichas estructuras se pueden aplicar a problemas de diseño de software similares. Se puede decir que un patrón de diseño es conocimiento pre-existente y documentado. Existen cientos de patrones que pueden aplicarse en diferentes

momentos del diseño de sistemas. Como el nombre sugiere, los patrones arquitectónicos son patrones orientados al diseño de la arquitectura.

En contraste con las tácticas, los patrones de diseño de arquitectura son conceptos de diseño más detallados. Mientras las tácticas de diseño representan el ¿qué?, los patrones podrían representar el ¿cómo? Esto es, los patrones describen con cómo deben de ser implementados. Se debe notar también que un patrón podría implementar una o más tácticas, incuso de diferentes categorías.

Ya comentamos que las soluciones conceptuales descritas en un patrón son sintácticamente heterogéneas en el sentido de que no se elaboran atendiendo a un formato, plantilla o estilo de redacción estándar. Sin embargo, desde el punto de vista semántico las descripciones generalmente incluyen los siguientes elementos de información:

Contexto: los ámbitos de diseño y escenarios en los que esencialmente se puede aplicar el patrón. Por ejemplo: sistemas distribuidos, sistemas en capas, etc.

Problema: una descripción general de la problemática que pretende resolver el patrón.

Fuerzas: se mencionan las restricciones asociadas a la uso del patrón.

Solución: se definen las tareas la para la implementación del modelo del patrón.

Diagrama de solución: una representación gráfica para la descripción del patrón ilustrando por ejemplo los tipos de componentes y relaciones, etc.

Consecuencias de la solución: las consecuencias de la selección del patrón, algunas veces se incluye el contraste con otros patrones unificando la siguiente sección de patrones relacionados.

Patrones relacionados: se hace mención de otros patrones, comparando las maneras de implementación con las que resuelven el problema planteado. Aquí los patrones pueden aparecer en una simple lista o bien se puede dar una explicación más profunda.

A continuación se muestra un ejemplo de patrón arquitectónico en términos de los elementos de información antes descritos. El patrón es *Pipes and Filters* [19].

Contexto: Pipes and Filters

→ *Data Flow Architecture*

You have an integration solution that consists of several financial applications. The applications use a wide range of formats—such as the Interactive Financial Exchange (IFX) format, the Open Financial Exchange (OFX) format, and the Electronic Data Interchange (EDI) format—for the messages that correspond to payment, withdrawal, deposit, and funds transfer transactions.
Integrating these applications requires processing the messages in different ways. For example, converting an XML-like message into another XML-like message involves an XSLT transformation. Converting an EDI data message into an XML-like message involves a transformation engine and transformation rules. Verifying the identity of the sender involves verifying the digital signature attached to the message. In effect, the integration solution applies several transformations to the messages that are exchanged by its participants.

Problema:

→ How do you implement a sequence of transformations so that you can combine and reuse them independently?

Fuerzas:

→ Implementing transformations that can be combined and reused in different applications involves balancing the following forces:

- Many applications process large volumes of similar data elements. For example, trading systems handle stock quotes, telecommunication billing systems handle call data records, and laboratory information management systems (LIMS) handle test results.
- The processing of data elements can be broken down into a sequence of individual transformations. For example, processing XML messages typically involves a series of XSLT transformations.
- The functional decomposition of a transformation f(x) into g(x) and h(z) (where f[x]=g[x]?h[z]) does not change the transformation. However, when separate components implement g and h, the communication between them (that is, passing the output of g[x] to h[z]) incurs overhead. This overhead increases the latency of a g(x)?h(z) implementation compared to an f(x) implementation.

Solución:

→ Implement the transformations by using a sequence of filter components, where each filter component receives an input message, applies a simple transformation, and sends the transformed message to the next component. Conduct the messages through pipes that connect filter outputs and inputs and that buffer the communication between the filters.
The left side of Figure 1 shows a configuration that has two filters. A source application feeds messages through the pipe into filter 1.
The filter transforms each message it receives and then sends each transformed message as output into the next pipe. The pipe carries the transformed message to filter 2. The pipe also buffers any messages that filter 1 sends and that filter 2 is not ready to process. The second filter then applies its transformation and passes the message through the pipe to the sink application. The sink application then consumes the message. This configuration requires the following:

2 The output of the source must be compatible with the input of filter 1.
3 The output of filter 1 must be compatible with the input of filter 2.
4 The output of filter 2 must be compatible with the input of the sink.

Diagrama de solución:

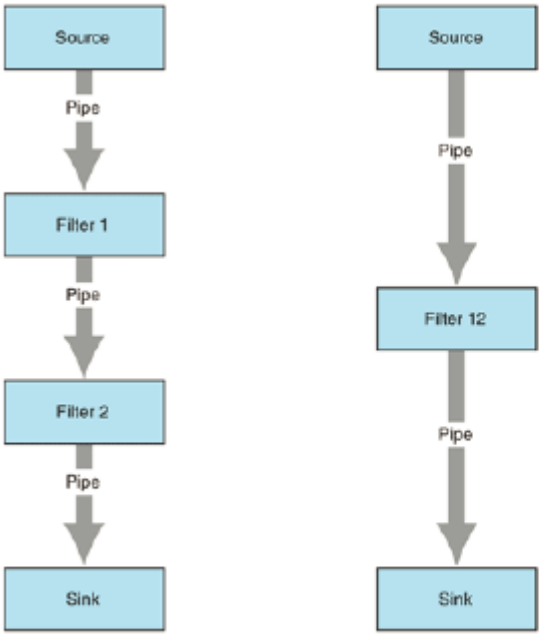


Figure 1. Using Pipes and Filters to break processing into a sequence of simpler transformations

The right side of Figure 1 shows a single filter. From a functional perspective, each configuration implements a transfer function. The data flows only one way and the filters communicate solely by exchanging messages. They do not share state; therefore, the transfer functions have no side effects. Consequently, the series configuration of filter 1 and filter 2 is functionally equivalent to a single filter that implements the composition of the two transfer functions (filter 12 in the figure).

Comparing the two configurations illustrates their tradeoffs:

4 The two-filter configuration breaks the transformation between the source and the sink into two simpler transformations. Lowering the complexity of the individual filters makes them easier to implement and improves their testability. It also increases their potential for reuse because each filter is built with a smaller set of assumptions about the environment that it operates in.

5 The single-filter configuration implements the transformation by using one specialized component. The one hop that exists between input and output and the elimination of the interfilter communication translate into low latency and overhead.

In summary, the key tradeoffs in choosing between a combination of generic filters and a single specialized filter are reusability and performance.

In the context of pipes and filters, a transformation refers to any transfer function that a filter might implement. For example, transformations that are commonly used in integration solutions include the following:

1. Conversion, such as converting Extended Binary Coded Decimal Interchange Code (EBCDIC) to ASCII.
2. Enrichment, such as adding information to incoming messages.
3. Filtering, such as discarding messages that match a specific criteria.

4. Batching, such as aggregating 10 incoming messages and sending them together in a single outgoing message.
5. Consolidation, such as combining the data elements of three related messages into a single outgoing message.

Consecuencias de la solución:

→ Using Pipes and Filters results in the following benefits and liabilities:

Benefits

1. Improved reusability. Filters that implement simple transformations typically encapsulate fewer assumptions about the problem they are solving than filters that implement complex transformations. For example, converting a message from one XML encapsulation to another encapsulates fewer assumptions about that conversion than generating a PDF document from an XML message. The simpler filters can be reused in other solutions that require similar transformations.
2. Improved performance. A Pipes and Filters solution processes messages as soon as they are received. Typically, filters do not wait for a scheduling component to start processing.
3. Reduced coupling. Filters communicate solely through message exchange. They do not share state and are therefore unaware of other filters and sinks that consume their outputs. In addition, filters are unaware of the application that they are working in.
4. Improved modifiability. A Pipes and Filters solution can change the filter configuration dynamically. Organizations that use integration solutions that are subject to service level agreements usually monitor the quality of the services they provide on a constant basis. These organizations usually react proactively to offer the agreed-upon levels of service. For example, a Pipes and Filters solution makes it easier for an organization to maintain a service level agreement because a filter can be replaced by another filter that has different resource requirements.

Liabilities

a) Increased complexity. Designing filters typically requires expert domain knowledge. It also requires several good examples to generalize from. The challenge of identifying reusable transformations makes filter development an even more difficult endeavor.
b) Lowered performance due to communication overhead. Transferring messages between filters incurs communication overhead.
c) This overhead does not contribute directly to the outcome of the transformation; it merely increases the latency.
d) Increased complexity due to error handling. Filters have no knowledge of the context that they operate in. For example, a filter that enriches XML messages could run in a financial application, in a telecommunications application, or in an avionics application.
e) Error handling in a Pipes and Filters configuration usually is cumbersome.
f) Increased maintainability effort. A Pipes and Filters configuration usually has more components than a monolithic implementation (see Figure 2). Each component adds maintenance effort, system management effort, and opportunities for failure.
g) Increased complexity of assessing the state. The Pipes and Filters pattern distributes the state of the computation across several components. The distribution makes querying the state a complex operation.

→ For more information about Pipes and Filters, see the following related patterns:

Patrones relacionados:

- Implementing Pipes and Filters with BizTalk Server 2004. This pattern uses the Global Bank scenario to show how you can use BizTalk Server 2004 to implement Pipes and Filters.
- Pipes and Filters [Shaw96, Buschmann96, Hohpe03].
- Intercepting Filter [Trowbridge03]. This version of Intercepting Filter discusses the pattern in the context of Web applications built using the Microsoft .NET Framework. Developers can chain filters to implement preprocessing and post-processing tasks such as extracting header information and rewriting URLs.
- In-band and Out-of-band Partitions [Manolescu97]. This pattern remedies the lack of a component that has a global context in Pipes and Filters systems. The out-of-band partition is context-aware; therefore, it can configure the filters and handle errors.

2.5 Representación del Conocimiento

La representación del conocimiento es un área de ciencias de la computación cuyo objetivo es representar el conocimiento de un dominio de manera que facilite la inferencia a partir de dicho conocimiento. Básicamente es un medio para permitir que la información que convierta en conocimiento cuando se le da una interpretación.

La representación del conocimiento es utilizada para crear esquemas de datos en un dominio o ámbito de conocimiento, agregando su significado y relación, de tal manera que facilite su procesamiento y análisis por distintos agentes (máquinas, personas, etc.), y se permita la reutilización de ese conocimiento con diferentes fines. Actualmente la representación del conocimiento es muy utilizada en la web semántica, pero su uso en otras áreas también es popular.

Existen diferentes métodos de representación del conocimiento que ayudan a la representación de los conceptos de un dominio. En este trabajo se ha seleccionado el uso de *ontologías*. Una ontología es un modelo que resulta de seleccionar un dominio y aplicar sobre él un método con el fin de obtener una representación de los conceptos que contiene (también llamados clases) y de las relaciones que existen entre dichos conceptos con la finalidad de facilitar la comunicación y el intercambio de información entre diferentes sistemas y entidades.

Hemos comentado que el objetivo general de este proyecto es asistir a los arquitectos novatos en la selección de patrones que promueven el atributo de calidad desempeño. Actualmente, los datos relacionados a este atributo de calidad no se encuentran de una manera estructurada (por ejemplo una base de datos) como para facilitar su procesamiento y análisis. Entonces en este trabajo se formularon dos ontologías.

La primera ontología la definimos para la representar conceptos clave relacionados al atributo de calidad desempeño. Los conceptos los hemos extraído de modelos de calidad que incluyen el atributo desempeño así como tácticas, *métricas* y *trade-offs* relacionados este atributo.

La segunda ontología la definimos para la representar palabras comúnmente utilizadas en sentencias en las que se habla positiva o negativamente del atributo de calidad desempeño. Estas palabras las hemos extraído de los textos de los patrones e incluyen sustantivos, verbos, conjunciones, etc.

Para dar una idea de la estructura de una ontología, la Figura 5 muestra gráficamente los conceptos relacionados a la primera ontología.

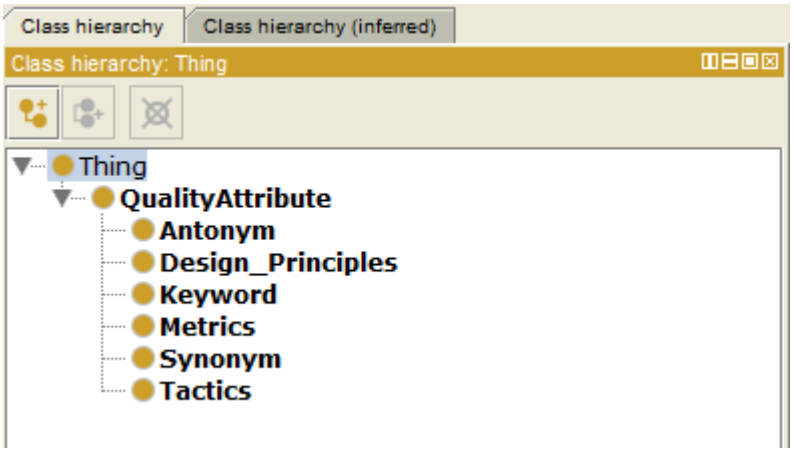


Figura 5. Ejemplo gráfico de la estructura de una ontología.

Para modelar un dominio se dispone de lenguajes y estándares de representación como RDF y OWL (basados en XML) y tecnologías como Protégé [20].

2.6 Extracción de Información

El procesamiento de lenguaje natural es un área de ciencias de la computación cuyo objetivo es estudiar las interacciones entre las computadoras y el lenguaje humano. Dentro de esta área se encuentra la extracción de información que se ocupa de recuperar tipos predefinidos de información desde información textual. Entre las aplicaciones comunes de la extracción de información están la búsqueda automática en pasajes de texto considerando información sintáctica y semántica.

La importancia de la extracción de información viene dada por la creciente cantidad de información no estructurada (por ejemplo, sin metadatos asociados)

existente en internet. Todo este conocimiento podría ser más accesible si se transformara a una forma relacional o fuese marcado utilizando etiquetas, como los tags que usa XML.

Un ejemplo de extracción de información sería la identificación de instancias de corporaciones para una categoría, por ejemplo la categoría “compañía”. De esta forma, de una noticia del tipo: "Ayer, Google compró la creciente compañía Facebook", un sistema sería capaz de encontrar los datos de tal manera que estableciera que "compañía" es "Google" y "Facebook".

Al usar extracción de la información no se recupera un subconjunto de documentos que son relevantes a una consulta usando palabras clave. Al usar extracción de la información se recuperan pasajes (de documentos) que describen hechos relevantes acerca de los tipos predefinidos de eventos, entidades o relaciones. Datos relevantes a estos hechos son generalmente especificados en ontologías.

Existen múltiples herramientas utilizadas en la extracción de información. General Architecture for Text Engineering (GATE) [21] es un toolkit de software Java desarrollado por la universidad de Sheffield en 1995 y ahora utilizado en todo el mundo, siendo la extracción de información una de sus funciones. Calais [22] es un servicio web de extracción de información de Thomson Reuters. Calais usa procesamiento de lenguaje natural y otro tipo de técnicas. Este servicio web busca en el texto y localiza las entidades (personas, lugares, productos, etc.), hechos (Juan trabaja para UC3M) y eventos (Juan fue acreditado como profesor titular) y después procesa esas entidades, hechos y eventos y los devuelve en formato RDF. Otras aplicaciones/software comercial que hacen posible la extracción de información incluyen Anderson Analytics [23], Attensity [24], Textalytics[25], TextAnalyst [26] , Textalyser [27] y WordStat [28] .

Como se mencionó anteriormente, en el método propuesto se utilizan técnicas de extracción de información para soportar la interpretación del texto de los patrones. La automatización de esta tarea se logró utilizando la API de la herramienta GATE [29]. Así, usando los conceptos en las ontologías definidas con técnicas de extracción de información se pueden localizar y etiquetar esos conceptos para detectar que se está hablando del atributo de calidad desempeño (*performance*) y proceder a la interpretación.

3 TRABAJOS RELACIONADOS

En la revisión bibliográfica realizada no se encontraron trabajos utilizando técnicas de representación del conocimiento y extracción de información que se enfoquen al análisis automático de patrones de diseño arquitectónico con el propósito de determinar si satisfacen o no atributos de calidad específicos. Sin embargo, sí se encontraron trabajos que usan técnicas de extracción y recuperación de información y representación de conocimiento para atender problemas en el área de la Ingeniería de Software. En este capítulo se describen estos trabajos.

3.1 Aplicaciones de Extracción y Recuperación de Información en Ingeniería de Software.

Investigadores del departamento de ciencias de la computación de la Universidad de Loyola en Maryland [30] realizaron un estudio para identificar los problemas que enfrentan los ingenieros de software y describir cómo métodos de recuperación y extracción de información han sido utilizado para resolver algunos de éstos.

En este trabajo principalmente se identificaron problemas que se clasificaron en tópicos relevantes en la Ingeniería de Software: ingeniería de requerimientos, mantenimiento de repositorios de software, links de trazabilidad entre documentos generados en los proyectos de software, reutilización de software y métricas de software.

En cuanto a los métodos de recuperación y extracción de información que se usaron para resolverlos están VSM (Vector Space Model), LSI (Latent Semantic Indexing), Tokenización (Tokenization) y algoritmos de pre-procesamiento. A continuación describimos de forma general estos métodos.

Los vectores VSM se usan para soportar el filtrado, extracción e indexación de información en documentos de texto, así como también para soportar la clasificación de su relevancia. Los vectores VSM son modelos basados en álgebra lineal, que permiten representar documentos de texto como vectores de *pesos de términos*. Esto es, un documento doc se representa como un vector

$$doc = (pT1, pT2, \ldots, pTn),$$

Donde pTi es el peso de un término en el documento. La naturaleza de los términos representados en el vector depende del dominio de aplicación, pero usualmente se refieren a palabras o frases cortas relevantes al dominio al que se refiere el documento. De esta forma, el peso de un término denota la importancia de éste en el documento. La forma más simple de asignar el peso a un término es la frecuencia con la que éste ocurre en el documento. Aunque lo más común es usar un enfoque denominado *tf-idf* (term frequency—inverse document frequency) que permite asignar un peso a un término considerando la frecuencia del término en todos los documentos de una colección.

LSI es un método, basado también en vectores de pesos, utilizado en el proceso de indexación de documentos. Sin embargo, se usan vectores de menor tamaño pues éste método se basa en el principio de que las palabras que se utilizan en los mismos contextos tienden a tener significados similares. Por ejemplo, supongamos que usamos LSI para indexar una colección de textos de arquitectura de software. Si las palabras *patrón de diseño*, *patrón arquitectónico*

y *patrón capas* aparecen juntas en gran cantidad de textos, el algoritmo de búsqueda se dará cuenta de que los tres términos son semánticamente cercanos. Una búsqueda de *patrón de diseño* por tanto, devolverá un conjunto de textos que contengan esa palabra (el mismo resultado que obtendríamos con una búsqueda normal), y también los textos que contienen las palabras *patrón arquitectónico* y *patrón capas* al estar relacionadas. Con esto se obtiene un conjunto más amplio de los resultados que con una búsqueda por palabra clave simple. De esta forma, el uso de LSI no requiere una coincidencia exacta para devolver resultados útiles*.*

Tokenization consiste en la segmentación de texto en palabras y oraciones. Antes de cualquier tratamiento el texto necesita ser segmentado en unidades lingüísticas tales como palabras, signos de puntuación, números, etc. Este es entonces un método muy común en este contexto.

Algo muy común es también el uso de ciertos algoritmos de *preprocesamiento* para soportar la discriminación de términos a considerar cuando se definen los vectores. Los algoritmos de *stopping words* remueven palabras que son de poca relevancia para las consultas de un documento como por ejemplo *"the", "about"* y *"can".* Los algoritmos de *stemming* reducen una palabra a su raíz (*stem*) de forma que se puedan tratar eficientemente las variantes de una palabra. Un ejemplo de esto sería eliminar los sufijos de las conjugaciones de verbos como am/is/are/being/been a simplemente "*be"*, o hacer que las palabras connection/connections/connective/connected/connected/connecting queden reducidas a *"connect".*

En las siguientes secciones explicamos el uso de estos métodos para resolver problemas específicos en los tópicos antes mencionados.

INGENIERÍA DE REQUERIMIENTOS

Como lo discutimos en el Capítulo 1, en el ciclo de vida del desarrollo de software, los objetivos de la etapa de Análisis son identificar el problema que se quiere resolver y especificar los comportamientos (ej. requerimientos de usuario o funcionales) y características (ej. requerimientos no funcionales) que debe satisfacer el sistema de software. En este contexto la Ingeniería de Requerimientos comprende todas las tareas necesarias para alcanzar estos objetivos.

Asumiendo que los requerimientos se encuentran escritos en especificaciones, un problema común es que si no se sigue un estilo consistente en su redacción, fácilmente se da pie a diferentes interpretaciones y muchas veces se ignoran sentencias que definen requerimientos importantes. De esta forma, la aplicación de extracción de información en la etapa de identificación de requerimientos se realiza para ayudar al descubrimiento de requerimientos que han sido omitidos, debido a la manera en la que se encuentran escritos en las especificaciones.

Considerando un contexto de desarrollo en donde se requiere descubrir requisitos faltantes para reconstruir un sistema legado, una propuesta es usar un repositorio de especificaciones de requerimientos de proyectos anteriores y similares al sistema legado que se intenta reconstruir [31]. Usando *VSM* y *LSI* se representan los términos clave en las especificaciones de proyectos anteriores similares al sistema legado. Entonces con los términos clave del nuevo requerimiento se pueden hacer búsquedas en el repositorio de especificaciones de requerimientos anteriores y en la nueva especificación. Si la búsqueda no produce resultados, significa que se ha identificado un requerimiento potencialmente faltante y se presentan al usuario para su consideración.

MANTENIMIENTO REPOSITORIOS DE SOFTWARE

El uso de repositorios de software se origina a partir de la necesidad de los programadores de compartir artefactos, como librerías o funciones; durante el desarrollo de un sistema. En términos generales, un repositorio de software es una ubicación de almacenamiento colaborativo que facilita la creación, mantenimiento y distribución de artefactos relacionados a proceso de desarrollo de un sistema de software.

Aunque existen estándares como formatos de archivos, estructuras y metadatos, los repositorios de software de mayor tamaño son cada vez más complejos en el sentido que estos van incrementando la cantidad y tipo de artefactos almacenados. Esto contribuye a que su uso sea complicado y se dé cierta desconfianza sobre la naturaleza de la información disponible y recuperada.

El tratamiento de repositorios mediante técnicas de recuperación y extracción de información permite tener la información de diferentes artefactos organizada, de manera que los artefactos existentes puedan proveer satisfactoriamente las necesidades de los ingenieros de software en la de búsqueda con la información precisa que necesitan.

E*n este contexto GHSOM (Growing Hierarchical Self-Organizing Map), ha sido utilizado para* automatizar la indexación de repositorios, mostrándolos en una jerarquía de capas de mapas simples o SOM (Self-Organizing Map), que son un tipo de red neuronal artificial [32]. Cuando se están construyendo repositorios de software, los VSM inicialmente contienen todos los términos extraídos para representar artefactos como documentos de requerimientos, documentos de diseño, archivos de código fuente, manuales y casos de prueba. Aplicando el método de *stemming y* algoritmos como *stopping words* y *hard/soft words, para* discriminar conceptos que no contribuyen a la descripción de los artefactos, agregando a los vectores VSM solo términos clave que son importantes en cada documento.

LINKS DE TRAZABILIDAD ENTRE DOCUMENTOS GENERADOS EN LOS PROYECTOS DE SOFTWARE

La trazabilidad de artefactos en un proyecto de software se refiere a la relación que existe entre los artefactos que se van generando en las etapas de desarrollo de un proyecto. Ejemplos de estos artefactos incluyen los documentos iniciales o finales de especificación de requerimientos, modelos de diseño, reportes, archivos de código fuente o casos de prueba.

Mantener la trazabilidad con todos esos documentos no es una tarea trivial, implicando un proceso manual costoso en tiempo y esfuerzo. Mediante técnicas de recuperación y extracción de información se propone el restablecimiento de links entre artefactos de software, por ejemplo encontrar los links entre artefactos de requerimientos y casos de prueba.

En este tópico la técnica de LSI junto con la herramienta ADAMS, son la manera más popular para encontrar links candidatos entre artefactos [33]. Primero indexando sobre una serie de artefactos obteniendo una matriz de conceptos clave, para después hacer el proceso para obtener conceptos relacionados mediante la técnica LSI. Para el ejemplo mencionado para reconocer links entre artefactos de requerimientos y casos de prueba se pudieran encontrar conceptos relacionados como: "*use case to test"* o *"use case to design"* que ayudan a la identificación de links entre requerimientos y casos de prueba.

REUTILIZACION DE SOFTWARE

En ingeniería de software se pretende la creación de componentes reutilizables que sean usados en el mismo contexto de un problema, teniendo beneficios potenciales en mejorar la calidad del software, productividad y mantenibilidad de los nuevos sistemas.

Originalmente estos componentes eran construidos por expertos, involucrando una alta inversión en su creación, estructuración y mantenimiento. Se han aplicado técnicas de recuperación y extracción de información sobre componentes existentes, localizando conceptos clave que pueden ser usados por los ingenieros de software para hacer mas eficientes las búsquedas sobre estos componentes.

Las herramientas de recuperación y extracción de información que se proponen para hacer búsquedas sobre artefactos, específicamente de código fuente, son *Wood y Sommerville* [34], diseñadas para hacer la búsqueda de componentes de software basados en *onframes* que son descriptores de componentes diseñados por expertos. Describiendo la función del componente, los objetos o partes del componente que desarrollan la función y las acciones de los objetos. La ventaja de búsquedas sobre *onframes* es que contienen descripciones más precisas basadas en conceptos clave.

Otra herramienta es *PATRicia (Program Analysis Tool for Reuse*) [35], enfocada en analizar artefactos de código, escritos bajo el paradigma orientado a objetos, que han sido escritos sin pensar en su reutilización, lo que dificulta la identificación de los componentes útiles para su uso. *PATRicia* encuentra componentes a lo largo de comentarios e identificadores, para esto utiliza un enfoque lingüístico en aspectos como comentarios e identificadores y otro aspecto no lingüístico para aspectos de POO (*Programación orientada a objetos)* como la jerarquía de clases; generando un ontología de términos y definiciones que describen los componentes de software.

METRICAS DE SOFTWARE

Las métricas en software son utilizadas para medir diferentes aspectos relacionados al desarrollo de un sistema. Estas medidas son importantes, por ejemplo, para determinar la calidad de un sistema o bien para facilitar la toma de decisiones relacionadas al desarrollo del mismo.

Recientemente se han utilizado técnicas de extracción de información para la estimación de métricas que no son tan fáciles de observar como lo son aquellas que miden aspectos estructurales del sistema (ej. líneas de código, número de atributos relacionados en un método, etc.). Estas métricas requieren mayor esfuerzo para su estimación. Un ejemplo de estas métricas es *acoplamiento*, usada para medir el grado que un componente de software necesita de otros para hacer su trabajo y *cohesión* que mide el grado que necesitan los elementos de un módulo para trabajar juntos

Para la estimación de métricas mediante extracción de información se propone principalmente la técnica de LSI. Por ejemplo, para determinar el grado de acoplamiento o cohesión de los componentes de un sistema a partir de su código fuente, se calculan las siguientes tres métricas basadas en el método LSI. La similitud conceptual entre clases (*CoCC*) es definida en términos de similitud conceptual entre un método y una clase (*CSMC*) y la similitud entre métodos (*CSMM*). Estas tres similitudes son definidas usando las siguientes formulas:

$$\text{let } x = \frac{vm_k^T \times vm_j}{(||vm_k||_2 \times ||vm_j||_2} (\text{the cosine similarity of } vm_k \text{ and } vm_j)$$

$$CSMM(m_k, m_j) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$CSMC(m_k, c_j) = \frac{\sum_{i=1}^{t} CSMM(m_k, m_{ji})}{t}$$

$$CSCC(c_l, c_j) = \frac{\sum_{i=1}^{t} CSMC(m_{j_i}, c_l)}{t}$$

Donde *c* denota una clase, *m* un método, *vm* el vector para un método.

Basado en el cálculo de *CSCC*, la primera métrica, el Acoplamiento Conceptual de una Clase *(c)* es definido como:

$$CoCC(c) = \sum_{i=1}^{n} CSCC(c, c_i))/(n-1) \text{ where } c \neq c_i.$$

Si la clase (c) tiene un grado alto de acoplamiento entonces el resultado al aplicar esta fórmula CoCC(c), a las demás clases será muy cercano a 1.

La segunda métrica $C^3(c)$, mide la Cohesión Conceptual de una Clase, para ello se utiliza la fórmula de *CSMM*. En la formula *CSMM* se calcula el valor promedio de todos los métodos en una clase considerando que la cohesión es alta mientras el valor de este promedio sea menor a 0.

La tercera métrica *LCSM*, mide la falta de similitud conceptual entre los métodos de una clase, que sería la medida inversa de una cohesión alta, donde valores altos indicarían baja cohesión. LCSM es el número de métodos de una clase que comparten similitudes, menos el número de métodos que no tienen nada en común.

3.2 Descubrimiento y Clasificación de Requerimientos en Proyectos de Software Open Source.

Este trabajo fue desarrollado en la universidad de Georgia, en el 2011, y su objetivo es el descubrimiento y clasificación automática requerimientos de proyectos open source usando técnicas recuperación y extracción de información [36].

En este tipo de proyectos las especificaciones de requerimientos se comunican en diversos medios, como chats, e-mails o foros, a través de textos no-estructurados, informales e incluso usando diversos idiomas. Generalmente, estos textos también contienen información no exclusiva de requerimientos, por ejemplo segmentos de código o secciones de noticias. Por la cantidad de texto o el tamaño de los proyectos el texto que describe las especificaciones de estos requerimientos es más propenso a errores de redacción por parte de sus autores y de interpretación por parte de sus lectores. Esto impacta negativamente en la calidad del diseño e implementación de estos requerimientos. La problemática principal radica en que tareas como el descubrimiento y análisis de requerimientos sobre especificaciones, requieren de una gran inversión de tiempo.

La manera en la que se hace el procesamiento sobre documentos de especificaciones es mediante un proceso que considera dos tareas: *reconocimiento* y *clasificación*. El reconocimiento de requerimientos consiste en localizar sentencias que especifican un requerimiento. La clasificación de los requerimientos consiste en identificar su categoría: funcional o no funcional. Para el segundo caso, considerando principalmente los factores de calidad del modelo de calidad de McCall [16]. Ambas tareas son automatizadas usando la herramienta GATE [29] y están soportadas por una ontología que incluye 6 niveles (L0 al L5). Como lo explicamos antes, una ontología es modelo que resulta de seleccionar un dominio y aplicar sobre él un método con el fin de obtener una representación de los conceptos que contiene (también llamados clases) y de las relaciones que existen entre los conceptos, con la finalidad de facilitar la comunicación y el intercambio de información entre diferentes sistemas y entidades.

En esta ontología los niveles más bajos (L0, L1) corresponden a conceptos de gramática del idioma inglés, los niveles intermedios (L2, L3 y L4) corresponden a conceptos de sentencias lógicas que definen la presencia de un requerimiento, y en el último nivel (L5) los conceptos son usados para la clasificación de las sentencias de requerimientos. En la Figura 6 se muestran estos niveles.

RCM Level	Description	Elements covered
L0: Token	Defines basic elements of text commonly included in all types of communication.	Word, punctuation, symbol, list, filename, url, email address, separator/delimiter, sentence, phrase
L1: POS	Defines most common Parts-of-Speech (POS) elements.	Adjective, adverb, conjunction, preposition, determiner, negation, noun, verb
L2: Qualification	Identifies expressions pointing to a context which might indicate the presence of a requirement.	Belief, certainty, necessity, preference, qualifier, quantifier, qualifying phrase
L3: Entities	Identifies the three fundamental elements of a requirement.	Subject(S)/actor, action(A)/verb, object (O)
L4: Requirement	Discovers parts of text identified as requirements.	SAO triples, SAO extensions, SAO atomic elements
L5: Classification	Classifies pieces of text identified at previous level and elements of lists.	SAO triples, SAO extensions, list items and introductory phrases

Figura 6. Niveles de ontología para la clasificación de requerimientos.

En el nivel 0 (L0) se definen los conceptos que representan elementos básicos identificados en los textos: palabras, signos de puntuación y elementos específicos a los proyectos analizados: direcciones de correo electrónico, URLs y referencias de archivos.

Los conceptos del nivel 1 (L1) definen partes comunes del idioma inglés y son conocidos como POS (*Parts of speech*): conjunciones (por ejemplo and, but also, than, or, else, otherwise, although), preposiciones (por ejemplo: over, under, on, upon, in, at, inside, on, at, in), sustantivos y verbos.

En el nivel 2 (L2) se definen los conceptos que se refieren a cualificadores y cuantificadores comúnmente utilizados para referirse a un requerimiento, por ejemplo: think/belief, necessity, preference.

En el nivel 3 (L3) se definen tres conceptos fundamentales de un requerimiento: sujeto (S), acción (A) y objeto (O)..

En el nivel 4 (L4) se definen estructuras gramaticales usadas para especificar un requerimiento como la unión de calificadores y cuantificadores del nivel L2 y sujetos (S), acciones (A) y objetos (O) del nivel L3.

Por último, el nivel 5 (L5) se definen conceptos del dominio específico para la clasificación de las sentencias de requerimientos inidentificadas en el nivel 4 como funcionales y no funcionales usando conceptos del modelo de McCall y otros que son parte de otros estándares o fueron encontrados en proyectos existentes para definir requerimientos. A este complemento de conceptos le llamaron modelo McCall+.

Para identificar los conceptos definidos del nivel L0 hasta el nivel L5 se definen *anotaciones* correspondientes a estos niveles, que son etiquetas que se agregan a un texto a nivel de palabras, sentencias, párrafos, secciones o incluso todo un documento. Las *anotaciones* pueden ser vistas también como metadatos esto es, datos que describen otros datos [37]. Las anotaciones son especificadas mediante la definición de reglas JAPE (*Java Annotation Patterns Engine*)..

De esta forma, un texto puede tener múltiples anotaciones provenientes de los distintos niveles (L0 – L5). La complejidad de las reglas JAPE aumenta según el nivel, debido que reglas de niveles superiores toman en consideración conceptos de niveles anteriores. Por ejemplo, una regla JAPE del nivel L5 para el criterio 12 del modelo de McCall que es Comunicación (comunicativeness), asociado al factor 5 que es Usabilidad (usability) (ver Tabla 3. Modelo de McCall), tiene la siguiente estructura:

```
Rule: L5_Comunicativeness
(
 {L4.valid == "Yes", L4_Requirement contains KW_F5C12}
):L5_ComunicativenessFired
-->
:L5_ComunicativenessFired.Comunicativeness = {category = "F5C12"}
```

La primer parte de la regla se refiere a su nombre que es *L5_Comunicativeness*, compuesto por el nivel de la ontología al que corresponde (L5) y el criterio de calidad para el que es usada (*Comunicativeness*). Después de esto viene la especificación de la expresión regular:

```
(
 {L4.valid == "Yes", L4_Requirement contains KW_F5C12}
) :L5_ComunicativenessFired
```

Donde se especifica que sea una anotación que ha sido aceptada como válida en el nivel L4 de la ontología (*L4.valid == "Yes")* y además se especifica que el texto del requerimiento (*L4_Requirement)* contenga conceptos clave asociados al factor 5 y criterio 12 del modelo de McCall (*contains KW_F5C12).* A esto se asigna la anotación temporal llamada *L5_ComunicativenessFired.*

En la segunda parte de la regla:

-->
:L5_ComunicativenessFired.Comunicativeness = {category = "F5C12"}

Se retoma la anotación temporal para definir la anotación final como: *Comunicativeness (L5_ComunicativenessFired.Comunicativeness*), y se asigna a la propiedad *category el valor "F5C12",* para precisar que ese requerimiento pertenece al criterio 12 del factor 5 del modelo de McCall.

En la Figura 7 vemos un ejemplo de la sentencia de un requerimiento, indicando las anotaciones que se detectan desde el nivel L2 hasta el nivel L5 (en los niveles L0 y L1 se encuentran anotaciones generales del idioma inglés).

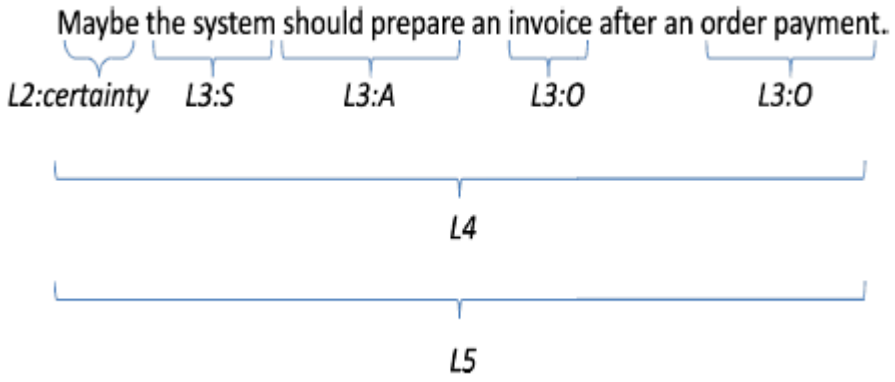


Figura 7. Anotaciones identificadas a partir de ontología en la sentencia de un requerimiento.

Para el nivel L2 encontramos el concepto *maybe* como cualificador, en el nivel L3 se encuentran como sujeto (S): *the system y como* acción (A)*: should prepare* y dos objetos (O): *invoice* y *order payment.* El nivel L4 representa la estructura gramatical del requerimiento uniendo los conceptos de los niveles L2 y L3. Y finalmente la sentencia del requerimiento es clasificada como funcional o no funcional usando los conceptos del nivel L5.

En el ejemplo de la Figura 8 se muestran otros ejemplos de clasificación sentencias de requerimientos usando conceptos definidos en el nivel L5. El ejemplo ilustra que una misma sentencia puede ser considerada por dos o más

anotaciones que corresponden a conceptos diferentes definidos en el nivel L5 (classifications).

Feature Request	Classifications
Perhaps this could be made an option in the export screen.	Expandability Operability
It could even be conditionalized, so that the option to export a real Excel file is not available unless this PEAR module is installed.	Expandability Modularity Simplicity
Using this Excel export module should also fix the fact that Excel for Mac and Excel for Windows require differing CSV formats	ExecutionEfficiency Modularity Operability Simplicity StorageEfficiency Traceability

Figura 8. Ejemplo de clasificaciones de sentencias usando el modelo de calidad McCall y McCall+.

La evaluación del proceso propuesto para el descubrimiento y clasificación de requerimientos se realizó con dos experimentos, explicados a continuación.

- En el primer experimento se aplicó el proceso a textos de 16 proyectos con más de 2,347,407 palabras y con más de 700 sentencias de requerimientos.

  A su vez este primer experimento se subdividió en las siguientes dos partes:

  a) Clasificación de requerimientos con el modelo de calidad de McCall extendido o McCall+.

  b) Clasificación de requerimientos con solo el modelo de calidad de McCall.

Estos experimentos arrojaron los siguientes resultados:

- En promedio el tiempo de procesamiento de cada proyecto fue de 6 minutos.
- Fueron reconocidos un promedio de 146,716 conceptos relacionados a requerimientos..
- Fueron reconocidas un promedio de 69.8% de sentencias de requerimientos.
- Con el conjunto de conceptos definidos en el modelo McCall+ se logró una clasificación del 56.4% de las sentencias de requerimientos.
- Con el modelo básico de McCall solo se alcanzó la clasificación de del 25.6% de las sentencias de requerimientos.

• En el segundo experimento se compararon los resultados del proceso automático, con el proceso que realizó un humano experto de forma manual.

Para este experimento se dividió cada proyecto en pequeños segmentos de datos y algunos párrafos extensos, después se seleccionaron aleatoriamente 20 ejemplos de esos segmentos.

Una vez que el experto realizó la clasificación manual de estos textos, se compararon los resultados obtenidos con los resultados de la clasificación automática. Las métricas para realizar esta comparación fueron:

1. *Precision:* Que se define como el número de instancias recuperadas que son relevantes. El valor perfecto de *precision* es de 1.0 lo que significa que cada resultado regresado en la búsqueda es relevante.

2. *Recall*: Que se define como el número de instancias relevantes que han sido recuperadas. El valor perfecto de *recall* es de 1.0 lo que significa que todos los documentos relevantes fueron devueltos en nuestra búsqueda.

Con instancias relevantes nos referimos a aquellas que esperamos encontrar en nuestra búsqueda. Por ejemplo, asumamos que tenemos 60 páginas relevantes. Si un motor de búsqueda, en una consulta dada, retorna 30 páginas y sólo sólo 20 son relevantes este motor tendrá entonces una precisión de 20/30 = 0.6 mientras que su recall es 20/60 = 0.3.

3. *F-Measure:* Que se define como la combinación de las métricas *precision* y *recall* promediando los porcentajes de estas dos métricas.

De este experimento los resultados obtenidos para las anteriores métricas se muestran en la Figura 9. La primera columna representa el porcentaje de reconocimiento de requerimientos y la segunda las clasificaciones con respecto al modelo de McCall y McCall+.

	Requirement	All annotation types
Precision	0.94	0.58
Recall	0.64	0.70
F-Measure	0.76	0.46

Figura 9. Métricas de comparación con un experto en el descubrimiento y clasificación de requerimientos.

En estas métricas el 1.0 representa el 100%, por lo tanto el porcentaje de descubrimiento de requerimientos es el más alto indicando que se han identificado un 94% de requerimientos correctos. La precisión en la clasificación es más baja con 58%. Esto indica que el proceso abarca buen porcentaje en las expresiones regulares definidas para la identificación de requerimientos pero no tanto para la clasificación.

3.3 Clasificación de Requerimientos No Funcionales en Documentos SRS.

Este trabajo fue desarrollado por la universidad Concordia en Montreal, Canadá; en el año 2013 [38]. El objetivo de este trabajo es apoyar a los ingenieros de software con métodos de análisis semánticos para encontrar requerimientos escritos en los textos de documentos SRS (Software Requirements Specification).

En un documento SRS se encuentran todos los requerimientos que un sistema debe cumplir y son típicamente clasificados en requerimientos funcionales que describen características del sistema en la fase de desarrollo, y requerimientos no funcionales que incluyen los atributos de calidad, restricciones de diseño, entre otros. Es bien sabido que los requerimientos no funcionales tienen un gran impacto sobre el costo y el tiempo total del proceso de desarrollo del sistema. Se propone un procesamiento automatizado de documentos SRS con el fin de encontrar ambigüedades u omisiones de requerimientos no funcionales. El enfoque para superar estos desafíos es basado en el procesamiento de lenguaje natural usando la herramienta GATE [29] y ontologías.

Previo a la definición del proceso, se creó un *gold estándar* [37] para requerimientos no funcionales. El gold estándar es una colección de documentos con anotaciones, generadas manualmente por varias personas, usado para evaluar herramientas de recuperación y extracción de información. En este trabajo a partir del *gold estándar* se generó una ontología, en la que se categorizaron las sentencias encontradas de requerimientos no funcionales. La mayoría de las clases de la ontología se basan en el estándar ISO/IEC 9126 [17]. Algunas clases de la ontología y su estructura se muestran en la Figura 10.

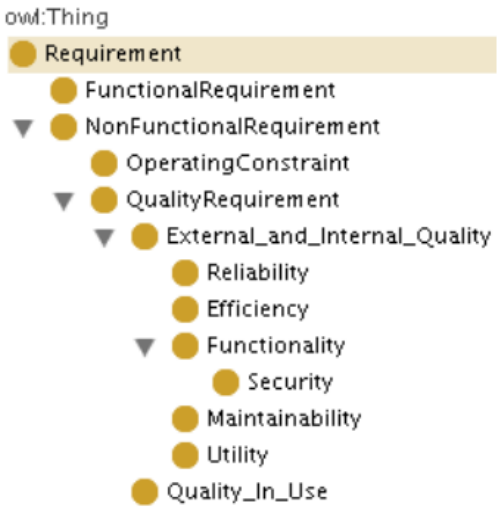


Figura 10. Ontología de requerimientos funcionales FR y requerimientos no funcionales NFR.

El proceso para el análisis de los documentos de requerimientos se implementó usando recursos de la herramienta GATE siguiendo 2 pasos:

1. Pre-procesamiento: en este paso los documentos son procesados para que pueda realizarse posteriormente la clasificación de sus sentencias. En este paso se utilizaron recursos predefinidos de GATE como *standard tokenization* que permite descomponer un texto, casi siempre en elementos individuales como palabras, signos de puntuación, símbolos, etc*., sentence splitting* que permite hacer una descomposición a nivel de sentencias*.*

2. Clasificación: en este paso se construyó un módulo para clasificar las sentencias encontradas anteriormente. Se definieron 3 categorías principales, en la de *NFRs* se identifican varios atributos de calidad: requerimientos funcionales, restricciones de diseño y *NFRs* (*security, efficiency*, reliability*, functionality* and *usability/utility*).

En la Figura 10 se muestra un ejemplo de las anotaciones generadas para la clasificación de NFRs en un texto.

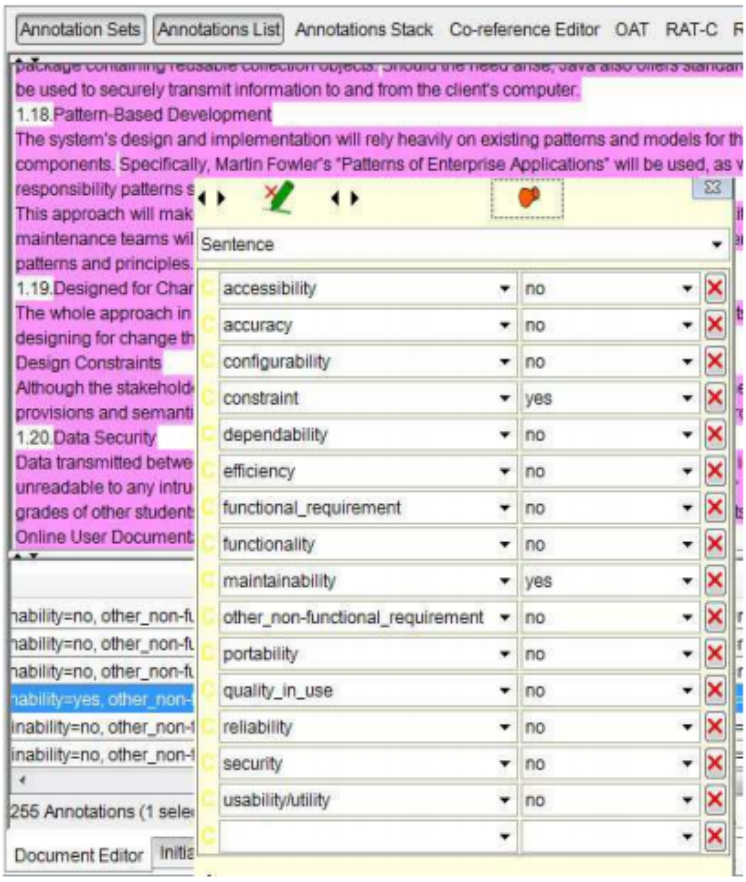


Figura 10. Ejemplo de clasificación de NFR en la interfaz de GATE.

Después de los pasos de pre-procesamiento y clasificación, se pobla la ontología de NFR con instancias (valores) a sus correspondientes clases. Se utilizó OwlExporter, que es un plugin de GATE para exportar las instancias de un documento en formato OWL (Web Ontology Language), lenguaje que permite publicar y compartir datos usando ontologías. Por último el uso de SPARQL (Protocol and RDF Query Language) que es un lenguaje para ejecutar query's sobre ontologías, por ejemplo si se quisieran consultar todas las sentencias de seguridad (security).

La evaluación de la efectividad en la clasificación de NFRs fue con las métricas *Precision*, *Recall* and *F-Measure (definidas en la sección anterior)* por cada una de las clases definidas en la ontología. Los porcentajes obtenidos de esta propuesta son altos para *Recall*, al encontrar información relacionada pero no información exacta, que es el porcentaje obtenido al aplicar la fórmula de *Precision*, si se quisieran mejorar estos resultados se tendrían que refinar a las categorías de clasificación, especialmente para las clases *Reliability, Efficiency y Functionality.*

De manera general los porcentajes promedio alcanzados en esta propuesta se consideran altos ya que para *Recall, Precision y F-measure es de* 0.84, recordando en que el 1.0 es el 100% de efectividad.

4 DESCRIPCIÓN DE MÉTODO PARA ANÁLISIS DE PATRONES

En este capítulo se describe el método definido para realizar análisis automático de textos sobre patrones de diseño arquitectónico con el propósito de determinar si satisface o no el atributo de calidad desempeño. Como lo indicamos antes, los patrones sobre los que trabaja el método están escritos en lenguaje natural en el idioma inglés. La automatización de las fases del método se realizó a través de una herramienta, denominada Orión, que usa recursos provistos por la API de la herramienta GATE.

4.1 Etapas del Método de Análisis de Patrones

El método definido para el análisis de patrones está compuesto de cuatro etapas principales, que son clásicas en un proceso de extracción de información:

1. **Preparación:** enfocada en hacer un pre-procesamiento del contenido del archivo que describe el patrón, o conjunto de patrones, con el propósito de identificar un conjunto de tokens y sentencias para su tratamiento en las etapas posteriores.
2. **Extracción:** enfocada en hacer la extracción de un subconjunto de conceptos clave y otras palabras, del conjunto de tokens obtenidos previamente, relacionados al atributo de calidad desempeño.
3. **Clasificación:** enfocada en encontrar coincidencias de sentencias que incluyen conceptos clave y otras palabras, identificados de la etapa de extracción, con el propósito de determinar si el patrón promueve (*promotes en inglés*) o inhibe *(inhibits en inglés)* el atributo de calidad desempeño.
4. **Presentación:** enfocada en mostrar las sentencias identificadas en la etapa de clasificación. Esto es, mostrar las sentencias que indican si el patrón promueve o inhibe el atributo de calidad desempeño.

Como se muestra en la Figura 11 estas etapas se realizan de forma secuencial usando un enfoque de procesamiento data-flow. Esto es, se tiene una cadena de etapas; cada etapa produce una salida que es utilizada como entrada por la siguiente etapa en la cadena.

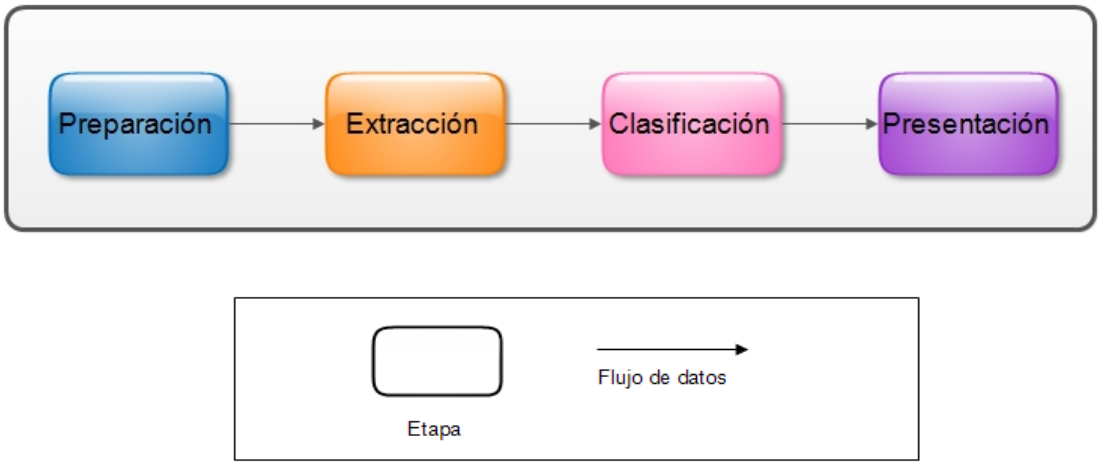


Figura 11. Etapas de método de análisis de patrones.

En las siguientes secciones se describen los detalles correspondientes al diseño e implementación estas etapas**.**

4.1.1 Etapa de Preparación

Esta etapa está enfocada en hacer un pre-procesamiento del *corpus* sujeto al análisis. En este contexto, un *corpus* es una colección de textos elaborados a partir de fuentes típicas que se pueden utilizar como textos que serán objeto de análisis. En este caso el *corpus* son textos contenidos en varios archivos que describen un patrón o un conjunto de patrones.

El pre-procesamiento que se realiza sobre el *corpus* consiste inicialmente en identificar un universo de *tokens y sentencias.* Los *tokens* son un conjunto de caracteres típicamente a nivel de palabras, aunque también pueden ser símbolos, signos de puntuación, etc. Las *sentencias* son secuencias de *tokens;* su inicio está marcado por una letra mayúscula y su fin por un salto de línea o por caracteres como un punto, puntos suspensivos o bien dos puntos.

Retomando lo explicado en el Capítulo 3 sobre *anotaciones,* y en el contexto de la API de la herramienta GATE, las *anotaciones* (a las cuales también nos referiremos como etiquetas *o tags)* definen metadatos sobre los *tokens* o *sentencias* de un *corpus.*

Así, la etapa de preparación genera como salida *tokens* y *sentencias* con *anotaciones* que sirven como entrada para la etapa de extracción.

4.1.2 Etapa de Extracción

Esta etapa está enfocada en hacer la identificación de un subconjunto de *anotaciones* de *tokens* relevantes para el atributo de calidad desempeño, a partir del conjunto de *tokens* identificados en la etapa de preparación.

Para identificar *tokens* relevantes, en esta etapa usamos la API de la herramienta GATE la cual se apoya de *diccionarios y listas*. En GATE una lista es una ontología almacenada en un archivo de texto plano. La API de la herramienta GATE cuenta con archivos de *listas* predefinidos que permiten generar las *anotaciones* correspondientes en los *tokens*. Por ejemplo, por medio de estas *listas* se puede asociar a un *token* una *anotación* que diga que el *token* es el nombre de una ciudad, de una organización, de un título de persona, de un día de la semana, de una moneda, etc. Por otra parte, un *diccionario* es un archivo que utiliza la API de la herramienta GATE para definir cuáles y en qué orden cargar los archivos de estas listas*.*

Ya hemos mencionado que en este trabajo se definieron dos ontologías: una para la representar conceptos clave relacionados al atributo de calidad

desempeño (ej. tácticas, *métricas* y *trade-offs*) y una para la representar palabras comúnmente utilizadas en sentencias, en textos de patrones, para referirse positiva o negativamente al atributo de calidad desempeño (ej. sustantivos, verbos, conjunciones, etc.). Ambas ontologías fueron creadas "desde cero" pues GATE no tiene ontologías predefinidas de este tipo. Para crear estas ontologías fue necesario realizar un *modelado del dominio*. La Figura 12 muestra el proceso realizado para hacer este modelado. Los cuadros grises representan las actividades realizadas, mientras los cuadros blancos representan artefactos de entrada o salida de cada actividad. A continuación describimos las actividades del proceso.

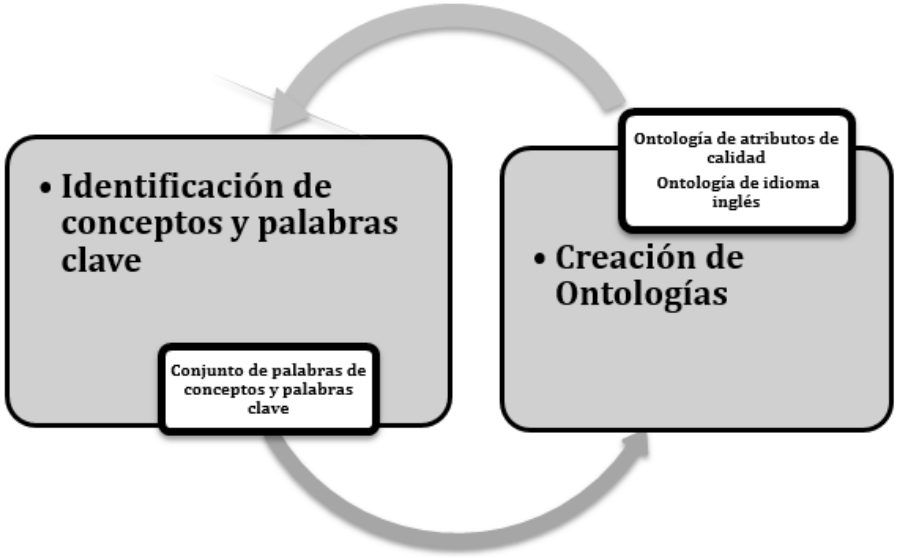


Figura 12. Actividades realizadas para el modelado del dominio del atributo de calidad y artefactos generados.

Identificación de Conceptos y Palabras Clave

En la primer actividad se revisó literatura relevante sobre modelos de calidad, métricas, tácticas y trade-offs, entre otra, con el fin de identificar conceptos y palabras clave relacionadas al atributo de calidad desempeño. La identificación de conceptos y palabras fue realizada "a mano" utilizando como estrategia una notación basada en colores. Haciendo uso de la notación de colores, la Figura 13 muestra las categorías de los conceptos y palabras considerados en este trabajo y una explicación breve de cada una de estas.

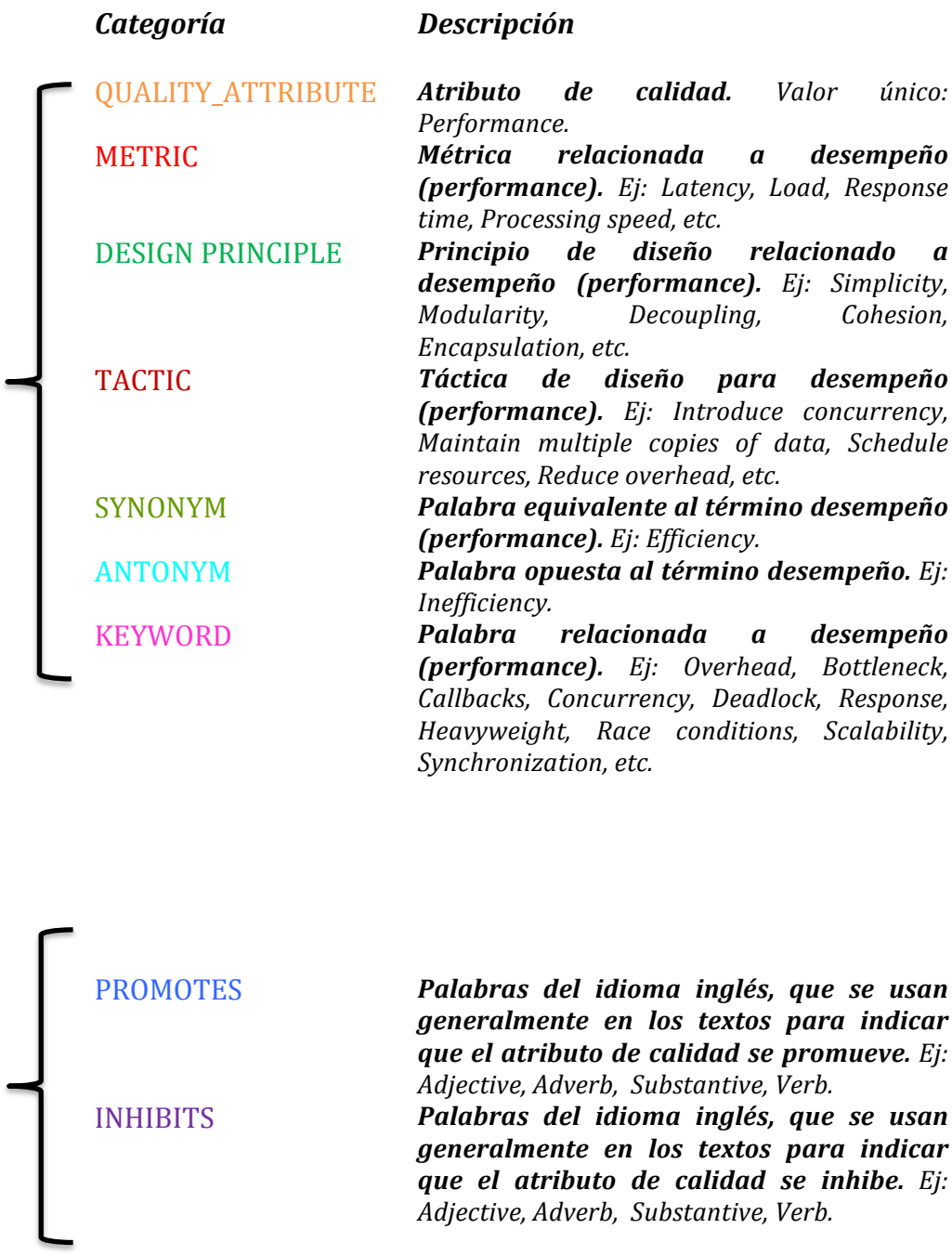
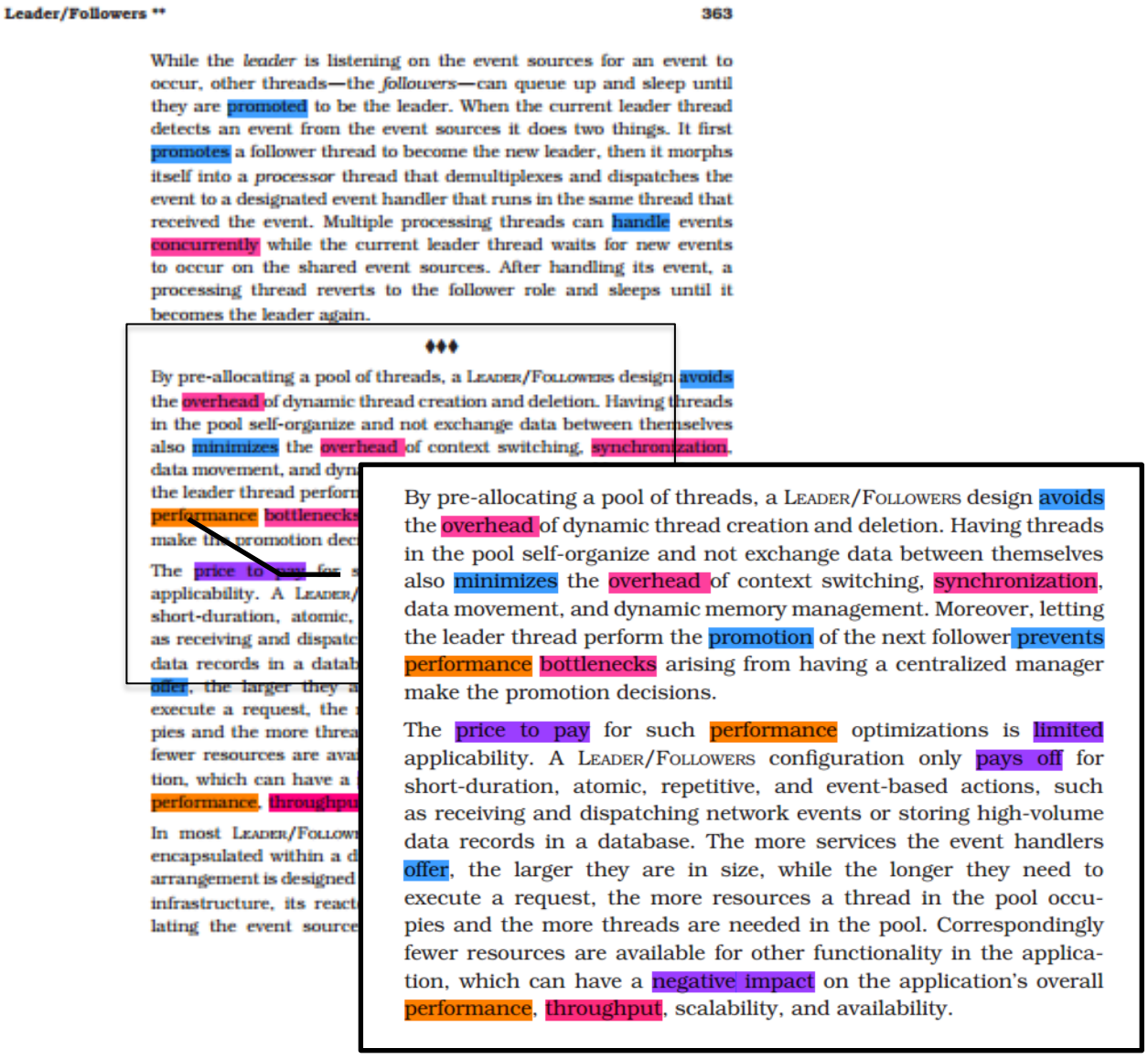


Figura 13. Categorías de conceptos y palabras considerados en las ontologías definidas en este trabajo.

A continuación se muestra un ejemplo de esta actividad sobre el texto del patrón *Leader/Followers*. Para identificar conceptos clave y palabras relacionadas al dominio del atributo de calidad desempeño usando la notación de colores.



Leader/Followers ** 363

While the *leader* is listening on the event sources for an event to occur, other threads—the *followers*—can queue up and sleep until they are promoted to be the leader. When the current leader thread detects an event from the event sources it does two things. It first promotes a follower thread to become the new leader, then it morphs itself into a *processor* thread that demultiplexes and dispatches the event to a designated event handler that runs in the same thread that received the event. Multiple processing threads can handle events concurrently while the current leader thread waits for new events to occur on the shared event sources. After handling its event, a processing thread reverts to the follower role and sleeps until it becomes the leader again.

♦♦♦

By pre-allocating a pool of threads, a LEADER/FOLLOWERS design avoids the overhead of dynamic thread creation and deletion. Having threads in the pool self-organize and not exchange data between themselves also minimizes the overhead of context switching, synchronization, data movement, and dynamic memory management. Moreover, letting the leader thread perform the promotion of the next follower prevents performance bottlenecks arising from having a centralized manager make the promotion decisions.

The price to pay for such performance optimizations is limited applicability. A LEADER/FOLLOWERS configuration only pays off for short-duration, atomic, repetitive, and event-based actions, such as receiving and dispatching network events or storing high-volume data records in a database. The more services the event handlers offer, the larger they are in size, while the longer they need to execute a request, the more resources a thread in the pool occupies and the more threads are needed in the pool. Correspondingly fewer resources are available for other functionality in the application, which can have a negative impact on the application's overall performance, throughput, scalability, and availability.

Figura 14. Ejemplo de identificación de conceptos clave sobre un patrón usando las categorías de la Figura 13.

Creación de Ontologías

En esta actividad se crearon y poblaron las dos ontologías --la de conceptos clave relacionados al atributo de calidad desempeño y de palabras comúnmente utilizadas en sentencias, en textos de patrones, para referirse positiva o negativamente al atributo de calidad desempeño. Esta actividad es muy similar al diseño orientado a objetos donde se definió una estructura general en términos de clases para posteriormente establecer valores específicos o instancias de estas clases. En la Figura 15 se muestran las clases para la ontología de conceptos clave que fueron definidas considerando los conceptos identificados en la etapa anterior.

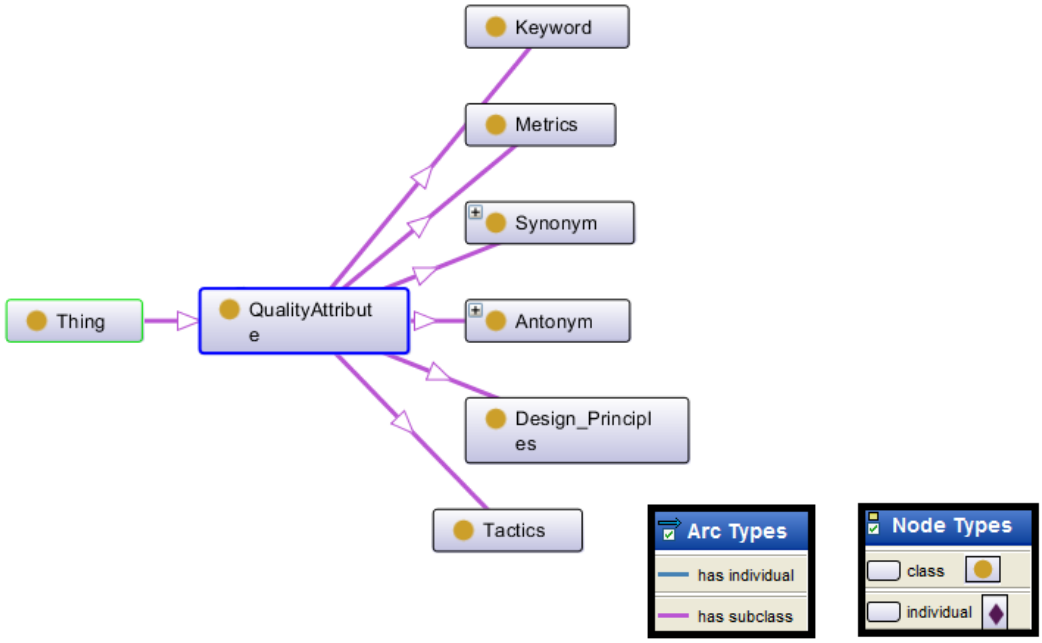


Figura 15. Clases de ontología de conceptos clave relacionados al atributo de calidad desempeño.

En la Figura 16, se muestran las instancias de las clases definidas en la ontología de conceptos clave relacionados al atributo de calidad desempeño. Sin embargo, como puede observarse, no se han agregado instancias para todas las clases debido que en los textos de los patrones analizados no se identificó la ocurrencia valores concretos para *Design_Principles* y *Tactics.*

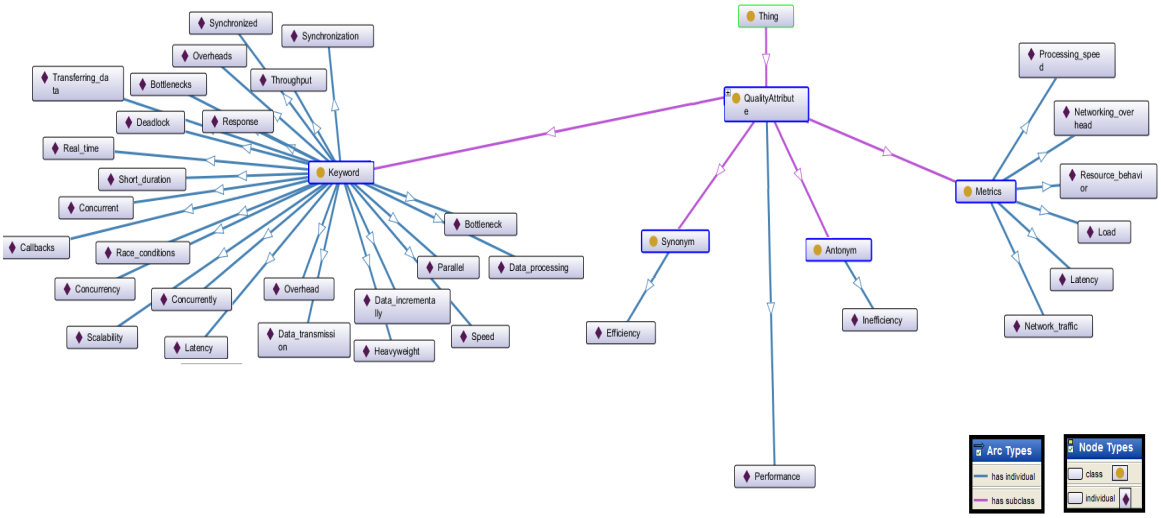


Figura 16. Instancias de la ontología de conceptos clave relacionados al atributo de calidad desempeño.

En la Figura 17, se muestran las clases de la ontología para la representar palabras utilizadas comúnmente en sentencias, en textos de patrones, para referirse positiva o negativamente al atributo de calidad desempeño. Como puede apreciarse fueron definidas usando las categorías promueve (promotes) e inhibe (inhibits). También en la figura se muestran algunas instancias de estas clases.

Estas dos ontologías, la de conceptos clave y la de palabras en sentencias promueve o inhibe, son la materia prima para la definición de expresiones regulares (explicadas en la siguiente etapa), para identificar sentencias en las que se habla positiva (*promueve*) o negativamente (*inhibe*) del atributo de calidad.

Figura 17. Instancias de ontología de palabras en sentencias promueve o inhibe.

Es importante hacer mención que ya se ha realizado bastante trabajo en este campo de modelado en la gramática para el idioma inglés y existen ontologías públicas, como la que ofrece WordNet [39], muy completas. Sin embargo, estas son demasiado generales para lograr la identificación de sentencias en las que se habla positiva o negativamente del atributo de calidad desempeño. De esta forma, aunque también se hace uso de estas ontologías predefinidas de propósito general, fue necesario definir “desde cero” las dos ontologías que hemos descrito en esta sección para lograr la extracción de tokens relevantes y el análisis de sentencias.

4.1.3 Etapa de Clasificación

Esta etapa está enfocada en encontrar coincidencias de sentencias que incluyen tokens que corresponden a conceptos clave y otras palabras identificadas de la etapa de extracción, con el propósito de determinar si en la sentencia se habla positivamente (*promotes*) o negativamente (*inhibits*) sobre desempeño. Para ello se generan anotaciones que corresponden a estructuras de sentencias específicas usando *expresiones regulares.*

En esta etapa, entonces, se definen expresiones regulares usando *JAPE (Java Annotation Pattern Engine)* [29]. Como lo mencionamos en el Capítulo 3, JAPE es un recurso exclusivo de GATE que permite definir y reconocer anotaciones de forma automática. En lo sucesivo, nos referiremos a estas expresiones regulares simplemente como reglas JAPE.

Estructuralmente una regla JAPE se compone de dos partes, parte izquierda (LHS) y parte derecha (RHS):

1. LHS (*left hand side*): En esta parte de la regla se escribe la expresión regular que deberá ser encontrada apoyándose de texto y los siguientes símbolos:
   + (1..n)
   ** (0..n)*
   | (boolean OR)
   ? (opcional)

2. RHS (*right hand side*): En esta parte de la regla se define la anotación y sus características que serán generadas para todas las instancias de la expresión regular descubiertas. Se usa como separador un *';'* y se tiene la siguiente sintaxis:
   {LHS} --> {Annotation type};{attribute1}={value1};...;{attribute n}={value n}

Las partes LHS y RHS son separadas por el símbolo '-->'.

Las sentencias se componen de tokens que corresponden a conceptos clave y otras palabras, utilizando POS (*part-of-speech tagging*). POS permite asignar a cada una de las palabras su categoría gramatical de acuerdo a la definición de la palabra o su relación con las palabras adyacentes en una frase, oración o párrafo. Esto ayudó a identificar en una sentencia la función gramatical del token, por ejemplo si se trata de sustantivo, verbo, adjetivo, adverbio, etc. Esto también permite observar el orden de los tokens e identificar repeticiones y colocación en la sentencia, para poder agruparlas en una expresión regular.

A continuación se muestran algunos ejemplos de las reglas JAPE para la clasificación de una sentencia con la anotación *Promotes* (en la Tabla 7) y la

estructura de una regla JAPE para la clasificación de una sentencia con la anotación *Inhibits* (en la Tabla 8*).*

Identificador	Regla JAPE	Ejemplo de Texto
promotes_01	**Rule: promotes_01** **(** **{Promotes}** **(** **{Token}** **)[0,3]** **{QualityAttribute}** **(** **({Token})?** **(** **({Keyword})** **)** **)?** **):prom01** **-->** **:prom01.Promotes_01 =** **{kind="Promotes_01",rule =** **"Promotes_01"}**	*Such a configuration can further **increase** **system** **performance** **and** **throughput**, as some filter instances can start processing new data streams while others are processing previous data streams.*

Tabla 7. Ejemplo de regla JAPE para identificar una sentencia refiriéndose a que se promueve el atributo de calidad desempeño.

En el ejemplo de la Tabla 7, la primer parte es el nombre que lleva la regla, en este ejemplo sería: **promotes_01** (la regla promotes con el identificador uno). En la parte LSH tenemos lo siguiente:

1. Debe existir una anotación de tipo ***Promotes*** en la oración.

{Promotes}

Esta anotación es derivada de la aparición de alguna de las palabras de la clase *Promotes* de la ontología mostrada en la Figura 17. En nuestro ejemplo tenemos la palabra *increase*:

*Such a configuration can further **increase** system performance and throughput.*

2. Pueden aparecer de 0 a 3 anotaciones de tipo **Token**.

(
{Token}
)[0,3]

La anotación de tipo ***Token***, es un sustantivo general identificado con el POS Tagger de la API de GATE.

*Such a configuration can further increase **system** performance and throughput.*

3. Debe existir una anotación del tipo ***QualityAttribute.***

{QualityAttribute}

Esta anotación es derivada de la aparición de algún concepto clave de la clase QualityAttribute de la ontología mostrada en la Figura 16. En nuestro ejemplo tenemos la palabra *performance*:

*Such a configuration can further increase system **performance** and throughput.*

4. Puede aparecer como complemento una anotación del tipo ***Keyword*,** correspondiente a la clase *Keyword* de la ontología de la Figura 15, y puede ser precedida opcionalmente por una anotación de tipo **Token.**

(
({Token})?
(
({Keyword})
)
)?

En esta parte de la regla no es obligatorio la aparición de algún concepto clave de la clase ***keyword*** de la ontología mostrada en la Figura 16. En nuestro ejemplo tenemos el concepto *throughput*:

*Such a configuration can further increase system performance **and throughput.***

Aunque no es obligatoria su ocurrencia se consideró porque proporciona más certeza de que se promueve el atributo de calidad.

5. Al final de la estructura de la expresión regular, se asigna el nombre que llevará la anotación de manera temporal:

: prom01

En la segunda parte de la regla, RHS se definen los siguientes elementos:

prom01.Promotes_01 = {kind="Promotes_01",rule = "Promotes_01"}

Utiliza el nombre de la anotación temporal **prom01** para asignarle el nombre final de la anotación como **Promotes_01,** así como el tipo **Promotes_01** y la regla a la que pertenece **Promotes_01.**

Identificador	Estructura	Ejemplo
inhibits_01	**Rule: inhibits_01** **(** **{Inhibits}** **({Token})?** **{QualityAttribute}** **{Inhibits}** **):inhi01** **-->** **:inhi01.Inhibits_01 =** **{kind="Inhibits_01",rule =** **"Inhibits_01"}**	*An ACTIVE OBJECT arrangement also introduces a heavyweight request handling and execution infrastructure, which can **cause performance penalties** for components that only implement short-duration methods.*

Tabla 8. Ejemplo de regla JAPE para identificar una sentencia refiriéndose a que se inhibe el atributo de calidad desempeño.

En el ejemplo de la Tabla 8 la regla se llama **inhibits_01** (la regla inhibits con el identificador uno). En la parte LSH tenemos lo siguiente:

1. Debe existir una anotación de tipo ***Inhibits*** en la oración.

{Inhibits}

Esta anotación es derivada de la aparición de alguna de las palabras de la clase *Inhibits* de la ontología mostrada en la Figura 17. En nuestro ejemplo tenemos la palabra *cause*:

which can ***cause*** *performance penalties for components that only implement short-duration methods.*

2. Pueden aparecer alguna anotación de tipo ***Token***.

({Token})?

En la oración de nuestro ejemplo no ocurre este escenario.

3. Debe existir una anotación del tipo ***QualityAttribute***.

{QualityAttribute}

Esta anotación es derivada de la aparición de algún concepto clave de la clase QualityAttribute de la ontología mostrada en la Figura 16. En nuestro ejemplo tenemos la palabra *performance*:

which can cause ***performance*** *penalties for components that only implement short-duration methods.*

4. Después, nuevamente debe existir otra anotación de tipo ***Inhibits***:
   {Inhibits}

Esta anotación es derivada de la aparición de alguna de las palabras de la clase *Inhibits* de la ontología mostrada en la Figura 17. En nuestro ejemplo tenemos la palabra *penalties*:

which can cause performance ***penalties*** *for components that only implement short-duration methods.*

5. Al final de la estructura de la expresión regular, se asigna el nombre que llevará la anotación de manera temporal:
   : inhi01

En la segunda parte de la regla, RHS se definen los siguientes elementos:

inhi01.Inhibits_01 = {kind="Inhibits_01",rule = "Inhibits_01"}

Las reglas de la categoría *Inhibits* utilizan el mismo estándar de las reglas de la categoría *Promotes,* para el nombre de la anotación temporal **inhi01**, el nombre final de la anotación como **Inhibits_01**, así como el tipo **Inhibits_01** y la regla a la que pertenece **Inhibits.**

En el *Anexo 1: Estructura de reglas JAPE,* ubicado en la siguiente url:

https://www.dropbox.com/s/xswap5ieq56ux5u/Anexo%201%20-%20Estructura%20de%20reglas%20JAPE.pdf?dl=0

se puede consultar la lista completa de las reglas definidas para poder hacer la clasificación de sentencias con anotaciones *Promotes* y/o *Inhibits* para el atributo de calidad desempeño definidas en este trabajo.

4.1.4 Etapa de Presentación

Esta etapa está enfocada en mostrar el resultado del análisis a nivel de sentencias donde se encuentran las anotaciones identificadas en la etapa de clasificación. Para ello se sigue un enfoque del área de Question-Answering [40] que son sistemas de búsqueda de respuestas, y trabajan con extracción de información y el procesamiento del lenguaje natural. Estos sistemas procesan preguntas de dominio abierto y devuelven las respuestas específicas tras consultar colecciones de documentos de texto, con el objetivo de obtener información relevante para el usuario.

En el área de Question-Answering, se ha trabajado mucho sobre algoritmos y estructuras para responder preguntas de diferentes tipos, como las preguntas *W's* que son del tipo *why, what, where* y *who. Pero* también existe una estructura para responder preguntas del tipo *Yes/No* [41], [42]. En esta etapa se da respuesta a una pregunta *Yes/No* para saber el patrón analizado promueve o no el atributo desempeño. *La estructura de las preguntas Yes / No, tienen los siguientes elementos:*

Questions type: YES/NO

BE + SUBJECT + Adjective/Noun

Are you Canadian? Yes / No

Is she tall? Yes / No

DO/DOES + SUBJECT + VERB

Do you play baseball? Yes / No

Does she live in Japan? Yes / No

Entonces, la formulación de la pregunta a responder tiene la siguiente estructura:

Does the <PATTERN> promote <QUALITY_ATTRIBUTE>?

Does the <Pipes-And-Filters> promote <performance>? Yes / No

El criterio que se ha utilizado para dar respuesta a esta pregunta es por medio del conteo de las anotaciones en las categorías *Promotes* e *Inhibits*. Si el número de anotaciones *Promotes* es mayor al de *Inhibits* entonces la respuesta es ***yes.*** En caso contrario, si el número de anotaciones *Inhibits* es mayor entonces se responde con ***no.*** Cuando la cantidad de anotaciones en ambas categorías es la misma, queda a criterio de quien ejecuta el análisis determinar si es un si o no y se muestra la etiqueta **unanswered.** También se considera el escenario de cuando en el número de anotaciones es cero, la respuesta mostrada también será **unanswered**.

4.2 Diseño y Descripción de la Herramienta Orión

La automatización de las fases del método definido para el análisis de patrones, descritas antes, se realizó a través de una herramienta denominada Orión. En la Figura 18 se muestra su estructura en términos de sus principales subsistemas: ***GUI*** y **Motor de Análisis**. Cada uno de estos subsistemas a su vez contiene un conjunto de componentes. En la figura se usan los mismos colores de la Figura 11 para que el lector pueda mapear cada uno los componentes con las etapas del método que estos soportan. En el Motor de Análisis se usan recursos provistos la API de la herramienta GATE.

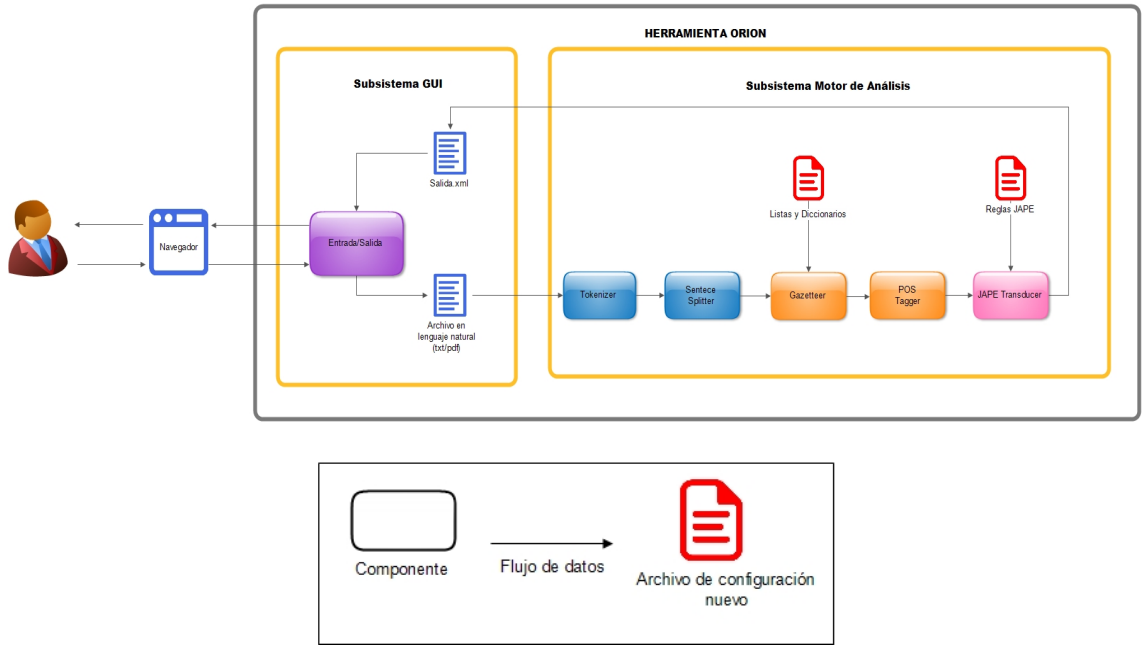


Figura 18. Estructura de la Herramienta Orión en términos de sus principales componentes.

4.2.1 Subsistema GUI (Entrada de datos)

El subsistema GUI (Interfaz Gráfica de Usuario) incluye un componente denominado Entrada/Salida que tienen la responsabilidad de ser el punto de interacción entre el usuario y el subsistema Motor de Análisis. Por esta razón indicamos en la Figura 18 que, de alguna forma, este componente soporta la etapa de presentación (denotada en la Figura 11 con color violeta). Como explicamos antes esta etapa se enfoca en contestar la pregunta relacionada a saber si se promueve o no desempeño en el patrón analizado y mostrar las sentencias de texto que soportan la respuesta. Como se puede apreciar, el

acceso a este componente es través de un navegador. La apariencia de este componente al recibir información de entrada se muestra en la Figura 19.

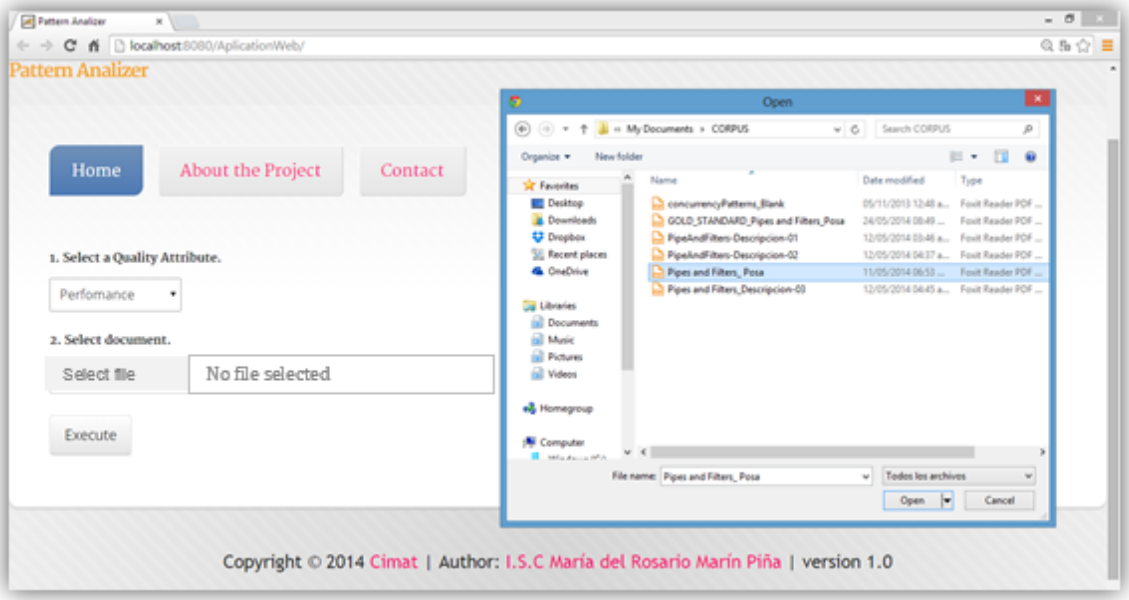


Figura 19. Apariencia del subsistema GUI.

El flujo de ejecución para un análisis inicia cuando el usuario a través del componente Entrada/Salida*:*

1. Selecciona el atributo de calidad de interés: *performance.*

2. Carga el archivo con el o los patrones a analizar (denotado en la Figura 13 como Archivo en lenguaje natural). En esta implementación sólo se soportan los formatos: *.txt* y .pdf sin contenidos de gráficos o imágenes.

Una vez provista esta información se envía al subsistema Motor de Análisis. A continuación se describe la funcionalidad de este subsistema.

4.2.2 Subsistema Motor de Análisis

Los componentes del subsistema Motor de Análisis se apoyan en recursos provistos por la API de la herramienta GATE. A continuación se describen estos componentes.

Tokenizer y Sentence Splitter

El método definido para el análisis de patrones está compuesto de cuatro etapas principales. La primera de esta preparación se enfoca en hacer un pre-procesamiento del contenido del archivo que describe el patrón, o conjunto de patrones, con el propósito de identificar un conjunto de tokens y sentencias para su tratamiento en las etapas posteriores.

Como se aprecia en la Figura 18, para soportar esta etapa (denotada en la Figura 11 con color azul) se definen los componentes Tokenizer y Sentence Splitter.

El componente Tokenizer tiene la responsabilidad de identificar tokens. Para soportar esta función este componente usa un recurso predefinido de la API de GATE llamado *Tokenizer.* El componente Sentence Splitter tiene la responsabilidad de identificar sentencias (secuencias de tokens). Para soportar esta función este componente usa un recurso predefinido de la API de GATE llamado *Sentence Splitter.*

***G*azetteer y POS Tagger**

La segunda etapa del método definido para el análisis de patrones es extracción y se enfocada en hacer la extracción de un subconjunto de conceptos clave y otras palabras, del conjunto de tokens obtenidos previamente, relacionados al atributo de calidad desempeño.

Como se aprecia en la Figura 18, para soportar esta etapa (denotada en la Figura 11 con color naranja) se definen los componentes *Gazetteer y* POS *Tagge*r*.*

Ya hemos explicado antes, sección 4.1.1., que para realizar esto es necesario el uso de ontologías que pueden definirse de forma simple como listas y diccionarios que indican como cargarlas. Como se aprecia en la Figura 18 el componente *Gazetteer* utiliza un conjunto de listas que contienen los conceptos relacionados al atributo de calidad desempeño y los diccionarios correspondientes. El componente *Gazetteer* fue creado a partir de un recurso existente en la API de GATE llamado *Gazzetteer*. Sin embargo, las listas fueron creadas especialmente para este trabajo.

El componente POS Tagger fue creado a partir de un recurso existente en la API de GATE llamado POS Tagger. Este componente tiene la responsabilidad de asignar a cada una de las palabras y/o conceptos una categoría gramatical de acuerdo a su definición o su relación con las palabras y/o conceptos adyacentes en la sentencia. Los nombres de las categorías gramaticales están representados por abreviaciones. Algunos ejemplos de estas abreviaciones son:

NN - noun - singular or mass
NNP - proper noun - singular: All words in names usually are capitalized but titles might not be.
NNPS - proper noun - plural: All words in names usually are capitalized but titles might not be.
NNS - noun - plural
NP - proper noun - singular
NPS - proper noun – plural
VBD - verb - past tense
VBG - verb - gerund or present participle
VBN - verb - past participle
VBP - verb - non-3rd person singular present
VB - verb - base form: subsumes imperatives, infinitives and subjunctives.
PP - personal pronoun

JAPE Transducer

La tercer etapa del método definido, denominada clasificación, está enfocada en encontrar coincidencias de sentencias que incluyen conceptos clave y otras palabras, identificadas de la etapa de extracción, con el propósito de determinar si el patrón promueve (promotes en inglés) o inhibe (inhibits en inglés) el atributo de calidad desempeño.

Como se aprecia en la Figura 18, para soportar esta etapa (denotada en la Figura 11 con color rosa) se define el componente *JAPE Transducer.*

Igualmente, este componente usa un recurso existente en la API de GATE llamado *JAPE Transducer que* permite la ejecución de las reglas JAPE definidas. Como se parecía en la Figura 13, el resultado generado es almacenado en un archivo llamado *Salida.xml*.

4.2.3 Subsistema GUI (Resultado del análisis)

Como se aprecia en la Figura 18, el componente Entrada/Salida del subsistema GUI muestra el resultado del análisis. Se da respuesta a la pregunta y se permite al usuario visualizar y navegar a través de los resultados obtenidos de sentencias que indican si el patrón promueve o inhibe el atributo de calidad desempeño. Esto se ilustra en las Figuras 20 y 21 respectivamente.

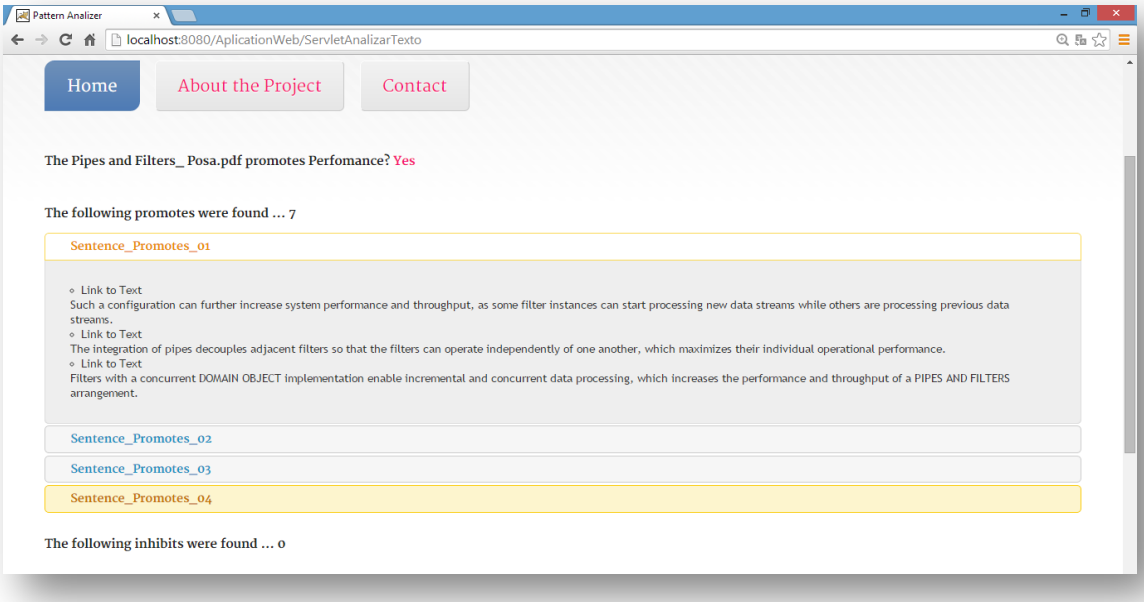


Figura 20. Presentación de sentencias que indican que se promueve o inhibe el atributo de calidad desempeño.

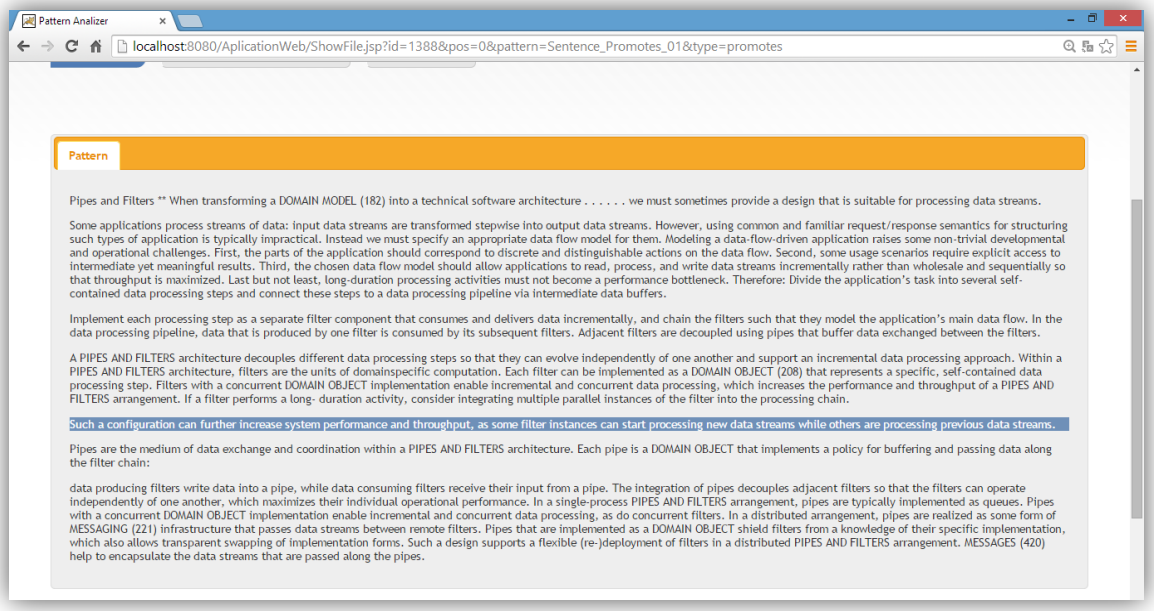


Figura 21. Navegación sobre las sentencias que indican que se promueve o inhibe el atributo de calidad desempeño.

5 EVALUACIÓN DE LA PROPUESTA

En este capítulo se discuten los resultados de la evaluación del método definido para realizar análisis automático de textos sobre patrones de diseño arquitectónico con el propósito de determinar si satisface o no el atributo de calidad desempeño. Inicialmente se explican las métricas utilizadas para la evaluación. Posteriormente se describe la manera en la que se diseñaron los experimentos, así como los datos recolectados después de su ejecución y la interpretación de los resultados obtenidos.

5.1 Métricas Utilizadas

5.1.1 Precision y Recall

Para evaluar si el método propuesto tiene una exactitud en el resultado del análisis cercana a la que tendría un arquitecto experimentado, que era uno de nuestros objetivos, se utilizaron dos métricas muy populares para evaluar los sistemas de extracción de información: *precision* y *recall*.

El conjunto de sentencias relevantes para el cálculo de estas métricas están contenidas en el *Anexo 2: Gold Estándar,* ubicado en la siguiente url:

https://www.dropbox.com/s/xzousxfomaqrjzh/Anexo%202%20-%20Gold%20Est%C3%A1ndar.pdf?dl=0

El término Gold Estándar fue explicado en el Capítulo 3. En la Figura 22 se muestra el tipo de información a la que hacen referencia las métricas de precisión y recall.

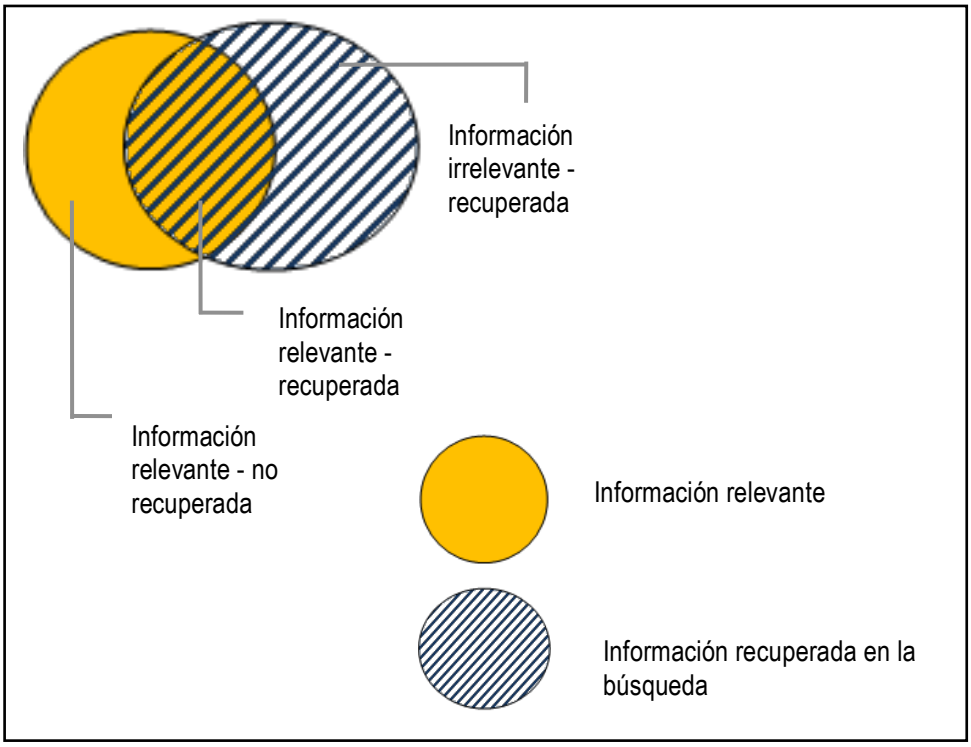


Figura 22. Tipos de información a la que hacen referencia las métricas de Precision y Recall.

También usaremos los términos mostrados a continuación para explicar las fórmulas para el cálculo de *precisión* y *recall*:

	Relevantes	***Irrelevantes***
Instancias recuperadas	***True Positives (tp)***	***False Positives (fp)***
Instancias no recuperadas	***False Negatives (fn)***	***True Negatives (tn)***

Precision

Se define como el número de instancias (por ejemplo documentos, páginas web, registros en una base de datos o, en el caso de este trabajo, sentencias) recuperadas que son relevantes. Es decir, es una medida para determinar cuánta información extraída automáticamente es correcta. El valor perfecto de *precision* es de 1.0 lo que significa que todas las sentencias recuperadas en la búsqueda son relevantes. En la Figura 23, se muestra la fórmula de *precision*, donde tenemos:

$$\text{Precision} = \frac{\text{Número de sentencias relevantes recuperadas } (\textbf{\textit{true positives}})}{\text{Número total de sentencias recuperadas } (\textbf{\textit{true positives + false positives}})}$$

C: Número de sentencias irrelevantes.

A: Número de sentencias relevantes recuperadas.

$$\text{PRECISION: } \frac{A}{A+C} \times 100\%$$

Figura 23. Descripción gráfica de la fórmula de precisión.

Recall

Se define como el número de instancias (por ejemplo documentos, páginas web, registros en una base de datos o, en el caso de este trabajo, este trabajo sentencias) relevantes que han sido recuperadas. Es decir, es una medida para determinar cuánta información relevante se ha extraído automáticamente. El valor perfecto de *recall* es de 1.0 lo que significa que todas las sentencias relevantes existentes en la colección fueron recuperadas en nuestra búsqueda.

En la Figura 24, se muestra la fórmula de *recall*, donde tenemos:

$$\text{Recall} = \frac{\text{Número de sentencias relevantes recuperadas (}\mathbf{\text{true positives}}\text{)}}{\text{Número total de sentencias relevantes existentes (}\mathbf{\text{true positives + false negatives}}\text{)}}$$

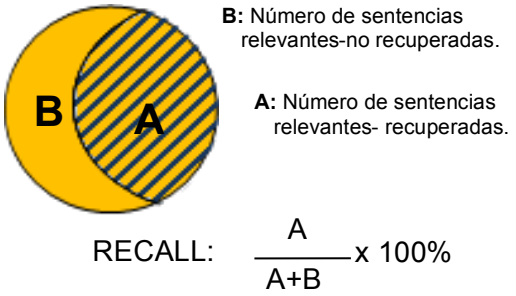


Figura 24. Descripción gráfica de la fórmula de recall.

5.1.2 Experiencia en Diseño y Desarrollo de Sistemas de los Participantes en los Experimentos

Esta medida es utilizada para medir el nivel de experiencia en el diseño y desarrollo de sistemas de los participantes en los experimentos y permite evaluar si nuestro método es útil para asistir a los arquitectos novatos en la identificación y selección de patrones durante el diseño de la arquitectura, que es otro de nuestros objetivos.

De esta forma se definieron los niveles *Bajo*, *Medio* y *Alto*. El nivel se considera *Bajo* si el participante ha colaborado en un rango de 1 a 4 proyectos a nivel escolar, el siguiente nivel es *Medio* si ha colaborado en un rango de 1 a 4 proyectos en la industria y por ultimo *Alto* si ha colaborado en 5 o más proyectos en la industria.

5.1.3 Tiempo de Análisis

Esta medida en utilizada para medir el tiempo invertido por un participante para realizar el análisis de textos sobre patrones con el propósito de determinar si satisface o no el atributo de calidad desempeño. Esta métrica permite evaluar si con nuestro método se puede disminuir el tiempo que invierte un arquitecto novato en el análisis “manual” del texto de un patrón, que es uno de nuestros objetivos.

5.2 Descripción de Experimentos

En esta sección se explican los detalles de cómo se estructuraron los experimentos para la obtención de los datos necesarios en el cálculo de las métricas descritas anteriormente.

5.2.1 Personas Consideradas

La Figura 25 lista los tres grupos de participantes considerados en los experimentos realizados: estudiantes de licenciatura de 6 semestre de la carrera de ingeniería de software (denotado como nivel licenciatura), estudiantes de segundo año de maestría en ingeniería de software (denotado como nivel maestría) y trabajadores en industria de diseño y desarrollo de software (denotado como nivel industria).

El tamaño de cada grupo fue de 15 participantes. Entonces, el número total de participantes en los tres experimentos fue de 45. Un factor importante en la selección de participantes es que tuvieran nociones sobre arquitectura de software, patrones de diseño y atributos de calidad.

Figura 25. Tres grupos de participantes considerados en los experimentos.

5.2.2 Diseño de los Experimentos

Cada experimento consistió en la ejecución de tres fases cada una con diferente grado de complejidad. Los datos más significativos de cada esta fase se listan en la Figura 26.

Fase 1

- Atributo de calidad buscado : **Performance.**
- Número de patrones analizados: **1 patrón.**
- Nombre de patrones analizados: **Pipes and Filters.**
- Pertenece a un mismo autor.

Fase 2

- Atributo de calidad buscado : **Performance**
- Número de patrones analizados: **3 patrones**
- Nombre de patrones analizados: **Pipes and Filters**
- Pertenecen a tres autores distintos.

Fase 3

- Atributo de calidad buscado : **Performance**
- Número de patrones analizados: **4 patrones**
- Nombre de patrones analizados: **Half-Sync / Half-Async, Leader Followers, Active Object y Monitor Object**
- Pertenecen a un mismo autor.

Figura 26. Fases de los experimentos.

La fase 1 consiste en el análisis de un documento de texto describiendo el patrón Pipes and Filters. La fase 2 consiste en el análisis de tres documentos de texto describiendo el patrón Pipes and Filters; escritos por diferentes autores. La fase 3 consiste el análisis de tres documentos de texto describiendo los patrones Half-Sync / Half-Async, Leader Followers, Active Object y Monitor Object; escritos por el mismo autor. Todos los patrones antes mencionados promueven desempeño y su longitud es de 2 a 4 páginas cada uno.

La definición de estas fases se realizó de ésta forma porque consideramos que permiten evaluar la escalabilidad del análisis con respecto a la heterogeneidad de la redacción del texto del patrón así como el tiempo que invierte un arquitecto novato en el análisis “manual” del texto de un patrón, que son dos de nuestros objetivos.

Cada una de estas fases consistió en los siguientes pasos:

- Llenado de datos generales del participante en el experimento: nombre, correo, experiencia en el diseño y desarrollo de sistemas.
- Registro de la hora de inicio del experimento.
- Lectura del texto del patrón o patrones de diseño provisto.
- Subrayado de las sentencias en donde se hable de ***performance (desempeño)*** indicando:
  1. El número de la sentencia.
  2. La clasificación de la sentencia dependiendo de la interpretación del participante sobre cómo se habla del atributo de calidad; es decir, indicar si el atributo se promueve o se inhibe.
- Conteo y registro del número total de sentencias, promueve o inhibe, identificadas.
- Respuesta a la pregunta básica. Por ejemplo:
  ¿El patrón **Pipes and Filters** promueve **Desempeño/Performance?**
- Registro de la hora de término del experimento.

Para facilitar la ejecución de cada experimento se diseñó un formato para capturar los datos. Este formato se entregó a cada participante.

5.3 Resultados obtenidos

Para ilustrar con más detalle los resultados de los experimentos realizados hemos tomado como ejemplo representativo el experimento a Nivel Licenciatura que por su nivel de experiencia pueden considerarse como arquitectos novatos. Sin embargo, todos los resultados obtenidos de cada experimento se pueden consultar en la siguientes urls:

https://www.dropbox.com/s/9g5mgrl0yey5hk7/Experimento_GRAFICAS%20%28Licenciatura%29.xlsx?dl=0

https://www.dropbox.com/s/d71dqw36e7vby1d/Experimento_GRAFICAS%20%28Maestria%29.xlsx?dl=0

https://www.dropbox.com/s/hwjdzwz9v19pwv1/Experimento_GRAFICAS%20%28Industria%29.xlsx?dl=0

La Figura 27 muestra la distribución del total de participantes con respecto a su nivel de experiencia. Como le explicamos antes el nivel se considera *Bajo* si el participante ha colaborado en un rango de 1 a 4 proyectos a nivel escolar, el siguiente nivel es *Medio* si ha colaborado en un rango de 1 a 4 proyectos en la industria y por ultimo *Alto* si ha colaborado en 5 o más proyectos en la industria.

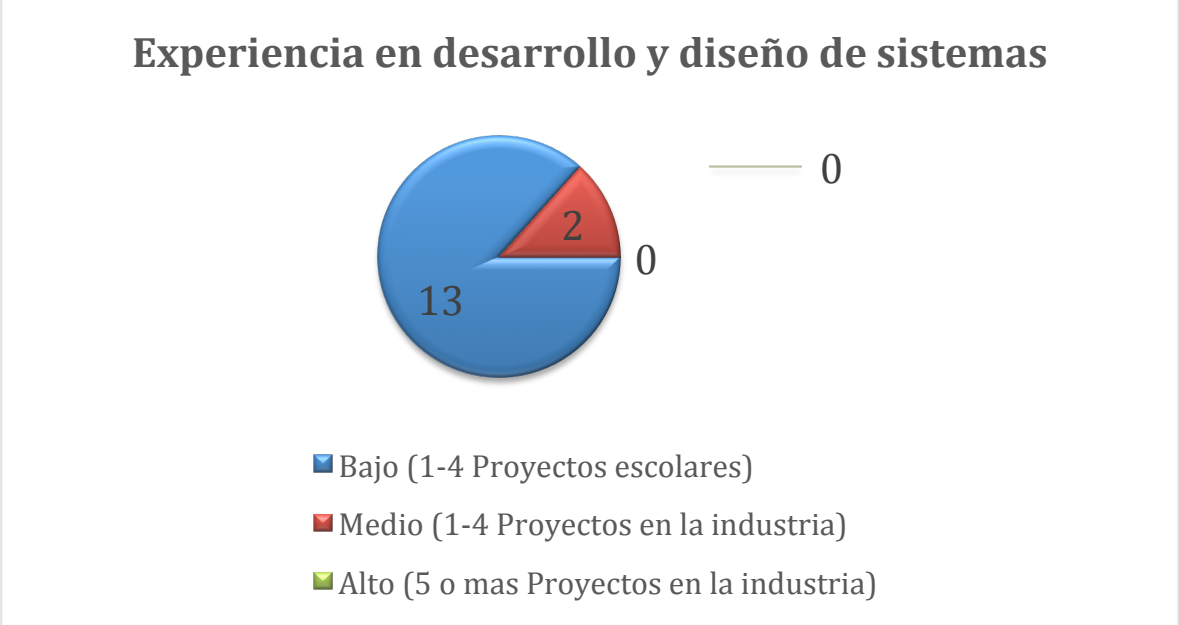


Figura 27. Nivel de experiencia en desarrollo y diseño de sistemas a nivel licenciatura.

Los datos de la Figura 27 revelan que casi el 90% de los participantes del grupo de nivel de licenciatura, poseen nivel *Bajo* con relación a la experiencia en desarrollo y diseño de sistemas. Esto se alinea a nuestro objetivo de que queremos asistir a arquitectos novatos.

5.3.1 Fase 1

En la Figura 28 se muestran los resultados obtenidos con respecto al tiempo de análisis durante la ejecución de la **Fase 1 (**análisis del patrón Pipes and Filters). Como se observa el tiempo mínimo que registraron los participantes es de ***6 minutos*** y el tiempo máximo es de ***18 minutos***.

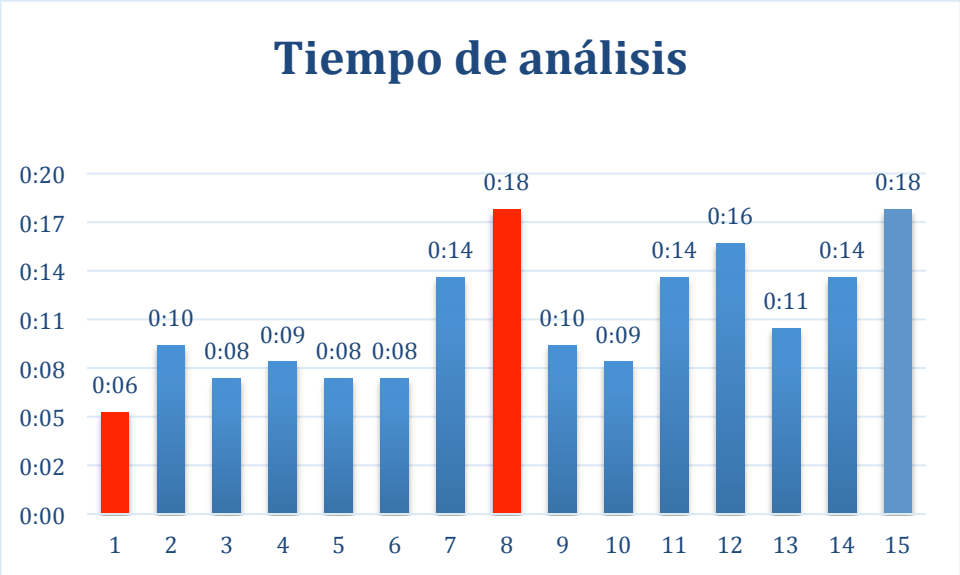


Figura 28: Tiempo de análisis de la Fase 1 en el grupo de licenciatura.

En la Figura 29 se muestran los resultados obtenidos para la métrica de *recall* de este experimento. Se observa que sólo tres participantes obtuvieron valores perfectos de *recall*. Esto es, solo tres participantes recuperaron todas las sentencias relevantes existentes en el texto del patrón. Descartando los valores de 1, solo tres participantes obtuvieron valores de al menos .80.

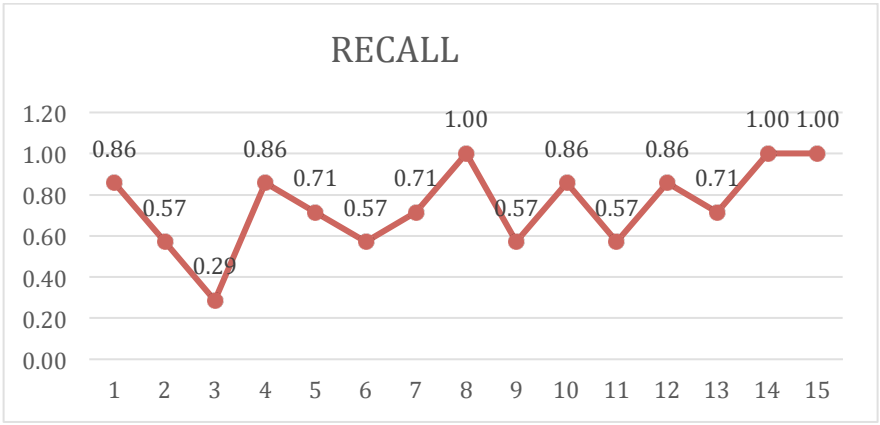


Figura 29. Resultados obtenidos para *recall* en la Fase 1 en el grupo de licenciatura.

En la Figura 30 se muestran los resultados obtenidos para la métrica *precision*. Solo 6 participantes obtuvieron valor de 1, lo que significa que todas las sentencias que recuperaron son relevantes.

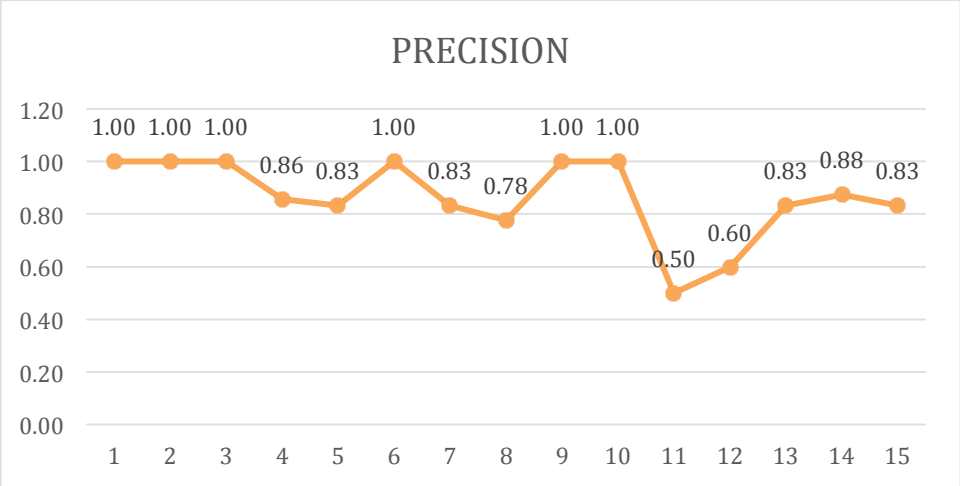


Figura 30. Resultados obtenidos para *precision* en la fase 1 en el grupo de licenciatura.

En la Figura 31 se hace la comparativa entre los valores obtenidos de *recall* y *precisión.* En ocho de los participantes se observa el fenómeno de que a menor *recall* mayor *precision*. Ver el participante 3, donde la *precision* es del 100% y su *recall* es de 29%.

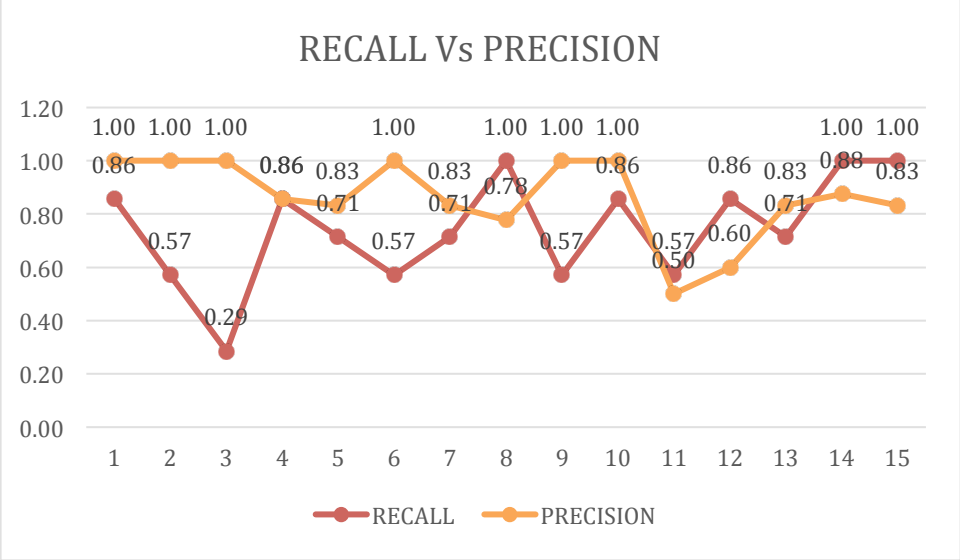


Figura 31. Comparativa de resultados de *precision* y *recall* en la fase 1 en el grupo de licenciatura.

Finalmente en la Figura 32 se muestran los resultados obtenidos en las respuestas de los participantes para de la pregunta formulada en esta fase del experimento:

Este patrón promueve ***performance (rendimiento/desempeño)***?

En el formato de los experimentos se consideraron principalmente las respuestas *'SI'* y *'NO'*, aunque también se agregó una tercera opción, por si el participante no tenía la certeza de contestar con las anteriores respuestas, seleccionando *'NO SÉ'*.

Pipes and Filters

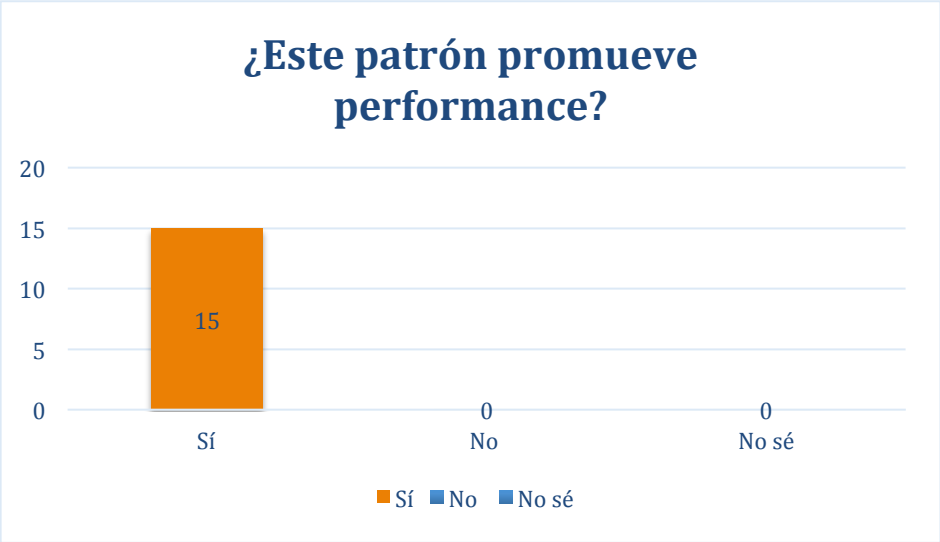


Figura 32. Respuestas obtenidas de los participantes en la fase 1 en el grupo de licenciatura.

El resultado en esta fase el 100% de los participantes respondieron que el patrón analizado que fue *Pipes and Filters* promueve *performance*, lo cual es correcto.

5.3.2 Fase 2

Para la **Fase 2** del experimento se obtuvieron los siguientes resultados:

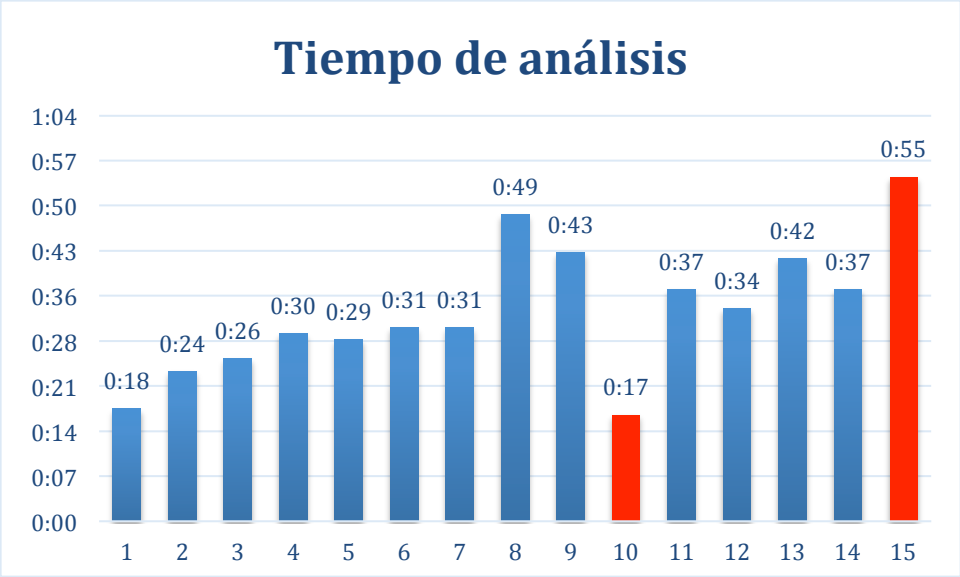


Figura 33. Tiempo de análisis de fase 2 en el grupo de licenciatura.

En esta fase el tiempo mínimo fue de **17 minutos** y el tiempo máximo registrado es de **55 minutos,** tomando en cuenta que el total de patrones analizados en esta fase fueron **3,** pero los textos estaban escrito por diferentes autores.

En la Figura 34 se muestran los resultados de la métrica de *recall*. Se observa que sólo cuatro participantes obtuvieron valores perfectos de *recall*. Esto es, solo cuatro participantes recuperaron todas las sentencias relevantes existentes en los textos de los patrones. Descartando los valores de 1, solo 2 participantes obtuvieron valores de al menos .80. Lo que indica que la recuperación de sentencias irrelevantes aumentó en esta fase comparado con la anterior.

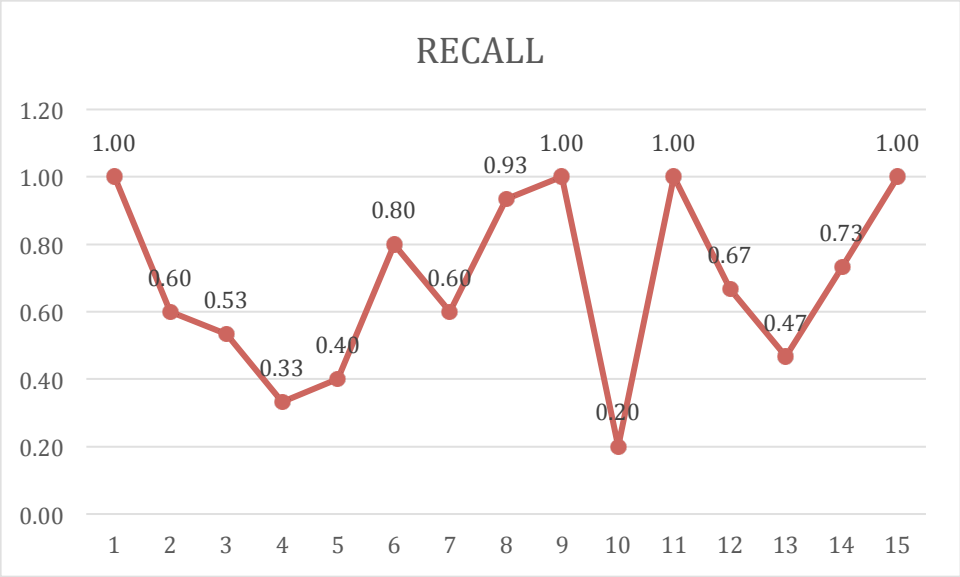


Figura 34. Resultados obtenidos para recall en la fase 2 en el grupo de licenciatura.

En la Figura 35 se muestran los resultados obtenidos para la métrica *precision*. Ningún participante obtuvo un valor de 1. El mejor resultado es el del participante 9 con un valor de .8.

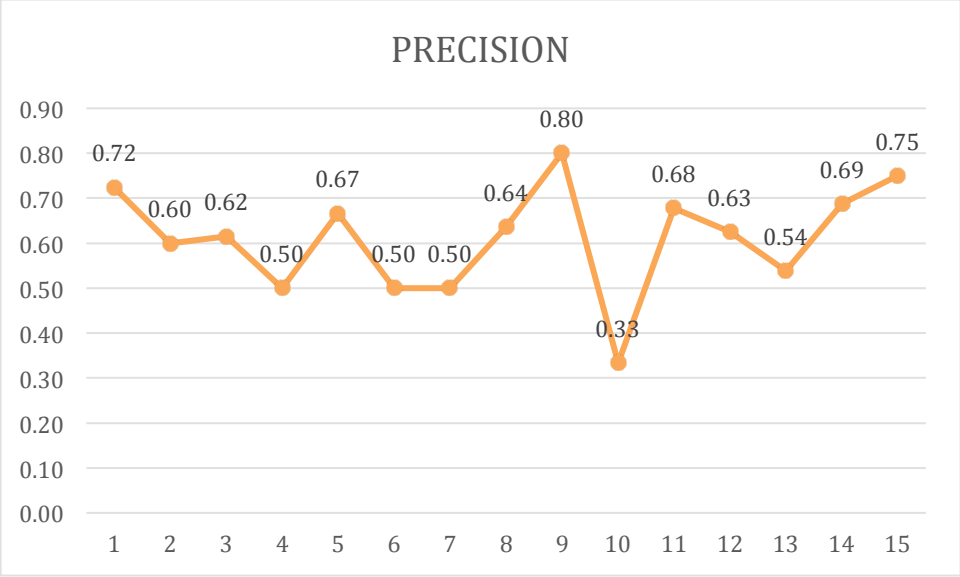


Figura 35. Resultados obtenidos para precision en la fase 2 en el grupo de licenciatura.

En la Figura 36 se hace la comparativa entre los valores obtenidos de *recall* y *precisión.* En nueve de los participantes se observa el fenómeno de que a mayor *recall* menor *precision*.

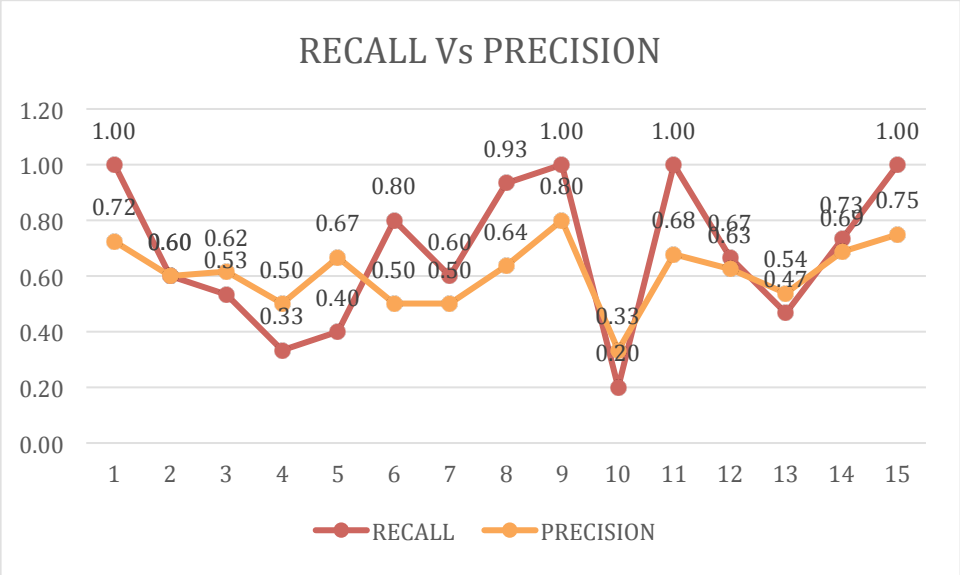


Figura 36. Comparativa de resultados de *precision* y *recall* en la fase 2 en el grupo de licenciatura.

Pipes and Filters

Figura 37. Respuestas obtenidas de los participantes en la fase 2 en el grupo de licenciatura para el patrón *Pipes and Filters*.

En esta fase el 86% de los participantes respondieron que los patrones analizados que fueron *Pipes and Filters,* redactados por diferentes autores, promueven *performance*, lo cual es correcto.

5.3.3 Fase 3

En la **Fase 3** se obtuvieron los siguientes datos:

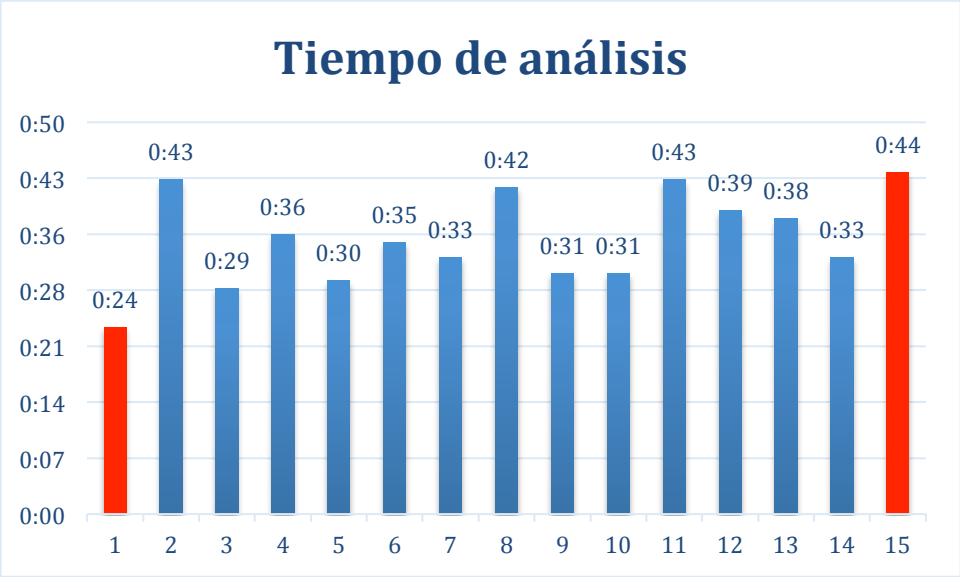


Figura 38. Tiempo de análisis de fase 3 en el grupo de licenciatura.

En esta fase el tiempo mínimo registrado fue de **24 minutos** y el tiempo máximo es de **44 minutos,** tomando en cuenta que el total de patrones analizados en esta fase fueron **4**.

En la Figura 39 se muestran los resultados obtenidos para *recall*. Se observa que sólo 4 participantes obtuvieron valores perfectos de *recall*. Esto es, solo 4 participantes recuperaron todas las sentencias relevantes existentes en el texto del patrón. Descartando los valores de 1, solo 2 participantes obtuvieron valores de al menos .80.

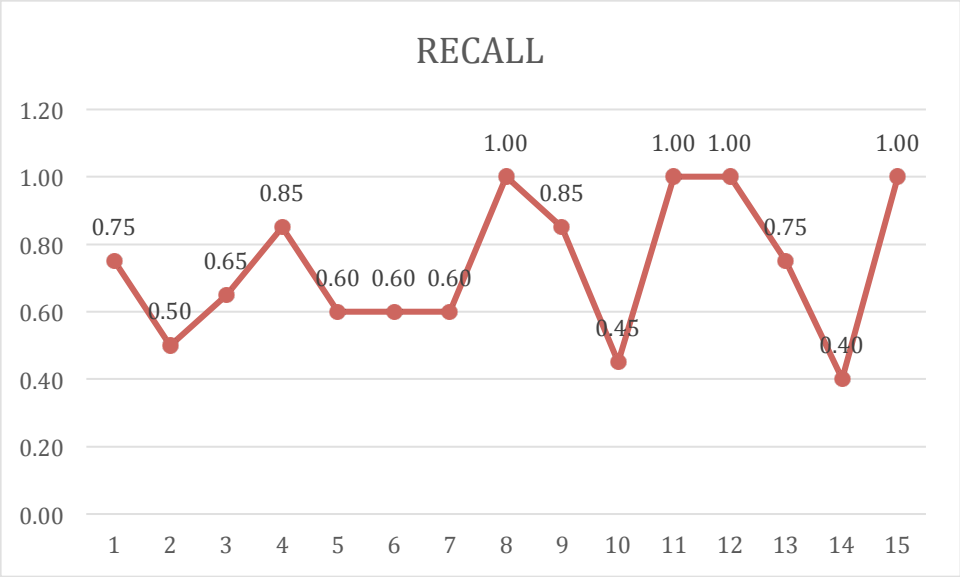


Figura 39. Resultados obtenidos para recall en la fase 3 en el grupo de licenciatura.

La Figura 40 muestra los valores de *precision* obtenidos. Ningún participante obtuvo un valor de 1. El mejor resultado fue de .85.

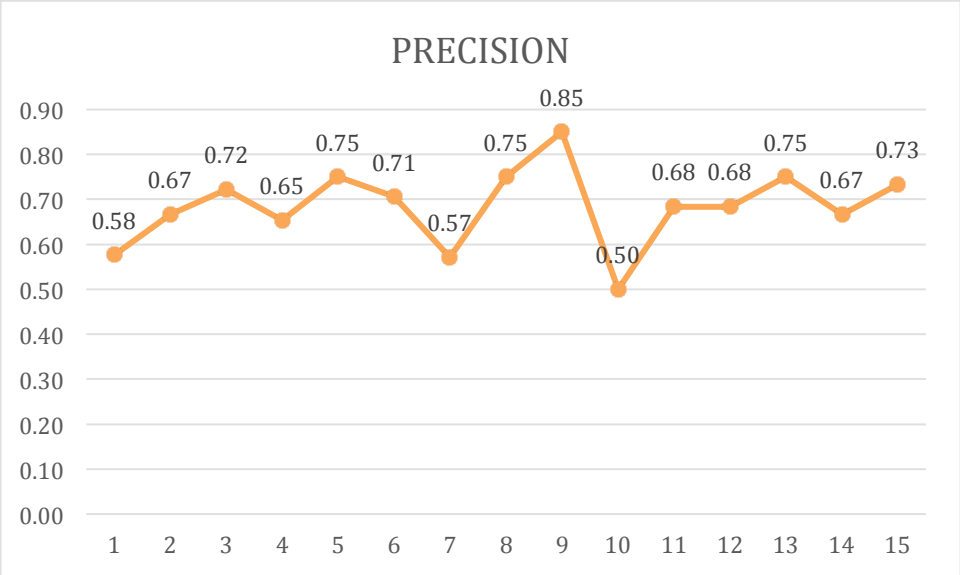


Figura 40. Resultados obtenidos para precision en la fase 3 en el grupo de licenciatura.

En la Figura 41 se hace la comparativa entre los valores obtenidos de *recall* y *precisión.* No se observa un fenómeno de los comentados antes que se dé de forma mayoritaria.

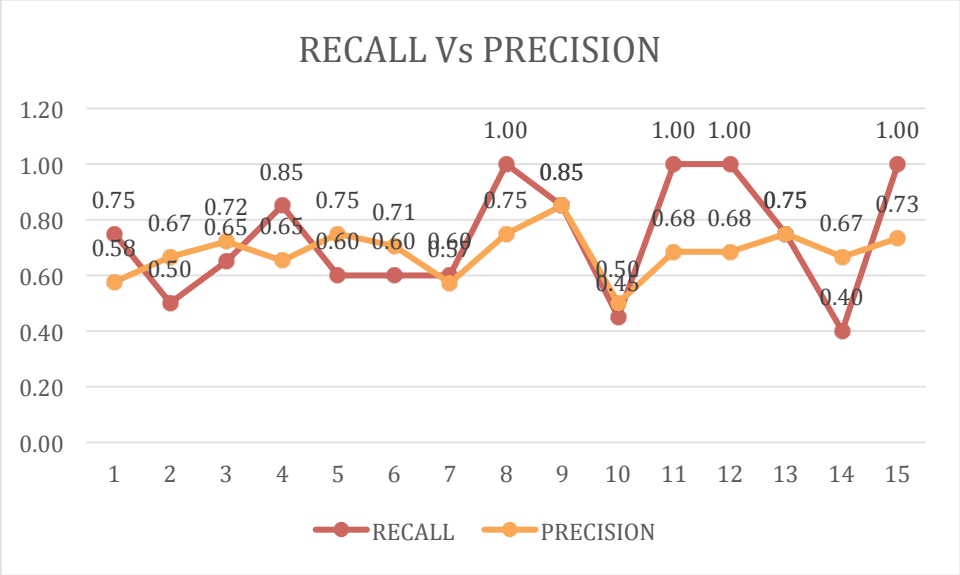


Figura 41. Respuestas obtenidas de los participantes en la fase 1 en el grupo de licenciatura.

En esta fase del experimento se analizaron 4 patrones: Half-Sync / Half-Async, Leader Followers, Active Object y Monitor Object, y la pregunta fue formulada individualmente para cada patrón, teniendo las siguientes respuestas:

Half-Sync / Half-Async

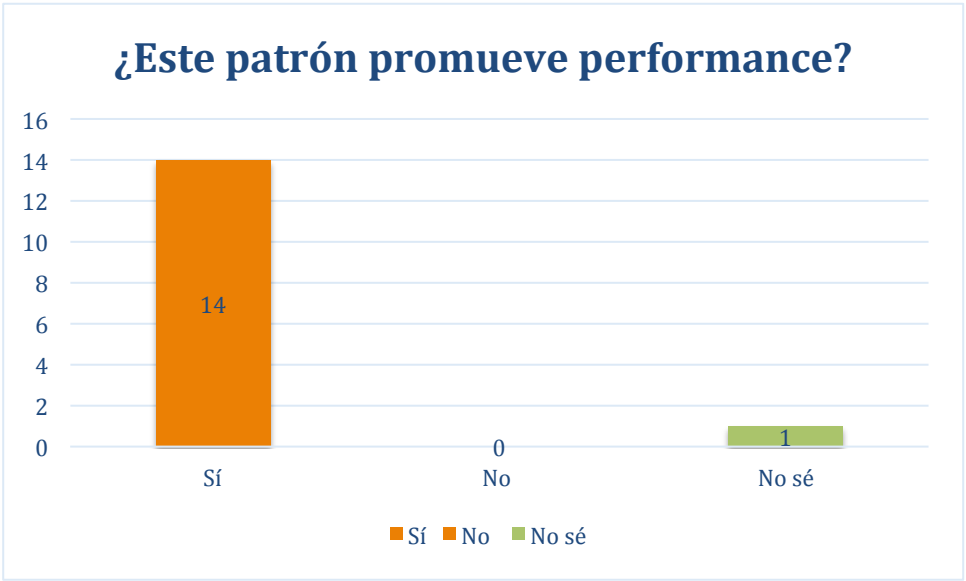


Figura 42. Respuestas obtenidas de los participantes en la fase 3 en el grupo de licenciatura para el patrón *Half-Sync / Half-Async.*

Leader Followers

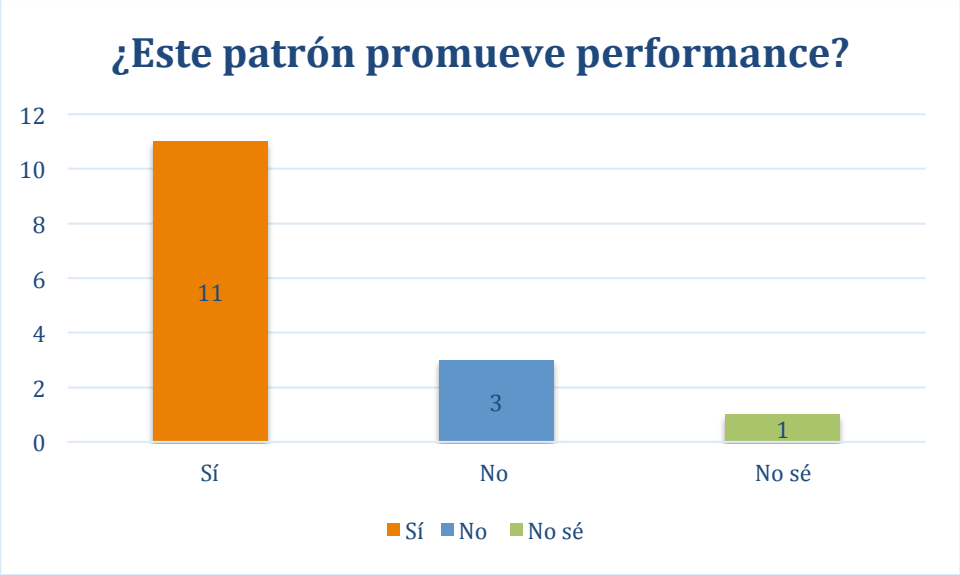


Figura 43. Respuestas obtenidas de los participantes en la fase 3 en el grupo de licenciatura para el patrón *Leader Followers.*

Active Object

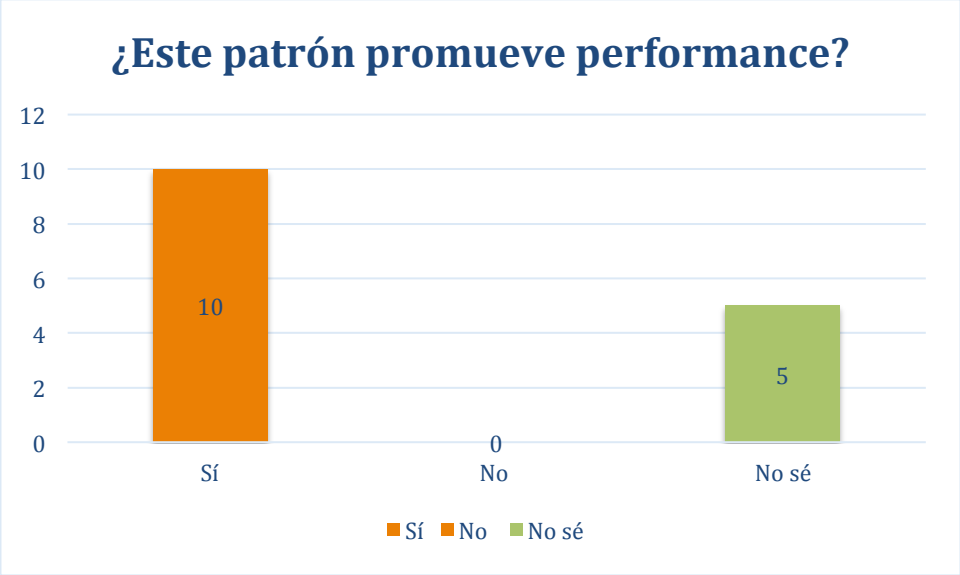


Figura 44. Respuestas obtenidas de los participantes en la fase 3 en el grupo de licenciatura para el patrón *Active Object.*

Monitor Object

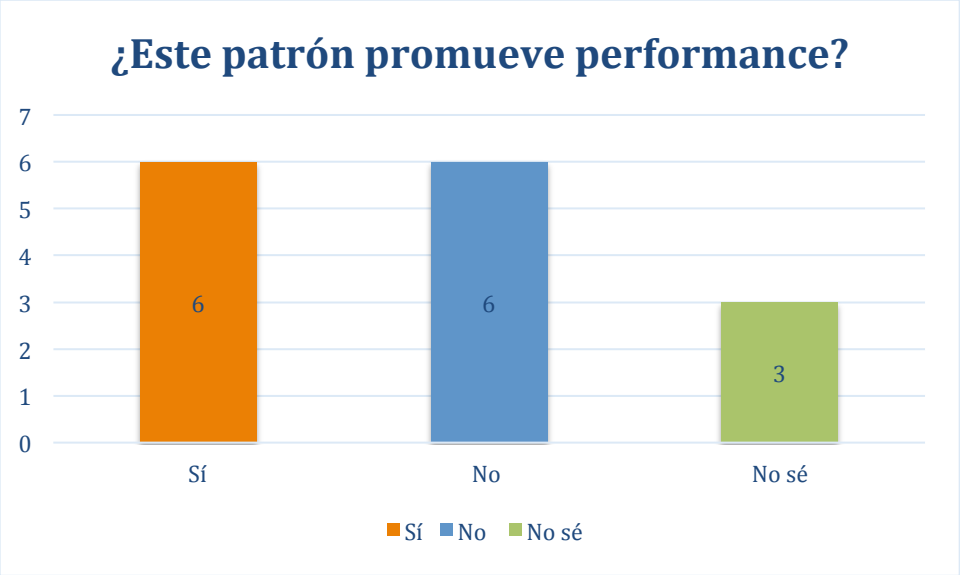


Figura 45. Respuestas obtenidas de los participantes en la fase 3 en el grupo de licenciatura para el patrón Monitor Object.

Los resultados obtenidos en las respuestas de los patrones ***Half-Sync / Half Async*** y ***Leader Followers*** al menos el 70% de los participantes respondieron que estos patrones ***SI*** promueven performance, mientras en las respuestas de los patrones ***Active Object*** y ***Monitor Object,*** el tipo de respuestas son más variadas. Por ejemplo, en las respuestas para el patrón ***Monitor Object*** *el 40%* dieron como respuesta ***SI****, otro 40% dio una respuesta* ***NO*** *y el 20% restante dio como respuesta* ***NO SÉ****. Y para* ***Active Object*** *un 66% respondio* ***SI*** *mientras un 33% respondio* ***NO.*** *Estos resultados quizá puedan estar relacionados al hecho de que no es tan evidente encontrar las sentencias donde se habla de performance en los patrones* ***Active Object*** y ***Monitor Object****.*

En las Tablas 9, 10 y 11 se comparan los resultados obtenidos del análisis manual realizado por los participantes versus el análisis automatizado realizado por la herramienta orión, con el objetivo de hacer evidente si se alcanzaron o no los objetivos de este trabajo.

En el *Gold Estándar* el número de sentencias relevantes para la Fase 1 son 7 (ver *Anexo 2: Gold Estándar,* ubicado en la siguiente url: *https://www.dropbox.com/s/xzousxfomaqrjzh/Anexo%202%20-%20Gold%20Est%C3%A1ndar.pdf?dl=0*). En las filas de la Tabla 9 se indican los resultados obtenidos para las métricas: tiempo de análisis, *recall*, *precision* y

la respuesta a la pregunta formulada. Por otra parte, las columnas refieren al análisis manual realizado por participantes y análisis automatizado realizado por la herramienta Orión. Donde podemos ver que en promedio en el análisis manual los participantes encontraron 6, de las 7 sentencias relevantes, mientras la herramienta orión es capaz de detectar las 7 sentencias relevantes. En cuanto al tiempo de análisis en promedio los participantes se tardaron 11 minutos en analizar el texto de 1 patrón, mientras la herramienta hace este proceso en menos de 1 minuto. Después para *recall* y *precision* los porcentajes de la herramienta orión, son bastante aceptables; al estar en un 96% en *recall* en comparación al análisis manual donde el porcentaje promedio obtenido es más bajo con un 74% y aunque en *precision* el 86% del análisis manual es más alto que el análisis automatizado que registra un 85%, esta diferencia se hace mínima en comparación al tiempo del análisis manual y el análisis automatizado. Y por último vemos en la tabla que se ha obtenido el mismo resultado al responder la pregunta en el análisis manual y el análisis automatizado con la herramienta Orión.

	Análisis Manual realizado por participantes (promedio)	Análisis Automatizado realizado por la Herramienta Orión
NUMERO DE SENTENCIAS RELEVANTES	6	7
TIEMPO DE ANÁLISIS	11:00 Minutos	00:40 Segundos
RECALL	0.74	0.96
PRECISION	0.86	0.85
RESPUESTA A PREGUNTA FORMULADA	SI	SI

Tabla 9: Comparación de resultados en análisis manual con análisis automatizado en Fase1 de Experimento 1.

El número de sentencias relevantes para la Fase 2 en nuestro *Gold Estándar* son 15 (ver *Anexo 2: Gold Estándar,* ubicado en la siguiente url: *https://www.dropbox.com/s/xzousxfomaqrjzh/Anexo%202%20-%20Gold%20Est%C3%A1ndar.pdf?dl=0*). En las filas de la Tabla 10 se indican los resultados obtenidos para las métricas: tiempo de análisis, *recall*, *precision* y la respuesta a la pregunta formulada. Por otra parte, las columnas refieren al análisis manual realizado por participantes y análisis automatizado realizado por la herramienta Orión. Donde podemos ver que en promedio en el análisis manual los participantes encontraron 12, de las 15 sentencias relevantes, mientras la herramienta orión solo detecto 9. Pero esto se ve compensado con los tiempos tiempo de análisis registrados donde el análisis manual en promedio las participantes se tardaron 33 minutos en leer tres textos del mismo patrón pero de tres autores distintos y en cambio la herramienta orión se tardo menos de 1 minuto. Para *recall*, el análisis automatizado de la herramienta orión tiene un 77% de efectividad en comparación al análisis manual con 68%. En *precision* también el análisis automatizado por la herramienta orión está por encima del porcentaje promedio obtenido en el análisis manual por los participantes y aunque no es considerable la diferencia, volvemos a hacer énfasis en la mejora en cuanto al tiempo, obtenida con la herramienta orión. Y por último vemos en la tabla que se ha obtenido el mismo resultado al responder la pregunta en el análisis manual y el análisis automatizado con la herramienta Orión.

	Análisis Manual realizado por participantes (promedio)	Análisis Automatizado realizado por la Herramienta Orión
NUMERO DE SENTENCIAS RELEVANTES	12	9
TIEMPO DE ANÁLISIS	33:00 Minutos	00:56 Segundos
RECALL	0.68	0.77
PRECISION	0.61	0.62
RESPUESTA A PREGUNTA FORMULADA	SI	SI

Tabla 10: Comparación de resultados en análisis manual con análisis automatizado en Fase2 de Experimento 1.

En la Fase 3 el número de sentencias relevantes en nuestro *Gold Estándar* son 20 (ver *Anexo 2: Gold Estándar,* ubicado en la siguiente url: *https://www.dropbox.com/s/xzousxfomaqrjzh/Anexo%202%20-%20Gold%20Est%C3%A1ndar.pdf?dl=0*). En las filas de la Tabla 11 se indican los resultados obtenidos para las métricas: tiempo de análisis, *recall*, *precision* y la respuesta a la pregunta formulada. Por otra parte, las columnas refieren al análisis manual realizado por participantes y análisis automatizado realizado por la herramienta Orión. Donde podemos ver que en promedio en el análisis manual los participantes encontraron 16, de las 20 sentencias relevantes, mientras la herramienta orión detecto 25, es decir encontró 5 sentencias de más. Teniendo un *recall* de 75%, pero impactando a la *precision* lo cual es más baja con un 72%, aunque comparando estos porcentajes con los obtenidos en el análisis manual son mejores tanto en *recall* como en *precision* donde en promedio se ha obtenido un 73% para *recall* y un 68% para *precision*. Mientras el tiempo promedio del análisis manual fue de 35 minutos, el análisis automatizado con la herramienta orión es de 1.15 minutos y por último se ha obtenido el mismo resultado al responder la pregunta en el análisis manual y el análisis automatizado con la herramienta Orión.

	Análisis Manual realizado por participantes (promedio)	Análisis Automatizado realizado por la Herramienta Orión
NUMERO DE SENTENCIAS RELEVANTES	16	25
TIEMPO DE ANÁLISIS	35 Minutos	1.15 Minutos
RECALL	0.73	0.75
PRECISION	0.68	0.72
RESPUESTA A PREGUNTA FORMULADA	SI	SI

Tabla 11: Comparación de resultados en análisis manual con análisis automatizado en Fase 3 de Experimento.

6 RESUMEN Y TRABAJO FUTURO

En este capítulo se presenta resumen de este trabajo y propuestas de trabajo futuro.

6.1 Resumen

La arquitectura de un sistema de software es un modelo, que se genera en etapas tempranas de la fase de diseño. La arquitectura es un artefacto importante porque representa la estructuración general del sistema y la manera en que se estructura un sistema tiene un impacto directo sobre la capacidad de éste para satisfacer requerimientos. Especialmente, requerimientos de atributos de calidad.

Existen diversos métodos para soportar el diseño de la arquitectura. En muchos de estos métodos se hace uso del concepto de patrones de diseño. Así, estos métodos asumen que los arquitectos que los utilizan conocen este concepto y sus características relevantes; por ejemplo qué atributos de calidad promueve o inhibe determinado patrón. En la práctica, sin embargo, esto no es así; o al menos no es así para arquitectos novatos. La falta de formación académica en materia de patrones de diseño así como factores como:

a) el número de patrones que existe actualmente,
b) el surgimiento de nuevos patrones,
c) la descripción heterogénea de los patrones,
d) el idioma en el que están escritos los patrones y
e) la falta de herramientas para soportar estas tareas

(que hemos explicado en el Capítulo 1) hace que la identificación y selección de patrones sea una tarea generalmente costosa en tiempo y esfuerzo.

Este trabajo contribuye a mitigar muchos de estos factores mediante la definición de un método, y herramienta de soporte, para asistir a los arquitectos novatos en la identificación y selección de patrones de diseño. Específicamente, patrones que promueven el atributo de calidad desempeño.

Consideramos que los objetivos definidos al inicio de este proyecto se han cumplido y a continuación respaldamos esta afirmación:

a) *permitir el análisis automático de patrones de diseño arquitectónico, descritos en lenguaje natural en el idioma inglés, con el propósito de responder a la pregunta si un determinado patrón satisface o no el atributo de calidad desempeño*.

   Se definió un método, y la herramienta Orión que lo automatiza, para este propósito usando técnicas de extracción de información y de representación del conocimiento. Los detalles del método y la herramienta Orión están descritos en el Capítulo 4.

b) *lograr la escalabilidad del análisis con respecto a la heterogeneidad de la redacción del texto del patrón*.

   Se realizaron un conjunto de experimentos demostrándose que con la herramienta Orión es posible analizar textos de patrones escritos por diferentes autores de forma efectiva. Ver, por ejemplo la Tabla 10 del Capítulo 5.

c) *disminuir el tiempo que invierte un arquitecto novato en el análisis "manual" del texto de un patrón*.

   Se realizaron un conjunto de experimentos demostrándose que mediante el uso de la herramienta Orión es posible reducir significativamente el tiempo de análisis de un patrón comparado con el análisis "manual" realizado por un arquitecto novato. Ver, por ejemplo las Tablas 9, 10 y 11 del Capítulo 5.

d) *tener una exactitud en el resultado del análisis cercana a la que tendría un arquitecto experimentado*.

   Se realizaron un conjunto de experimentos demostrándose que mediante el uso de la herramienta Orión es posible lograr una precisión cercana a la del análisis "manual" realizado por un arquitecto experimentado. Ver, por ejemplo las Tablas 9, 10 y 11 del Capítulo 5.

6.2 Trabajo Futuro

A continuación se discuten tres posibles líneas de trabajo futuro.

a) Extender la herramienta para incluir más atributos de calidad.

Por cuestiones de practicidad y alcance, en este trabajo se decidió trabajar con el atributo de desempeño debido a que es un atributo de calidad importante para prácticamente cualquier tipo de sistema y existen muchos patrones de diseño que promueven este atributo de calidad.

De esta forma, sería interesante extender la herramienta considerando otros atributos de calidad como por ejemplo los incluidos en modelos de calidad descritos en el Capítulo 2.

b) Mejorar el diseño de las ontologías.

En este trabajo se definieron y poblaron dos ontologías: una de conceptos clave relacionados al atributo de calidad desempeño y una de palabras comúnmente utilizadas en sentencias, en textos de patrones, para referirse positiva o

negativamente al atributo de calidad desempeño. Sin embargo, estas ontologías fueron definidas usando listas en el contexto de la API de la herramienta GATE.

Las listas podrían considerarse como *ontologías descriptivas* que permiten incluir descripciones, taxonomías de conceptos, relaciones entre los conceptos y propiedades, pero no permiten inferencias lógicas. De esta forma, como trabajo futuro sería bueno explorar qué tipo de inferencias son relevantes en el contexto de ese trabajo y considerar el uso de una *ontología lógica*.

c) Mejorar el algoritmo utilizado para responder a la pregunta YES/NO.

Una vez detectadas las sentencias en la cuales se habla positiva o negativamente del atributo de calidad desempeño, la herramienta Orión responde la pregunta de tipo Yes/No:

Does the <pattern_name> promote performance?

En la versión actual de Orión el algoritmo que responde a esta pregunta cuenta las anotaciones *Promotes* e *Inhibits* detectadas. Si el número de anotaciones *Promotes* es mayor al de *Inhibits* entonces la respuesta es *yes.* En caso contrario, la respuesta es *no.* Cuando la cantidad de anotaciones en ambas categorías es la misma, queda a criterio de quien ejecuta el análisis determinar si es un si o no y la herramienta muestra la etiqueta unanswered.

Sin embargo, en el idioma inglés existen palabras que impactan en cómo debe interpretarse una sentencia diciendo que se promueve o inhibe el atributo de calidad. Por ejemplo, los auxiliares *can*, *might*, *could*, *may.* De esta forma, sería conveniente considerar el uso de pesos o ponderaciones asociados a conceptos y palabras claves en las dos ontologías y redefinir este algoritmo.

7 REFERENCIAS BIBLIOGRÁFICAS

[1] L. Bass, P. Clements and R. Kazman. Software Architecture in Practice Third Edition, 2012.

[2] R. Wojcik and P. Clements, “Attribute-Driven Design (ADD), Version 2 . 0”. November, 2006.

[3] A. J. Lattanze. Architecting Software Intensive Systems: A Practitioner’s Guide, 2008.

[4] N. Rozanski and E. Woods. Software Systems Architecture Working with Stakeholders Using Viewpoints and Perspectives, 2012.

[5] Hibernate. http://hibernate.org/.

[6] Spring. http://spring.io/.

[7] F. Buschmann, K. Henney, and D. Schmidt. Pattern-Oriented Software Architecture Volume 4, A Pattern Language for Distributed Computing, 2007.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, 1995.

[9] T. Rischbeck and T. Erl. SOA Design Patterns (The Prentice Hall Service-Oriented Computing Series from Thomas Erl), 2009.

[10] M. Fowler. Patterns of Enterprise Application Architecture, Addison-Wesley Longman Publishing Co., 2002.

[11] R. Daigneau. Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web ServicesAddison Wesley, 2011.

[12] Speed Reading Facts. http://www.execuread.com/facts/.

[13] A. Jansen, J. van der Ven, P. Avgeriou, and D. K. Hammer. Tool Support for Architectural Decisions. In Proceedings of the Working IEEE/IFIP Conference on Software Architecture, 2007.

[14] M. A. Babar and I. Gorton. A tool for managing software architecture knowledge. In Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, 2007.

[15] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas. A web-based tool for managing architectural design decisions. In proceedings of the Workshop on SHaring ARchitectural Knowledge (SHARK), 2006.

[16] H. Cunningham, I. Roberts, and M. A. Greenwood, "Developing Language Processing Components with GATE Version 7 (a User Guide)," vol. 7, no. Gate 2, 2012.

[17] T. Tudorache, C. Nyulas, N. F. Noy, and M. a Musen, "WebProtégé: A Collaborative Ontology Editor and Knowledge Acquisition Tool for the Web.," *Semant. Web*, vol. 4, no. 1, pp. 89–99, Jan. 2013.

[18] D. Jamwal, "Analysis of Software Quality Models for Organizations," vol. 1, no. 2, pp. 19–23, 2010.

[19] M. C. M. Ing, "Software Quality Model Requirements for Software Quality Engineering," no. 2000, 2003.

[20] K. Wiegers and J. Beatty. Software Requirements, 3rd Edition Developer Best Practices), Microsoft Press, 2013.

[21] Pipes and Filters. http://msdn.microsoft.com/en-us/library/ff647419.

[22] Gate. https://gate.ac.uk.

[23] Calais: Connect Everything. http://www.opencalais.com/.

[24] Anderson Analytics. http://www.andersonanalytics.com/.

[25] Attensity. http://www.attensity.com/.

[26] Textalytics. https://textalytics.com/inicio.

[27] TextAnalyst. http://megaputer.com/site/textanalyst.php.

[28] Textalyser. http://textalyser.net/.

[29] WordStat. http://provalisresearch.com/es/productos/software-de-analisis-de-contenido/.

[30] D. Binkley and D. Lawrie, "Information Retrieval Applications in Software Development.", 2011.

[31] Lee, S.W.; Rine, D. Missing requirements and relationship discovery through proxy viewpoints model. In Proceedings of ACM Symposium on Applied

Computing, Nicosia, Cyprus, March 2004; 1513 – 1518, no. March, p. 2004, 2004.

[32] Maarek, Y.; Berry, D.; Kaiser, G. An information retrieval approach for automatically constructing software libraries. IEEE Trans. Softw. Eng. 1991, 17 (8), 800–813.

[33] De Lucia, A. Fasano, F. Oliveto, R. and Tortora, G.Recovering traceability links in software artefact management systems using information retrieval methods. ACM Trans. Softw. Eng. Method.2007, 16(4), 13–70.

[34] Wood, M.; Sommerville, I. An information retrieval system for software components. SIGIR Forum, 1988, 22(3–4), 11–28..

[35] Yao, H.; Etzkorn, L.; Virani, S. Automated classification and retrieval of reusable software components. J. Am. Soc. for Info. Sci. Technol.,2008, 59(4), 613–627.

[36] R. Vlas and W. N. Robinson, "A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects Related research," pp. 1–10, 2011.

[37] G. Teamware, U. Guide, M. Petrillo, and J. Baycroft, "Introduction to Manual Annotation," no. April, 2010.

[38] A. Rashwan, O. Ormandjieva, and R. Witte, "Ontology-Based Classification of Non-functional Requirements in Software Specifications: A New Corpus and SVM-Based Classifier," *2013 IEEE 37th Annu. Comput. Softw. Appl. Conf.*, no. 1, pp. 381–386, Jul. 2013.

[39] WordNet. http://wordnet.princeton.edu/.

[40] L. Hirschman and R. Gaizauskas, "Natural language question answering: the view from here," *Nat. Lang. Eng.*, vol. 7, no. 04, pp. 275–300, Feb. 2002.

[41] E. Hovy, U. Hermjakob, D. Ravichandran, and M. Rey, "A Question / Answer Typology with Surface Text Patterns."

[42] H. Kanayama and P. John, "Answering Yes / No Questions via Question Inversion," vol. 1, no. December 2012, pp. 1377–1392.