Un Enfoque Inteligente para Derivar *Use Case Maps* a partir de Requerimientos de Software

Brian Caimmi, Guillermo Rodriguez, Álvaro Soria

ISISTAN-UNCPBA-CONICET, Facultad de Ciencias Exactas, Campus Universitario, Paraje Arroyo Seco, Tandil, Buenos Aires, Argentina
{brian.caimmi; guillermo.rodriguez; alvaro.soria}@isistan.unicen.edu.ar

Resumen. Los requerimientos de software descritos en lenguaje natural son la base para realizar la creativa pero exhaustiva tarea de diseñar una arquitectura de software. No obstante, suele existir una amplia brecha conceptual entre los requerimientos y la arquitectura de software que describe su funcionalidad; más aún, poco se ha explorado en este campo para reducir esta brecha. En este trabajo, proponemos un enfoque inteligente para derivar una descripción arquitectónica de un sistema, en particular utilizando la notación Use Case Maps (UCMs) a partir de requerimientos redactados en lenguaje natural con el objetivo de facilitar la transición entre la especificación de requerimientos y el diseño arquitectónico. El enfoque combina algoritmos de clasificación de texto, procesamiento de lenguaje natural y clusterización de texto para generar UCMs de la arquitectura de software prevista. La experimentación llevada a cabo con varios casos de estudio muestra resultados prometedores en cuanto al potencial del enfoque para asistir a los arquitectos de software durante las primeras etapas del proceso de desarrollo de software.

Palabras clave: Software Architecture, Use Case Maps, Machine Learning, Natural Language Processing

1. Introducción

Para diseñar una arquitectura de software, los arquitectos de software se basan en requerimientos de software que suelen ser especificados sin demasiado formalismo en lenguaje natural. Por el contrario, las arquitecturas son descritas formalmente ya que siguen estándares muy precisos y deben ser fácilmente comprendidas en fases de desarrollo posteriores. Además de diferencias en cuanto a la formalidad, los arquitectos de software deben lidiar con una gran cantidad de requerimientos para construir arquitecturas de software de calidad. Esto induce a que una de las transiciones más complejas y creativas del proceso de desarrollo de software ocurra entre el análisis de requerimientos de software y el diseño arquitectónico. Para facilitar esta transición, la notación Use Case Map (UCM)[1] es una herramienta capaz de brindar una visión general del comportamiento deseado de un sistema. La notación UCM se genera a partir de la deducción de responsabilidades de estos requerimientos que se asocian a componentes conceptuales. Asimismo, los UCMs describen trazas de ejecución a partir

de las relaciones causales que existen entre las responsabilidades. De esta forma, esta notación bastante intuitiva es muy útil para comunicar el comportamiento deseado de un sistema sin perderse en demasiados detalles.

En este trabajo se presenta un enfoque basado en técnicas de *Machine Learning* y de Procesamiento de Lenguaje Natural (NLP, *Natural Language Processing*) para analizar requerimientos de software textuales y derivar escenarios funcionales expresados en la notación UCM. En primer lugar, los requerimientos son clasificados en requerimientos funcionales y requerimientos no funcionales utilizando algoritmos de clasificación de texto como Naïve Bayes, k-Nearest-Neighbor (k-NN) y Support Vector Machines (SVM). Los requerimientos funcionales son posteriormente analizados, mientras que los no funcionales no son considerados en el análisis. En segundo lugar, el texto de cada requerimiento funcional es analizado para detectar acciones que el software deba realizar. Estas acciones representan las responsabilidades de algún componente en la arquitectura final. En tercer lugar, se detectan secuencias de responsabilidades que determinen posibles trazas de ejecución en la notación UCM. Finalmente, las responsabilidades son agrupadas semánticamente mediante técnicas de clusterización como K-Means, Partitioning Around Medoids (PAM), Hierarchical, Expectation-Maximization (EM), DBScan y X-Means. De esta forma, el enfoque brinda una posible descomposición funcional de un sistema en responsabilidades y trazas de ejecución para asistir a los arquitectos de software en el diseño arquitectónico. Una evaluación preliminar con varios proyectos de software demostró resultados prometedores en cuanto a la utilidad del enfoque para generar una primera descomposición funcional y escenarios comportamentales de una arquitectura de software.

El resto del trabajo se encuentra organizado de la siguiente forma. La Sección 2 discute los trabajos relacionados. La Sección 3 presenta el enfoque propuesto. La Sección 4 resume la evaluación empírica del enfoque. Finalmente, la Sección 5 presenta conclusiones del trabajo y trabajos futuros.

2. Trabajos Relacionados

En los últimos tiempos, varios autores han abordado el problema de clasificar automáticamente requerimientos de software en las categorías 'requerimiento funcional' y 'requerimiento no funcional'. Cleland Hund et al. [2] propuso un enfoque que clasifica requerimientos entre nueve diferentes categorías no funcionales al emplear un clasificador basado en técnicas de *Information Retrieval*. Posteriormente, en [3,4,5,6] se utilizaron técnicas de clasificación de texto basadas en *Machine Learning* para determinar si ciertos requerimientos de software eran funcionales o no funcionales. En estos últimos trabajos, los clasificadores predecían la categoría no funcional específica de cada requerimiento no funcional. En cambio, el enfoque que proponemos se centra en diferenciar entre requerimientos funcionales y no funcionales. Dado que nuestro enfoque sólo utiliza requerimientos funcionales durante la descomposición funcional del sistema, no es necesario clasificar los requerimientos no funcionales en categorías más específicas.

Por otro lado, existen pocos enfoques que asistan en la transformación de requerimientos funcionales en una primera abstracción funcional de una especi-

ficación arquitectónica. En [4] se presentó un enfoque que analiza requerimientos funcionales para identificar potenciales responsabilidades de software y componentes conceptuales de un sistema a ser desarrollado. En cuanto a la generación de diagramas dinámicos, [7,8,9] propusieron distintas herramientas que generan diagramas UML de secuencia al analizar especificaciones de caso de uso.

El objetivo de nuestra propuesta es derivar Use Case Maps (UCMs) a partir de requerimientos de software para obtener una mejor comprensión del sistema en su totalidad. El enfoque propuesto se diferencia de los trabajos referenciados en varias cuestiones. Primero, el enfoque que proponemos es uno de los primeros en analizar requerimientos para generar UCMs. Segundo, debido a que los diagramas de secuencia se centran en las operaciones mientras que los UCMs se centran en las responsabilidades[10], es más factible analizar requerimientos funcionales para derivar responsabilidades que derivar operaciones. Finalmente, varios estudios generan diagramas dinámicos como los diagramas UML de secuencia al analizar especificaciones de casos de uso. Los casos de uso presentan una estructura que describe el flujo (paso a paso) de un escenario, por lo que las relaciones causales entre pasos son suministradas dentro de la estructura. En cambio, nuestro enfoque evalúa características de la semántica de los requerimientos textuales para primero detectar responsabilidades y luego determinar relaciones causales entre estas responsabilidades.

3. Enfoque

La Figura 1 presenta un esquema de nuestro enfoque para derivar Use Case Maps (UCMs), como especificaciones arquitectónicas, a partir de requerimientos de software escritos en Inglés. El enfoque se compone de cinco pasos: clasificación de requerimientos, identificación de responsabilidades, secuencialización de responsabilidades, clusterización de responsabilidades y generación de UCMs.

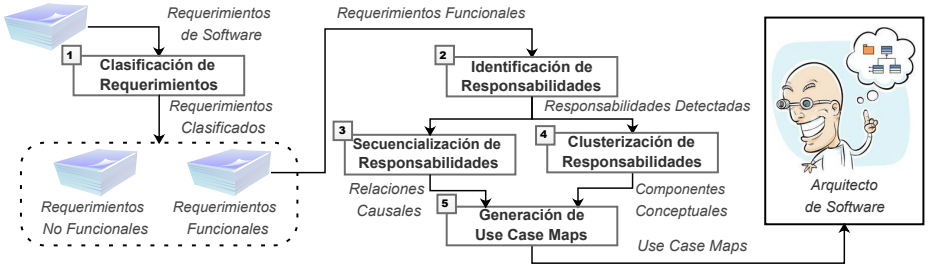


Figura 1: Diagrama del enfoque propuesto.

3.1. Clasificación de Requerimientos

El flujo del enfoque comienza con la etapa de clasificación de requerimientos de software en donde cada requerimiento se categoriza como 'funcional' o

'no funcional'. Sin embargo, previo a llevar a cabo la clasificación, los requerimientos deben ser representados en un formato comprensible por las técnicas de *Machine Learning*. Una representación de documentos de textos muy utilizada y efectiva es el Modelo de Espacio Vectorial (VSM, *Vector Space Model*)[11]. En este modelo cada requerimiento es identificado mediante un vector característico n-dimensional, donde cada dimensión se corresponde a un término distinto que se asocia a un valor numérico que indica su importancia.

Para representar los requerimientos de acuerdo al VSM, el enfoque realiza un pre-procesamiento sobre los requerimientos que involucra tareas de normalización, eliminación de stop-words y *stemming*. Primero, la normalización implica eliminar números y signos de puntuación; y luego, convertir el texto a minúscula. Segundo, las stop-words son palabras sin significado como artículos, pronombres y preposiciones, que tienen una alta probabilidad de ocurrencia en cualquier texto. Por lo tanto, al eliminar stop-words se conservan únicamente las palabras significativas relacionadas al tópico del texto[12], mejorando la predicción de las categorías de los requerimientos. En este trabajo, utilizamos una lista de 319 stop-words en Inglés[1]. Tercero, se aplica un algoritmo de *stemming*[13] para reducir las variantes morfológicas de las palabras restantes. Finalmente, utilizando la función $tf - idf$ se asignan los valores numéricos a los lemas resultantes y generar la representación en el VSM[11].

Contando con la representación de los requerimientos de software en el VSM, el siguiente paso consiste en identificar los requerimientos funcionales del conjunto de requerimientos de entrada. Para esto, el enfoque aplica técnicas supervisadas de *Machine Learning*. En particular, en este trabajo se optó por los clasificadores de texto Naïve Bayes, k-Nearest-Neighbor (k-NN), y Support Vector Machines (SVM)[11]. La razón de la selección de estos algoritmos se basa en que obtuvieron resultados alentadores en [3] y en [5]. Inicialmente, se utiliza un conjunto de requerimientos funcionales y no funcionales que se encuentren previamente etiquetados para aprender un modelo de clasificación de texto. Luego, el enfoque evalúa cada requerimiento representado en el VSM por medio de uno de estos clasificadores para determinar un conjunto de requerimientos funcionales y otro de requerimientos no funcionales. Al finalizar la etapa de clasificación de requerimientos, los requerimientos funcionales continúan en el flujo del enfoque mientras que los no funcionales son descartados.

3.2. Identificación de Responsabilidades

Una vez que el enfoque dispone de un conjunto de requerimientos funcionales, se procede a analizar el texto de estos requerimientos para identificar responsabilidades candidatas. El enfoque deriva estas responsabilidades al analizar las frases verbales y nominales contenidas en las oraciones. Para ello, el enfoque se basa en la combinación de *Part-of-Speech tagging* (POS)[14] y el analizador de dependencias de Stanford[15]. POS es un proceso que asigna una clase léxica o etiqueta (es decir, sustantivo, verbo, preposición, proposición, adjetivo) a cada

[1] http://snowball.tartarus.org/algorithms/english/stop.txt

palabra de una oración. Para el caso de un verbo, la etiqueta indica el tiempo verbal correspondiente. En este contexto, un algoritmo POS recibe el texto de cada requerimiento funcional y devuelve las etiquetas POS correspondientes para cada palabra. Paralelamente, el analizador de dependencias de Stanford detecta relaciones gramaticales entre las palabras de los requerimientos. Una relación gramatical es una 3-tupla que expresa un tipo de relación entre un término subordinante y uno subordinado. Utilizando las relaciones gramaticales detectadas, el enfoque es capaz de reconocer frases verbales (es decir, un verbo junto a un objeto directo), frases nominales (es decir, el sujeto de una oración), frases preposicionales, y proposicionales. Cada una de las frases verbales detectadas es seleccionada como una responsabilidad candidata. A continuación, el enfoque utiliza las etiquetas POS para descartar aquellas frases que no presenten verbos en un tiempo verbal adecuado. Es decir, las frases verbales que contengan un verbo en pasado simple no son consideradas como responsabilidades candidatas. Asimismo, las frases verbales que no tengan un objeto directo asociado también son descartadas. Al finalizar esta etapa, el enfoque dispone de un conjunto de responsabilidades candidatas que son la base para generar diagramas UCM.

3.3. Secuencialización de Responsabilidades

Teniendo el conjunto de responsabilidades candidatas, el enfoque analiza el contexto semántico de los requerimientos textuales para determinar relaciones causales entre las responsabilidades candidatas. Estas relaciones surgen del análisis de las sentencias condicionales y las preposiciones presentes en el texto de los requerimientos. Las sentencias condicionales describen situaciones hipotéticas y sus consecuencias. Una sentencia condicional posee una cláusula de condición y una cláusula de consecuencia. Para detectar estas cláusulas, en el lenguaje inglés existen varios conectores condicionales para introducir cláusulas condicionales (e.g.: 'If', 'Unless', 'Even if', 'Until') y cláusulas de consecuencia (e.g. 'Then'). En este contexto, el enfoque primero detecta sentencias condicionales con cualquier tipo de término introductorio y luego las re-escribe utilizando la estructura 'If <cláusula de condición> then <cláusula de consecuencia>', simplificando así el análisis. Posteriormente, el enfoque determina un conjunto de relaciones causales, siendo las relaciones que recaigan en la cláusula de condición las que precedan a las responsabilidades pertenecientes a la cláusula de consecuencia. Por otro lado, los requerimientos generalmente presentan preposiciones temporales que pueden indicar relaciones causales entre responsabilidades. El enfoque se limita a detectar a las preposiciones 'After' (es decir, 'Después') y 'Before' (es decir, 'Antes') en los requerimientos. Luego, el enfoque detecta las responsabilidades que se mencionen en la frase preposicional y en el resto de la oración para determinar relaciones causales. Si la preposición es 'After', las responsabilidades explícitas en la frase preposicional preceden a las responsabilidades del resto de la oración. Por el contrario, si la preposición es 'Before', las responsabilidades del resto de la oración anteceden a las responsabilidades mencionadas en la frase preposicional.

3.4. Clusterización de Responsabilidades

Paralelamente a la secuencialización de responsabilidades, la etapa de clusterización de responsabilidades define el conjunto de componentes conceptuales de un UCM mediante la utilización de un algoritmo de clusterización de texto y las responsabilidades candidatas. Para realizar esto, cada una de las responsabilidades detectadas es representada en el VSM. Luego, un algoritmo de clusterización toma por entrada al conjunto de representaciones de las responsabilidades en el VSM y lo trata de particionar en clusters disjuntos. En este trabajo se optó por los algoritmos de clusterización K-Means, Partitioning Around Medoids (PAM), Hierarchical, Expectation-Maximization (EM), DBScan[16] y X-Means[17]. Es importante notar que en la partición conseguida cada cluster representa un agrupamiento funcional del sistema en desarrollo. Sin embargo, asignarle un nombre a cualquiera de los componentes conceptuales resulta difícil. Por tal razón, el enfoque nombra a cada uno de los clusters con el sustantivo que aparezca un mayor número de veces en el subconjunto de responsabilidades.

3.5. Generación de Use Case Maps

En este punto, el enfoque combina las relaciones causales entre las responsabilidades y los componentes conceptuales para generar escenarios UCMs. Utilizando el conjunto de relaciones causales detectadas, el enfoque genera todas las posibles trazas de ejecución. Se define como traza de ejecución a una secuencia de responsabilidades relacionadas de forma causal, que comienza con un punto de inicio, continua con puntos intermedios y termina con un punto de fin. Los puntos de inicio serán las responsabilidades que no dependan de ninguna otra y que tengan alguna relación causal asociada. Los puntos intermedios serán las posibles secuencias de relaciones causales detectadas. Asimismo, los puntos de fin serán las responsabilidades que no precedan a otra responsabilidades y que tengan alguna relación causal asociada. Cada una de las trazas de ejecución detectadas representa un escenario funcional. Luego de generar todas las posibles trazas de ejecución, el enfoque asigna cada responsabilidad de estas trazas a un componente conceptual de acuerdo a la partición en clusters obtenida.

Finalmente, el enfoque le brinda los escenarios funcionales representados en UCM al arquitecto de software. De esta manera, el arquitecto dispondrá de una primera descomposición funcional del sistema en desarrollo. Además, el arquitecto podrá tener una vista general de los aspectos comportamentales que se encuentran descritos en los requerimientos funcionales del sistema.

4. Evaluación

Para evaluar al enfoque, experimentamos sobre cada etapa del enfoque: clasificación de requerimientos, identificación de responsabilidades, secuencialización de responsabilidades, y clusterización de responsabilidades. El primer experimento consistió en evaluar el proceso de clasificación de requerimientos. Para esto, entrenamos a los clasificadores de texto Naïve Bayes, k-NN y SVM en pos de comparar su habilidad al clasificar requerimientos de software en las categorías

'funcional' y 'no funcional'. El segundo experimento comprendió la evaluación de la habilidad del enfoque para identificar y secuencializar responsabilidades en los requerimientos funcionales. Los objetivos de este experimento son analizar si el enfoque es capaz de identificar un alta proporción de responsabilidades de los requerimientos funcionales y si el enfoque tiene la capacidad necesaria para detectar relaciones causales entre las responsabilidades reales de los proyectos. El tercer experimento comprendió la comparación del desempeño de los algoritmos de clusterización K-Means, PAM, Hierarchical, EM, DBScan y X-Means para agrupar responsabilidades similares en componentes conceptuales.

4.1. Configuración de la Evaluación

Para evaluar el enfoque utilizamos 17 proyectos de software propios de 2 repositorios. El repositorio PROMISE[18] ofrece 626 requerimientos de 15 proyectos de software. Los restantes 2 proyectos que brindan 16 requerimientos pertenecen a un trabajo practico de la carrera de Ingeniería de Software de la Universidad de Nueva York[2] (por simpleza a este repositorio lo denominamos NY). El primer experimento involucró a todos los requerimientos de los 17 proyectos. En cambio, en el segundo y tercer experimento usamos los proyectos 2, 8 y 9 de PROMISE junto a los dos proyectos de NY. Estos dos últimos proyectos permitieron contrastar el desempeño del enfoque al analizar diferentes tipos de requerimientos funcionales. Los requerimientos funcionales del repositorio PROMISE son generalmente de corta longitud mientras que los requerimientos funcionales de NY poseen un mayor grado de detalle y longitud. La Tabla 1 lista la distribución de requerimientos funcionales y no funcionales de los 17 proyectos de software.

Para medir los resultados de los dos primeros experimentos generamos matrices de confusión que representan los posibles resultados de la tarea de identificación al ser aplicada sobre información de prueba. Una matriz de confusión separa los siguientes cuatro valores: *True Positives (TP),True Negatives (TN)*, *False Positives (FP) y False Negatives (FN)*.

Utilizando estos valores es posible calcular las métricas *Accuracy*, *Precision*, *Recall* y *F-measure*. *Accuracy* (1) mide la exactitud general de las predicciones realizadas. *Precision (2)* mide la fracción de instancias recuperadas que son relevantes. *Recall (3)* mide la fracción de instancias relevantes que son recuperadas.

[2] http://www.nyu.edu/classes/jcf/CSCI-GA.2440-001/handouts/Assignment2.htm

Repositorio	PROMISE															NY	
Proyecto	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Req. Func.	21	11	48	25	37	27	15	20	17	38	0	0	0	0	0	6	8
Req. No Func.	8	29	32	30	36	47	8	73	7	15	13	22	19	16	12	0	2

Tabla 1: Distribución de requerimientos funcionales (Req. Func) y no funcionales (Req. No Func.) por proyecto de software.

F-measure (4) es el promedio armónico entre las métricas *Precision* y *Recall* que se utiliza para comparar el desempeño de diferentes métodos de recuperación.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F_{Measure} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{4}$$

Por otro lado, para la evaluación de la etapa de clusterización de responsabilidades se utilizaron las métricas de calidad más comunes en el análisis de soluciones de clusterización: *Entropy*[19], *Purity*[19], *F-Measure*[20] y *Adjusted Rand Index (ARI)*[21]. *Entropy* cuantifica la exactitud de clusters disjuntos o clusters en un nivel de una solución de clusterización jerárquica. Esta métrica evalúa cómo se distribuyen las distintas clases de muestras dentro de los clusters. *Purity* se centra en la frecuencia de la clase más común en cada cluster. Un alto valor de *Purity* sugiere que el cluster es un subconjunto puro de una clase predominante. *F-measure* considera a cada cluster como si fuera el resultado de una consulta y cada clase como si fuera el conjunto deseado de muestras para esa consulta. Por lo tanto, es posible calcular *Precision* y *Recall* de los clusters para cada clase dada como si fuera un problema de clasificación. Finalmente, *ARI* mide la concordancia entre dos particiones: una dada por el proceso de clusterización y otra definida por un criterio externo.

4.2. Clasificación de Requerimientos

En este experimento, evaluamos el proceso de clasificación de requerimientos del enfoque al comparar el uso de los clasificadores Naïve Bayes, k-NN y SVM. Con respecto a k-NN, utilizamos las variaciones de este clasificador al configurar el parámetro k con los valores 1, 2 y 3.

En el contexto de clasificación de requerimientos de software en las categorías 'funcional' y 'no funcional', las variables TP, TN, FP y FN toman valores específicos. TP es el número de requerimientos funcionales correctamente clasificados como 'funcional'; TN es el número de requerimientos no funcionales correctamente clasificados como 'no funcional'; FP es el número de requerimientos no funcionales incorrectamente clasificados como 'funcional'; y FN es el número de requerimientos funcionales incorrectamente clasificados como 'no funcional'.

Utilizamos la técnica de 10-fold Cross-Validation[22] con 10 iteraciones para evaluar de manera separada el desempeño de cada clasificador de texto. Por cada iteración, generamos una matriz de confusión para realizar un análisis más detallado que la proporción de predicciones correctas. Finalmente, promediamos los resultados de las iteraciones para producir una única estimación de las métricas.

La Tabla 2 presenta la comparación de los resultados de las métricas estadísticas obtenidas a partir de a técnica 10-fold Cross-Validation para los clasificadores de texto Naïve Bayes, k-NN (k = 1, 2 y 3) y SVM. Además, los valores más altos

Clasificador de Texto	*Accuracy*	*Precision*	*Recall*	*F-measure*
Naïve Bayes	0.8300	0.7827	0.8354	0.8056
k-NN (k = 1)	0.8443	0.8051	0.8435	0.8206
k-NN (k = 2)	0.7935	0.7046	**0.9033**	0.7887
k-NN (k = 3)	0.8238	0.8057	0.7745	0.7869
SVM	**0.9068**	**0.8891**	0.8925	**0.8896**

Tabla 2: Comparación de los valores de métricas logrados por los clasificadores Naïve Bayes, k-NN (k = 1, 2, y 3) y SVM, en la técnica 10-fold Cross-Validation.

de las métricas se encuentran resaltados. A simple vista, se puede ver que todos los clasificadores lograron buenos resultados para la métrica *Accuracy*, siendo el SVM el sobresaliente. El clasificador k-NN con un valor de k igual a 2 alcanzó el mayor valor de *Recall* pero el menor valor de *Precision* a la misma vez. Esto último significa que si el enfoque usara este clasificador de texto, el enfoque tenderá a mantener a casi todos los requerimientos funcionales para el procesamiento llevado a cabo en las etapas siguientes. Sin embargo, una gran cantidad de requerimientos clasificados como 'funcional' serán requerimientos no funcionales. Finalmente, el clasificador SVM obtuvo el mayor valor de *F-measure*; por lo que este clasificador puede ser considerado como la mejor alternativa para tener una proporción balanceada entre valores aceptables de *Precision* y *Recall*.

4.3. Identificación y Secuencialización de Responsabilidades

En el segundo experimento evaluamos el desempeño de las etapas de identificación de responsabilidades y secuencialización de responsabilidades sobre cinco proyectos de software. Para hacer esto, primero analizamos manualmente los requerimientos funcionales de los proyectos 2, 8, 9, 16 y 17 para determinar el número de responsabilidades reales (es decir, responsabilidades que el enfoque debería identificar), el número de frases verbales que son responsabilidades potenciales y las relaciones causales entre las responsabilidades. La Tabla 3 resume los resultados de este análisis manual sobre los cinco proyectos.

El proceso de identificación de responsabilidades tomó por entrada a los cinco proyectos de software y devolvió las responsabilidades detectadas en cada caso como salida. Luego, sabiendo de antemano el número de responsabilidades reales y de frases verbales, generamos las correspondientes matrices de confusión. En este contexto, TP es el número de responsabilidades reales correctamente identificadas; TN es el número de frases verbales que no representan responsabilidades y que fueron correctamente descartadas; FP es el número de frases verbales que no representan responsabilidades incorrectamente identificadas; y FN es el número de responsabilidades reales no identificadas. Por consiguiente, utilizamos estos valores para calcular las métricas *Accuracy*, *Precision, Recall* y *F-measure*.

Por otra parte, la etapa de secuencialización de responsabilidades analizó las responsabilidades reales de los mismos cinco proyectos de software retornando

Proyecto	Responsabilidades Reales	Frases Verbales	Relaciones Causales
2	15	23	2
8	26	36	11
9	18	32	7
16	12	24	7
17	19	28	7

Tabla 3: Resumen de las responsabilidades reales, las frases verbales (responsabilidades potenciales) y relaciones causales en los proyectos 2, 8, 9, 16 y 17.

las relaciones causales detectadas. Luego, conociendo las relaciones causales de referencia, generamos las matrices de confusión para cada proyecto redefiniendo TP, TN, FP y FN. En este caso, TP es el número de relaciones causales reales detectadas; TN es el número de relaciones causales indeseadas no identificadas; FP es el número de relaciones causales indeseadas incorrectamente identificadas; y FN es el número de relaciones causales reales no identificadas. Utilizando estos valores se calcularon las métricas *Precision, Recall* y *F-measure*.

La comparación del desempeño de la etapa de identificación de responsabilidades se muestra en la Tabla 4. Esta etapa obtuvo un alto desempeño al analizar el proyecto 2 ya que se detectaron la mayoría de las responsabilidades existentes. Estos buenos resultados se deben a que la mayoría de las frases verbales de los requerimientos funcionales eran responsabilidades reales. Por lo tanto, el enfoque tiene pocas probabilidades de equivocarse al identificar responsabilidades. Además, se lograron resultados aceptables al analizar los proyectos 8, 16 y 17. En estos tres proyectos, el enfoque logró tasas de *Recall* que variaron desde una base de 65.38 % hasta 83.33 %, lo que implica que en cuatro de los cincos proyecto se detectó al menos el 65.38 % de responsabilidades. En términos de *Precision*, al menos el 71.43 % de las responsabilidades fueron detectadas en estos tres proyectos. De lo anterior, pudimos establecer que el rango aceptable de valores de *F-measure* va desde 72.34 % a 96 %. Sin embargo, los peores resultados surgieron al analizar el proyecto 9. En este caso, todas las métricas tuvieron tasas cercanas

Proyecto	Identificación de Responsabilidades				Secuencialización de Responsabilidades		
	Accuracy	*Precision*	*Recall*	*F-measure*	*Precision*	*Recall*	*F-measure*
2	0.9565	1.0000	0.9375	0.9677	1.0000	1.0000	1.0000
8	0.6389	0.8095	0.6538	0.7234	0.8889	0.7273	0.8000
9	0.4688	0.5333	0.4444	0.4848	1.0000	0.4286	0.6000
16	0.7500	0.7143	0.8333	0.7692	1.0000	0.5714	0.7273
17	0.6786	0.7500	0.7895	0.7692	0.6250	0.7143	0.6667

Tabla 4: Valores de métricas logrados por las etapa de identificación y secuencialización de responsabilidades al analizar los proyectos 2, 8, 9, 16 y 17.

al 50 % debido al gran número de frases verbales existentes en comparación con el número de responsabilidades reales. El proyecto 9 presenta casi el doble de frases verbales que las responsabilidades existentes; en consecuencia, el enfoque detectó muchas responsabilidades falsas. Asimismo, el valor de *Recall* fue bajo en este proyecto ya que muchas de las responsabilidades reales no tenían asociado un objeto directo o sus verbos representaban acciones en tiempo pasado; por lo tanto, el enfoque descartó un número considerable de responsabilidades.

Por otro lado, la Tabla 4 también detalla los resultados de la etapa de secuencialización de responsabilidades. Los valores de *Precision* fueron 100 % en tres de los cinco proyectos de software; sin embargo, en los proyectos 8 y 17 fueron 62.50 % y 88.89 %, respectivamente. Esto se debe a que en estos dos últimos proyectos existen requerimientos que mezclan frases condicionales y preposicionales, de manera que se detectan relaciones causales no deseadas. En cuanto a *Recall*, esta etapa fue capaz de detectar la mayoría de relaciones causales en los proyectos 2, 8 y 17. No obstante, en los proyectos 9 y 16 sólo se logró detectar respectivamente el 42.86 % y el 57.14 % de relaciones existentes. En este sentido, en cuatro de los cinco proyectos se identificaron por lo menos el 57.14 % de relaciones causales reales. A partir de estos resultados, pudimos establecer que los valores aceptables de *F-measure* deberían ser superiores a 60 %.

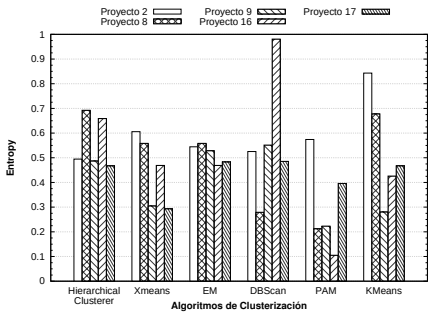
4.4. Agrupación de Responsabilidades

En el último experimento se comparó la habilidad de los algoritmos de clusterización K-Means, Hierarchical, PAM, X-Means, EM y DBScan para generar agrupaciones de las responsabilidades reales de los proyectos 2, 8, 9, 16 y 17 en componentes conceptuales. Los algoritmos X-Means, Expectation-Maximization (EM) y DBScan no dependen de un valor k para definir el número de clusters a generar; sin embargo, los algoritmos Hierarchical, PAM, y K-Means si requieren la especificación de este valor k de antemano. Para definir el valor k en cada caso, ejecutamos estos algoritmos variando el valor k desde 1 hasta el número de responsabilidades asociadas al proyecto de software correspondiente. Por cada ejecución, se calculó el coeficiente de silhouette. Un valor alto de este coeficiente indica una mejor calidad en la solución de clusterización[16]. En consecuencia, el coeficiente de silhouette máximo determina el valor k a utilizar.

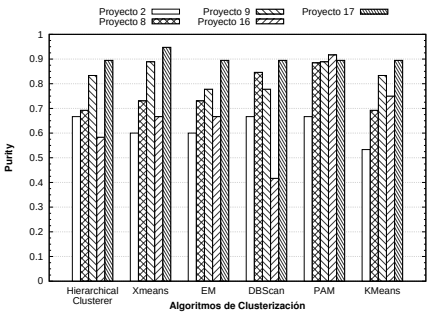
La Figura 2 muestra cuatros sub-figuras que detallan los resultados obtenidos de la descomposición funcional para las métricas *Entropy*(2a), *Purity*(2b), *F-measure(2c)* y *Adjusted Rand Index (ARI)*(2d). Analizando los valores de *Entropy* logrados en los cinco proyectos, se determinó que los algoritmos de clusterización PAM y X-Means lograron los mejores resultados. Mientras más bajos sean los valores de *Entropy* mejor es la solución, ya que en estos casos los clusters agrupan a un mayor número de responsabilidades que pertenecen al mismo componente conceptual. En cuanto a *Purity*, la mayoría de los algoritmos de clusterización obtuvieron valores similares para los cincos proyectos. Sin embargo, PAM se destacó al haber obtenido los mejores valores de *Purity* para cuatro de los cincos proyectos. Por otro lado, los algoritmos X-Means, Hierarchical y EM lograron valores de *F-measure* superiores a 60 % en los cincos proyectos.

En contraste, K-Means, DBScan y PAM no lograron tan buen promedio como el subconjunto anterior de algoritmos ya que alcanzaron valores atípicos en los proyectos 8 y 9. Finalmente, se utilizó la métrica *ARI* para evaluar el grado de similitud entre la descomposición funcional obtenida con la descomposición funcional de referencia de estos proyectos. Considerando que la métrica *ARI* varía en el rango [-1;1], se debe destacar que X-Means logró valores positivos superiores a 0.25 para esta métrica en los cinco proyectos. Esto indica que X-Means produjo las soluciones que más se asemejaron a las descomposiciones funcionales de referencias. Además, Hierarchical y EM lograron descomposiciones funcionales aceptables en la mayoría de los proyectos de software. No obstante, Hierarchical tuvo un mal desempeño en los proyectos 2 y 17, y EM en el proyecto 17. Asimismo, PAM obtuvo atípicamente la mejor descomposición funcional que se asemeja a la arquitectura de referencia para el proyecto 16. Teniendo en cuenta los resultados de las cuatro métricas, se confirmó experimentalmente que el algoritmo X-Means es el más indicado para llevar a cabo la descomposición funcional.

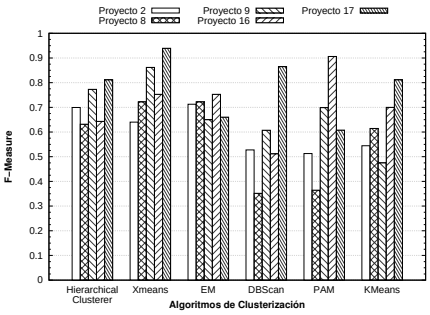
Finalmente, utilizando las responsabilidades detectadas, las relaciones causales generadas y los componentes conceptuales, el enfoque genera los escenarios



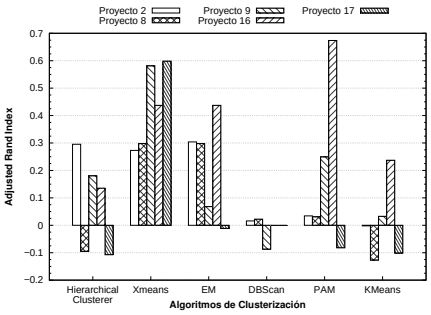
(a) *Entropy*



(b) *Purity*



(c) *F-measure*



(d) *Adjusted Rand Index (ARI)*

Figura 2: Resultados de la clusterización para la asignación de responsabilidades.

UCM correspondientes y se los presenta al arquitecto de software. Analizando los escenarios generados, el arquitecto dispone de una compresión general del comportamiento deseado para la arquitectura en desarrollo a la vez que puede continuar con el proceso de diseño al refinar estos escenarios.

5. Conclusiones y Trabajos Futuros

En este artículo hemos presentando un enfoque inteligente para derivar Use Case Maps (UCMs) a partir de requerimientos escritos en lenguaje natural. Utilizamos 17 proyectos de software para evaluar la etapa de clasificación mientras que un subconjunto seleccionado al azar de 5 proyectos fue considerado para la evaluación de las restantes etapas debido a limitaciones de tiempo. Respecto a la etapa de clasificación de requerimientos, se concluyó que el clasificador de texto SVM es el más adecuado para identificar requerimientos funcionales ya que logró valores altos y balanceados de *Accuracy*, *Precision*, *Recall* y *F-measure*. En cuanto a la etapa de identificación de responsabilidades, el enfoque identificó al menos el 65.38 % de responsabilidades explícitas en los requerimientos en cuatro de los cinco proyectos de software. Por otro lado, la etapa de secuencialización logró detectar al menos el 57.14 % de relaciones causales en cuatro de cincos proyectos de software. Por último, la evaluación de la etapa de clusterización sugirió que el algoritmo de clusterización X-Means es el más adecuado para generar descomposiciones funcionales de una arquitectura.

Con la evidencia de la viabilidad de este enfoque, se planea continuar trabajando en este campo. En primer lugar, se espera incorporar el *feedback* del arquitecto de software en cada etapa del proceso integrado. Otra línea de investigación que vale la pena seguir explorando es la incorporación de información referida a la arquitectura de software de referencia para mejorar la agrupación de responsabilidades. A su vez, la incorporación de requerimientos estructurados como casos de uso, y hasta *user stories*, permitirían enriquecer la asistencia al analizar la estructura de estos tipos de documentos. Finalmente, si bien los resultados de la evaluación de los 5 pasos del enfoque son significativos para una validación preliminar, estamos planeando repetir los experimentos con el conjunto de 17 proyectos para corroborar dichos resultados. Asimismo, tenemos planeado evaluar los 5 pasos del enfoque de manera conjunta.

Agradecimientos

Agradecemos al Ing. C. Lomagno por su invaluable aporte en el desarrollo e implementación de la herramienta que materializa al enfoque propuesto.

Referencias

1. Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., Little, R.: Documenting software architectures: views and beyond. Pearson Education (2002)
2. Cleland-Huang, J., Settimi, R., Zou, X., Solc, P.: Automated classification of non-functional requirements. Requirements Engineering **12**(2) (2007) 103–120

3. Casamayor, A., Godoy, D., Campo, M.: Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. Information and Software Technology **52**(4) (2010) 436–445
4. Casamayor, A., Godoy, D., Campo, M.: Functional grouping of natural language requirements for assistance in architectural software design. Knowledge-Based Systems **30** (2012) 78–86
5. Slankas, J., Williams, L.: Automated extraction of non-functional requirements in available documentation. In: Natural Language Analysis in Software Engineering (NaturaLiSE), 2013 1st International Workshop on, IEEE (2013) 9–16
6. Nguyen, T.H., Grundy, J., Almorsy, M.: Rule-based extraction of goal-use case models from text. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM (2015) 591–601
7. Thakur, J.S., Gupta, A.: Automatic generation of sequence diagram from use case specification. In: Pro. of the 7th India Software Engineering Conf., ACM (2014)
8. De Souza, F.C., de Castro Giorno, F.A.: Automatic generation of sequence diagrams and updating domain model from use cases. In: SOFTENG 2015 : The First Intl. Conf. on Advances and Trends in Software Engineering, Citeseer (2015)
9. Thakur, J.S., Gupta, A.: Anmodeler: a tool for generating domain models from textual specifications. In: Proc. of the 31st IEEE/ACM International Conf. on Automated Software Engineering, ACM (2016)
10. Reekie, J., McAdam, R., McAdam, R.: A software architecture primer. Software Architecture Primer (2006)
11. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to information retrieval (2008)
12. Schneider, K.M.: Techniques for improving the performance of naive bayes for text classification. In: Intl. Conf. on Intelligent Text Processing and Computational Linguistics, Springer (2005)
13. Porter, M.F.: An algorithm for suffix stripping. Program **14**(3) (1980) 130–137
14. Toutanova, K., Klein, D., Manning, C.D., Singer, Y.: Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proc. of the 2003 Conf. of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1, Association for Computational Linguistics (2003)
15. Chen, D., Manning, C.D.: A fast and accurate dependency parser using neural networks. In: EMNLP. (2014) 740–750
16. Han, J., Pei, J., Kamber, M.: Data mining: concepts and techniques. Elsevier (2011)
17. Pelleg, D., Moore, A.W., et al.: X-means: Extending k-means with efficient estimation of the number of clusters. In: ICML. Volume 1. (2000)
18. Menzies, T., Caglayan, B., Kocaguneli, E., Krall, J., Peters, F., Turhan, B.: The promise repository of empirical software engineering data. (2012)
19. Zhao, Y., Karypis, G.: Evaluation of hierarchical clustering algorithms for document datasets. In: Proceedings of the eleventh international conference on Information and knowledge management, ACM (2002) 515–524
20. Larsen, B., Aone, C.: Fast and effective text mining using linear-time document clustering. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM (1999) 16–22
21. Hubert, L., Arabie, P.: Comparing partitions. Journal of classification **2**(1) (1985)
22. Michie, D., Spiegelhalter, D.J., Taylor, C.C.: Machine learning, neural and statistical classification. (1994)