Un Enfoque para Automatizar la Refactorización de Casos de Uso

Alejandro Rago[1,2], Paula Frade[2], Miguel Ruival[2], Claudia Marcos[1,2]

[1] Instituto Superior de Ingeniería de Software Tandil (ISISTAN), CONICET
Campus Universitario, Paraje Arroyo Seco, B7001BBO, Tandil, Bs. As., Argentina

[2] Facultad de Ciencias Exactas (ISISTAN), UNICEN University
Campus Universitario, Paraje Arroyo Seco, Tandil, Bs. As., Argentina

{arago,cmarcos}@exa.unicen.edu.ar - {maripau07,miguebt}@gmail.com

Resumen Llevar a cabo las actividades de captura y modelamiento de requerimientos no es una tarea sencilla. Ésta requiere realizar un análisis profundo de las necesidades de los clientes y demanda cierto grado de experiencia de los analistas. Para comunicar satisfactoriamente los requerimientos, se deben aprovechar los instrumentos provistos por las técnicas de especificación (por ejemplo, relaciones entre casos de uso) de forma tal que se evite la redundancia y se promueva el reuso y abstracción de comportamiento. En la práctica, estos instrumentos no son utilizados tanto como deberían y es necesario que los analistas revisen los documentos periódicamente para preservar la calidad de los requerimientos. Desafortunadamente, inspeccionar los documentos manualmente es una tarea compleja y ardua. Por ello, en este artículo presentamos un enfoque semi-automático que permite encontrar defectos en las especificaciones de casos de uso y solucionarlos por medio de refactorizaciones. El enfoque implementa heurísticas avanzadas para localizar los defectos (por ej., comportamientos duplicados) y mecanismos que agilizan su resolución. El enfoque fue evaluado en sistemas reales, obteniendo resultados preliminares prometedores.

1. Introducción

Para que un proyecto de software sea exitoso, es vital comprender correctamente las necesidades del cliente. Una buena especificación de requerimientos es un aspecto primordial para lograr este cometido, permitiendo documentar y así comunicar las propiedades de un sistema de manera efectiva a través de un proceso de desarrollo [9]. A pesar de la experiencia adquirida por la industria en las últimas décadas respecto a la Ingeniería de Requerimientos (RE), es común que las organizaciones tengan dificultades para elicitar, documentar y administrar los requerimientos de sus clientes. Muchos proyectos fracasan al momento de entregar la funcionalidad dentro del tiempo y presupuesto acordado con el cliente por tener requerimientos de mala calidad [11].

Los requerimientos son generalmente descriptos por medio de especificaciones escritas en lenguaje natural. Una de las técnicas más difundidas y potentes

para documentar requerimientos son los casos de uso. Los casos de uso son sencillos de comunicar [10] y están soportados por guías de buenas prácticas [5]. Desafortunadamente, los casos de usos desarrollados en la práctica contienen un gran número de defectos, tales como la duplicación de funcionalidad [4], la falta de abstracción [13], o requerimientos dispersos y entremezclados [3].

Las especificaciones defectuosas, además de ser complejas de entender y comunicar, acarrean dificultades a lo largo de todo el proceso de desarrollo de un sistema. En los últimos años, varios investigadores han estudiado estrategias para poder identificar los defectos presentes en las especificaciones y han desarrollado mecanismos que permiten solucionarlos [18,19,4]. Estos mecanismos son, generalmente, agrupados en lo que se denominan catálogos de refactorizaciones. Lamentablemente, la aplicación de dichos catálogos es responsabilidad del analista, el cual debe inspeccionar los requerimientos a mano en la ardua tarea de encontrar los defectos para así mejorar la calidad de sus especificaciones. En este contexto, este trabajo presenta un enfoque iterativo denominado *ReUse*, el cual ayuda a los analistas a identificar defectos potenciales en especificaciones de casos de uso mediante técnicas de procesamiento de texto avanzadas y los asiste en su resolución utilizando un catálogo de refactorizaciones. El enfoque esta implementado en un herramienta prototipo que facilita la interacción entre los analistas y el enfoque durante el proceso de mejora de los requerimientos. Este trabajo realiza dos contribuciones al estado del arte. Primero, la definición de un enfoque iterativo que permite la mejora progresiva de las especificaciones de requerimientos mediante una herramienta semi-automática, guiando al analista tanto en la búsqueda de defectos como en la selección y ejecución de refactorizaciones. Segundo, la implementación de un componente que permite descubrir funcionalidad duplicada en los documentos mediante la combinación de técnicas de *Procesamiento de Lenguaje Natural* (NLP) y *Aprendizaje de Maquina* (ML) con técnicas de *Alineamiento de Secuencias* (usadas generalmente en el área de investigación genética). Adicionalmente, se llevó a cabo una evaluación del prototipo sobre varios sistemas, obteniendo resultados preliminares alentadores.

El resto de este trabajo esta organizado de la siguiente manera. La Sección 2 analiza trabajos que tratan la temática de defectos en requerimientos. La Sección 3 introduce el enfoque y describe la organización de sus componentes. La Sección 4 reporta los resultados de los experimentos. Finalmente, la Sección 5 presenta las conclusiones y discute futuras lineas de investigación.

2. Trabajos Relacionados

En la actualidad es posible encontrar una gran diversidad de trabajos que proponen estrategias para mejorar la calidad de las especificaciones de requerimientos. El proceso general planteado para la mejora de requerimientos se puede organizar en tres partes: la evaluación (o análisis) de las especificaciones textuales, la identificación de defectos, y la aplicación de soluciones basadas en diversas técnicas.

Pocos trabajos han explorado la búsqueda de indicios de problemas que resultan útiles para guiar y enfocar el proceso de mejora general de documentos de requerimientos. El trabajo de Ciemniewska y otros [4] provee técnicas para identificar defectos en especificaciones de casos de uso. Los defectos se organizan en tres niveles: a nivel de especificaciones, de casos de uso, y de pasos. El *nivel de especificaciones* considera la duplicación de comportamiento. El *nivel de casos de uso* considera casos de uso muy cortos o largos, o extensiones complicadas, entre otros. El *nivel de pasos* considera estructuras sintácticas complejas, u omisión de actores, entre otros. Aunque este trabajo no propone refactorizaciones para solucionar los defectos, es relevante porque provee heurísticas sencillas para identificar los defectos en el texto.

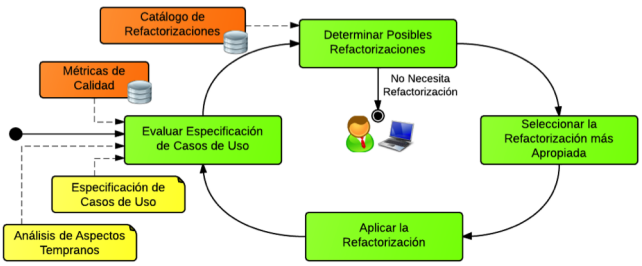
También existen una serie de trabajos que basan el proceso de mejora de requerimientos en la definición de catálogos de refactorización. Algunos de estos trabajos formalizan el proceso de análisis de la documentación mediante la técnica *Goal Question Metrics* (GQM) [17]. GQM prescribe la construcción de un plan de evaluación y de mejoras, definiendo un modelo de métricas que permite determinar la calidad de los artefactos de software. Ramos y otros han instanciado el framework GQM en el contexto de casos de uso [13]. Su enfoque, llamado *AIRDoc*, permite mejorar la calidad de los casos de uso mediante evaluaciones y refactorizaciones progresivas. Una evaluación permite determinar dónde yace un problema, mientras que una refactorización ayuda a resolver el problema. Para cada problema definido, el enfoque describe mecanismos que permiten solucionarlos. Sin embargo, la identificación de los defectos debe ser realizada por un analista de forma manual.

Otros trabajos dejan la formalización de lado, y se enfocan más en la mejora propiamente dicha. Rui y otros estudiaron la aplicación de técnicas de refactorización en casos de uso [14]. Las refactorizaciones son organizadas usando un meta-modelo de casos de uso. Este meta-modelo está conformado por conceptos en distintos niveles, tales como casos de uso, escenarios, actores, entre otros. Sin embargo, la tarea de determinar "dónde" y "cómo" una refactorización debe ser aplicada es incierto, y se deja esta responsabilidPd al criterio de los analistas. En la misma linea, Yu y otros sugieren que para asegurar la construcción de casos de uso con una granularidad y nivel de abstracción adecuada, es necesario tener en cuenta la funcionalidad común, eliminando el comportamiento redundante y motivando el reuso de funcionalidad [19]. Con este propósito, este trabajo introduce un catálogo de refactorizaciones focalizado en resolver defectos recurrentes en las especificaciones. El catálogo utiliza el concepto de episodios para describir las refactorizaciones. Un episodio es definido de manera recursiva como: una comportamiento complejo que esta compuesto de escenarios más simples o una unidad atómica de comportamiento (es decir, una interacción). Para emplear una refactorización, se debe alinear el modelo de episodios a la especificación y luego realizar las transformaciones necesarias. Similarmente a [14], la identificación de los defectos esta fuera del alcance de este trabajo.

3. *ReUse*: Un Enfoque para la Refactorización de Casos de Uso

Con el objetivo de simplificar los esfuerzos de los analistas a la hora de inspeccionar los documentos de requerimientos y asegurar que los requerimientos cumplen con ciertos criterios de calidad, hemos definido un enfoque de naturaleza iterativa que permite automatizar la tarea de refactorizaciones de casos de uso. Este enfoque, denominado *ReUse* (Refactoring assistant for Use cases), permite encontrar defectos en las especificaciones de casos de uso de manera automática y asiste a los analistas en la resolución de dichos defectos mediante refactorizaciones. El enfoque *ReUse* toma como entrada un conjunto de *especificaciones de casos de uso* y el *análisis de aspectos tempranos* (opcional). Adicionalmente a las entradas, el enfoque utiliza también una serie de artefactos desarrollados específicamente para este trabajo. Estos artefactos son las *métricas de calidad* para casos de uso y un *catálogo de refactorizaciones*. Las métricas definen propiedades que permiten evidenciar los defectos. El catálogo de refactorizaciones es un conjunto de soluciones preexistentes a defectos recurrentes en casos de uso.

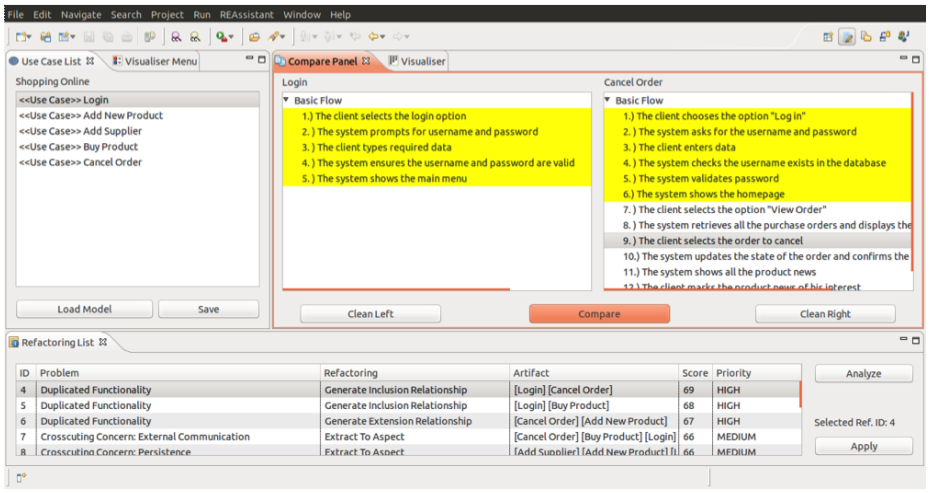
Figura 1. Enfoque para la Refactorización de Casos de Uso



El enfoque *ReUse* está dividido en cuatro actividades principales que llevan a cabo la mejora de la especificación (ver Figura 1). La actividad Evaluar Especificación de Casos de Uso tiene como objetivo determinar los defectos presentes en los documentos de requerimientos utilizando las métricas de calidad definidas. La salida de esta actividad es un conjunto de defectos identificados, tales como la duplicación de comportamiento, la falta de abstracción y la definición de relaciones inapropiadas entre casos de uso. La actividad Determinar Posibles Refactorizaciones tiene como propósito determinar qué refactorizaciones se ajustan mejor para resolver los defectos identificados previamente. Como resultado, esta actividad genera un conjunto de sugerencias de refactorizaciones para cada defecto. La actividad Seleccionar la Refactorización Apropiada tiene como objetivo determinar cuál es la sugerencia (entre el conjunto de posibles soluciones) que provee el mayor beneficio para mejorar la calidad de los

requerimientos. Con este propósito, se priorizan las refactorizaciones sugeridas en base a la combinación de distintos criterios de comparación, generando una especie de "ranking". Este ranking es presentado a los analistas, los cuales podrán seleccionar la sugerencia de mayor prioridad según el enfoque o bien alguna otra sugerencia particular según su criterio personal y experiencia. Por último, la actividad Aplicar la Refactorización toma la refactorización seleccionada y la aplica sobre la especificación. Esto significa realizar una serie de pasos o transformaciones que permiten resolver el defecto de manera sistemática. En caso de que la ejecución de la refactorización requiera información adicional (por ej., el nombre de un nuevo casos de uso), esta es solicitada al analista. La salida de esta actividad es una especificación de casos de uso parcialmente mejorada, la cual constituye la entrada de la siguiente iteración del enfoque.

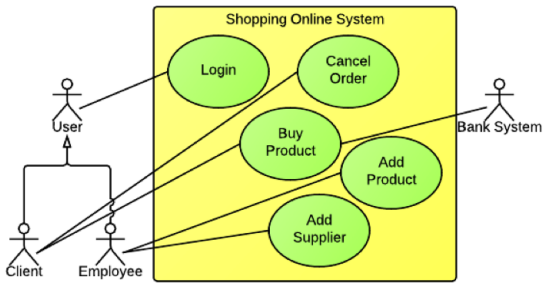
Figura 2. Interfaz Gráfica de la Herramienta Prototipo



El enfoque propuesto se encuentra materializado como un conjunto de plugins de Eclipse, los cuales permiten a los analistas interactuar durante todo el proceso iterativo del enfoque. La Figura 2 muestra cómo la herramienta presenta al analista una serie de defectos detectados en un sistema. Para comprender mejor el enfoque propuesto, se explicarán las actividades del enfoque mediante un ejemplo. Considere el siguiente subconjunto de casos de uso de un sistema de compras en línea (Figura 3). La especificación de dicho sistema cuenta con cinco casos de uso. El caso de uso *Login* permite que cada usuario, sea empleado o cliente, se identifique en el sistema. Los casos de uso *Add Product* y *Add Supplier* permiten a los empleados de la compañía dar de alta en el sistema productos y proveedores. Finalmente, los casos de uso *Buy Product* y *Cancel Order* permiten a los clientes realizar la compra de productos y la cancelación de

pedidos, respectivamente. A continuación se describen en detalle las actividades que constituyen el enfoque *ReUse*.

Figura 3. Casos de Uso del Sistema de Compras en Línea



3.1. Evaluar Especificación de Casos de Uso

El objetivo de esta actividad es recolectar información que permita identificar los defectos existentes en una especificación. Al analizar los casos de uso se obtiene un conjunto de medidas. Por ejemplo, luego de analizar los casos de uso *Login* y *Cancel Order*, la herramienta encuentra evidencias de defectos de funcionalidad duplicada, dado que ambos casos de uso describen los pasos necesarios para ingresar al sistema.

Para determinar los defectos en las especificaciones, nuestro enfoque define un esquema de objetivos, preguntas y métricas basadas en la técnica GQM (Goal-Question-Metric) [17]. Esta técnica permite establecer los parámetros a tener en consideración durante la inspección de los casos de uso de forma ordenada y clara. Primero, se definieron un conjunto de objetivos de calidad y de preguntas que permiten responder si dichos objetivos se cumplen. Se definieron dos objetivos de calidad: (i) *Reuso / Modificabilidad*, que determinan el grado de reuso y la capacidad de modificación presentes en los documentos observando principalmente el encapsulamiento y la duplicación de información, y (ii) *Entendibilidad / Mantenibilidad*, que permite determinar el grado de entendibilidad y mantenibilidad de los documentos, teniendo en cuenta principalmente indicadores de simpleza y claridad, como así también la correcta utilización de los distintos instrumentos de UML involucrados en los modelos. Con el propósito de determinar si se satisfacen los objetivos de calidad definidos, se definieron varias preguntas que permiten responder si estos están satisfechos y un conjunto de métricas de calidad para poder responder las preguntas (ver Tabla 1). Estas métricas poseen a su vez un proceso de recolección asociado que indica cómo debe llevarse a cabo la medición. Dichos procesos pueden ser simples, como por

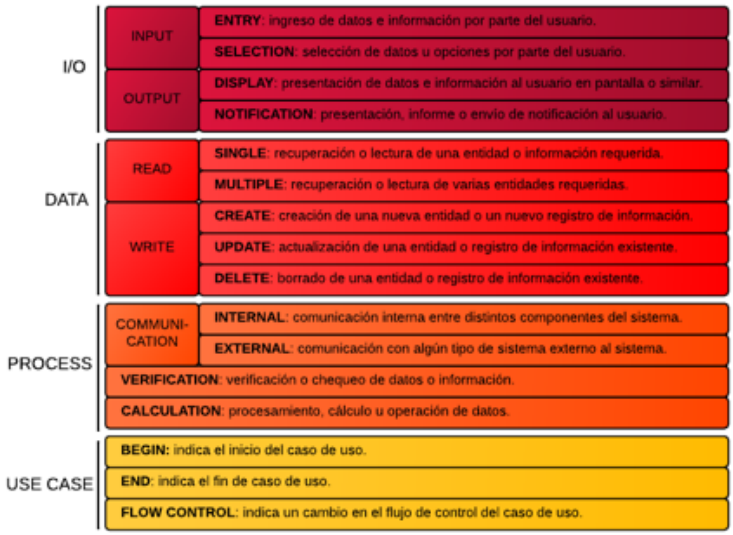
Tabla 1. Preguntas y Métricas asociadas a cada Objetivo de Calidad

Objetivo de Calidad	**Preguntas**	**Métricas**
Reuso / Modificabilidad	Q1: ¿Existen bloques de funcionalidad considerados duplicados? ¿Existen secciones de documentación que, aunque no en los mismos términos, definen el mismo comportamiento?	M.1.1: Bloques de funcionalidad duplicada
	Q2: ¿Se encuentran encapsulados o modularizados los requerimientos no funcionales?	M.2.1: Requerimientos no funcionales no modularizados
	Q3: ¿Se encuentran encapsulados o modularizados los requerimientos funcionales?	M.3.1: Requerimientos funcionales no modularizados
Entendibilidad / Mantenibilidad	Q4: ¿Existen elementos carentes de valor o significado dentro del modelo? ¿Se respeta la semántica de los casos de uso?	M.4.1: Actores mal definidos o sin sentido M.4.2: Casos de uso mal definidos o sin sentido M.4.3: Relaciones inadecuadas entre casos de uso
	Q5: ¿La especificación es simple y a la vez representativa?	M.5.1: Casos de uso demasiado extensos M.5.2: Casos de uso poco extensos M.5.3: Casos de uso "felices" (sin flujos alternativos)

ejemplo evaluar la longitud de los casos de uso, o más bien complejos, tales como determinar secciones que definan funcionalidad duplicada. El proceso de recolección más interesante de nuestro enfoque y sobre el cual haremos hincapié en este artículo es la detección de funcionalidad duplicada. Automatizar esta tarea no es sencillo debido a que las especificaciones están escritas en lenguaje natural (acarreando problemas tales como la sinonimia, ambigüedades, etc.). Para sortear estas dificultades, hemos combinado técnicas de *Aprendizaje de Maquina* (ML) [1,12] y algoritmos de *Alineamiento de Secuencias* (SA) [16].

La estrategia propuesta para comparar los eventos (que describen la interacción con el sistema) entre casos de uso consiste en realizar una abstracción que permita interpretar la semántica de los mismos. Esta abstracción elimina las particularidades de cada expresión y permite tratar los eventos según su intención y no por los términos que la definen. La Figura 4 presenta el conjunto de categorías definido (es decir, abstracciones) relacionadas al dominio de los casos de uso. Dichas categorías, denominadas "Acciones de Dominio" (DAs), forman parte de una jerarquía que tiene en cuenta la intención de un determinado evento a medida que se acerca a sus hojas. Las DAs son un refinamiento del trabajo presentado en [15]. Las especificaciones de los casos de uso son analizadas por un clasificador de DAs previamente entrenado, generando una discretización de los eventos según su intención o significado semántico.

Figura 4. Jerarquía de Acciones de Dominio para Casos de Uso



Luego de la clasificación, la búsqueda de funcionalidad duplicada se lleva a cabo comparando las especificaciones de casos de uso utilizando las DAs. Para ello, se transforman las especificaciones a secuencias de DAs y, bajo esta nueva representación, se utiliza un algoritmo de alineamiento de secuencias (JAligner[3]) que (tratando las especificaciones como cadenas de ADN) calcula la similitud entre diferentes bloques de funcionalidad de los casos de uso. La comparación realizada por este algoritmo encuentra la posición relativa entre dos secuencias que maximice su similitud (de comportamiento) en base a un sistema configurable de puntuación. Para más información, el lector puede consultar [16].

3.2. Determinar Posibles Refactorizaciones

Esta actividad tiene como objetivo determinar mecanismos que resuelvan defectos identificados previamente. Para ello, se analizan diferentes refactorizaciones para cada defecto encontrado. Una refactorización es una solución para un defecto particular que tiene sentido cuando se cumplen ciertas condiciones. El enfoque *ReUse* incluye un catálogo de refactorizaciones derivado de trabajos relacionados [14,18,19]. El mismo fue definido teniendo en cuenta la (semi-)automatización del proceso de mejora. Las refactorizaciones disponibles son listadas en la Tabla 2. Cada refactorización cuenta un contexto de aplicación (es decir, las condiciones), una descripción general de la solución, un mecanismo sistemático para ejecutar la refactorización, una prioridad, y un conjunto

[3] http://jaligner.sourceforge.net

Tabla 2. Refactorizaciones Incluidas en el Enfoque *ReUse*

Refactorización	Descripción	Prioridad
Generar Relación de Inclusión	Agrega una relación de inclusión entre casos de uso evitando la duplicación de funcionalidad	Alta
Generar Relación de Extensión	Agrega una relación de extensión entre casos de uso evitando la duplicación de funcionalidad	Alta
Generar Relación de Generalización	Agrega una relación de generalización entre casos de uso promoviendo la abstracción de funcionalidad común	Alta
Extraer Aspecto Temprano	Encapsula un conjunto de crosscutting concerns relacionados en un aspecto temprano	Media
Extraer Caso de Uso	Divide un caso de uso que describe varios requerimientos funcionales, disminuyendo su complejidad	Media
Unificar Casos de Uso	Unifica varios casos de uso que describen un mismo requerimiento funcional cuya complejidad individual es trivial, disminuyendo los costos de mantenibilidad del modelo	Media
Eliminar Relación de Inclusión / Extensión / Generalización	Remueve una relación que, debido a la evolución de la especificación, no tiene sentido en el modelo y no respeta la semántica de casos de uso (e.g. incluir un caso de uso que nadie más incluye)	Baja
Eliminar Caso de Uso / Actor	Permite eliminar un elemento que carece de sentido en el modelo de CU al encontrarse duplicado o al haber quedado obsoleto durante la evolución del sistema	Baja

de ejemplos que ilustran la refactorización. A modo de ejemplo se describe la refactorización Generar Relación de Inclusión en la Tabla 3.

Para determinar si una refactorización es aplicable, se analiza el modelo de casos de uso junto con las medidas recolectadas, evaluando si se cumplen las condiciones necesarias. Por ejemplo, consultando la medida relacionada a la duplicación de comportamiento es posible detectar una refactorización cuyo objetivo sea el reuso, como por ejemplo Generar Relación de Inclusión. Además, en dicho caso, se debe chequear que la duplicación ocurra en dos flujos básicos, lo cual es requisito en este tipo de relación. Teniendo en cuenta esta refactorización, la Figura 5 muestra un extracto de la especificación del sistema de compras en línea con una de las refactorizaciones aplicables. El conjunto de las refactorizaciones sugeridas y su impacto sobre los casos de uso puede ser visualizado de forma gráfica en la herramienta prototipo, como es mostrado en la Figura 2.

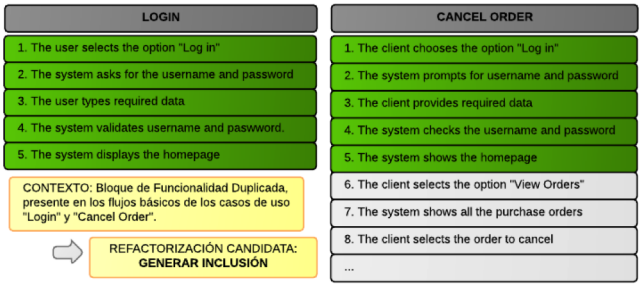
3.3. Seleccionar Refactorización más Apropiada

En esta actividad se le permite al analista seleccionar alguna de las refactorizaciones encontradas para su ejecución en la iteración actual. Dado que ciertos defectos son más importantes y acarrean mayores beneficios en la calidad de los

Tabla 3. Detalle de la Refactorización Generar Relación de Inclusión

Generar Relación de Inclusión	
Contexto	Existe una porción de funcionalidad duplicada en los flujos básicos de dos o más CU, la cual no se encuentra afectada por condición alguna. Puede ocurrir que uno de los CU contenga en su flujo básico únicamente esta porción de funcionalidad identificada como duplicada, constituyendo un caso especial más simple de esta refactorización.
Solución	Crear un nuevo CU que encapsule la porción de funcionalidad duplicada y generar una relación de inclusión desde los CU base al caso de uso a incluir, eliminando la funcionalidad duplicada de los CU base.
Mecanismo	1. Crear un nuevo CU en caso que la funcionalidad repetida no esté encapsulada en algún CU del modelo. Referenciarlo como el CU incluido. 2. Nombrar apropiadamente el CU. 3. Identificar la secuencia de acciones a ser removidas de cada CU base. 4. Mover la secuencia de eventos duplicada de los CU base al CU incluido. 5. Mover los actores que ya no formen parte de los CU base y sólo lo sean del CU incluido; y copiar los que participen en ambos. 6. Mover los flujos alternativos de los CU base que sean excepciones dentro de la secuencia de eventos movida al CU incluido. 7. Remover de los CU base la información asignada al CU incluido. 8. Generar relación de inclusión entre los CU base y el CU incluido. 9. Modificar la especificación de los CU base haciendo una referencia explícita al CU incluido.

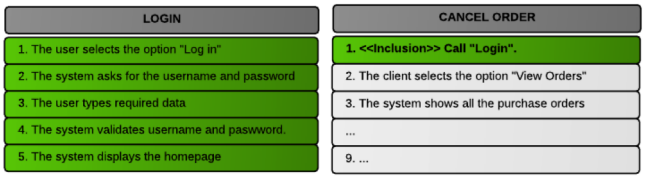
Figura 5. Determinación de Refactorizaciones en Casos de Uso



requerimientos, nuestro enfoque asiste a los analistas mediante la generación de un ordenamiento o "ranking" de las refactorizaciones sugeridas. La idea subyacente es que el analista resuelva el defecto de mayor importancia cuanto antes. La puntuación de cada refactorización se calcula teniendo en cuenta dos factores: la confianza de la detección del defecto y la prioridad del refactoring. El valor de confianza representa la certeza de que el defecto identificado es correcto, y está determinado por los resultados del procesamiento del texto y el análisis de los casos de uso. Por ejemplo, en el caso de la funcionalidad duplicada, la técnica de alineamiento de secuencias utilizada retorna información acerca de la certeza de las cadenas repetidas encontradas. La prioridad es un valor asociado a cada refactorización, directamente relacionado con la importancia del problema por el cual surge (ver Tabla 2). Adicionalmente, existe información específica sobre cada refactorización que permite ponderar la prioridad. Por ejemplo, en el caso de una refactorización de inclusión, se tiene en cuenta el porcentaje de eventos afectados por la duplicación. Utilizando estos parámetros es posible confeccionar una lista de refactorizaciones ordenada.

3.4. Aplicar Refactorización

Figura 6. Ejecución de una Refactorización



La actividad Aplicar Refactorización comprende el último paso de la organización iterativa de *ReUse* y tiene como objetivo aplicar la refactorización seleccionada sobre las especificaciones textuales. Para "ejecutar" la refactorización, se consulta el catálogo de refactorizaciones y se realiza la secuencia de pasos indicada en la sección de mecanismo correspondiente. Esto transforma la especificación defectuosa en una de mejor calidad. Considerando el ejemplo introducido anteriormente, la Figura 6 muestra el resultado de aplicar la refactorización Generar Relación de Inclusión al defecto de duplicación de funcionalidad. Dado que la funcionalidad duplicada ya se encuentra encapsulada en el caso de uso *Login*, no es necesario crear un nuevo caso de uso (ver Figura 5). Respecto del caso de uso base *Cancel Order*, se deben remover sólo los eventos duplicados (numerados del 1 al 5), ya que no hay actores ni flujos alternativos que deban ser eliminados. Una vez sustraídos estos eventos, se agrega una inclusión desde el caso de uso base *Cancel Order* hacia el caso de uso *Login*, incluyendo una referencia explícita del primero al segundo en la especificación.

4. Evaluación Preliminar

Con el propósito de validar nuestro enfoque, se llevaron a cabo experimentos con casos de estudio disponibles públicamente. El objetivo de esta evaluación preliminar fue corroborar la habilidad del enfoque para encontrar defectos (principalmente, de comportamiento duplicado) y sugerir refactorizaciones que permitan resolverlos. Los experimentos fueron llevados a cabo en cinco casos de estudio, a saber: DLibraCRM [4], MobileNews [4], WebJSARA [4], HWS [8], y CRS [2]. Los primeros tres sistemas están compuestos por un conjunto de especificaciones de casos de uso desarrolladas como parte del proyecto Software Development Studio (SDS). La documentación de DLibraCRM, MobileNews y WebJSARA consisten de 15, 15 y 29 casos de uso, respectivamente. Asimismo, la extensión de las especificaciones de dichos casos de estudio es de 13, 12 y 22 páginas de texto, respectivamente. El cuarto sistema es el Course Registration System (CRS) [2], un sistema distribuido que será utilizado para administrar los cursos y inscripciones de una universidad. La documentación de CRS consiste de 8 casos de uso (aprox. 20 páginas). El quinto y último sistema es el Health Watcher System (HWS) [8], un sistema web que sirve como intermediario entre los ciudadanos y el gobierno municipal para atender cuestiones relacionadas a la salud. Las especificaciones de HWS contienen 9 casos de uso (aprox. 19 páginas).

Para poder cuantificar la performance de nuestro enfoque, hemos decidido utilizar métricas derivadas del área de Recuperación de Información [12]. Evaluaciones similares han utilizado estas métricas anteriormente [4], observando los valores de *Precision*, *Recall* y *F-Measure* para arribar a conclusiones empíricas. En el contexto de nuestro trabajo, hay dos factores que deben ser considerados: la interpretación de las métricas en este tipo de experimentos, y la solución de referencia utilizada para comparar la salida del enfoque y así poder calcular las métricas. Por un lado, la *Precision* permite medir la tasa de sugerencias correctas (TP: verdaderos positivos) en contraste con las sugerencias incorrectas (FP: falsos positivos). Los falsos positivos pueden ocurrir tanto de la sugerencia de defecto que no lo es, como así también de la sugerencia de un refactoring incorrecto para un defecto correcto. Esta métrica es calculada como: $Precision = \frac{tp}{tp+fp}$. El *Recall*, por otro lado, permite calcular la tasa de sugerencias correctas (TP) en comparación al número real de defectos en los casos de estudio. Esto significa que para poder realizar el cálculo de *Recall*, es necesario poder reconocer los defectos del caso de estudio que el enfoque omitió (estos son conocidos como FN: falsos negativos). Esta métrica se calcula con la siguiente formula: $Recall = \frac{tp}{tp+fn}$.

A efectos de no sesgar la comparación, se encomendó el análisis de las especificaciones de los casos de estudios a cuatro analistas funcionales senior. Estos estuvieron a cargo de inspeccionar los documentos textuales de forma manual, identificando las porciones defectuosas de las especificaciones y determinando alternativas que permitirían mejorar dichos casos de uso. Los analistas tuvieron aproximadamente ocho horas para llevar a cabo esta tarea, y trabajaron individualmente. Posteriormente, se organizó una reunión entre los analistas, en la cual pudieron exponer sus descubrimientos y, luego de una discusión constructiva, pudieron arribar a un acuerdo mutuo acerca de los defectos encontrados en

los casos de estudio. El resultado de este ejercicio permitió elaborar una solución de referencia, la cual representa el razonamiento de los analistas y permite hacer comparaciones para calcular las métricas.

La evaluación intenta averiguar si el enfoque es capaz de descubrir los defectos ocultos en los casos de uso, y también qué tan bien funciona cuando se utiliza en especificaciones provenientes de sistemas reales. Los experimentos consistieron en ejecutar *ReUse* en los cinco casos de estudio, sin tener en cuenta la intervención de un analista humano. Los defectos encontrados y las refactorizaciones fueron aplicados de manera progresiva, seleccionando en cada iteración la sugerencia de mayor importancia (según el ranking elaborado por el enfoque). Los resultados de este experimento fueron comparados con las soluciones de referencia de cada caso de estudio, respectivamente. De esta manera, pudimos calcular las métricas de *Precision* y *Recall*. Debido a que los casos de estudio no presentaban errores simples, tales como casos de uso sin sentido o actores inútiles, la evaluación se enfocó principalmente en la funcionalidad duplicada. Es importante destacar que durante los experimentos, nos vimos forzados a realizar ciertas salvedades con respecto a las sugerencias de nuestro prototipo. Particularmente, el enfoque tuvo la habilidad de reconocer especificaciones extraordinariamente similares entre dos o más casos de uso, los cuales no necesariamente indicaban la presencia de un defecto. Por ejemplo, casos de uso relacionados con interacciones del sistema para realizar altas-bajas-modificaciones (CRUD) estaban escritas de manera muy parecida, y consecuentemente se sugerían como potenciales defectos. En dichos casos, preferimos omitir las sugerencias del enfoque por el bien del experimento.

Cuadro 4. Resultados de los Experimentos con *ReUse*

	DLibraCRM	MobileNews	WebJSARA	CRS	HWS	Total
TP	9	5	12	3	13	43
FP	6	1	10	1	7	25
FN	0	1	6	0	0	7
Precision	0.60	**0.83**	0.54	**0.75**	0.65	0.63
Recall	**1.00**	**0.83**	0.66	**1.00**	**1.00**	**0.86**

La Tabla 4 resume los resultados del experimento. El prototipo pudo recuperar la mayoría de los defectos como así también sugerir las refactorizaciones correspondientes (˜85 % *recall*), sin cometer un número significativo de errores en el proceso (˜65 % *precision*). En algunos de los casos de estudio, como son DLibraCRM, CRS, y HWS, el prototipo reconoció la totalidad de los defectos presentes en las especificaciones, logrando un *recall* del 100 %. Los defectos detectados estuvieron acompañados de sus respectivas refactorizaciones, los cuales se correspondieron principalmente a Generar Relacion de Inclusion, Generer Relacion de Extensión, y Unificar Casos de Uso. Los casos de estudio MobileNews y WebJSARA obtuvieron un *recall* de ˜80 % y ˜65 %, respectivamente. Particularmente, se pudo apreciar que el prototipo tuvo un nú-

mero considerable de FN en el caso de estudio WebSJARA. Esta situación se atribuye a una identificación deficiente del comportamiento duplicado entre casos de uso, el cuál acarreó la omisión de algunas refactorizaciones (principalmente, Generar Relación de Generalización y de Inclusión).

Con respecto a la métrica de *precision*, se lograron resultados aceptables. En MobileNews and WebJSARA, la herramienta obtuvo valores de *precision* superiores al 75 %. En los sistemas restantes se obtuvieron mediciones de *precision* sensiblemente inferiores, las cuales en promedio fueron del 60 %. Esta disminución de *precision* se atribuye a la detección defectuosa de funcionalidad duplicada, la cual acarreó la sugerencia incorrecta de refactorizaciones tales como Generar Relación de Inclusión y Generar Relación de Extensión. Sin embargo, los bloques recuperados por el prototipo compartían similitudes léxicas (es decir, términos) y semánticas (las acciones de dominio) significativas.

Los experimentos demostraron un gran potencial del prototipo. Para una herramienta de asistencia, creemos que es deseable tener una *precision* aceptable en favor de lograr un alto *recall*. Esta linea de razonamiento también se ha utilizado recientemente en otras evaluaciones de herramientas de RE automatizadas [6,7]. No obstante, es necesario llevar a cabo una evaluación más rigurosa para determinar la utilidad de la herramienta. Particularmente, sería interesante tratar de cuantificar la calidad de la identificación de los bloques de funcionalidad duplicada y los beneficios de la ponderación de las refactorizaciones sugeridas.

5. Conclusiones

En este artículo, presentamos un enfoque denominado *ReUse* que facilita las tareas del analista a la hora de determinar defectos en la especificación de requerimientos y mejorar la calidad de las mismas. El enfoque esta implementado en una herramienta prototipo que soporta el análisis de casos de usos combinando técnicas de procesamiento de texto avanzadas. Particularmente, *ReUse* aprovecha las bondades de un clasificador específico del dominio (de casos de uso) para obtener una representación abstracta de las especificaciones y, con éste conocimiento, adapta una técnica de alineamiento de secuencias para encontrar funcionalidad duplicada. Posteriormente, la funcionalidad redundante identificada es comparada con un catálogo de refactorizaciones, sugiriendo la mejor alternativa que permita solucionar dicho defecto de forma asistida.

Se realizaron experimentos con cinco casos de estudio que abarcan un amplio rango de dominios de software. Los resultados logrados fueron alentadores, obteniendo un muy buen *Recall* y una *Precision* aceptable. Obtener un alto *Recall* significa que el prototipo fue capaz de identificar la mayoría de los defectos de las especificaciones, lo cual es muy importante en el contexto de la problemática planteada. La *Precision* se mantuvo en rangos más bajos, lo que significa que la herramienta comete algunos errores a la hora de encontrar defectos y sugerir refactorizaciones. Aún así, somos optimistas respecto a que un analista podrá descartar las sugerencias incorrectas con un mínimo esfuerzo.

A pesar que los resultados son satisfactorios, durante la experimentación pudimos apreciar algunas limitaciones. Por ejemplo, las técnicas utilizadas fallaron al diferenciar aquellos comportamientos escritos de forma similar de los que realmente son similares (por ej., los casos de uso CRUD). Asimismo, los límites de los bloques de funcionalidad duplicada no siempre fueron identificados correctamente. Como trabajo futuro, estamos analizando alternativas para refinar la jerarquía de acciones de dominio para describir mejor la semántica de las especificaciones. También, pensamos ajustar los parámetros de la técnica de alineamiento de secuencias para mejorar la detección de bloques. Adicionalmente, es necesario llevar a cabo un mayor número de experimentos y contar con más casos de estudio para confirmar los resultados.

Referencias

1. Baeza-Yates, R., Ribeiro-Neto, B., et al.: Modern information retrieval, vol. 463. ACM press New York. (1999)
2. Bell, R.: Course registration system. `http://sce.uhcl.edu/helm/RUP_course_example/courseregistrationproject/indexcourse.htm` (2011)
3. Chernak, Y.: Building a foundation for structured requirements. aspect-oriented re explained - part 1. Better Software (January 2009)
4. Ciemniewska, A., Jurkiewicz, J.: Automatic Detection of Defects in Use Cases. Master's thesis, Poznan University of Technology (2007)
5. Cockburn, A.: Writing effective use cases, vol. 1. Addison-Wesley Reading (2001)
6. Cuddeback, D., et al.: Automated requirements traceability: The study of human analysts. In: 18th IEEE Int. Req. Eng. Conf. pp. 231 –240 (October 2010)
7. Dekhtyar, A., et al.: On human analyst performance in assisted req. tracing: Statistical analysis. In: 19th IEEE Int. Req. Eng. Conf. pp. 111 –120 (2011)
8. Greenwood, P.: Tao: A testbed for aspect oriented software development. `http://www.comp.lancs.ac.uk/~greenwop/tao/` (2011)
9. Hull, E., Jackson, K., Dick, J.: Requirements Engineering. Springer (2010)
10. Jacobson, I., et al.: The unified software development process. A-W (1999)
11. Kamata, M.I., Tamai, T.: How does req. quality relate to project success or failure? In: Procs. of the 15th IEEE Int. Req. Eng. Conf. pp. 69–78 (2007)
12. Manning, C., et al.: Introduction to Information Retrieval. CUP (2008)
13. Ramos, R., et al.: Quality improvement for use case model. In: SBES'09. pp. 187–195. IEEE (2009)
14. Rui, K., Butler, G.: Refactoring use case models: the metamodel. In: Procs. of the 26th Australasian Computer Science Conf. pp. 301–308 (2003)
15. Sinha, A., et al.: An analysis engine for dependable elicitation of natural language use case description and its application to industrial use cases. IBM Report (2008)
16. Smith, T., Waterman, M.: Identification of common molecular subsequences. Journal of Molecular Biology 147(1), 195 – 197 (1981)
17. Van Solingen, R., Berghout, E.: The Goal/Question/Metric Method: a practical guide for quality improvement of software development. McGraw-Hill (1999)
18. Xu, J., Yu, W., Rui, K., Butler, G.: Use case refactoring: a tool and a case study. In: 11th APSE Conf. pp. 484–491. IEEE (2004)
19. Yu, W., Li, J., Butler, G.: Refactoring use case models on episodes. In: Procs. of the 19th Int. Conf. on Aut. Soft. Eng. pp. 328–335. IEEE (2004)