UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Revista Científica

http://revistas.udistrital.edu.co/ojs/index.php/revcie/index

CIDC

Artículo de investigación científica y tecnológica

Recuperación y clasificación de arquitecturas software en GitHub para reutilización, soportado por ontologías

Retrieval and Classification of Software Architectures on GitHub for Reuse, Supported by Ontologies

Recuperação e classificação de arquiteturas de software no GitHub para reutilização, suportado por ontologias

Hugo Ordóñez[1]
Camilo Ordóñez[2]
Víctor Buchelli[3]

Recibido: enero de 2021 **Aceptado:** abril de 2021

Para citar este artículo: Ordóñez, H., Ordóñez, C. y Buchelli, V. (2021). Recuperación y clasificación de arquitecturas software en GitHub para reutilización, soportado por ontologías. *Revista Científica*, *41*(2), 242-251. https://doi.org/10.14483/23448350.17644

Resumen

Para definir una arquitectura de un nuevo proyecto de software es clave reutilizar componentes existentes en proyectos previos. Estos proyectos previos pueden ser de la organización o estar disponibles en internet a través de GitHub. Los componentes reutilizados permiten tomar decisiones arquitecturales y así economizar tiempo y recursos. Sin embargo, actualmente buscar componentes de software en GitHub es ineficiente, pues las búsquedas son por cadenas de texto. En este trabajo se presenta OntoGitHubSearch como un modelo de búsqueda que permite la recuperación y clasificación de arquitecturas de software almacenadas en GitHub. El modelo recupera todos los repositorios que coincidan con los conceptos de arquitectura buscados; para darle semántica y contexto a la búsqueda, el modelo implementa la ontología de dominio específico Service-Oriented Architecture Ontology Version 2.0. Adicionalmente, se implementa procesamiento de lenguaje natural para analizar el texto del repositorio. Los repositorios son identificados y clasificados según el contexto que se expresa en el texto y en los conceptos encontrados en las descripciones del mismo. Para el proceso de evaluación del modelo propuesto se desarrolló una aplicación web denominada WebOntoGitHubSearch, la cual permite la interacción con usuarios en el proceso de consulta. La evaluación se realizó con la participación de desarrolladores y arquitectos de software de varias empresas colombianas, se utilizaron métricas de recuperación de información tales como Precision at k. Los resultados obtenidos en el proceso de evaluación son promisorios y permiten verificar la eficacia del modelo propuesto.

1. Ph. D. Universidad del Cauca. Popayán, Colombia. hugoordonez@unicauca.edu.co.
2. Fundación Universitaria de Popayán. Popayán, Colombia.
3. Ph. D. Universidad del Valle. Cali, Colombia.

Palabras clave: arquitectura de software; búsquedas; clasificación; GitHub; reúso.

Abstract
However, searching for software components on GitHub is currently inefficient, as searches are based on text strings. This paper introduces OntoGitHubSearch as a search model that allows the retrieval and classification of software architectures stored on GitHub. This proposed model retrieves all repositories that match the searched architectural concepts; to provide the search with semantics and context, it implements the domain-specific ontology Architecture Ontology Version 2.0, as well as a natural language processing module to analyze the text in the repository. The repositories are automatically identified and classified according to the text and concepts found in their descriptions. To evaluate the proposed model, we developed a web application called WebOntoGitHubSearch, which allows interaction with users during the search process. The evaluation of the model was carried out with the participation of software developers and architects from several colombian businesses. Information retrieval metrics such as Precision at k. The results obtained during the evaluation process are promising and allow verifying the effectiveness of the proposed model.
Keywords: classification; GitHub; reuse; searches; software architecture.

Resumo
Para definir uma arquitetura para um novo projeto de software, é fundamental reutilizar componentes existentes em projetos anteriores. Esses projetos anteriores podem ser da própria organização ou estar disponíveis na internet pelo GitHub. Os componentes reutilizados permitem que você tome decisões de arquitetura e, assim, economize tempo e recursos. No entanto, pesquisar atualmente por componentes de software no GitHub é ineficiente, pois as pesquisas são feitas por strings de texto. Neste trabalho, OntoGitHubSearch é apresentado como um modelo de busca que permite a recuperação e classificação de arquiteturas de software armazenadas no GitHub. O modelo recupera todos os repositórios que correspondem aos conceitos de arquitetura pesquisados, para fornecer semântica e contexto à pesquisa, o modelo implementa a ontologia específica de domínio da Ontologia de Arquitetura Orientada a Serviços Versão 2.0. Além disso, o processamento de linguagem natural é implementado para analisar o texto do repositório. Os repositórios são identificados e classificados de acordo com o contexto que está expresso no texto e nos conceitos encontrados nas suas descrições. Para o processo de avaliação do modelo proposto, foi desenvolvida uma aplicação web denominada WebOntoGitHubSearch, que permite a interação com os usuários no processo de consulta. A avaliação foi realizada com a participação de desenvolvedores de software e arquitetos de diversas empresas colombianas, métricas de recuperação de informação como Precisão em K. Os resultados obtidos no processo de avaliação são promissores e permitem verificar a eficácia do modelo proposto.
Palavras-chaves: arquitetura de software; classificação; GitHub; pesquisas; reutilização.

Introducción

Comenzar un proyecto de software hoy en día requiere de una estrategia cautelosa (Guillén, Crespo, Gómez y Sanz, 2016) que permita controlar y gestionar la arquitectura del proyecto. La estrategia debe contemplar varias opciones tales como: 1) tener un lugar (repositorio preferiblemente en la nube) para almacenar copias de respaldo que generalmente incluye control de versiones o seguimiento y gestión de revisiones (Borges y Valente, 2018); 2) registro de incidencias: cada proyecto debe tener un sistema de seguimiento de problemas al estilo de tickets, que permita a los desarrolladores saber los detalles de un problema que tenga el software o una explicación sobre una funcionalidad que deba ser implementada; 3) trabajo en equipo: los repositorios son un lugar perfecto para trabajar conjuntamente con colegas que pueden estar distribuidos a lo largo del mundo, además permiten la colaboración, haciendo así que todos los desarrolladores lean y escriban directamente sobre el repositorio (Contreras, 2018); 4)

versionamiento de código, es decir, poder guardar en un momento específico los cambios efectuados a un proyecto a través de un archivo o conjunto de archivos, con la posibilidad de tener acceso al historial de cambios, ya sea para consultar una versión anterior o para hacer comparaciones entre versiones (Jiang et al., 2021); 5) contribuir: es posible, a partir de cada copia de un proyecto, hacer ajustes que arreglen bugs o introduzcan una nueva funcionalidad y finalmente la contribución se fusiona con el código original (Hu, Wang, Ren y Choo, 2018).

Como punto clave para el surgimiento y el éxito de un proyecto software aparecen las plataformas colaborativas como GitHub (Zöller, Morgan y Schröder, 2020; H. Ordóñez et al., 2014), la cual, por su naturaleza de intercambio de información distribuida y colaborativa, permite la capacidad de interactuar y compartir información entre desarrolladores de software, con más de 28 millones de desarrolladores y más de 67 millones de repositorios creados colaborativamente. Debido a lo anterior, GitHub se ha convertido en el repositorio de proyectos software de mayor demanda en el mundo (Montandon, Valente y Silva, 2021).

Dentro de los aspectos relevantes de la arquitectura de un proyecto software, se encuentra la reutilización de componentes existentes de proyectos anteriores, los cuales se seleccionan de los repositorios existentes en GitHub (Coelho, Valente, Milen y Silva, 2020), con el fin de crear instancias que se conectan entre sí para definir o concretar los compontes a reutilizar. Los componentes reutilizados permiten tomar daciones arquitecturales por parte de desarrolladores o arquitectos en un proyecto nuevo (Golzadeh, Decan, Legay y Mens, 2020).

Con base en lo anterior, encontrar uno o varios repositorios en GitHub que contengan compontees software que puedan ser reutilizados se convierte en un fuerte desafío, el cual demanda gran cantidad de tiempo y esfuerzo, debido al alto volumen de repositorios existentes en GitHub. En este sentido, se hacen necesarios mecanismos que permitan encontrar o descubrir de forma ágil arquitecturas de software que se encuentran en los repositorios de GitHub. Para hacer frente a este desafío, GitHub ha desarrollado una aplicación web que se encuentra en su página principal. Aunque la aplicación permite hacer búsquedas por palabra clave, estas carecen de una contextualización debido a que se hacen de manera específica sobre cada uno los elementos (*positories*, *code*, *commits*, *issues*, *discussions*, *packages*, *marketplace*, *topics*, *wikis*, *users*) almacenados en los repositorios de GitHub, haciendo así que sus resultados sean mixtos, es decir, ordenados por medio de las votaciones que han tenido por parte de los desarrolladores registrados en GitHub, mas no según las necesidades del usuario que está realizando la búsqueda.

En este trabajo se propone a OntoGitHub-Search, un modelo de búsqueda que permite la recuperación y calificación de arquitecturas de software almacenadas en los repositorios de GitHub que se ha convertido en un importante corpus de conocimiento (De Graaf et al., 2014). El modelo recupera todos los repositorios que coincidan con los conceptos de búsqueda que el usuario utilice. Para darle semántica y contexto a la búsqueda, el modelo implementa una ontología de dominio específico denominada arquitectura de software orientada a servicios 2.0 (Service-Oriented Architecture Ontology Version 2.0) (De Graaf, Tang, Liang y van Vliet, 2012). La ontología proporciona un vocabulario común a las diferentes partes interesadas (arquitectos, desarrolladores, analistas), el cual describe los recursos de información (conceptos) con suficiente detalle, sin restricciones de rango y dominio (Rocha et al., 2018), además proporciona soporte de razonamiento a las búsquedas (Paredes-Valverde et al., 2018). Además, se implementa un mecanismo de procesamiento de lenguaje natural que analiza cada una de las descripciones de los componentes (*code*, *commits*, *description*, *issues*, *packages*, *topics*, *discussions*, entre otros) de los repositorios para identificar los repositorios a clasificar. Los repositorios

son descubiertos y clasificados según el contexto expresado en la consulta y los conceptos encontrados en las descripciones de sus componentes (Figueroa et al., 2016).

Los repositorios clasificados son desplegados al usuario final, con el propósito de que pueda revisar y reusar los componentes de arquitectura de software que se encuentren registrados en cada uno de los repositorios clasificados. El reúso se puede realizar en proyectos software nuevos o en proyectos en ejecución.

Para el proceso de evaluación del modelo propuesto se desarrolló una aplicación web denominada WebOntoGitHubSearch, la cual permite la interacción con usuarios en el proceso de consulta. La evaluación se realizó con la participación de desarrolladores y arquitectos de software de varias empresas colombianas; en este proceso se evaluó la relevancia en la lista de resultados, utilizando métricas clásicas de recuperación de información tales como Precision at k. Los resultados obtenidos en el proceso de evaluación son promisorios y permiten verificar la eficacia del modelo propuesto.

Metología

Un aspecto fundamental del que habla la ingeniería de software es la reutilización de la arquitectura de los proyectos; esto no solo consiste en volver a utilizar funcionalidades pre-desarrolladas y empaquetadas en clases, componentes o servicios (Thapar y Sarangal, 2020). También puede reutilizarse el conocimiento y la experiencia de otros desarrolladores. En este sentido, GitHub se ha convertido en un gran aliado de los desarrolladores y arquitectos, ya que alberga miles de proyectos software que son gratuitos y disponibles para reutilización en cualquier momento. Además, son de código abierto con acceso público ilimitado. GitHub alberga una cantidad impresionante de proyectos en alta calidad. Sin embargo, seleccionar un repositorio que contenga un proyecto o arquitectura reutilizable es una tarea desafiante.

Con base en lo anterior, a continuación se describen algunos de los trabajos que se han desarrollado para acceder a la información de GitHub.

Trabajos relacionados

GitHub se ha convertido en un foco de mucho interés para desarrolladores e investigadores, ya que permite reutilización de proyectos software. En este sentido, algunos autores, tales como Montandon et al. (2021), han aplicado técnicas de *machine learning* para identificar repositorios, para lo cual definen roles técnicos como los derivados de la experiencia en tecnologías de programación particulares (por ejemplo, lenguajes de programación) o componentes arquitectónicos (por ejemplo *frameworks frontend*), ya que representan tecnologías y técnicas que cada uno domina. Las técnicas utilizadas para clasificar los repositorios son Random Forest y Naive Bayes. De la misma forma, en Jiang et al. (2021) y Coelho et al. (2020) se aplican técnicas de *machine learning* basadas en redes neuronales para hacer recomendación de repositorios a partir de las etiquetas que contienen elementos del repositorio, tales como *code*, *title*, *commits*. Siguiendo en la misma línea, en Kallis, Di Sorbo, Canfora y Panichella (2020) se propone Ticket Tagger, una aplicación de GitHub que analiza el título y la descripción del repositorio mediante técnicas de *machine learning* para reconocer automáticamente los tipos de informes enviados en GitHub y asignar etiquetas a cada repositorio en consecuencia, aplicando algoritmos de clasificación de texto basados en clasificadores lineales, además de un enfoque de Deep Learning usando redes neuronales.

Por otra parte, algunos trabajos se centran generalmente en distinguir a los grupos de desarrolladores de entre ocho y veinte personas en redes sociales (Zöller et al., 2020), analizando la transferencia de habilidades y la delegación de tareas y deberes. Para cada repositorio de interés, se construye un grafo dirigido a partir de una red de interacciones entre desarrolladores exitosos usando una librería de Python. Para determinar qué tan

similares son dos redes, se realiza análisis de redes complejas utilizando la técnica NetSimile. En la misma línea, Hu et al. (2018) realizan análisis de los datos que representan las interacciones sociales de los usuarios de GitHub y proponen un enfoque basado en la actividad de Follow-Star-Fork para medir la influencia del usuario en la red social de desarrolladores de Github. Para esto reprocesan los datos de GitHub y construyen la red social. Luego, analizan la influencia del usuario en la red social, en términos de popularidad, centralidad, valor del contenido, contribución y actividad. Finalmente, analizan la correlación del usuario. Así mismo Hou, Yao y Gong (2021) combinan la información de la topología de la red social y la información de la interacción de los desarrolladores para calcular la intensidad de cooperación del desarrollador, con el fin de explorar en profundidad la relación entre los desarrolladores desde las propiedades topológicas y semánticas. Para esto, se propone un algoritmo de detección de comunidad ABDCI basado en la intensidad de cooperación de los desarrolladores. Finalmente, este método se aplica a muchos tipos diferentes de redes de desarrolladores en el ecosistema de software a través de GitHub.

A partir del análisis anterior, es posible evidenciar que los trabajos citados se centran en aplicación de técnicas de *machine learning* o identificación de grupos en redes sociales para clasificar los repositorios de GitHub, utilizando datos de las descripciones identificando relaciones entre los repositorios a clasificar; aunque tienen buenos resultados dejan de lado los intereses del usuario, además no utilizan una base de conocimiento específico en arquitectura de software como una ontología, la cual permite encontrar con mayor nivel de similitud los repositorios que sean de verdadera utilidad al usuario dentro de un dominio de arquitectura de software.

Modelo propuesto

El modelo propuesto se compone de dos capas: 1) capa de búsqueda, encargada de capturar los conceptos que el usuario seleccione de la ontología, los cuales pueden ser complementados con palabras clave para aumentar la expresividad de la consulta; una vez se define la consulta por parte del usuario, esta es enviada a GitHub utilizando el api que provee para identificar los repositorios que cumplen con la consulta; 2) capa de resultados, se encarga de procesar las descripciones de cada uno de los compontes de un repositorio aplicando un mecanismo basado en lenguaje natural, posteriormente los repositorios son clasificados y ordenados para ser desplegados al usuario. Para validar el modelo propuesto se implementó una aplicación web denominada WebOntoGitHubSearch; en la Figura 1 se muestran los componentes de la aplicación web desarrollada. Estos componentes se explican a continuación.

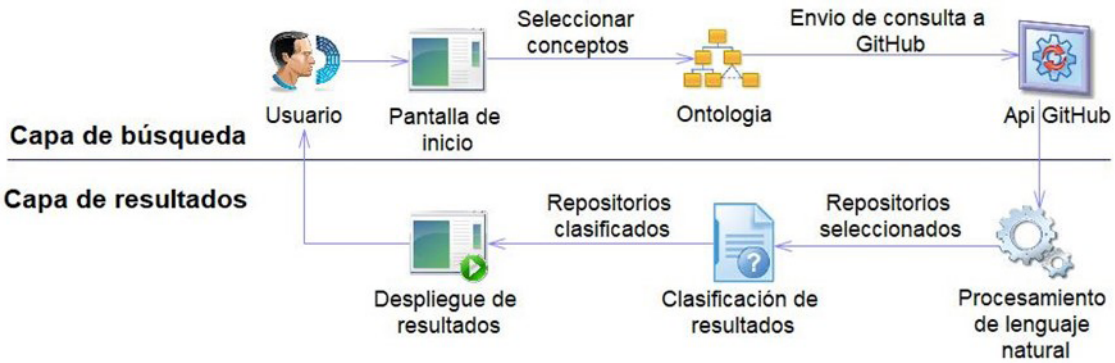

Figura 1. Componentes de la aplicación web desarrollada.
Fuente: elaboración propia.

- **Pantalla de inicio**: muestra la interfaz web de la aplicación. Esta interfaz cuenta con una caja de texto para capturar los conceptos que el usuario seleccione de la ontología; tiene la opción de ampliar la consulta con las palabras clave que él crea necesarias para formar la consulta, además se encuentra un botón que despliega los resultados de la búsqueda. WebOntoGitHubSearch cuenta con una interfaz web sencilla y usable.
- **Ontología**: Service-Oriented Architecture Ontology Version 2.0 fue diseñada por el Grupo SOA de Open Group, que participa en un programa de trabajo para producir definiciones, análisis, recomendaciones, modelos de referencia y estándares para ayudar a los profesionales de negocios y tecnologías de la información a comprender y adoptar SOA.

La ontología contiene clases y propiedades correspondientes a los conceptos de SOA. Las definiciones formales de OWL se complementan con descripciones en lenguaje natural de los conceptos y las relaciones de uso entre ellos. Las definiciones de lenguaje natural y OWL que contiene esta especificación constituyen la definición normalizada de la ontología. La Figura 2 muestra la representación de las ontologías en Protege.

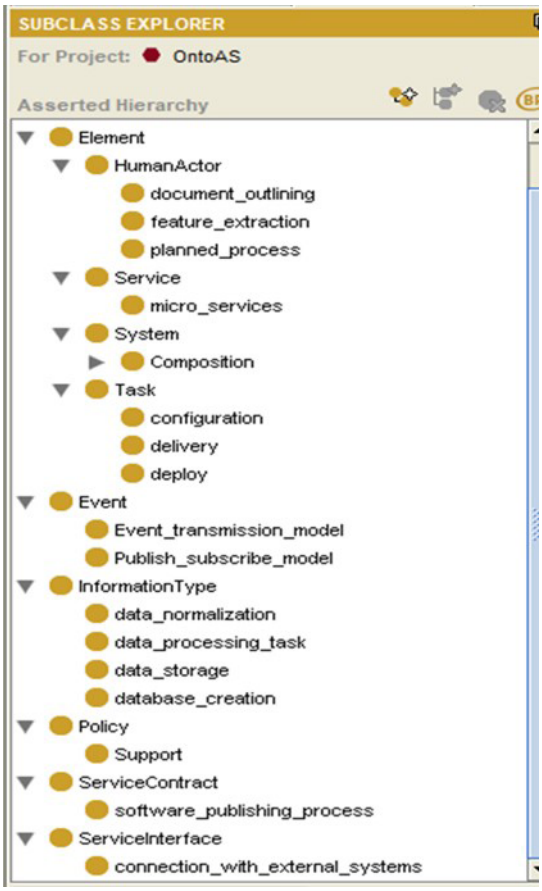

Figura 2. Ontología (Service-Oriented Architecture Ontology Version 2.0).
Fuente: elaboración propia.

- **Api GitHub**: este componente utiliza el api *rest* de GitHub para extraer los repositorios que tengan alguna similitud con los conceptos de la consulta. De esta se extraen las descripciones de los siguientes componentes: *users*, *repositories*, *code*, *commits*, *issues* y *pull requests*, *title*, tags y *topics*. Con estas descripciones se crea una matriz de conceptos (m*n) denominada MC en la que cada fila representa un repositorio m y cada columna a un componente n.
- **Procesamiento de leguaje natural**: en este componente se escriben reglas (por ejemplo, doneby, GeneratedBy ↔ Generates, IsInputAt↔HasInput, involvesParty ↔ isPartyTo, performedBy ↔ performs, entre otras) de reconocimiento de patrones estructurales en el texto de las descripciones recuperadas de GitHub, empleando el formalismo gramatical definido en la ontología. A partir de estas reglas, se calculan las frecuencias de aparición de diferentes unidades lingüísticas (por ejemplo, palabras, frases) en texto de las descripciones. Con las frases de mayor frecuencia se crea una nueva matriz denominada matriz de frases *MF*.
- **Clasificación de resultados**: en este componente se crean índices de clasificación en memoria. En estas estructuras se indexan las descripciones tomadas de la matriz *MF*. En los índices las descripciones son ponderadas a través del modelo vectorial propuesto por

Manning, Raghavan y Schutze (2008), utilizando la ecuación 1, en la que $F_{i,j}$ es la frecuencia observada del concepto *j* en el repositorio *i*, $Max(F_j)$ es la mayor frecuencia observada en el repositorio *i*, *N* es el número de repositorios en el índice y n_j es el número de repositorios en los que aparece el concepto *j*.

$$w_{i,j} = \frac{f_{i,j}}{\max(f_i)} * log\left(\frac{N}{n_j+1}\right) \quad \text{Ecuación 1}$$

Una vez es creado el índice en memoria, los repositorios son clasificados y ordenados aplicando la distancia de cosenos (comúnmente usada en el modelo vectorial de recuperación de información), utilizando la ecuación 2; los resultados que superan un umbral de similitud (70%) entre las frases almacenadas en el índice y la consulta son recuperados para ser desplegados al usuario.

$$Cos(r,q) = \frac{\sum_{i=1}^{n}(r_i * q_i)}{\sqrt{\sum_{i=1}^{n} d_i^2} * \sum_{i=1}^{n} q_i^2} \quad \text{Ecuación 2}$$

- **Despliegue de resultados**: se listan los repositorios que contiene las descripciones de arquitectura de software para que el usuario los visualice, después de que han pasado por el proceso de filtrado y se han escogido los que más se relacionan con las necesidades de búsqueda del usuario. Los resultados se listan en orden de acuerdo a la similitud que presentan con respecto a la consulta realizada.

Resultados

El proceso de evaluación del modelo propuesto se llevó a cabo de dos maneras: 1) evaluación con arquitectos y desarrolladores que trabajan en empresas de desarrollo nacionales e internacionales; 2) evaluación con estudiantes de últimos semestres que asisten a la materia de arquitectura de software. Estas estuvieron enfocadas en medir la precisión, con el propósito de verificar la satisfacción del usuario en cuanto a la relevancia de los resultados reportados por el sistema.

Para la evaluación de la relevancia de los resultados, se utilizó una métrica que permite la evaluación de colecciones no controladas de ítems de información (como lo es este caso internet) que es comúnmente usada para evaluar sistemas que presentan resultados en forma ordenada (ranking). Esta métrica es la Precisión en k (Precision at k) de resultados ordenados. Esta métrica tiene la ventaja de no requerir el tamaño estimado del conjunto de ítems de información relevantes (Ordóñez, Corrales y Cobos, 2014). Esta métrica fue aplicada teniendo como fundamento que a los usuarios lo que más les importa es qué tan buenos son los resultados en las primeras páginas (ya que normalmente ellos no revisan toda la lista de resultados). Esto motiva a que la evaluación involucre una medición de los valores de precisión en ciertos puntos de la lista de los resultados recuperados, como por ejemplo los primeros 10 o 20 documentos.

- **Evaluación con arquitectos y desarrolladores**. Para este proceso se contó con la participación de tres arquitectos, de los cuales uno pertenece a una empresa de desarrollo de software multinacional y dos son parte de una empresa nacional. Además, se contó con quince desarrolladores, de los cuales cinco pertenecen a la empresa multinacional y diez a la nacional. El proceso de evaluación incluyó 20 búsquedas de arquitecturas de software con los conceptos tomados de la ontología, en las que cada uno de los usuarios seleccionó al azar un concepto para ejecutar cada consulta, en total se realizaron 360 consultas y se revisaron 3600 repositorios. En la Figura 3 se presenta la precisión promedio en diferentes valores de k, con resultados comprendidos entre 95% y 93% para los tres primeros ítems, además de 81% y 73% para los ítems del 8 al 10; esto demuestra que el modelo

propuesto tiene un alto grado de precisión y relevancia para las consultas realizadas por parte de los arquitectos y desarrolladores. Adicionalmente, la gráfica permite observar que para la gran mayoría de los arquitectos y desarrolladores los repositorios ranqueados en las primeras posiciones de la lista de resultados contienen arquitecturas de software que pueden ser reutilizadas por ellos en proyectos que lo requieran.

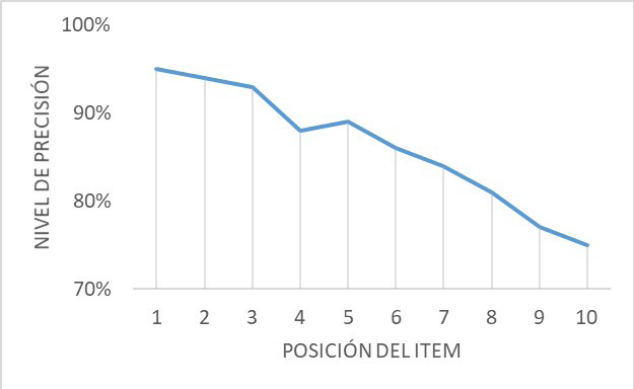

Figura 3. Precisión en k, arquitectos y desarrolladores.
Fuente: elaboración propia.

- **Evaluación con estudiantes de últimos semestres**. En este proceso se contó con 60 estudiantes de diferentes universidades (3) quienes se encuentran cursando la materia de arquitectura de software. Debido a la poca experiencia de los estudiantes, para realizar este proceso se tomaron diez conceptos de la ontología tales como: *planned process docker*, *software publishing process*, *connection with external systems*, *publish subscribe model*, *data processing task* Python, *micro services* Java, entre otros. Algunos de los conceptos fueron complementados con palabras clave tales como: Java, Python, Docker, entre otras, para darle oportunidad al estudiante de entender de mejor manera la consulta. La Figura 4 muestra que los resultados, que son muy relevantes para los estudiantes, en estos se tiene que los tres primeros resultados de la lista contienen entre 95% y 85% de relevancia; de la misma forma, los últimos resultados se encuentran entre 80% y 75% de relevancia. Esto demuestra que para los estudiantes los resultados reportados tienen alto grado de relevancia, es decir, los repositorios recuperados de alguna manera contienen elementos de arquitectura de software que pueden ser fácilmente reutilizables en algún proyecto que estén desarrollando, por ejemplo, un proyecto de curso o proyecto de aula.

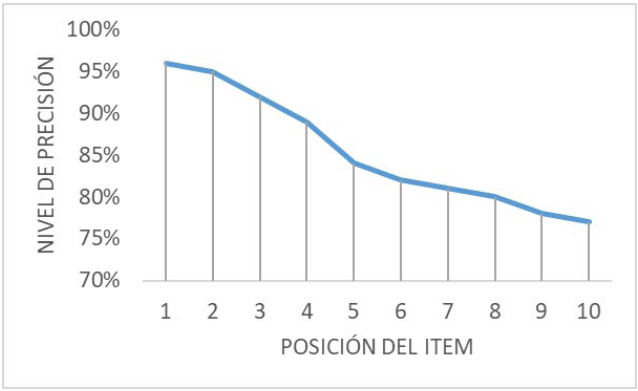

Figura 4. Precisión en k, Estudiantes.
Fuente: elaboración propia.

Conclusiones

En este trabajo se definió un modelo que permite recuperar arquitecturas de software que se encuentran almacenadas y compartidas en GitHub. El modelo a través de sus componentes permite clasificar las arquitecturas de software por medio de las descripciones que estas tengan en cada uno de sus elementos.

GitHub es un lugar donde existe gran cantidad de información, y buscar información en este mega repositorio es un reto para los desarrolladores de software. A partir de la introducción y utilización de las ontologías en los procesos que conforman la fase de clasificación y organización de la información, los resultados de las búsquedas serán mejores y el usuario podrá obtener de una manera más fácil y automática lo que necesita.

La ontología utilizada en el modelo de búsqueda permite dar mayor expresividad y

conceptualización a la consulta realizada por un usuario, ya que contiene conceptos específicos de arquitectura de software. Por otra parte, los conceptos de la ontología utilizados en las búsquedas aumentan el nivel de precisión de los resultados desplegados al usuario.

El proceso de evaluación se desarrolló en dos partes con usuarios específicos. Para los usuarios arquitectos y desarrolladores de software de empresas se pudo evidenciar que los primeros resultados entregados por el modelo tienen alto grado de relevancia: entre el 95% y 93%, demostrando así que los repositorios recuperados de alguna forma contienen componentes arquitecturales de software que pueden ser reutilizados por ellos en algún momento o proyecto que lo requiera. En la evaluación con estudiantes de últimos semestres se utilizaron palabras clave como complemento a los conceptos de la ontología.

Los resultados obtenidos en la primera procesión se encuentran entre 95% y 85%. Aunque este nivel de precisión es bueno, esto permite evidenciar que debido a la poca experiencia de los estudiantes el nivel de precisión tiende a bajar de 93% a 85%. Como trabajo a futuro se tiene el propósito de adicionar al modelo una ontología en ciencias de la computación, lo cual permitirá ampliar el dominio de la información que se desee recuperar de GitHub, además de evaluar la aplicación web en relación con la usabilidad y la satisfacción de usuario.

Agradecimientos

El profesor Hugo Ordóñez agradece por el apoyo y el tiempo asignado para realizar la investigación a la Universidad del Cauca, en la cual labora como profesor asociado; de la misma forma el profesor Camilo Ordóñez agradece a la Fundación Universitaria de Popayán. Finalmente, el profesor Víctor Buchelli agradece a la Facultad de Ingeniería de la Universidad del Valle por el apoyo y el tiempo asignado para realizar la investigación.

Referencias

Borges, H., Valente, M. T. (2018). What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. *Journal of Systems and Software*, *146*, 112-129. https://doi.org/10.1016/j.jss.2018.09.016

Coelho, J., Valente, M. T., Milen, L., Silva, L. L. (2020). Is This GitHub Project Maintained? Measuring the Level of Maintenance Activity of Open-Source Projects. *Information and Software Technology*, *122*, e106274. https://doi.org/10.1016/j.infsof.2020.106274

Figueroa, C., Ordóñez, H., Corrales, J. C., Cobos, C., Krug Wives, L., Herrera-Viedma, E. (2016). Improving Business Process Retrieval Using Categorization and Multimodal Search. *Knowledge-Based Systems*, *110*(15), 49-59. https://doi.org/10.1016/j.knosys.2016.07.014

Contreras, A. L. (2018). Gestión de la motivación en escenarios organizacionales. *Investigación e Innovación en Ingenierías*, *6*(1), 84-92. https://doi.org/10.17081/invinno.6.1.2777

Golzadeh, M., Decan, A., Legay, D., Mens, T. (2021). A Ground-Truth Dataset and Classification Model for Detecting Bots in GitHub Issue and PR Comments. *Journal of Systems and Software*, *175*, e110911. https://doi.org/10.1016/j.jss.2021.110911

De Graaf, K. A., Liang, P., Tang, A., van Hage, W. R., van Vliet, H. (2014). An Exploratory Study on Ontology Engineering for Software Architecture Documentation. *Computers in Industry*, *65*(7), 1053-1064. https://doi.org/10.1016/j.compind.2014.04.006

De Graaf, K. A., Tang, A., Liang, P., van Vliet, H. (2012). Ontology-Based Software Architecture Documentation. En *Joint Working Conference on Software Architecture and European Conference on Software Architecture* (pp. 121-130). https://doi.org/10.1109/WICSA-ECSA.212.20

Guillén, A. J., Crespo, A., Gómez, J. M., Sanz, M. D. (2016). A Framework for Effective Management of Condition Based Maintenance Programs in the Context of Industrial Development of E-Maintenance

Strategies. *Computers in Industry*, *82*, 170-185. https://doi.org/10.1016/j.compind.2016.07.003

Hou, T., Yao, X., Gong, D. (2021). Community Detection in Software Ecosystem by Comprehensively Evaluating Developer Cooperation Intensity. *Information and Software Technology*, *130*, e106451. https://doi.org/10.1016/j.infsof.2020.106451

Hu, Y., Wang, S., Ren, Y., Choo, K, R. (2018). User Influence Analysis for Github Developer Social Networks. *Expert Systems with Applications*, *108*, 108-118. https://doi.org/10.1016/j.eswa.2018.05.002

Jiang, J., Wu, Q., Cao, J., Xia, X., Zhang, L. (2021). Recommending Tags for Pull Requests in GitHub. *Information and Software Technology*, *129*, e106349. https://doi.org/10.1016/j.infsof.2020.106394

Kallis, R., Di Sorbo, A., Canfora, G., Panichella, S. (2020). Predicting Issue Types on GitHub. *Science of Computer Programming*, *205*(1), e102598. https://doi.org/10.1016/j.scico.2020.102598

Manning, C. D., Raghavan, P., Schutze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Montandon, J. E., Valente, M. T., Silva, L. L. (2021). Mining the Technical Roles of GitHub Users. *Information and Software Technology*, *131*, e106485. https://doi.org/10.1016/j.infsof.2020.106485

Ordóñez, H., Corrales, J., Cobos, C., Wives, L., Thom, L. (2014). Collaborative Evaluation to Build Closed Repositories on Business Process Models. En *Proceedings of the 16th International Conference on Enterprise Information Systems* (pp. 311-318). https://doi.org/10.5220/0004881203110318

Ordóñez, Hugo, Corrales, J. C., Cobos, C. (2014). MultiSearchBP: entorno para la búsqueda y agrupación de modelos de procesos de negocio. *Polibits*, *49*, 29-38. https://doi.org/10.17562/pb-49-3

Paredes-Valverde, M. A., Salas-Zárate, M. P., Colomo-Palacios, R., Gómez-Berbís, J. M., Valencia-García, R. (2018). An Ontology-Based Approach with Which to Assign Human Resources to Software Projects. *Science of Computer Programming*, *156*(1), 90-103. https://doi.org/10.1016/j.scico.2018.01.003

Rocha, R., Araújo, A., Cordeiro, D., Ximenes, A., Teixeira, J., Silva, G., da Silva, D., Espinhara, D., Fernandes, R., Ambrosio, J., Duarte, M., Azevedo, R. (2018). DKDOnto: An Ontology to Support Software Development with Distributed Teams. *Procedia Computer Science*, *126*, 373-382. https://doi.org/10.1016/j.procs.2018.07.271

Thapar, S. S., Sarangal, H. (2020). Quantifying Reusability of Software Components Using Hybrid Fuzzy Analytical Hierarchy Process (FAHP)-Metrics Approach. *Applied Soft Computing*, *88*, e105997. https://doi.org/10.1016/j.asoc.2019.105997

Zöller, N., Morgan, J. H., Schröder, T. (2020). A Topology of Groups: What GitHub Can Tell Us about Online Collaboration. *Technological Forecasting and Social Change*, *161*, e120291. https://doi.org/10.1016/j.techfore.2020.120291