Supporting Expertise Location in Coding Phase of Software Development Process

J. R. Martínez, R. R. Palacio, A. Vizcaíno, J. Cortez and V. H. Menéndez

***Abstract*— Software organizations attempt to reduce the time and cost of software projects to maintain competition between organizations, one way to achieve this goal is by applying knowledge gained from previous projects or resources used by members of the organization. However, software developer teams generally do not benefit from this knowledge as it is rarely stored. This is a problem since to solve problems or doubts developers need to seek expertise (knowledge of the best level at the right time) to maintain the level of competition and the fact of not having information of this knowledge makes it difficult to identify and in some cases, this is lost when the provider leaves the company. This work addresses the *expertise* location problem through an agent-based architecture for coding phase of software development process, which was designed from the information obtained by a focal group and with a case study of the search for expertise in software development organizations.**

***Keywords*— coding phase; software engineering; expertise location; software agents.**

I. INTRODUCCIÓN

LAS organizaciones de software tratan de reducir los tiempos y costos de los proyectos de software con el fin de mantener el nivel de competencia, una manera de lograr este objetivo es mediante la reutilización de conocimientos adquiridos en proyectos anteriores [1, 2]. Dicho conocimiento según [3] puede encontrarse en las siguientes fuentes:

- *Artefactos*: se refiere al conocimiento que se encuentra en la documentación (e.g., libro de requerimientos, documento de visión), código fuente, repositorios donde los desarrolladores consultan documentos físicos o digitales para completar sus actividades diarias (e.g., blogs, manuales, marcadores, control de versiones) y en las herramientas para administrar los proyectos.
- *Personas:* se refiere a los individuos con cierto grado de conocimiento específico de un área que pudiera ser de utilidad para resolver alguna problemática en las actividades de desarrollo, siendo cualquier persona dentro de la organización una potencial fuente de conocimiento.

Sin embargo los equipos de desarrolladores generalmente no se benefician de este conocimiento, debido a que durante el proceso de desarrollo de un software se generan diferentes artefactos (e.g. requerimientos del sistema, módulos, componentes del software, manuales, etc.) y dichos artefactos no están conectados o relacionados con su/s creador/es (programadores, arquitectos de software, analistas, etc.), lo cual causa que el conocimiento sea más difícil de identificar, no esté disponible o se pierda cuando el proveedor deja la organización llevándose dicho conocimiento con él [4-6]. Esto es un inconveniente puesto que para resolver problemas o dudas los desarrolladores realizan búsquedas de *expertise* (conocimiento de mejor nivel en el momento adecuado) con el fin de mantener un nivel de competencia adecuado para este tipo de trabajo [7].

Esta problemática ha sido abordada por proyectos que se han enfocado a un solo tipo de fuente de conocimiento ya sea en personas o en artefactos. Sin embargo, los artefactos de una empresa son creados por desarrolladores que pueden facilitar el conocimiento a los colegas mediante la compartición de su conocimiento tácito (experiencia) y/o explícito (artefactos).

En este sentido, para abordar la problemática de administrar el conocimiento de los desarrolladores, es factible el uso de agentes inteligentes, ya que dichos agentes podrían acceder al conocimiento de otros colegas y compartir el propio. Esto es porque cada agente es un sistema informático que se encuentra en un ambiente y es capaz de una acción autónoma en este entorno con el fin de cumplir con sus objetivos [8]. Además, que los agentes según [9] tienen propiedades como: *reactividad* porque son capaces de percibir su entorno y responder de manera oportuna a los cambios que se producen en él; y *proactividad* porque son capaces de mostrar un comportamiento dirigido a un objetivo.

En este artículo se aborda el problema de localización de expertise en la fase de codificación en el desarrollo de software, a través de una arquitectura de agentes inteligentes, la cual fue diseñada a partir de la información recopilada tras la ejecución de grupos focales y un estudio de caso de la búsqueda de expertise en organizaciones de desarrollo de software.

El artículo se organiza de la siguiente manera: en la sección II se presentan los trabajos relacionados. Después, en la sección III se describe el método utilizado para este trabajo de investigación. La sección IV explica la arquitectura basada en agentes para apoyar la localización del expertise. En la sección V se muestran los resultados obtenidos. Y finalmente en la sección VI se presentan las conclusiones del trabajo.

J.R. Martínez, Instituto Tecnológico de Sonora (ITSON), Cd. Obregón, Sonora, México, joseramonmg26@gmail.com

R. R. Palacio, Instituto Tecnológico de Sonora (ITSON), Navojoa, Sonora, México, ramon.palacio@itson.edu.mx

A. Vizcaíno, Universidad Castilla-La Mancha (UCLM), Ciudad Real, España, aurora.vizcaino@uclm.es

J. Cortez, Instituto Tecnológico de Sonora (ITSON), Cd. Obregón, Sonora, México, joaquin.cortez@itson.edu.mx

V. H. Menéndez, Universidad Autónoma de Yucatán (UADY), Mérida, Yucatán, México, mdoming@gmail.com

II. TRABAJOS RELACIONADOS

La localización del expertise en el desarrollo de software es un tema complejo, ya que el conocimiento se encuentra en los desarrolladores de software (experiencia) y los recursos que utilizan para solucionar problemas en distintos repositorios. Esto ha provocado que se hayan realizado esfuerzos para gestionar la localización del expertise mediante el uso de agentes como los trabajos de [5, 10] donde a través de una arquitectura multi-agente se apoya la gestión de la información y el conocimiento que se genera durante cierto proceso del desarrollo de software. Dichos trabajos utilizan tecnologías basadas en web para apoyar la interacción entre los agentes utilizando diferentes técnicas de razonamiento para generar nuevo conocimiento a partir de información previa y de aprendizaje de su propia experiencia. Dicho razonamiento está basado en los distintos casos que sucedieron en algún proyecto anterior o previamente en el mismo, así como también en la experiencia documentada por los stakeholders. En ese mismo sentido otra propuesta que utiliza agentes es el trabajo de [11] que presenta una implementación inicial de un sistema que trabaja con agentes que ayudan a la localización del expertise (expertos) teniendo como tema de dominio el lenguaje de programación en java, donde los agentes se encargan de programar citas para el intercambio de conocimiento, detectar proactivamente cuándo se necesita ayuda, proporcionar información adicional durante la interacción y ajustar sus propios mecanismos de perfiles de acuerdo a los comentarios de los usuarios.

También existen plugins o complementos para entornos de desarrollo de software como SNIFF [12] que es un plugin para el entorno de desarrollo eclipse para el lenguaje de programación java, el cual se basa en la premisa de que las bibliotecas de sus clases están bien documentadas, lo que facilita la búsqueda de las clases disponibles del dominio de programación en java. En ese mismo sentido, Exemplar [13] es una herramienta para la búsqueda de proyectos de software de gran relevancia con el fin de reutilizar el código fuente. La búsqueda se realiza utilizando palabras clave del proyecto y la descripción del mismo, de manera que se hace una coincidencia entre las palabras y la descripción. También se encuentra BluePrint [14], el cual describe el diseño, implementación y evaluación de una interfaz de búsqueda web integrada al entorno de desarrollo Adobe Flex Builder que ayuda a los usuarios a localizar ejemplos de código de proyectos anteriores utilizando palabras clave (e.g., lenguaje de programación, framework, nombre de la clase y/o método).

Además, existen propuestas de sistemas de apoyo a la localización del *expertise* como QuME [15], que es un prototipo de interfaz web personalizada para usuarios de comunidades de ayuda en línea de java, el cual presenta un mecanismo para inferir el nivel de conocimiento en java de los usuarios, que se calcula utilizando parámetros como las preguntas que coloca en el foro, la frecuencia de respuesta, las palabras clave de su perfil y otros aspectos más que proporcionan el grado de *expertise* de la persona. En ese mismo sentido, Expertise Recommender [16] es un sistema de recomendación de expertos que utiliza una arquitectura de recomendación general que se basa en un estudio de campo de localización de experiencia, el cual se calcula de acuerdo a las necesidades del usuario y al entorno en el cual es solicitado dicho experto.

Finalmente, Knowledge Condensation tool [17], presenta un mecanismo de clasificación basado en el etiquetado manual (como en las redes sociales) realizado por los desarrolladores durante sus interacciones en medios electrónicos textuales no estructurados (como Skype or Instant Messaging). De esta forma, las interacciones se etiquetan y guardan en un repositorio, por lo que los miembros del equipo de trabajo pueden buscar conocimiento arquitectónico en función de las etiquetas de clasificación insertadas.

Los trabajos citados muestran cómo los investigadores han realizado distintos esfuerzos para localizar el expertise en ambientes de desarrollo de software. Algunos han permitido ubicar el nivel de conocimiento de las personas con base a parámetros específicos y el historial de conocimientos que se han compartido en un foro especializado. También se utiliza razonamiento para generar nuevo conocimiento a partir de un conocimiento base, plataformas de software para ubicar artefactos de los proyectos de desarrollo, sistemas que permiten la ubicación de personas expertas en tópicos específicos. Sin embargo, ninguno de los trabajos encontrados ha logrado integrar artefactos y expertos, pues dichos trabajos se limitan a la recolección del conocimiento para utilizarlo en ese momento sin compartirlo, de manera que nadie más puede tener acceso a él o conocer quién es el proveedor. La mayoría de los trabajos citados recolectan el expertise individual de los usuarios pero no trabajan en la manera de integrarlo y compartirlo para hacerlo accesible para todos los desarrolladores en una organización, además están enfocados a un solo lenguaje de programación y/o recurso (e.g. código reutilizado), esto es importante debido a que en el desarrollo de software la experiencia es un factor importante para las entregas a tiempo como lo resaltan en [3], pues se estaría apoyando de manera importante a la reutilización de conocimiento organizacional [7].

En la Tabla I se presenta un resumen de las herramientas encontradas para la localización del expertise en el desarrollo de software.

TABLA I. TRABAJOS RELACIONADOS LOCALIZACIÓN DEL EXPERTISE.

Propuesta	artefactos	personas
BluePrint [14]	X	
Agents for Expertise Location[11]		X
SNIFF [12]	X	
Exemplar [13]	X	
QuME [15]		X
Expertise Recommender		X
Arquitectura de agentes para proceso de mantenimiento[10]	X	
Exemplar [13]	X	
Arquitectura multi-agente de tres niveles [5]		X
Knowledge Condensation tool [17]	X	

III. BÚSQUEDA DEL EXPERTISE

En esta sección se presenta el modelo utilizado para este trabajo de investigación, el cual consistió en lo siguiente:

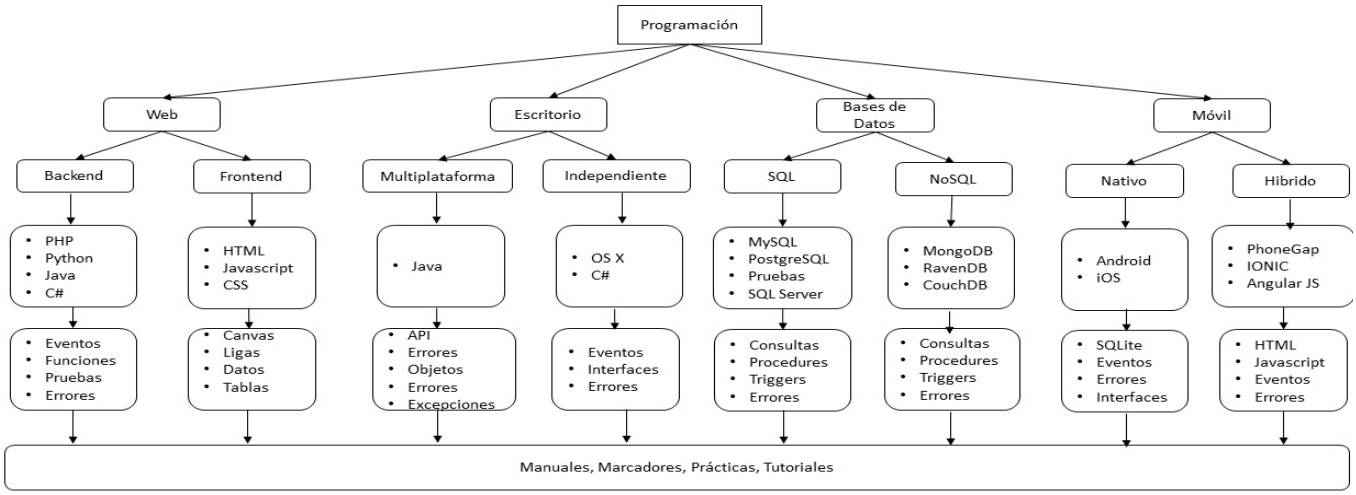

Figura 1. Clasificación del conocimiento del programador.

A. *Grupo focal*

Para realizar esta actividad se invitó a siete programadores que trabajan en una Fábrica de Software, los cuales tenían en promedio 3.1 años de experiencia (mín = 1; max = 10), el grupo focal fue diseñado para discutir tres tópicos principales, estos eran: i) forma en que los programadores buscan sus recursos (expertos o artefactos) para resolver problemáticas o dudas durante la construcción del software; ii) tipo de conocimiento y cómo almacenan el conocimiento; y iii) proceso de comunicación con el experto y criterios de selección.

A partir de los resultados del Grupo Focal se obtuvo la clasificación del conocimiento en el dominio de programación (ver Fig. 1) en cuatro niveles, específicamente en el dominio de programación de software.El primer nivel de la clasificación se relaciona con las plataformas que se pueden utilizar para desarrollar un proyecto (por ejemplo, Web, móvil, de escritorio, de base de datos), y luego el segundo nivel es el tipo de programación de acuerdo a la plataforma seleccionada (por ejemplo, backend, frontend), el tercer nivel es el lenguaje que se utiliza en la plataforma seleccionada (por ejemplo, Java, Python, PHP, etc.), el cuarto nivel es el tema relacionado con los conocimientos (palabras clave) y el último nivel es el formato en el que se representa el conocimiento (por ejemplo, manuales, tutoriales en vídeo, código fuente).

B. *Búsqueda de expertise en la fase de codificación mediante agentes inteligentes*

Basados en los resultados obtenidos del Grupo Focal se diseñó el proceso metodológico para representar el conocimiento durante la construcción de software, el cual se basa en 4 Fases: Extracción, Indentificación, Búsqueda y Recomendación (ver Fig. 2). Este modelo ha sido implementado mediante una arquitectura de agentes inteligentes que incluye herramientas para el almacenamiento, clasificación y búsqueda de expertise. Esta arquitectura facilita al usuario la recomendación del expertise sugerido ya sea de personas o artefactos mediante la ejecución secuencial de las distintas fases del modelo del proceso metodológico. Cada componente de la arquitectura de agentes implementa algunas de las fases del proceso de búsqueda. Las características de este modelo se describen a continuación:

- *Extracción:* el usuario proporciona la cadena de búsqueda que describe la necesidad o problema a resolver, de donde se extrae toda la información

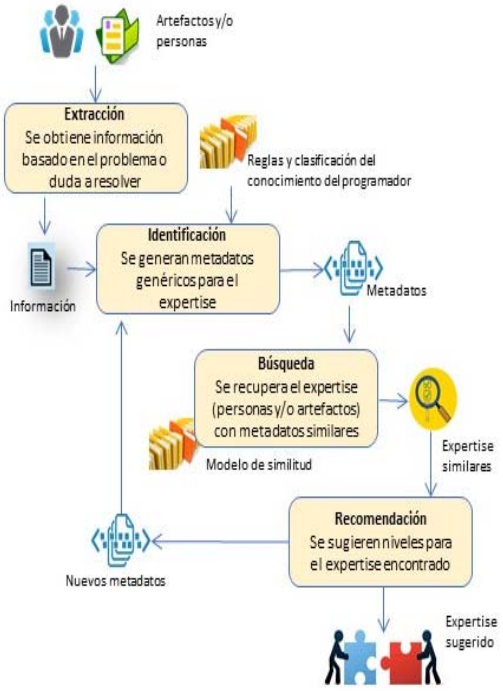

Figura 2. Proceso para la localización de expertise del programador de software.

textual.

- *Identificación:* esta fase presenta un conjunto de metadatos iniciales obtenidos de la información extraída. Metadatos como datos de la persona, tipo de conocimientos en programación, lenguaje, base de datos, ruta de almacenamiento de los recursos, entre otros. Para ciertos recursos (doc, pdf, etc.) se utilizan metadatos generados por el programa de edición creados al momento de almacenarlos (título, autor, versión etc.). A partir de un conjunto predefinido de reglas y la clasificación del conocimiento del programador se infiere quién es el dueño, el tipo de contenido, tipo de documento y datos de creación y modificación.
- *Búsqueda:* Los metadatos son identificados y comparados con los metadatos del expertise de los colegas para determinar su grado de similitud. Para cada expertise similar se hace una comparación de sus valores utilizando una medida de similitud de sus metadatos. Esto genera un conjunto de expertise ordenados por similitud a los valores propuestos por el usuario, que son presentados al usuario para su revisión y elección.
- *Recomendación:* Todos los metadatos encontrados son presentados en un formulario para la modificación de la búsqueda por parte del usuario. Los metadatos restantes pueden o no ser modificados para facilitar el proceso de identificación. Los resultados son ordenados con base al grado de similitud y pueden ser utilizados como parte del proceso. En cualquier momento es posible invocar al navegador de los metadatos para visualizar los valores que están almacenados en el repositorio.

C. Arquitectura de agentes

La arquitectura general del prototipo consistió de tres capas, las cuales son:

- *Cliente*, consiste en un formulario que permite la interacción del usuario con el sistema para hacer consultas de búsqueda de expertise.
- *Middleware*, permite registrar y realizar la ejecución de los métodos que permiten la extracción, identificación, búsqueda y sugerencias del sistema a partir de los recursos disponibles de los usuarios registrados. Esta parte está compuesta por las tecnologías de Java Agente Development Framework (JADE por sus siglas en inglés) java, y la API de MongoDB para hacer conexiones utilizando java.
- *Base de datos*, la arquitectura utiliza base de datos NoSQL mediante el gestor de bases de datos MongoDB.

El prototipo de la arquitectura está compuesto por los siguientes agentes inteligentes:

- *Agente Usuario* (UA): este agente cuenta con una interfaz gráfica que permite al usuario ingresar su conocimiento, crear un perfil, actualizar el conocimiento y realizar búsquedas, cumpliendo con la propiedad de *proactividad* de los agentes al buscar cumplir sus objetivos a través de un razonamiento basado en la búsqueda y captura de conocimiento de los desarrolladores.
- *Agente Central de Búsqueda* (CSA): encargado de realizar todos los registros, modificaciones y consultas en la base de datos con el conocimiento de todos los usuarios registrados en el sistema, cumpliendo con la propiedad de *reactividad* al estar al pendiente de todas las actualizaciones del conocimiento de los usuarios o del ingreso de nuevos usuarios al sistema.

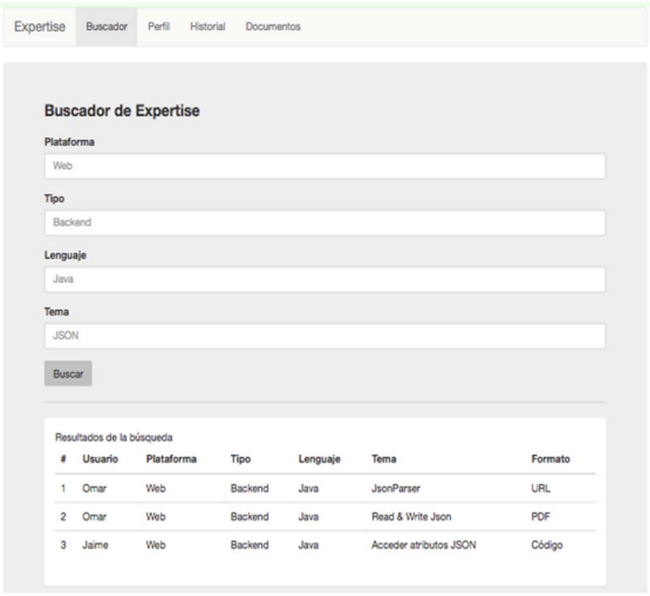

Figura 3. Interfaz gráfica de usuario.

La comunicación y la búsqueda de los objetivos de la arquitectura de agentes está basado en un formalismo, el cual consiste en un conjunto de secuencias ordenadas de datos (tuplas) de la forma:

$$K <P, T, L, S, F>$$

P = Plataforma donde $P \in$ {web, escritorio, bases de datos, móvil}
T = Tipo relacionado a la plataforma (e.g., P=web) $\rightarrow T \in$ {backend, frontend}
L = Lenguaje de programación (e.g., P=web, T=backend) $\rightarrow L \in$ {Java, PHP,Python,..}
S = Tema (e.g., P=web, T=backend, L=PHP) $\rightarrow S \in$ {GUI, Scripts, Eventos...}
F = Formato de representación del conocimiento $F \in$ {manuales, código fuente,..}

Como se han propuesto en otros trabajos [18, 19], estas tuplas dan la inteligencia a los agentes a través de una simulación del razonamiento de los desarrolladores. Dichas tuplas hacen uso de la clasificación del conocimiento de programación presentada anteriormente (ver Fig. 1) el cual se encuentra presente en los paneles del sistema que se presentan en la Fig. 3, ya sea para registrar un usuario, actualizar su conocimiento o realizar una búsqueda. En el caso de las

búsquedas se tiene como propósito encontrar una coincidencia total en la tupla, dado el caso de no ser así el agente toma una decisión con base en otros recursos que hayan utilizado los otros programadores y que les haya sido útil o que haya sido adaptado para sus necesidades.

El proceso de comunicación entre los agentes consiste en tres pasos: 1) se realiza la solicitud de una acción por parte de uno de los UA registrados en el sistema, 2) posteriormente CSA recibe la petición para poder realizar la acción correspondiente en la base de datos y 3) por último este a su vez da una respuesta a UA con respecto a la petición que solicitó. UA presenta una interfaz gráfica con 4 paneles, dichos paneles son: panel de perfil de usuario, panel de historial de proyectos, panel de documentos (aportaciones), panel de búsqueda de expertise (ver Fig. 3).

El panel de perfil de usuario permite registrar de manera básica los usuarios en el sistema. Ya registrado el usuario puede utilizar el panel de búsqueda para identificar el expertise disponible, el cual utiliza la clasificación de conocimiento propuesta para buscar recursos asociados al problema o duda que se presente.

En el panel de historial se guardan los proyectos en los que ha participado dicho usuario, esto para ir definiendo lenguajes de programación o especialidades de un programador con base en los proyectos que ha trabajado. Tanto el panel de perfil, el de búsqueda y el de historial son parte de la *extracción* e *identificación* del proceso de localización de expertise.

En el caso del panel de documentos es donde el usuario puede subir sus aportaciones de recursos que le hayan sido útiles para resolver algún problema durante uno de los proyectos en los que haya participado. Todo documento agregado (aportación) debe estar asociado a algún proyecto (mediante metadatos), así se indica para qué problema fue usado tal recurso con el fin de que otra persona pueda consultar el mismo recurso si se enfrenta a una problemática similar. Al agregar el documento se activa el comportamiento encargado de obtener los campos del documento, para que posteriormente dicho documento sea identificado como posible expertise que puede servir de apoyo a un proyecto actual o futuro. Este panel facilita la parte de *búsqueda* del proceso de localización de expertise que se ha definido.

El Agente Central de Búsqueda (CSA) tiene un comportamiento autónomo que está monitorizando las peticiones de los usuarios; primeramente, revisando los mensajes enviados por los UA, para posteriormente realizar las acciones y responder al agente solicitante. Es éste quien hace posible la recomendación de los expertise registrados en el sistema, y es éste quien hace posible que se ejecuten los procesos de *búsqueda* y *recomendación* del proceso metodológico de localización de expertise.

IV. EVALUACIÓN DE CASO DE ESTUDIO

El objetivo de esta evaluación fue percibir de los usuarios potenciales el grado de usabilidad de la arquitectura de agentes en comparación a la manera tradicional de localizar el expertise para resolver una problemática que les impida avanzar en sus actividades de codificación. Buscar el expertise indicado es una práctica normal o un hábito entre los programadores, lo cual consume parte del tiempo que destinan a sus actividades de trabajo, aun cuando el expertise continuamente lo tienen sus colegas. Para localizarlo, los desarrolladores utilizan diferentes medios de comunicación síncronos (e.g. mensajeros instantáneos, redes sociales) como asíncronos (e.g. wikis, blogs) que les permiten en primera instancia localizar y en segunda obtener determinados conocimientos para solucionar obstáculos que les impiden completar su actividad de trabajo.

Para medir la usabilidad de la arquitectura de agentes se utilizó la métrica SUS (System Usability Scale), la cual consta de un cuestionario de 10 reactivos (Tabla II) con cinco respuestas cada uno, que van desde "totalmente de acuerdo" hasta "totalmente en desacuerdo" [20]. Se utilizó esta prueba dado que es sencilla de administrar a los participantes, puede ser usada en pequeños grupos y puede diferenciar entre un sistema usable y uno que no lo es [21, 22].

A. Diseño del estudio

Se realizó un estudio comparativo donde los participantes del estudio tenían que localizar expertise utilizando tanto la arquitectura de agentes como los medios tradicionales que ellos utilizan (e.g. navegadores, marcadores, documentos, teléfono, mensajería instantánea y facebook).

El estudio fue desarrollado utilizando un paradigma intra-sujeto, de tal forma que todos los participantes realizaron ambas actividades. Con este paradigma se tiene la ventaja de garantizar que las mismas personas trabajan en las diferentes condiciones del estudio.

La variable independiente fue: localización del expertise (empleo de la arquitectura de agentes o los medios tradicionales). Las variables dependientes fueron:

- Usabilidad: con el fin de obtener la percepción con respecto a la facilidad de localizar el expertise más apropiado que pudiera ayudar a resolver la situación que disparó la necesidad del expertise.
- Tiempo de búsqueda: el fin era determinar el tiempo que tardan en localizar expertise apropiado para facilitar la solución a la situación presentada.

B. Participantes

El grupo de sujetos estuvo integrado por 22 programadores de software, de los cuales 10 trabajan en una Fábrica de Software, y 12 en departamentos de sistemas donde ejercían como programadores. Sus edades fluctuaban entre los 23 y 30 años de edad. Todos contaban con la licenciatura concluida y solo 3 de ellos contaba con estudios de posgrado. Los años de experiencia de los participantes como programadores era de 3.2 años (min= 1; max= 5).

C. Procedimiento

Los sujetos del estudio participaron en un taller interactivo de 2 horas presenciales en su lugar de trabajo, donde se explicó la arquitectura de agentes y su funcionamiento, posteriormente ellos utilizaron la herramienta para registrar sus datos personales, proyectos en los que han participado, y los documentos que les ayudaron a resolver problemáticas. Además, se describió la funcionalidad de la arquitectura de

agentes, especialmente enfocadas a la localización de expertise de personas y artefactos.

Posteriormente, todos los sujetos realizaron al menos 2 actividades de localización de expertise durante sus labores (uno con la arquitectura y otro de manera tradicional), y no existía un límite de tiempo para su realización.

Al finalizar cada actividad, los sujetos respondieron de manera anónima la encuesta de usabilidad SUS (System Usability Scale) [20] con el fin de conocer su percepción en cuanto a la usabilidad de cada medio para localizar el expertise. Adicionalmente se les solicitó a los participantes que agregaran comentarios indicando qué les motivaría a utilizar o no la arquitectura de agentes en comparación con la manera tradicional.

D. Limitaciones

El caso de estudio descrito y los métodos utilizados para evaluar las formas de localización de expertise podrían tener limitaciones. Algunos aspectos que pudieron haber afectado el resultado del estudio son:

- La cantidad de sujetos era pequeña, por lo que se pretende repetir el experimento con un grupo mayor de participantes.
- Las respuestas de los participantes pudieron ser subjetivas debido a que estuvieron basadas en su percepción de usabilidad.
- La fiabilidad de las respuestas pudo ser un factor de afectación, con el fin de evitar que los sujetos contesten lo que el investigador quiere escuchar se pidió que las respuestas fueran anónimas.
- La complejidad de las actividades era variable, ya que se buscó que el escenario que disparara la necesidad de localizar el expertise estuviera relacionado con su contexto de trabajo.

E. Resultados

Para el registro de la percepción de usabilidad de los usuarios se empleó la encuesta SUS, la cual consta de 10 reactivos utilizando una escala Likert de 5 puntos en un rango de 1 (Completamente en desacuerdo) hasta 5 (Completamente de acuerdo). La encuesta proporciona el nivel de usabilidad percibido con valores desde 1% a 100%. Se utilizó una prueba t-student con muestras relacionadas para comparar las medias obtenidas de los casos (uso de arquitectura y tradicional) y determinar si existían diferencias significativas entre ellas.

En la Tabla II se presentan los resultados obtenidos del instrumento SUS por cada pregunta y la ponderación media obtenida de las respuestas de los participantes.

Estos resultados sirven para probar si la arquitectura propuesta de los agentes es más útil que la manera tradicional de localizar expertise en el ambiente de trabajo de la codificación de software. En este caso, el promedio de la media de usabilidad de los medios tradicionales fue de 69.04%, mientras que la arquitectura de agentes fue de 81.72%. La prueba t-student para muestras relacionadas reveló la existencia de una diferencia estadística significativa entre el uso de la arquitectura de agentes ($t_{.05}$= -7.257; p= 0.000). Algunas sugerencias interesantes en cuanto a la experiencia con la arquitectura de agentes fueron: "… cuando me cambio de aplicación de repente me pierdo de lo que estaba haciendo…*[participante 3]*", "…mi experiencia con la aplicación fue buena, porque me evita andar preguntando o molestando a mis compañeros…*[participante 7]*".

TABLA II. RESULTADOS DE USABILIDAD.

Pregunta	Medios Tradicionales	Arq. Agentes
1. Creo que me gustaría usar este sistema frecuentemente.	3.28 (1.37)	3.85 (1.02)
2. El sistema me pareció innecesariamente complejo.	2.09 (1.02)	1.60 (1.20)
3. Pienso que el sistema es sencillo de utilizar	4.00 (1.00)	4.20 (0.67)
4. Creo que necesitaría la ayuda de un técnico para poder usar este sistema.	1.56 (1.01)	1.54 (1.04)
5. Me pareció que las distintas funciones de este sistema estaban bien integradas.	2.59 (1.07)	3.79 (1.20)
6. Me pareció que había demasiadas incoherencias en este sistema.	2.00 (0.95)	1.90 (0.45)
7. Creo que casi todo el mundo aprendería a usar este sistema rápidamente.	3.88 (0.78)	4.78 (1.18)
8. Me pareció que el sistema es muy complicado de usar.	2.56 (1.04)	2.00 (0.90)
9. Me sentí muy cómodo utilizando el sistema.	3.56 (1.16)	4.10 (0.80)
10. Tuve que aprender muchas cosas antes de poder utilizar este sistema.	1.25(0.67)	1.10 (0.40)
Grado de usabilidad media	**69.04 (8.09)**	**81.72 (3.26)**

En relación a la localización de expertise se encontró que la media de tiempo invertido en la modalidad de medios tradicionales fue de 28.4 minutos y una desviación estándar de 15.09 minutos (min=9; max=65). En cambio con la arquitectura de agentes los participantes encontraron el expertise en un tiempo promedio de 5.95 minutos y una desviación estándar de 2.11 minutos (min=2; max=9). Como se puede notar la diferencia en tiempo es mucho menor con el uso de la arquitectura de agentes pues ayuda considerablemente a disminuir el tiempo invertido en la localización del expertise ($t_{.05}$ = 6.928; p= 0.000). Algunos comentarios con respecto a esto fueron "…esto me ayuda a saber si ya alguien hizo lo que necesito, pero fuera mejor si me ayudara a contactar a mi colega…[*participante 12]*", "… si esto estuviera funcionando aquí en la empresa me ahorraría tiempo … *[participante 3]*".

V. CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se presentó una arquitectura basada en agentes para localizar el expertise (artefactos o personas) en el ámbito de la construcción del software. Dicha arquitectura ha sido implementada en el ambiente de trabajo de programadores de software. La aceptación de la arquitectura ha sido aceptable ya que su uso ha sido sencillo para los usuarios, principalmente porque les ayuda a disminuir el tiempo de búsqueda con respecto al expertise que se encuentra entre sus colegas.

Podemos concluir que la usabilidad de la arquitectura para la localización del expertise entre programadores ha sido

aceptable al estar la media en 81.70 %. Sin embargo se identificaron algunos aspectos a mejorar, por lo que se pueden resaltar los siguientes puntos:

- Modificar la forma de acceder, se sugiere sea empotrada en los IDE para programar, y no sea una aplicación independiente, ya que tal y como está ahora los usuarios se salen del contexto del problema al salir del sistema y podría significar una distracción.
- Contemplar el contacto directo con los colegas a través de la misma arquitectura y además incluir la disponibilidad de estos para interactuar.
- Establecer mecanismos de búsqueda más avanzados (e.g, realizar búsquedas también en google) para que los usuarios expertos puedan tener acceso a artefactos externos a la construcción del software.

REFERENCIAS

[1] P. A. Nielsen and K. Kautz, *Software Processes & Knowledge: Beyond Conventional Software Process Improvement*. Aalborg, Denkmark: Software Innovation Publisher, 2008.

[2] K. Kautz and K. Thaysen, "Knowledge, learning and IT support in a small software company," *Journal of Knowledge Management,* vol. 5, pp. 349-357, 2001.

[3] I. Becerra-Fernandez and R. Sabherwal, *Knowledge management: systems and processes*: ME Sharpe, 2010.

[4] R. S. Pressman, *Software Engineering: A Practitioner's Approach, 7/e*: Mc Graw-Hill, 2009.

[5] J. P. Soto, A. Vizcaíno, and M. Piattini, "Fostering Knowledge Reuse in Communities of Practice by Using a Trust Model and Agents.," *International Journal of Information Technology and Decision Making,* vol. 16, pp. 1409-, 2017.

[6] G. Borrego, A. L. Morán, R. R. Palacio, and O. M. Rodríguez-Elias, "Understanding Architectural Knowledge Sharing in AGSD Teams: An Empirical Study.," presented at the 11th IEEE International Conference on Global Software Engineering, ICGSE 2016,, Orange County, CA, USA, August 2-5, 2016, 2016.

[7] K. A. Ericsson, M. J. Prietula, and E. T. Cokely, "The making of an expert," *Harvard Business Review,* vol. 85, p. 114, 2007.

[8] N. R. Jennings, "On agent-based software engineering," *Artificial intelligence,* vol. 117, pp. 277-296, 2000.

[9] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice.," *The knowledge engineering review,* vol. 10, pp. 115-152, 1995.

[10] O. M. Rodríguez, A. Vizcaíno, A. I. Martínez, M. Piattini, and J. Favela, "How to manage knowledge in the software maintenance process," in *Advances in Learning Software Organizations*, ed: Springer, 2004, pp. 78-87.

[11] A. Vivacqua, "Agents for expertise location," in *AAAI Spring Symposium Workshop on Intelligent Agents in Cyberspace*, 1999, pp. 9-13.

[12] S. Chatterjee, S. Juvekar, and K. Sen, "Sniff: A search engine for java using free-form queries," in *Fundamental Approaches to Software Engineering*, ed: Springer, 2009, pp. 385-400.

[13] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby, "A search engine for finding highly relevant applications," in *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, 2010, pp. 475-484.

[14] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: integrating web search into the development environment," in *SIGCHI Conference on Human Factors in Computing Systems*, 2010, pp. 513-522.

[15] J. Zhang, M. S. Ackerman, L. Adamic, and K. K. Nam, "QuME: a mechanism to support expertise finding in online help-seeking communities," in *20th annual ACM symposium on User interface software and technology*, 2007, pp. 111-114.

[16] D. W. McDonald and M. S. Ackerman, "Expertise Recommender: A Flexible Recommendation System and Architecture," in *ACM Conference on Computer Supported Cooperative Work*, 2000, pp. 231-240.

[17] G. Borrego, A. L. Morán, and R. R. Palacio, "Preliminary Evaluation of a Tag-Based Knowledge Condensation Tool in Agile and Distributed Teams," presented at the 12th IEEE International Conference on Global Software Engineering, ICGSE 2017, Buenos Aires, Argentina, 2017.

[18] L. F. Díez, A. Valencia, and J. Bermúdez, "Agent-based Model for the Analysis of Technological Acceptance of Mobile Learning," *IEEE LATIN AMERICA TRANSACTIONS,* vol. 15, pp. 1121-1127, 2017.

[19] K. Raya-Díaz, C. Gaxiola-Pacheco, and C. o. n.-P. M., "Agent-Based Model for Self-Management of Network Flows using Negotiation," *IEEE LATIN AMERICA TRANSACTIONS,* vol. 16, pp. 210-215, 2018.

[20] J. Brooke, "SUS-A quick and dirty usability scale," *Usability evaluation in industry,* vol. 189, pp. 4-7, 1996.

[21] A. Bangor, J. Miller, and P. Kortum, "Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale," *Journal of Usability Studies,* vol. 4, pp. 114-123, 2009.

[22] usability.gov. (July 5th, 2018). *System Usability Scale (SUS)*. Available: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

Martínez G. José R. was born in Navojoa, Sonora, México in 1989. He received his B.S. degree in Software Engineering from Instituto Tecnológico de Sonora, in 2014. his M.S. degree in Computer Science from the Instituto Tecnológico de Sonora, in 2016. Currently, he is studying a PhD degree in Computer Science at Universidad Autónoma de San Luis Potosi. His research interests include software engineering, knowledge management, agents and human computer interaction.

Palacio, Ramón R. is a professor of Software Engineering at Instituto Tecnológico de Sonora (ITSON) in Navojoa, Sonora, México, where he is member of Networks and Intelligent Systems research group. His research interests include Computer Network, Human-Computer Interaction (HCI), and Software Engineering. With his research group, he conducts studies to gain a better understanding of current work practices and based on this understanding, designs, develops and evaluates appropriate technologies for diverse work environments. He holds a Ph.D. in Computer Sciences from Universidad Autónoma de Baja California (UABC), México.

Vizcaíno, Aurora is a professor of Software Engeneering at University of Castilla-La Mancha. She is member of Alarcos Research Group. Her research interest include Global Software Engineering, Knowlege Management, Software Agents. She holds a Ph.D. in Informatics from Universidad de Castilla La-Mancha (UCLM), Spain.

Cortez, Joaquin, he received the Master's degree in electric engineering from CINVESTAV Guadalajara, in 2001 and the Ph. D. degree in electric engineering from CINVESTAV-Guadalajara, in 2008. He is currently an electrical and electronics engineering professor at the Technologic Institute of Sonora,. His teaching and research interests include digital communications and digital signal processing and HumanComputer Interaction.

Menéndez, Víctor H. is professor of Software Engineering at Universidad Autónoma de Yucatán (UADY) in Mérida, Yucatán, México, where he is member of Software Engineering and Educational Computing research group. His research interests include Information Search and Retrieval, Human-Computer Interaction (HCI), and Software Engineering. With his research group, he conducts studies to improve functionalities relatives to use of repositories for diverse work environments. He holds a Ph.D. in Advanced Computer Technologies from Universidad de Castilla La-Mancha (UCLM), Spain.